EXACT AND APPROXIMATE ALGORITHMS FOR THE CALCULATION OF SHORTEST PATHS

By

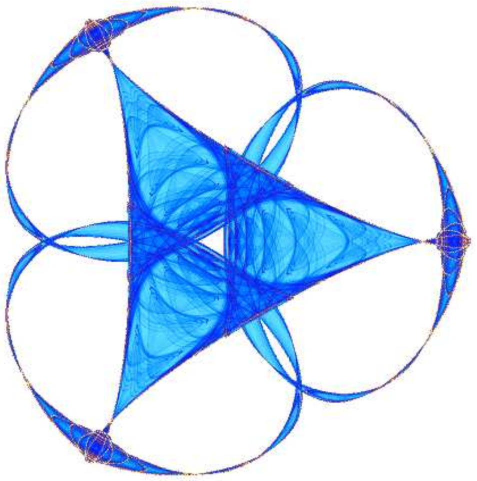
Fajie Li

and

Reinhard Klette

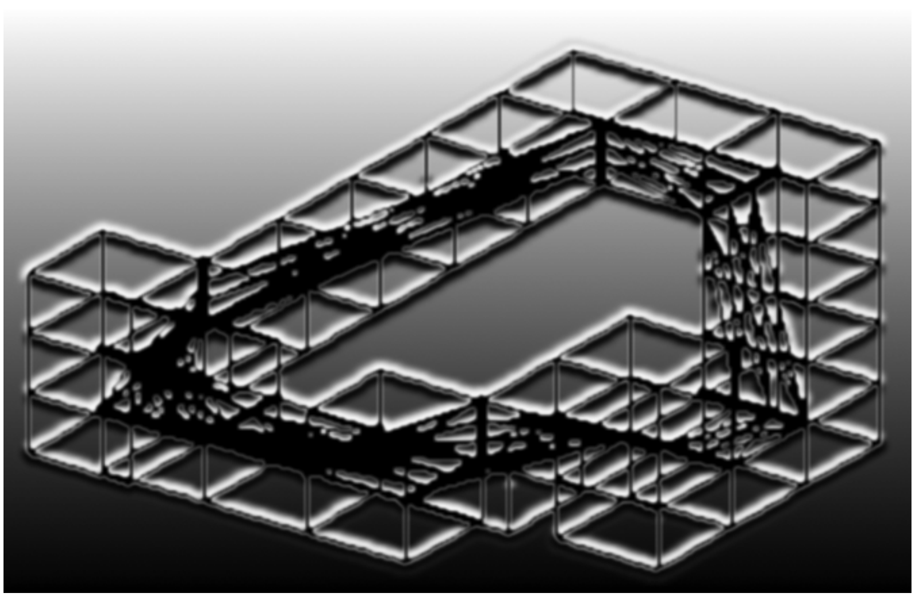
IMA Preprint Series # 2141

(October 2006)



Exact and Approximate Algorithms for the Calculation of Shortest Paths

Fajie Li and Reinhard Klette



Fajie Li
Reinhard Klette

Department of Computer Science
The University of Auckland
New Zealand

31 October 2006

(updated: 10 April 2007)

Ptolemy once asked Euclid whether there was any shorter way to a knowledge of geometry than by a study of the Elements, whereupon Euclid answered that there was no royal road to geometry.

Proclus Diadochus

Abstract

Keywords: Euclidean shortest paths, rubberband algorithm, computational geometry, digital geometry, cube curves, art gallery problems.

This report provides a complete discussion of an algorithm (called *rubberband algorithm*, which was proposed in 2000 for the calculation of minimum-length polygonal curves in cube curves in 3D space. The report transforms it "step-by-step" into a general, provably correct, and time-efficient algorithm which solves the indented task for simple cube curves of any complexity. Variations of this algorithm are then used to solve different Euclidean shortest path (ESP) problems, such as calculating the ESP inside a simple cube arc, inside a simple polygon, on the surface of a convex polytope, or inside a simply connected polyhedron. Finally, the developed methodology of rubberband algorithms is applied to improve various results of best algorithms for the touring polygons, parts cutting, safari and zookeper, and the watchman route problem.

Contents

Chapter 1

Introduction

This introductory chapter informs about the general context (in computational geometry and in the field of approximation algorithms), preliminaries for discussing either exact or approximation algorithms, and the original rubberband algorithm.

1.1 Exact versus Approximation Algorithms

An *algorithm* is any finite number of steps which are formulated with respect to a finite set of executable (with respect to a defined model of computation) operations. This report is about approximation algorithms for shortest path problems.

As common in algorithmic complexity theory, we assume that the performance of each basic operation requires one time unit. This report is only concerned with upper time bounds $\mathcal{O}(f(n))$, which represent all *time complexities* g with

$$g(n) \leq c \cdot f(n)$$

for some *asymptotic constant* $c > 0$ and for all $n \geq n_0$. Parameter n characterizes the size of the input data.

In general, an algorithm is also specified by a finite set of *free parameters* (such as an upper limit for the number of runs through a loop, or an approximation parameter ε), and we speak about *one* algorithm, which allows different parameter settings, rather than about a class of algorithms, all just different with respect to another parameter setting. In this sense, free parameters have to be considered as well when specifying time complexities.

The term *exact algorithm* seems to be clear to everybody: there is no difference between the solution obtained by the algorithm and the true solution, for each input instance to the algorithm. For example, an algorithm for mapping a two-dimensional (2D) or three-dimensional (3D) binary picture into topologically equivalent "skeletal curves" is based on "thinning", and there are many different proposals for elementary thinning steps (see, e.g., [85]), but the final result is "exact" in the sense that it follows the rules applied when designing the algorithm. The situation becomes more simple if only one solution is considered to be true (e.g., when calculating the convex hull of a finite set of points).

1. DEFINITION. *An algorithm is* exact *if there exists a criterion independent of the algorithm itself which characterizes an output to be true, and solutions obtained by the algorithm are true (i.e., 100% correct), for each input instance of the algorithm.*

On the opposite, materials on approximation algorithms, such as the books [13, 67, 74, 102, 149] and the web site [120], are about approximation rather than about exactitude. If we are dealing with a *Euclidean shortest path* (ESP) problem then there might be a true solution (e.g., is there a uniquely determined ESP between two given points in a given 3D manifold?); but the question arises whether we are actually able to obtain or express it; possibly, any algorithm is only (!) able to "approximate" the true solution?

For this report we apply the following definition of an *arithmetic algorithm*, motivated by the arithmetic nature of the shortest path problems in general:

2. DEFINITION. *An* arithmetic algorithm *consists of a finite number of steps of arithmetic operations, possibly also using input parameters from the field $\mathbb{Q}$ of rational numbers, using only the following basic operators: $+, -, \cdot, /$ or the k-th root, for $k \geq 2$.*

For example, let $p(x) = 84x^6 - 228x^5 + 361x^4 + 20x^3 + 210x^2 - 200x + 25$. This polynomial will be of importance for our geometric studies, and we prove in Chapter 7 (by applying Bajaj's theorem from 1988 and Berlekamp's factorization algorithm from 1967) that the problem of finding the roots of $p(x)$ is not solvable by radicals[1] over the field of rationals $\mathbb{Q}$. (Note that this is different from an approximation caused by rounding.) That is, the roots of $p(x)$ cannot be expressed in terms of elementary arithmetic operations $+, -, \cdot, /$ or the k-th root over the field $\mathbb{Q}$. In other words, there does not exist an arithmetic formula for a solution of the equation $p(x) = 0$.

It follows that there does not exist an exact arithmetic algorithm for finding the roots of the equation $p(x) = 0$. In other words, any arithmetic algorithm for finding the roots of polynomials cannot be called an exact algorithm unless it is able to convert (!) its output into a format which represents the true (complex-valued) roots of $p(x)$.

[1]A real number α is expressible in terms of radicals if there exist a sequence of expressions $\beta_1, \ldots,$ β_n such that $\beta_1 \in \mathbb{Q}$, and each $\beta_i \in \mathbb{Q}$ equals the sum, difference, product, quotient, or the k-th root of preceding β's; finally, $\beta_n = \alpha$ [16].

1. EXAMPLE. *We illustrate a final step of conversion into a true solution which is not part of the arithmetic algorithm. For example, we apply an algorithm to find a root of the equation*

$$\cos x = 0$$

where $x \in$ [1, 2]. The algorithm will output a real number close to the true root to be an approximative solution. We know that the exact solution is the intersection point between curve $y = \cos x$ and x-axis, which is at $x = \pi/2$. So we can convert an approximative solution into the true solution $x = \pi/2$.

3. DEFINITION. *An arithmetic algorithm is* eventually exact *if it provides also final (not necessarily arithmetic) steps for converting its approximate solution into the true solution.*

The $2^{2^{\mathcal{O}(n)}}$ and $2^{n^{\mathcal{O}(1)}}$ algorithms for ESP calculations, described in [125] and [121], respectively, are regarded as exact algorithms by [106], without specifying ways how to convert their outputs into true solutions. (Approximation algorithms for ESP calculations are specified in [106] in general without identifying final algorithmic steps how to convert their outputs into true solutions.)

This report focuses on minimization problems. For this class of problems, the following definition (see, e.g., [74]) is in common use:

4. DEFINITION. *An algorithm is an δ-*approximation algorithm*for a minimization problem P iff, for each input instance I of P, the algorithm delivers a solution that is at most δ times the optimum solution.*

Obviously, $\delta < 1$ is impossible, $\delta = 1$ defines an exact algorithm, and, for example, $\delta = 1.1$ defines a general (!) error limit of 10%. - We also propose the use of the following definitions, and will show their usefulness in this report.

5. DEFINITION. *An algorithm is* without guarantee *(of exactitude) if there exists a real number $\varepsilon_0 > 0$ and an input instance such that the difference – between the output for this input and the true solution – is larger than ε_0, for any choice of the algorithm's parameters.*

Note that an δ-approximation algorithm can also be without guarantee.

2. EXAMPLE. *[73] describes a so-called "2-approximation linear algorithm" for calculating a shortest path on the surface of a convex polytope, which is an example of an algorithm without guarantee.*

6. DEFINITION. *An algorithm is* within guaranteed error limits *if for each $\varepsilon > 0$ and each input instance, there is a set of parameters for this algorithm such that the deference between output for this input and true solution is smaller than ε.*

3. EXAMPLE. *The historically first δ-approximation (where $\delta = 1 + \varepsilon$) algorithm for the general* 3D ESP problem *is due to [119], and it is an algorithm within guaranteed error limits. It can produce a path guaranteed to be no longer than (1 + ε) times the length of the true ESP. It has the time complexity*

$$\mathcal{O}\left(\frac{n^4\left[b + log(n/\varepsilon)\right]^2}{\varepsilon^2}\right)$$

where b is the number of bits representing the coordinates of the vertices in the domain Π*, and n is the number of edges of* Π*.*

An algorithm runs in $h(\varepsilon, n)$ time iff its time complexity is in $\mathcal{O}(h(\varepsilon, n))$, where n specifies the problem size (possibly by combining multiple parameters) and $\varepsilon > 0$ the accuracy.

Example 3 already shows that the separation of time complexity $h(\varepsilon, n)$ into a product of a function f, which is independent of ε, and a function κ, which is independent of n, is not always straightforward (or even not possible).

7. DEFINITION. *We call an algorithm* κ-linear *iff its time complexity is in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$, and function κ does not depend on the problem size n.*

Because this report is focusing on algorithms for shortest path problems, we use the term *solution* for a concrete path which is completely contained in the specified domain.

The following definition differs from those in [13, 67, 102]; it specifies a special class of *iterative algorithms* which are defined by performing at first a constant number of steps before running into a loop, where the stop of the algorithm is controlled by one *accuracy parameter* $\varepsilon > 0$. Let L_n be the length of a solution obtained as a result of the n-th iteration.

8. DEFINITION. *An iterative algorithm is* approximative *(or* approximate*) iff*[2] *, for each $\varepsilon > 0$, there exists a natural number n_ε such that, if $n \geq n_\varepsilon$ then*

$$|L_{n+1} - L_n| < \varepsilon$$

By this definition, the family of approximative (or approximate) algorithms is a subset of the family of δ-approximation algorithms as considered in [74]. (Other books, such as [13, 67, 74, 102, 120], also consider so-called "absolute" or "relative approximation", and so forth, which are schemes which are basically not much different to the concept of δ-approximation; for this reason we will not recall these

[2]Read "if and only if".

concepts, and use δ-approximation as *the* concept of approximation algorithms in general.)

This report reports about algorithms; some of them use *procedures* in the sense of subroutines. Every algorithm or procedure in this report will be classified whether being, exact, eventually exact, or approximative. Note that algorithms within guaranteed error limits are presented in this report in form of iterative approximate algorithms.

1.2 MLPs and ESPs

A cube-curve g is a loop of face-connected grid cubes in the 3D orthogonal grid; the union g of those cubes defines the *tube* of g. The report discusses ESPs in such tubes, which are defined by *minimum-length polygonal* (MLP) *curves* (see Figure 1.1).

The general *ESP problem* is as follows:

9. DEFINITION. *Given a Euclidean space which contains (closed) polyhedral obstacles, denoted by* $O = \{o_1, o_2, \ldots, o_n\}$*; compute a path, denoted by* ρ*, which (i) connects two given points, denoted by* p *and* q*, in the space, (ii) does not intersect the interior of any obstacle, and (iii) is of minimum Euclidean length.* ρ *is called a* (general) solution *to the ESP problem with respect to* O*,* p *and* q*.*

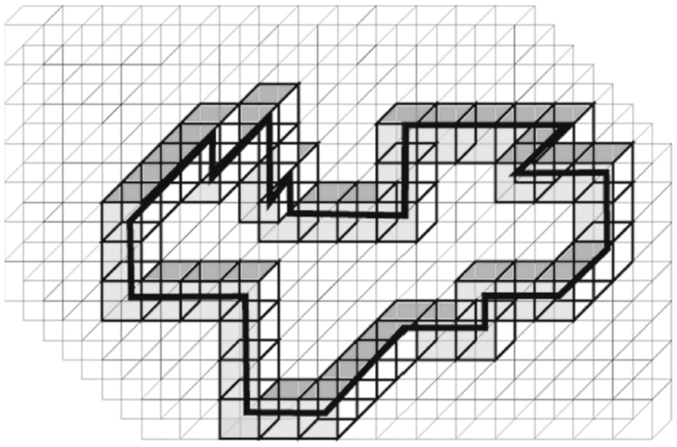


Figure 1.1: A cuboidal world: the bold curve is an initial guess for a 3D walk (or flight) through the given loop of shaded cubes. The length of the 3D walk needs to be minimized, which defines the MLP.

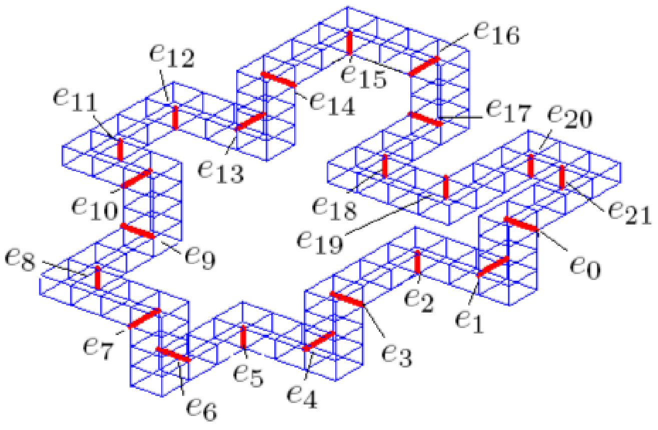


Figure 1.2: Example of a first-class simple cube-curve which has middle and end angles.

Let $O' = \{o'_1, o'_2, \ldots, o'_{n'}\}$ be a set of (closed) polyhedral obstacles such that for each $o_i \in O$, there exists $o'_{i'} \in O'$ such that $o_i \subseteq o'_{i'}$; let ρ' be a general solution to the ESP problem with respect to O', p and q. ρ' is called a restricted solution *to the ESP problem with respect to O, p and q.*

Finding a general solution to the general ESP problem (starting with dimension 3) is known to be NP-hard [27].

There are arithmetic δ-approximation algorithms solving the Euclidean shortest path problem in 3D in polynomial time, see [43]. Shortest paths or path planning in todays 3D robotics (see, for example, [92], or the annual ICRA conferences in general) seems to be dominated by heuristics rather than by general geometric algorithms. If a cuboidal world can be assumed then this report provides two general and κ-linear approximate shortest path algorithms (where the problem size n equals the number of critical edges in the given cube curve).

10. DEFINITION. *A critical edge of a cube-curve g is such a grid edge which is incident with exactly three different cubes contained in g.*

Figure 1.2 shows all the critical edges of a simple cube-curve.

3D MLP calculations generalize MLP computations in 2D; see, for example, [78, 127] for theoretical results and [52, 135] for 2D robotics scenarios. Shortest curve calculations in image analysis also use graph metrics instead of the Euclidean metric; see, for example, [130].

Interest in 3D MLPs was also raised by the issue of multigrid-convergent length estimation for digitized curves. The length of a simple cube-curve in 3D Euclidean space can be defined by that of the MLP; see [85, 128, 129], which is there characterized to be a *global approach* towards length measurement. A *local approach* for 3D length estimation, allowing only weighted steps within a restricted neighborhood, were considered in [76] and [77]. Alternatively to the MLP, the length of 3D digital curves can also be measured (within time linear in the number of grid points on the curve) based on DSS-approximations [46] (DSS = digital straight segment).

The computation of 3D MLPs was first time published in [22, 23, 24, 83], proposing a 'rubberband' algorithm[3]. This iterative algorithm was experimentally tested and showed "linear run-time behavior" with respect to a pre-selected accuracy constant $\varepsilon > 0$. It proved to be correct for tested inputs, where correctness was possible to be tested manually. However, in [22, 23, 24, 83], no mathematical proof was given for linear run time or general convergence (in the sense of our definition of approximate algorithms) to the exact solution.

This *original rubberband algorithm* is also published in the the book [85]. Recent applications of this algorithm are in 3D medical imaging; see, for example, [54, 151]. (The following section presents the original rubberband algorithm.)

The correctness and linearity problem of the original rubberband algorithm was approached along the following steps:

[93] only considered a very special class of simple cube-curves and developed a provable correct MLP algorithm for this class. The main idea was to decompose a cube-curve of that class into arcs at "end angles" (see Definition 3 in [93]), that means, the cube-curves have to have end-angles, then the algorithm can be applied.

[94] constructed an example of a simple cube-curve whose MLP does not have any of its vertices at a corner of a grid cube. It follows that any cube-curve with this property does not have any end angle, and this means that we cannot use the MLP algorithm as proposed in [93]. This was the basic importance of the result in [94]: we showed the existence of cube-curves which require further algorithmic studies.

[96] showed that the original rubberband algorithm requires a modification (in its Option 3) to guarantee that calculated curves are always contained in the tube g. This corrected rubberband algorithm achieves (as the original rubberband algorithm) minimization of length by moving vertices along critical edges.

[97] (finally) extended the corrected rubberband algorithm into the *edge-based rubberband algorithm* and showed that it is correct for any (!) simple cube-curve. [97] also presented a totally new algorithm, the *face-based rubberband algorithm*, and showed that it is also correct for any simple cube-curve. It was proved that both, the edge-based and the face-based rubberband algorithm, have time complexity in

[3]Not to be confused with a 2D image segmentation algorithm of the same name [99].

$\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where m is the number of critical edges in the given simple cube-curve, and

$$\kappa(\varepsilon) = (L_0 - L)/\varepsilon \tag{1.1}$$

where L_0 is the length of the initial path, L is the true (i.e., optimum) path. This report will report about those publications, as well as about applications of the rubberband algorithm (and its "derivatives").

1.3 The Original Rubberband Algorithm

Let $P_t = (p_0, p_1, \cdots, p_m)$ be a polygonal curve contained in a tube **g**. A polygonal curve Q is a *g-transform* of P iff Q may be obtained from P by a finite number of steps, where each step is a replacement of a triple a, b, c of vertices by a polygonal sequence $a, b_1, \cdots, b_k, c$ such that the polygonal sequence $a, b_1, \cdots, b_k, c$ is contained in the same set of cubes of g as the polygonal sequence a, b, c.

Assume a polygonal curve $P_t = (p_0, p_1, \cdots, p_m)$ and three pointers addressing vertices at positions $i-1, i$ and $i+1$ in this curve. There are three different *options* that may occur, and which define a specific g-transform:

(O_1) Point p_i can be deleted iff $p_{i-1}p_{i+1}$ is a line segment within the tube. Then the subsequence (p_{i-1}, p_i, p_{i+1}) is replaced in the curve by (p_{i-1}, p_{i+1}). In this case, the algorithm continues with vertices $p_{i-1}, p_{i+1}, p_{i+2}$.

(O_2) The closed triangular region $\triangle(p_{i-1}, p_i, p_{i+1})$ intersects more than just three critical edges of p_{i-1}, p_i and p_{i+1} (i.e., a simple deletion of p_i would not be sufficient anymore). This situation is solved by calculating a convex arc and by replacing point p_i by a sequence of vertices $q_1, \cdots, q_k$ on this convex arc between p_{i-1} and p_{i+1} such that the sequence of line segments $p_{i-1}q_1, \ldots, q_kp_{i+1}$ lies within the tube. In this case, the algorithm continues with a triple of vertices starting with the calculated new vertex q_k.

If (O_1) and (O_2) do not lead to any change, the third option may lead to an improvement (i.e., a shorter polygonal curve which is still contained and complete in the given tube):

(O_3) Point p_i may be moved on its critical edge to obtain an optimum position p_{new} minimizing the total length of both line segments $p_{i-1}p_{new}$ and $p_{new}p_{i+1}$. First, find $p_{opt} \in l_e$ such that $|p_{opt} - p_{i-1}| + |p_{opt} - p_{i+1}| = min_{p \in l_e} L(p)$ with $L(p) = |p - p_{i-1}| + |p - p_{i+1}|$. Then, if p_{opt} lies on the closed critical edge e, let $p_{new} = p_{opt}$. Otherwise, let p_{new} be that vertex bounding e and lying closest to p_{opt}.

Option 3 of this *original rubberband algorithm* is not asking for testing inclusion of the generated new segments within tube **g**. We realized that this test needs to be added.

1.4 Preliminaries

We use a few basic definitions and theorems of mathematical analysis (see, e.g., [17]), and state them here for completeness.

11. DEFINITION. *A sequence $X = (x_n)$ of real numbers is said to be a* Cauchy sequence *iff, for every $\varepsilon > 0$, there exists a natural number $H(\varepsilon)$ such that for all natural numbers n, $m \geq H(\varepsilon)$, the terms x_n, x_m satisfy $|x_n - x_m| < \varepsilon$.*

1. THEOREM. (Cauchy Convergence Criterion) *A sequence of real numbers is convergent iff it is a Cauchy sequence.*

2. THEOREM. (Monotone Convergence Theorem) *A monotone sequence of real numbers is convergent iff it is bounded.*

Convergence statements about our approximative algorithms are always based on the use of Theorem 2 and the more specific

1. COROLLARY. *A monotonously decreasing sequence of real numbers is convergent iff it is lower bounded.*

All our approximative algorithms apply Theorem 1 for deciding when to terminate: assume an accuracy parameter $\varepsilon > 0$ and let L_n be the solution obtained after the nth iteration. Then the algorithm will stop if $|L_{n+1} - L_n| < \varepsilon$; otherwise the algorithm goes into its next iteration.

One of the most striking properties of a convex function (see [20, 122, 123]) is that each local minimum is also a global minimum. Based upon this we will show that the solutions obtained by our algorithms (for solving the ESP problem for simple cube curves) are always the global solutions.

In generally, it is non-trivial to prove the uniqueness of a solution [148]. This implies that it is difficult (in general) to prove that a convex function is strictly convex since a strictly convex function will have a unique minimum [20, 122, 123]. We will apply basic theorems in topology to prove the uniqueness in Chapters 7 and 11.

1. LEMMA. ([26]) *A convex polygon is the shortest curve encircling a given finite set of planar points.*

Based on Lemma 1, we will frequently apply the *Melkman algorithm* for calculating the convex hull of a *simple polyline* (i.e., a finite planar set of vertices of a simple polygonal path or arc). It is not only linear in time complexity but also easy to implement [103, 133].

1.5 Structure of Report

In Chapter 2 we give two δ-approximation algorithms for computing approximate MLPs for simple cube-curves of "small size". They can be used to study the accuracy of the rubberband algorithm for small-sized inputs. The provided algorithms are actually impractical for big-sized inputs due to their time complexity $\mathcal{O}((mn)^4)$, where m is the number of subdivision points on each critical edge, and n is the number of critical edges of a given simple cube-curve.

Chapter 3 develops a δ-approximation algorithm to calculate MLPs for a special class of first-class simple cube-curves which can be decomposed into a number of simple cube-arcs.

In Chapter 4 we focus again on first-class simple cube-curves. For such a simple cube-curve, only Option 3 occurs in the original rubberband algorithm. We prove that the original rubberband algorithm will converge to the unique MLP of a given first-class simple cube-curve.

Chapter 5 still focuses on first-class simple cube-curves. We construct a non-trivial simple cube-curve such that each vertex of the MLP is not located at any endpoint of a critical edge of the given simple cube-curve. (This example answered an open problem published in the book [85].) Furthermore, we found two counterexamples to Option 2 of the original rubberband algorithm. We slightly corrected Option 3 of the original rubberband algorithm. We also prove that the original rubberband algorithm is κ-linear, with $\kappa(\varepsilon)$ as in Equation (1.1).

Chapter 6 finally applies the methodology developed in Chapter 4 to describe two provable correct approximate algorithms for computing the MLP of a general simple cube-curve.

Chapter 7 proves that there does not exist an exact algorithm for the general ESP problem. We also discuss the degenerate case (i.e., two or more vertices of the updated polygonal path are identical) of the simplified rubberband algorithm (which applies Option 3 of the original rubberband algorithm) and how to overcome such a case.

Chapters 8, 9 and 10 apply the basic idea of the original rubberband algorithm to solve some special but difficult ESP problems in 2D, on surfaces (restricted solution only), and in 3D domains.

Chapter 11 applies the basic idea of the original rubberband algorithm to solve some classic problems such as safari (restricted solution only), zookeeper (restricted solution only) and watchman route problems.

Chapters 12 concludes the report.

Chapter 2

An Algorithm with Guaranteed Error Limit

This chapter presents an arithmetic δ-approximation algorithm for computing the MLP of a general simple cube-curve. It has a time complexity in $\mathcal{O}(m^4n^4 + \kappa(\epsilon) \cdot n)$, where m is the number of possible subdivision points on any of the critical edges, and n is the total number of critical edges of a given simple cube-curve. Function $\kappa(\epsilon)$ is as in Equation (1.1).

2.1 Related Work

The problem of length estimation in picture analysis dates back to the beginning of the 1970's. For example, [108] presented a method for extracting a smooth polygonal contour from a digitized image aiming at a minimum-length polygonal approximation. For later work we cite [11, 14, 49, 53] as examples, which were focused on the problem in 2D. [84] compared the DSS and MLP techniques[1] for measuring the length of a digital curve in 2D (in fact, both techniques allow generalizations to 3D and beyond) based on time-efficient (linear) algorithms for both 2D techniques.

For the 3D case, [7] presents two methods for estimating the length of digitized straight lines. [36] gives four types of characterization schemes for this problem. [80] studies shortest paths between points on a digital 3D surface. [81] considers local length estimators for curves in 3D space, derived from their chain codes. [113] discusses the problem of approximating the length of a parametric curve $\gamma(t)$, with $\gamma : [0,1] \rightarrow \mathbb{R}^n$, from sampled points $q_i = \gamma(t_i)$, where the parameters t_i are unknown.

The computation of the length of a simple cube-curve in 3D Euclidean space was a subject in [77], but the proposed method may produce errors for specific curves. [24] presents the original rubberband algorithm (see Section 1.3) for computing an approximating MLP in the tube g of cube-curve g, with a measured run-time allowing to expect a general $\mathcal{O}(n)$ behavior, where n is the number of grid cubes in

[1]The *DSS technique* represents a digital curve, assumed to be a sequence of grid points, by subsequent DSSs of maximum length; the *MLP technique* considers a digital curve as a sequence of grid cells, calculating an MLP in the union of these cells.

the given cube-curve. However, no proof was given that this original rubberband algorithm actually converges towards the correct MLP, and also no analysis of its worst-case time complexity. This report (and related publications [96, 97]) provides for the first time proofs for the correctness and time complexity of this algorithm – however, after (minor, but necessary) modifications of the original rubberband algorithm.

This chapter presents two approximate algorithms for computing an MLP for a general simple cube-curve in $\kappa(\epsilon) \cdot \mathcal{O}(m^4 n^4)$ time, where $\kappa(\epsilon)$ is as defined in Equation (1.1), m is the number of possible subdivision points on any of the critical edges, and n is the total number of critical edges. The chapter is organized as follows: Section 2.2 provides used notations and theoretical results. Section 2.3 describes our algorithm and discusses its computational complexity. Section 2.4 gives the conclusions.

2.2 Basics

Following [24, 83], a grid point $(i, j, k) \in \mathbb{Z}^3$ is assumed to be the center point of a *grid cube*, with *faces* parallel to the coordinate planes, with *edges* of length 1, and *vertices* at its corners. *Cells* are either cubes, faces, edges, or vertices. The intersection of two cells is either empty or a joint *side* of both cells.

12. DEFINITION. *A* cube-curve *is an alternating sequence $g = (f_0, c_0, f_1, c_1, \ldots, f_n, c_n)$ of faces f_i and cubes c_i, for $0 \leq i \leq n$, such that faces f_i and f_{i+1} are sides of cube c_i, for $0 \leq i \leq n$ and $f_{n+1} = f_0$.*

A cube-curve is *simple* iff $n \geq 4$ and for any two cubes $c_i, c_k \in g$ with $|i - k| \geq 2$ (mod $n + 1$), if $c_i \bigcap c_k \neq \phi$ then either $|i - k| = 2$ (mod $n + 1$) and $c_i \bigcap c_k$ is an edge, or $|i - k| \geq 3$ (mod $n + 1$) and $c_i \bigcap c_k$ is a vertex.

A *simple cube-arc* is an alternating sequence $(f_0, c_0, f_1, c_1, \ldots, f_k)$ of faces f_i and cubes c_i, with $f_k \neq f_0$, which is a proper subsequence of some simple cube-curve.

We recall that a tube **g** is the union of all cubes contained in a cube-curve g. It is a compact set in $\mathbb{R}^3$; its frontier defines a polyhedron. A curve in $\mathbb{R}^3$ is *complete* in **g** iff it has a nonempty intersection with every cube contained in g. Following [83, 128, 129], we define:

13. DEFINITION. *A* minimum-length curve *of a simple cube-curve g is a shortest simple curve P which is contained and complete in tube **g**. The length of a simple cube-curve g is defined to be the length $\mathcal{L}(P)$.*

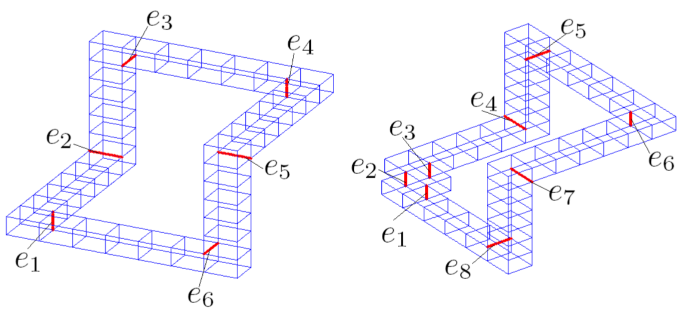


Figure 2.1: Left: a first-class simple cube-curve. Right: a non-first-class simple cube-curve.

It turns out that such a shortest simple curve P is always a polygonal curve (i.e., the MLP). The MLP is uniquely defined if the cube-curve is not only contained in a single layer of cubes of the 3D grid; see [97]. If it is contained in one layer, then the MLP is uniquely defined up to a translation orthogonal to that layer. We speak about *the* MLP of a simple cube-curve.

14. DEFINITION. *A simple cube-curve g is called* first-class *iff each critical edge of g contains exactly one vertex of the MLP of g.*

Figure 2.1 shows a first-class simple cube-curve (left) and a non-first-class simple cube-curve (right). For the latter one note that the vertices of the MLP must be in e_1, e_3, e_4, e_5, e_6, e_7 and e_8, but the critical edge e_2 does not contain any vertex of the MLP of this simple cube-curve.

Let $c_0, c_1, \ldots, c_{n-1}$ be the sequence of all consecutive cubes of cube-curve g. c_j is called *after* c_i if $j = i + 1 \pmod{n}$. Let e be a critical edge of g. Let c_i, c_j and c_k be three consecutive cubes of g such that $e \in c_i \cap c_j \cap c_k$, c_k is after c_j, and c_j is after c_i. Then, c_i is called *the first cube* of e in g and c_k is called *the third cube* of e in g.

15. DEFINITION. *Let e and f be two different critical edges of g. Let c_i be the third cube of e in g and c_j be the first cube of f in g. n is the total number of cubes in g. $\{c_i, c_{i+1}, ..., c_{j-1}, c_j\}$ (mod n) is* the arc *(in g)* between edges *e and f.*

2.2.1 Known Theorems

We recall a few theorems relevant for the discussion of our algorithms. Possible locations of vertices of MLPs have been studied in [83]:

3. THEOREM. *Let g be a simple cube-curve. Critical edges are the only possible locations of vertices of the MLP of g.*

[95] analyzed the original rubberband algorithm for first-class simple cube-curves:

4. THEOREM. *The original rubberband algorithm is correct for first-class simple cube-curves. The computational complexity is $\kappa(\epsilon) \cdot \mathcal{O}(n)$, where $\kappa(\epsilon)$ is as in Equation (1.1), and n is the number of critical edges of the simple cube-curve.*

We also recall a basic result in [66] on algorithms on graphs:

5. THEOREM. *Assume as input a directed graph G. The all-pairs shortest paths problem can be solved in $\mathcal{O}(n^3(loglogn/logn)^{5/7})$ time, where n is the number of vertices of G.*

2.2.2 New Theorems

This subsection provides a few technical results which will be useful later on. The reader may skip this section, and return later if looking for details of used formulas.

Let g_l, g_r, g_f, g_b, g_d and $g_u \in \mathbb{R}^3$ with $g_l \leq g_r$, $g_f \leq g_b$ and $g_d \leq g_u$. Let

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$

where $(x, y, z) \in [g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$. If there is a real number $M > 0$ such that $|f(x_1, y_1, z_1)| \geq M$, where $(x_1, y_1, z_1) \in [g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$, then we have the following:

2. LEMMA. *There is a real number $\delta > 0$ such that $\max\{|x_2-x_1|,\ |y_2-y_1|,\ |z_2-z_1|\} < \delta$, for each point*

$$(x_2, y_2, z_2) \in [g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$$

It also follows that

$$|f(x_2, y_2, z_2)| > M/2$$

for any of these points (x_2, y_2, z_2).

Proof. Since $[g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$ is a compact set, so $f(x, y, z)$ is uniformly continuous at each point of $[g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$. For $\varepsilon_0 = M/2 > 0$, it follows that there is a real number $\delta > 0$ such that, for any two points $(x_1, y_1, z_1), (x_2, y_2, z_2) \in [g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$ which satisfy that

$$\max\{|x_2 - x_1|,\ |y_2 - y_1|,\ |z_2 - z_1|\} < \delta$$

we have that

$$|f(x_2, y_2, z_2) - f(x_1, y_1, z_1)| < M/2$$

That is,

$$f(x_1, y_1, z_1)| - M/2 < f(x_2, y_2, z_2) < f(x_1, y_1, z_1)| + M/2$$

Now note that $|f(x_1, y_1, z_1)| \geq M$ implies that

$$f(x_1, y_1, z_1) \geq M \quad \text{or} \quad f(x_1, y_1, z_1) \leq -M$$

Therefore,

$$f(x_2, y_2, z_2) > M - M/2 = M/2 \quad \text{or} \quad |f(x_2, y_2, z_2)| < -M + M/2 = -M/2$$

With this we have $|f(x_2, y_2, z_2)| > M/2$. □

We apply Lemma 2 to prove Lemma 3 below, by which we are then able to prove the main theorem in this subsection.

Let $M_1 = \max\{|g_l|, |g_r|, |g_f|, |g_b|, |g_d|, |g_u|\} > 0$ and $|f(x_1, y_1, z_1)| \geq M_2$. For any two points (x_1, y_1, z_1) and $(x_2, y_2, z_2) \in [g_l, g_r] \times [g_f, g_b] \times [g_d, g_u]$ which satisfy that $\max\{|x_2 - x_1|,\ |y_2 - y_1|,\ |z_2 - z_1|\} < \delta$, we have the following:

3. Lemma. $|f(x_2, y_2, z_2) - f(x_1, y_1, z_1)| < 6M_1\delta/M_2$.

Proof. Without loss of generality, suppose that $x_1 < x_2$, $y_1 < y_2$ and $z_1 < z_2$. By the mean-value theorem,

$$\begin{aligned}
&|f(x_2, y_2, z_2) - f(x_1, y_1, z_1)| \\
&= |[f(x_2, y_2, z_2) - f(x_1, y_2, z_2)] + [f(x_1, y_2, z_2) - f(x_1, y_1, z_2)] \\
&\qquad + [f(x_1, y_1, z_2) - f(x_1, y_1, z_1)]| \\
&= |f_x(\xi_x, y_2, z_2)(x_2 - x_1) + f_y(x_1, \eta_y, z_2)(y_2 - y_1) + f_z(x_1, y_1, \zeta_z)(z_2 - z_1)| \\
&= \left| \frac{\xi_x}{\sqrt{{\xi_x}^2 + {y_2}^2 + {z_2}^2}}(x_2 - x_1) + \frac{\eta_y}{\sqrt{{x_1}^2 + {\eta_y}^2 + {z_2}^2}}(y_2 - y_1) \right. \\
&\qquad \left. + \frac{\zeta_z}{\sqrt{{x_1}^2 + {y_1}^2 + {\zeta_z}^2}}(z_2 - z_1) \right|
\end{aligned}$$

where $\xi_x \in (x_1, x_2)$, $\eta_y \in (y_1, y_2)$, and $\zeta_z \in (z_1, z_2)$. By Lemma 2, we have

$$\begin{aligned}
|f(x_2, y_2, z_2) &- f(x_1, y_1, z_1)| \\
&< 2M_1\delta/M_2 + 2M_1\delta/M_2 + 2M_1\delta/M_2 \\
&= 6M_1\delta/M_2
\end{aligned}$$

This proves our lemma. □

Let $e_1, e_2, \dots, e_n$ be all the consecutive critical edges (subsequent in some cyclic order) of g. Let the points

$$p_i(x_i, y_i, z_i), p'_i(x'_i, y'_i, z'_i) \in e_i$$

be such that

$$\max\{|x'_i - x_i|,\ |y'_i - y_i|,\ |y'_i - y_i|\} < \delta$$

where $i = 1, 2, \dots, n$. Furthermore, let

$$d_e(i) = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}$$

$$d'_e(i) = \sqrt{(x'_{i+1} - x'_i)^2 + (y'_{i+1} - y'_i)^2 + (z'_{i+1} - z'_i)^2}$$

where $i = 1, 2, \dots, n-1$, and

$$d_e(n) = \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2 + (z_1 - z_n)^2}$$

$$d'_e(n) = \sqrt{(x'_1 - x'_n)^2 + (y'_1 - y'_n)^2 + (z'_1 - z'_n)^2}$$

As above, let g_l, g_r, g_f, g_b, g_d and $g_u \in \mathbb{R}^3$ with $g_l \le g_r$, $g_f \le g_b$ and $g_d \le g_u$, and let

$$M_1 = \max\{|g_l|, |g_r|, |g_f|, |g_b|, |g_d|, |g_u|\} > 0$$

Let $d = \sum_{i=1}^{n} d_e(i)$ and $d' = \sum_{i=1}^{n} d'_e(i)$. By Lemma 3, we have the following:

6. THEOREM. *If* $|d_e(i)| \ge M_2$, *for* $i = 1, 2, \dots, n$, *then* $|d - d'| < 12nM_1\delta/M_2$.

Proof. Let

$$\begin{aligned}
\delta_{x_{i+1}} &= x_{i+1} - x'_{i+1},\ \delta_{x_i} = x'_i - x_i \\
\delta_{y_{i+1}} &= y_{i+1} - y'_{i+1},\ \delta_{y_i} = y'_i - y_i \\
\delta_{z_{i+1}} &= z_{i+1} - z'_{i+1},\ \delta_{z_i} = z'_i - z_i \\
\delta_x &= \delta_{x_{i+1}} + \delta_{x_i},\ \ \delta_y = \delta_{y_{i+1}} + \delta_{y_i},\ \text{ and }\ \delta_z = \delta_{z_{i+1}} + \delta_{z_i}
\end{aligned}$$

Then we have

$$|\delta_x| \le 2\delta,\ |\delta_y| \le 2\delta,\ \text{ and }\ |\delta_z| \le 2\delta$$

Since

$$\begin{aligned}
&(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i))^2 \\
= &[(x'_{i+1} - x'_i) + (x_{i+1} - x'_{i+1}) + (x'_i - x_i)]^2 \\
&+[(y'_{i+1} - y'_i) + (y_{i+1} - y'_{i+1}) + (y'_i - y_i)]^2 \\
&+[(z'_{i+1} - z'_i) + (z_{i+1} - z'_{i+1}) + (z'_i - z_i)]^2 \\
= &[(x'_{i+1} - x'_i) + \delta_x]^2 + [(y'_{i+1} - y'_i) + \delta_y]^2 + [(z'_{i+1} - z'_i) + \delta_z]^2
\end{aligned}$$

and Lemma 3, we have that

$$|d_e(i) - d'_e(i)| < 6M_1 \times 2\delta/M_2 = 12M_1\delta/M_2$$

Therefore, $|d - d'| = |\sum_{i=1}^{n}(d_e(i) - d'_e(i))| \leq \sum_{i=1}^{n}|d_e(i) - d'_e(i))| < 12nM_1\delta/M_2$. □

Theorem 6 gives an upper bound for the difference of the lengths of two polygonal curves which are completely contained in the same simple cube-curve.

2.3 Algorithm and Its Complexity

In this section we first present the algorithm. Then we illustrate it by some small examples. Finally we analyze the time complexity and the error limit of the algorithm.

2.3.1 The Algorithm

Based on Theorems 3, 4 and 6 we are now in a position to formulate the following algorithm. It is an application of graph theory, Dijkstra's algorithm, and the original rubberband algorithm.

1. ALGORITHM.

1. Initialize an integer $m \geq 2$. For each critical edge e_i, let $p_{i_0}, p_{i_1}, \ldots, p_{i_m}$ be subdivision points on e_i such that the Euclidean distance between p_{i_j} and $p_{i_{j+1}}$ equals $\frac{1}{m}$, for $j = 0, 2, \ldots, m-1$. (Note that p_{i_0} and p_{i_m} are the end points of e_i.)

2. We consider the weighted directed graph $G = [V, E]$, where

$$V = \{p_{i_j} : p_{i_j} \in e_i \wedge i = 1, 2, \ldots, n \wedge j = 1, 2, \ldots, m\}$$
$$E = \{p_{i_j}p_{k_l} : p_{i_j}p_{k_l} \text{ completely contained in the cube-arc between } e_i \text{ and } e_k \text{ in } g\}$$

The weight of straight segment $p_{i_j}p_{k_l}$ is defined to be the Euclidean distance between p_{i_j} and p_{k_l}, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

3. For each $v \in V$, let $S(v) = \{u : uv \in E\}$. Do the following:

3.1. For each $u \in S(v)$, apply Dijkstra's algorithm to compute the directed shortest path from u and v.

3.2. From this, find the local minimum-weight directed cycle which contains uv.

3.3. Let $CE(v)$ be the set of critical edges such that each of the vertices of the local minimum-weight directed cycle is on one of the edges in $CE(v)$.

4.1. Compare the local minimum-weight directed cycles, for each $v \in V$; the minimum cycle (for one particular v) is the global minimum-weight directed cycle, denoted by $AMLP$.

4.2. Let $CE = \emptyset$.

4.3. For each vertex $v \in V(AMLP)$ (this is the set of vertices of $AMLP$), find a critical edge e such that $v \in e$; let $CE = CE \cup \{e\}$.

5. By Theorem 4 apply the original rubberband algorithm on CE and $AMLP$, and compute the minimum-weight directed cycle (this is in fact an approximate MLP; see discussion later in this chapter).

2.3.2 Example for Algorithm 1

Table 2.1 shows the data of the critical edges of a (short) simple cube-curve g, shown in Figure 2.2, upper left. Figure 2.2, upper left, also shows the subdivision points of Step 1, where $m = 3$.

Figure 2.2, upper right, shows the graph, denoted by $G = [V, E]$, constructed in Step 2. Note that each edge of G is fully contained in tube **g**. To compute the weight of an edge, say, of $p_{2_3}p_{3_1}$ of G, see Table 2.1: the coordinates of p_{2_3} and p_{3_1} are $(0.5, 1, 0.5)$ and

$$(0.5, 1, 1.5) + \frac{1}{m} \times [(0.5, 2, 1.5) - (0.5, 1, 1.5)] = (0.5, 1.3333, 1.5)$$

respectively. It follows that the weight of segment $p_{2_3}p_{3_1}$ equals

$$d_e(p_{2_3}, p_{3_1}) = \sqrt{(0.5 - 0.5)^2 + (1.3333 - 1)^2 + (1.5 - 0.5)^2} = 1.0541$$

We illustrate Steps 3 and 4 by the (small) directed weighted graph shown in Figure 2.2, lower left. For Step 3, we have $V = \{v_1, v_2, v_3, v_4\}$, with $S(v_1) = \{v_3, v_4\}$, $S(v_2) = \{v_1\}$, $S(v_3) = \{v_2\}$, and $S(v_4) = \{v_3\}$.

Consider vertex v_1 and Step 3.1; for $v_3 \in S(v_1)$, the directed shortest path from v_1 to v_3 equals (v_1, v_2, v_3) with weight 3; the directed shortest path from v_1 to v_4 equals

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}
e_1	0.5	1	0.5	0.5	1	-0.5
e_2	-0.5	1	0.5	0.5	1	0.5
e_3	0.5	1	1.5	0.5	2	1.5
e_4	1.5	1	1.5	1.5	2	1.5
e_5	2.5	1	0.5	1.5	1	0.5
e_6	1.5	1	0.5	1.5	1	-0.5

Table 2.1: Coordinates of endpoints of critical edges in Figure 2.2.

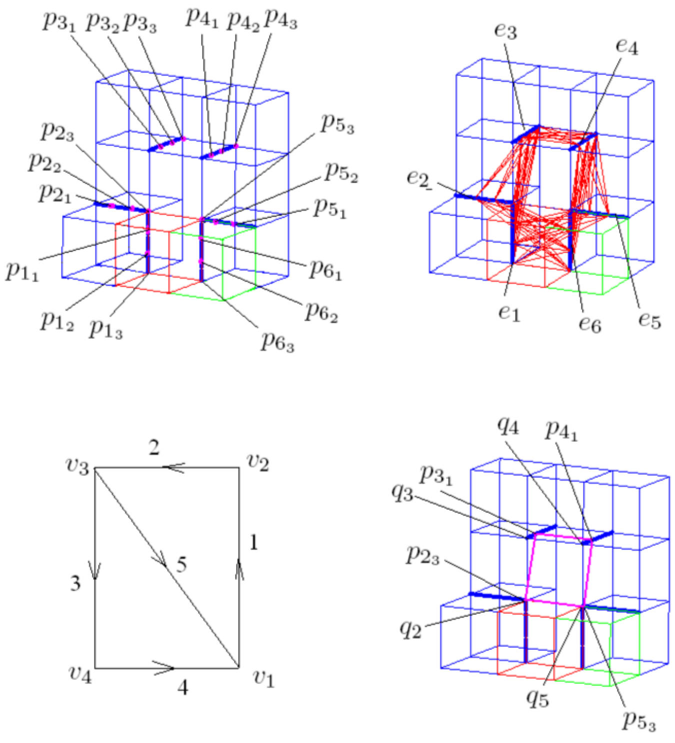


Figure 2.2: Illustration for steps of Algorithm 1 (see text for details).

(v_1, v_2, v_3, v_4) with weight 6. By Step 3.2, the local minimum-weight directed cycle, which contains v_3v_1, equals (v_1, v_2, v_3, v_1) with weight 8; $(v_1, v_2, v_3, v_4, v_1)$ (with weight 10) is the local minimum-weight directed cycle which contains v_4v_1.

Now consider vertex v_2 and Step 3.1; for $v_1 \in S(v_2)$, the directed shortest paths from v_2 to v_1 are (v_2, v_3, v_4, v_1) with weight 9, and (v_2, v_3, v_1) with weight 7. By Step 3.2, the local minimum-weight directed cycle, which contains v_1v_2, equals (v_2, v_3, v_1, v_2) with weight 8.

Next consider vertex v_3 and Step 3.1; for $v_2 \in S(v_3)$, the directed shortest paths from v_3 to v_2 are (v_3, v_4, v_1, v_2) with weight 8, and (v_3, v_1, v_2) with weight 6. By Step 3.2, the local minimum-weight directed cycle, which contains v_2v_3, equals (v_3, v_1, v_2, v_3) with weight 8.

Finally, consider vertex v_4 and Step 3.1; for $v_3 \in S(v_4)$, the directed shortest path from v_4 to v_3 equals (v_4, v_1, v_2, v_3) with weight 7. By Step 3.2, the local minimum-weight directed cycle, which contains v_3v_4, equals $(v_4, v_1, v_2, v_3, v_4)$ with weight 10.

For Step 4.1, compare all the local minimum-weight directed cycles obtained in Step 3; the global minimum-weight directed cycle equals (v_1, v_2, v_3, v_1) with weight 8.

The global minimum-weight directed cycle of G equals $(p_{2_3}, p_{3_1}, p_{4_1}, p_{5_3}, p_{2_3})$, which is denoted as $AMLP$; see Figure 2.2, lower right. This $AMLP$ has the weight 4.1082. The "associated" set of critical edges CE equals $\{e_2, e_3, e_4, e_5\}$.

For Step 5, apply the original rubberband algorithm on CE and $AMLP$ to compute the minimum-weight directed cycle $\rho = (q_2, q_3, q_4, q_5, q_2)$; see Figure 2.2, lower right. Cycle ρ has weight 4. (Actually, it is the true MLP of this simple cube-curve.)

2.3.3 Asymptotic Time Complexity and Error Limit

For Algorithm 1, the first two steps can be preprocessed. The main computation occurs in Step 3.1, which is in $\mathcal{O}(m^2n^2)$ (see, for example, [47], pages 595–601), where n is the number of critical edges and m the number of subdivision points on each critical edge. In Step 3.2, $|S(v)| \leq mn$. Thus, we have that the time complexity of Step 3 is in $\mathcal{O}(m^3n^3)$. Since $|V| \leq mn$, it follows that the time complexity of Step 4.1 is in $\mathcal{O}(m^4n^4)$. Since $|V(AMLP)| \leq n$, Step 4.3 can be computed in $\mathcal{O}(n^2)$. By Theorem 4, Step 5 runs in time $\kappa(\epsilon) \cdot \mathcal{O}(n)$, where $\kappa(\epsilon)$ is as in Equation (1.1). Therefore, the time complexity of Algorithm 1 is in $\mathcal{O}(m^4n^4 + \kappa(\epsilon) \cdot n)$.

Let L^* be the length of the approximate MLP computed by Algorithm 1. By Theorem 6, we have $0 < L^* - \mathcal{L}(P) < 12nM_1\delta/M_2$. This implies that

$$0 < L^* < \mathcal{L}(P) + 12nM_1\delta/M_2 = (1 + \frac{12M_1}{M_2\mathcal{L}(P)} n\delta)\mathcal{L}(P)$$

where M_1, M_2, n, and δ are as in Theorem 6. In other words, Algorithm 1 is an δ'-approximation, where

$$\delta' = 1 + \frac{12M_1}{M_2\mathcal{L}(P)} n\delta$$

Obviously, we may select m to be sufficiently large, say, $m > n^2$. It follows that $\delta < 1/n^2$, and we have

$$\delta' < 1 + \frac{12M_1}{M_2\mathcal{L}(P)} n \cdot \frac{1}{n^2} = 1 + \frac{12M_1}{M_2\mathcal{L}(P)} \cdot \frac{1}{n}$$

Therefore, mathematically, we can take a sufficiently large value of m to obtain the approximate MLP of g with good accuracy. However, in practice, our experiments for $N \leq 106$ and $n \leq 31$ showed that $m = 5$ is "quite sufficient", with a running time of about 173.1 seconds; see Table 2.2 for a Pentium 4 PC using Matlab 7.04.

N	n	m	*time*	$AMLP$	MLP
20	9	5	8.6	11.30	11.24
20	9	5	9.5	11.28	11.24
20	10	5	14.7	10.20	10.15
32	8	5	5.8	22.57	22.51
38	12	5	16.4	26.40	26.39
48	15	5	32.3	34.73	34.71
48	15	5	29.4	33.61	33.55
74	20	5	51.0	55.66	55.57
76	20	5	55.8	56.46	56.38
70	21	5	59.0	50.26	50.15
88	27	5	110.4	63.94	63.89
84	27	5	116.1	61.22	61.14
90	31	5	173.1	65.47	65.44
106	30	5	140.3	79.95	79.85

Table 2.2: Time complexity for Algorithm 1, where N is the number of cubes, n the number of critical edges, and m the number of subdivision points on each critical edge; the time is in seconds.

Following Lemma 3, if it would be possible to find better bounds than M_1 and M_2, then this would result into a smaller value of m.

2.4 Conclusions

We presented a δ'-approximation algorithm for computing the MLP of a general simple cube-curve, where

$$\delta' = 1 + \frac{12M_1}{M_2\mathcal{L}(P)}n\delta$$

M_1 , M_2, n, and δ are as in Theorem 6. It runs in time $\mathcal{O}(m^4n^4 + \kappa(\epsilon) \cdot n)$, where $\kappa(\epsilon)$ is as in Equation (1.1), m is the chosen number of possible subdivision points on any of the critical edges, and n is the number of critical edges.

Since the graph G, constructed in the algorithm, has mn vertices, by Theorem 5, the time complexity may be reduced to

$$\mathcal{O}\left(m^3n^3\left[\frac{loglog(mn)}{log(mn)}\right]^{5/7} + \kappa(\epsilon)\cdot n\right)$$

Chapter 3

MLPs of Decomposable Simple Cube Curves

This chapter develops a provably correct MLP algorithm for a special type of first-class simple cube-curves which can be decomposed (in a particular way) into a finite number of simple cube-arcs. [The next chapter will also show that the formulas derived for these simple cube-arcs are important for a correctness proof for the original rubberband algorithm, considering the same special type of cube-curves.]

3.1 Introduction

The previous chapter presented an $\mathcal{O}(m^4 n^4 + \kappa(\varepsilon) \cdot n)$ for general simple cube-curves by considering a pre-defined subdivision of all critical edges and applying Dijkstra's algorithm (well-known in algorithmic graph theory) and the original rubberband algorithm. The algorithm is reasonably time-efficient for "short" simple cube-curves. In this chapter we analyze the "geometric structure" of simple cube-curves. By introducing the concept of an "end angle" (see Definition 16 below), we will be able to decompose a special first-class simple cube-curve into finite simple cube-arcs in a unique way. (In other words, the tube of such a special first-class simple cube-curve is the union of the tubes of these simple cube-arcs.) We focus on such a special subset of the family of all first-class simple cube-curves [1] which have at least one end angle (e.g., as the cube-curve in Figure 1.2). We also study a non-trivial example of such a curve and show that the path calculated by the rubberband algorithm is converging against the MLP. We also presents an algorithm for the computation of approximate MLPs for the special class (as defined in this chapter) of simple cube-curves.

This chapter is organized as follows: Section 3.2 describes theoretical fundamentals for the length calculation of first-class simple cube-curves. Section 3.3.2 presents an algorithm for length computation. Section 3.4 gives experimental results for an example of a cube-curve, and discusses results obtained by the original rubberband algorithm for this particular input. Section 3.5 gives the conclusions.

[1]Note that we can classify a simple cube-curve in linear time to be first-class or not, by running the rubberband algorithm as described in [24]: the curve is first-class iff option (O_2) does not occur at all.

3.2 Basics

We begin by introducing basic terms used throughout the report.

16. DEFINITION. *Assume a simple cube-curve g and a triple of consecutive critical edges e_1, e_2, and e_3 such that $e_i \perp e_j$, for $i, j = 1, 2, 3$ and $i \neq j$. Let σ be a metavariable for x, y, or z. If e_2 is parallel to the σ-axis and the σ-coordinates of e_1 and e_3 are identical (for $\sigma = x$, $\sigma = y$, or $\sigma = z$), then we say that e_1, e_2 and e_3 form an* end angle.

Cube-curve g *has an end angle*, denoted by $\angle(e_1, e_2, e_3)$, in such a case. Otherwise we say that e_1, e_2 and e_3 form a *middle angle* $\angle(e_1, e_2, e_3)$, and g *has a middle angle* in this case.

Figure 1.2 shows a simple cube-curve which has five end angles $\angle(e_{21}, e_0, e_1)$, $\angle(e_4, e_5, e_6)$, $\angle(e_6, e_7, e_8)$, $\angle(e_{14}, e_{15}, e_{16})$), and $\angle(e_{16}, e_{17}, e_{18})$, and 16 middle angles, such as $\angle(e_0, e_1, e_2)$, $\angle(e_1, e_2, e_3)$, or $\angle(e_2, e_3, e_4)$.

17. DEFINITION. *A straight line l, which is incident with a critical edge e of g (i.e., $e \subset l$), is called a* critical line *(of e in g).*

A critical line and a critical edge are *corresponding* if they are incident.

18. DEFINITION. *Let $S \subseteq \mathbb{R}^3$. The set $\{(x, y, 0) : \exists z(z \in \mathbb{R} \wedge (x, y, z) \in S)\}$ is the xy-*projection *of S, or* projection *of S for short. Analogously we define the yz- or xz-*projection *of S.*

The next lemma is used in Section 3.4.1; it is a basic result from computational geometry, see [44], Lemma 1, or [125], Lemma 3.3, or [153].

4. LEMMA. *There is a uniquely defined shortest path, which passes through subsequent critical edges e_1, e_2, . . ., and e_k in this order.*

Let e_1, e_2, and e_3 be three (not necessarily consecutive) critical edges in a simple cube-curve, and let l_1, l_2, and l_3 be the corresponding three critical lines. As a general approach in this report, we consider points on critical lines in parametrized form, allowing to study changes in their positions by means of derivatives.

For example, point $p_2(t_2) = (x_2 + k_{x_2} t_2, y_2 + k_{y_2} t_2, z_2 + k_{z_2} t_2)$ on l_2 is parametrized by $t_2 \in \mathbb{R}$. Analogously, $p_1(t_1)$ or $p_3(t_3)$ on l_1 or l_3 are parametrized by t_1 or t_3, respectively. See Figure 3.1.

5. LEMMA. *Let $d_2(t_1, t_2, t_3) = d_e(p_1(t_1), p_2(t_2)) + d_e(p_2(t_2), p_3(t_3))$, for arbitrary reals t_1, t_2, and t_3. It follows that $\frac{\partial^2 d_2}{\partial {t_2}^2} > 0$.*

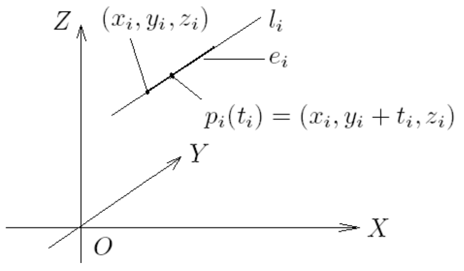


Figure 3.1: Point $p_i(t_i)$ on the critical line l_i which is incident with critical edge e_i; (x_i, y_i, z_i) is an end point of e_i.

Proof. Let the coordinates of $p_i(t_i)$ be $(x_i + k_{x_i} t_i, y_i + k_{y_i} t_i, z_i + k_{z_i} t_i)$, where i equals 1 or 3.

Assume that $p_i \in e_i \subset l_i$ (for $i = 1, 2, 3$), where e_i is an edge of the regular 3D orthogonal grid. It follows that only one of the values k_{x_i}, k_{y_i}, or k_{z_i} can be equal to 1; the other two must be equal to zero.

Let us look at one of these cases where the coordinates of p_1 be $(x_1 + t_1, y_1, z_1)$, the coordinates of p_2 be $(x_2, y_2 + t_2, z_2)$, and the coordinates of p_3 be $(x_3, y_3, z_3 + t_3)$. Then we have that $d_2(t_1, t_2, t_3) = d_e(p_1(t_1), p_2(t_2)) + d_e(p_2(t_2), p_3(t_3))$ equals

$$\sqrt{(t_2 - (y_1 - y_2))^2 + (x_1 + t_1 - x_2)^2 + (z_1 - z_2)^2} + \sqrt{(t_2 - (y_3 - y_2))^2 + (x_3 - x_2)^2 + (z_3 + t_3 - z_2)^2}$$

This can be written (in brief) as $d_2 = \sqrt{(t_2 - a_1)^2 + b_1^2} + \sqrt{(t_2 - a_2)^2 + b_2^2}$, where b_1 and b_2 are functions of t_1 and t_3, respectively. Then we have

$$\frac{\partial d_2}{\partial t_2} = \frac{t_2 - a_1}{\sqrt{(t_2 - a_1)^2 + b_1^2}} + \frac{t_2 - a_2}{\sqrt{(t_2 - a_2)^2 + b_2^2}} \tag{3.1}$$

and

$$\frac{\partial^2 d_2}{\partial {t_2}^2} = \frac{1}{\sqrt{(t_2 - a_1)^2 + b_1^2}} - \frac{(t_2 - a_1)^2}{[(t_2 - a_1)^2 + b_1^2]^{3/2}} + \frac{1}{\sqrt{(t_2 - a_2)^2 + b_2^2}} - \frac{(t_2 - a_2)^2}{[(t_2 - a_2)^2 + b_2^2]^{3/2}}$$

This simplifies to

$$\frac{\partial^2 d_2}{\partial {t_2}^2} = \frac{b_1^2}{[(t_2 - a_1)^2 + b_1^2]^{3/2}} + \frac{b_2^2}{[(t_2 - a_2)^2 + b_2^2]^{3/2}} > 0 \tag{3.2}$$

Other cases of positions of p_i lead, analogously, also to such a positive second derivative. □

With this lemma we introduced the very important (at least, for this report) concept that points $p_i(t_i)$ are studied with respect to their derivatives. This approach will be enforced repeatedly in this report.

Note that we do not use $p_i(t)$, because different critical edges may have different values of t, representing varying speed of movements along these edges (e.g., when adjusting the length of a curve which is passing through critical edges).

Lemma 5 is used in the proof of the following lemma. – Let e_1, e_2, and e_3 be three critical edges (again: not necessarily consecutive critical edges), and let l_1, l_2, and l_3 be their corresponding critical lines, respectively. Let p_1, p_2, and p_3 be three points such that p_i belongs to l_i, for $i = 1, 2, 3$. Let $p_2 = (x_2 + k_{x_2}t_2, y_2 + k_{y_2}t_2, z_2 + k_{z_2}t_2)$. Let $d_2 = d_e(p_1, p_2) + d_e(p_2, p_3)$.

6. LEMMA. *The function* $f(t_2) = \frac{\partial d_2}{\partial t_2}$ *has a unique real root.*

Proof. We refer to the proof of Lemma 5. Without loss of generality, we can assume that $a_1 \leq a_2$. Then, by Equation (3.1), we have that $f(a_1) \leq 0$ as well as $f(a_2) \geq 0$. The lemma follows with Equation (3.2). □

We use this lemma in Section 3.3.2 to describe our next algorithm. – Now let e_1, e_2 and e_3 be three consecutive (!) critical edges of a simple cube-curve g. Let $D(e_1, e_2, e_3)$ be the dimension of the linear space generated by e_1, e_2 and e_3. Let l_{13} be a line segment with one end point (somewhere) on e_1, and the other end point (somewhere) on e_3. Let $d_{e_ie_j}$ be the (Euclidean) Hausdorff distance between e_i and e_j (i.e., the minimum of all Euclidean distances between a point p on e_i, and a point q on e_j), where $i, j = 1, 2, 3$.

7. LEMMA. *The line segment* l_{13} *(for any choice of endpoints) is not completely contained in the tube **g** if* $D(e_1, e_2, e_3) = 3$, $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 1$, *and* $\max\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$, *or if* $D(e_1, e_2, e_3) \leq 2$ *and* $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$.

Proof. *Case 1.* Let $D(e_1, e_2, e_3) = 3$, $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 1$ and $\max\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$. We only need to prove that the conclusion is true when $\min\{d_{e_1e_2}, d_{e_2e_3}\} = 1$ and $\max\{d_{e_1e_2}, d_{e_2e_3}\} = 2$. In this case, the parallel projection [denoted by $g'(e_1, e_2, e_3)$] of all of g's cubes contained between e_1 and e_3 is illustrated in Figure 3.2, where AB is the projective image of e_1, EF that of e_3, and C that of one of the end points of e_2. Note that line segment AF must intercept grid edge BC at a point G, and intercept grid edge CD at a point H. And note that line segment GH is not completely contained in $g'(e_1, e_2, e_3)$. Therefore, if l_{13} is a line segment with one end point on e_1 and the other one on e_3, then l_{13} is not completely contained in **g**.

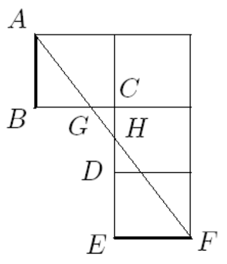


Figure 3.2: Illustration of Case 1 in the proof of Lemma 7.

Case 2. Let $D(e_1, e_2, e_3) = 2$ and $\min\{d_{e_1 e_2}, d_{e_2 e_3}\} \geq 2$. Without loss of generality, we can assume that $e_1 \parallel e_2$.

Case 2.1. e_1 and e_2 are on the same grid line; we only need to prove that the conclusion is true when $d_{e_1 e_2} = 2$ and $d_{e_2 e_3} = 2$. In this case, the projective image [denoted by $g'(e_1, e_2, e_3)$] of all of g's cubes contained between e_1 and e_3 is illustrated in Figure 3.3.

Case 2.1.1. $g'(e_1, e_2, e_3))$ is as on the left in Figure 3.3, where A and B are the projective images of either one end point of e_1 or e_2, respectively, and CD that of e_3. Note that line segment AD must intercept grid edge EC at a point F. Also note that line segments AD and AC are not completely contained in $g'(e_1, e_2, e_3)$. Therefore, if l_{13} is a line segment where one end point is on e_1, and the other on e_3, then l_{13} is not completely contained in **g**. Similarly, we can show that the conclusion is also true for Case 2.1.2, with $g'(e_1, e_2, e_3)$ as illustrated on the right in Figure 3.3.

Case 2.2. Assume that e_1 and e_2 are on different grid lines. We only need to prove that the conclusion is true when $d_{e_1 e_2} = \sqrt{5}$ and $d_{e_2 e_3} = 2$. In this case, the projective image [denoted by $g'(e_1, e_2, e_3)$] of all of g's cubes contained between e_1 and e_3 is illustrated in Figure 3.4, where A (B) is the projective image of one end point of e_1

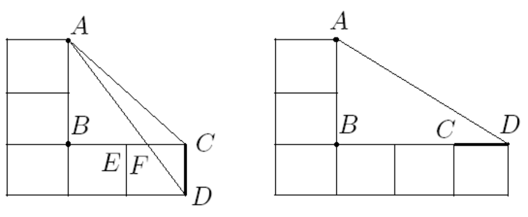


Figure 3.3: Illustration of Case 2.1 in the proof of Lemma 7.

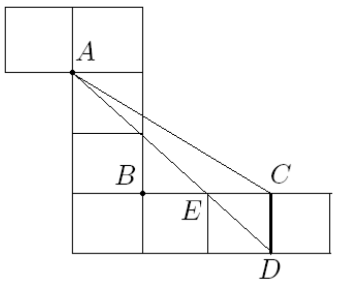


Figure 3.4: Illustration of Case 2.2 in the proof of Lemma 7.

(e_2), and CD that of e_3. Note that line segment AD must intercept grid edge EC at a point E. Also note that line segments AD and AC are not completely contained in $g'(e_1, e_2, e_3)$. Therefore, if l_{13} is a line segment with one end point on e_1, and one on e_3, then l_{13} is not completely contained in **g**.

Case 3. Let $D(e_1, e_2, e_3) = 1$ and $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$. Without loss of generality, we can assume that $e_1 \parallel e_2$.

Case 3.1. e_1 and e_2 are on the same grid line. We only need to prove that the conclusion is true when $d_{e_1e_2} = 2$ and $d_{e_2e_3} = 2$. In this case, the projective image [denoted by $g'(e_1, e_2, e_3)$] of all of g's cubes contained between e_1 and e_3 is illustrated on the left of Figure 3.5, where A, B, and C are projective images of one end point of e_1, e_2, and e_3, respectively. Note that line segment AC is not completely contained in $g'(e_1, e_2, e_3)$. Therefore, if l_{13} is a line segment with an end point on e_1 and another one on e_3, then l_{13} is not be completely contained in **g**.

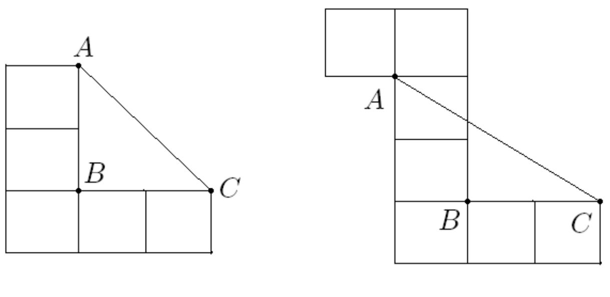


Figure 3.5: Illustration of both subcases of Case 3 in the proof of Lemma 7.

Case 3.2. Now assume that e_1 and e_2 are on different grid lines. We only need to prove that the conclusion is true when $d_{e_1e_2} = \sqrt{5}$ and $d_{e_2e_3} = 2$. In this case, the projective image [denoted by $g'(e_1, e_2, e_3)$] of all of g's cubes contained between e_1 and e_3 is illustrated on the right in Figure 3.5, where A,B,and C are the projective image of one end point of e_1, e_2, and e_3, respectively. Note that line segment AC is not completely contained in $g'(e_1, e_2, e_3)$. Therefore, if l_{13} is a line segment with end points on e_1 and e_3, then l_{13} is not be completely contained in **g**. □

This lemma and the following Lemmas 8 and 9 are used to prove the following Theorem 7 which states a sufficient condition for the first-class simple cube-curve.

Let g be a simple cube-curve such that any three consecutive critical edges e_1, e_2 and e_3 do satisfy that either $D(e_1, e_2, e_3)$ = 3, $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 1$ and $\max\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$, or $D(e_1, e_2, e_3) \leq 2$ and $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$. By Lemma 7, we immediately obtain:

8. LEMMA. *Every critical edge of g contains at least one vertex of g's MLP.*

Let g be a simple cube-curve, and assume that every critical edge of g contains at least one vertex of the MLP. Then we also have the following:

9. LEMMA. *Every critical edge of g contains at most one vertex of g's MLP.*

Proof. Assume that there exists a critical edge e such that e contains at least two vertices v and w of the MLP P of g. Without loss of generality, we can assume that v and w are the first (in the order on P) two vertices which are on e. Let u be a vertex of P, which is on the previous critical edge of P. Then line segments uv and uw are completely contained in **g**. By replacing $\{uv, uw\}$ by uw we obtain a polygon of length shorter than P, which is in contradiction to the fact that P is an MLP of g. □

Let g be a simple cube-curve such that any three consecutive critical edges e_1, e_2, and e_3 do satisfy that either $D(e_1, e_2, e_3)$ = 3, $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 1$ and $\max\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$, or $D(e_1, e_2, e_3) \leq 2$ and $\min\{d_{e_1e_2}, d_{e_2e_3}\} \geq 2$. By Lemma 8 and Lemma 9, we immediately obtain:

7. THEOREM. *The specified simple cube-curve g is first-class.*

Let e_1, e_2, and e_3 be three consecutive critical edges of a simple cube-curve g. Let p_1, p_2, and p_3 be three points such that $p_i \in e_i$, for $i = 1, 2, 3$. Let the coordinates of p_i be $(x_i + k_{x_i}t_i, y_2 + k_{y_i}t_i, z_i + k_{z_i}t_i)$, where $k_{x_i}, k_{y_i}, k_{z_i}$ are either 0 or 1, and $0 \leq t_i \leq 1$, for $i = 1, 2, 3$. Let d_2 = $d_e(p_1, p_2)$ + $d_e(p_2, p_3)$.

8. THEOREM. $\frac{\partial d_2}{\partial t_2} = 0$ *implies that we have one of the following representations for* t_3*: we can have*

$$t_3 = \frac{-c_2 t_1 + (c_1 + c_2) t_2}{c_1} \tag{3.3}$$

if $c_1 > 0$*; we can also have*

$$t_3 = 1 - \sqrt{\frac{c_1^2 (t_2 - a_2)^2}{(t_2 - t_1)^2} - c_2^2} \quad \textit{or} \tag{3.4}$$

$$t_3 = \sqrt{\frac{c_1^2 (t_2 - a_2)^2}{(t_2 - t_1)^2} - c_2^2} \tag{3.5}$$

if a_2 *equals either 0 or 1, and* c_1 *and* c_2 *are positive; and we can also have*

$$t_3 = 1 - \sqrt{\frac{(t_2 - a_2)^2 [(t_1 - a_1)^2 + c_1^2]}{(t_2 - b_1)^2} - c_2^2} \quad \textit{or} \tag{3.6}$$

$$t_3 = \sqrt{\frac{(t_2 - a_2)^2 [(t_1 - a_1)^2 + c_1^2]}{(t_2 - b_1)^2} - c_2^2} \tag{3.7}$$

if a_1*,* a_2*, and* b_1 *are either equal to 0 or 1, and* c_1 *and* c_2 *are positive reals.*

Proof. We have that $p_i = (x_i + k_{x_i} t_i, y_2 + k_{y_i} t_i, z_i + k_{z_i} t_i)$, with $k_{x_i}, k_{y_i}, k_{z_i}$ equals 0 or 1, and $0 \leq t_i \leq 1$, for $i = 1, 2, 3$. Note that only one of the values $k_{x_i}, k_{y_i}, k_{z_i}$ can be 1, and the other two must be equal to 0. It follows that for every $i, j \in \{1, 2, 3\}$, $d_e(p_i, p_j) = \sqrt{(t_j - t_i)^2 + c^2}$ or $\sqrt{(t_i - a)^2 + (t_j - b)^2 + c^2}$, where a, b are equal to 0 or 1, and $c > 0$. We have $c \neq 0$ because otherwise e_1 and e_2 would be on the same line, and that is impossible. Let d_2 = $d_e(p_1, p_2)$ + $d_e(p_2, p_3)$. The following three cases are possible:

Case 1. $d_2 = \sqrt{(t_2 - t_1)^2 + c_1^2} + \sqrt{(t_2 - t_3)^2 + c_2^2}$, with $c_i > 0$, for $i = 1, 2$. Then we have

$$\frac{\partial d_2}{\partial t_2} = \frac{t_2 - t_1}{\sqrt{(t_2 - t_1)^2 + c_1^2}} + \frac{t_2 - t_3}{\sqrt{(t_2 - t_3)^2 + c_2^2}}$$

and equation $\frac{\partial d_2}{\partial t_2} = 0$ implies the form of Equation (3.3).

Case 2. $d_2 = \sqrt{(t_2 - t_1)^2 + c_1^2} + \sqrt{(t_2 - a_2)^2 + (t_3 - b_2)^2 + c_2^2}$, with a_2, b_2 equals 0 or 1, and $c_i > 0$, for $i = 1, 2$. Then we have

$$\frac{\partial d_2}{\partial t_2} = \frac{t_2 - t_1}{\sqrt{(t_2 - t_1)^2 + c_1^2}} + \frac{t_2 - a_2}{\sqrt{(t_2 - a_2)^2 + (t_3 - b_2)^2 + c_2^2}}$$

and equation $\frac{\partial d_2}{\partial t_2} = 0$ implies the form of Equations (3.4) or (3.5).

Case 3. $d_2 = \sqrt{(t_2-a_1)^2+(t_1-b_1)^2+c_1^2} + \sqrt{(t_2-a_2)^2+(t_3-b_2)^2+c_2^2}$, with a_i, b_i equals 0 or 1, and $c_i > 0$, for $i = 1, 2$. Then we have

$$\frac{\partial d_2}{\partial t_2} = \frac{t_2-a_1}{\sqrt{(t_2-a_1)^2+(t_1-b_1)^2+c_1^2}} + \frac{t_2-a_2}{\sqrt{(t_2-a_2)^2+(t_3-b_2)^2+c_2^2}}$$

and equation $\frac{\partial d_2}{\partial t_2} = 0$ implies the form of Equations (3.6) or (3.7). □

This theorem will be used in Sections 3.3.1 and 3.4.1. The formulas are important in the correctness proof for the original rubberband algorithm, which will be given in Chapter 4.

The proof of Case 3 of Theorem 8, and Lemma 7 also show the following:

10. LEMMA. *Let g be a first-class simple cube-curve. If e_1, e_2 and e_3 form a middle angle of g then the vertex of the MLP of g on e_2 cannot be an endpoint (i.e., a grid point) on e_2.*

This lemma provides interesting information about the relationship between locations of vertices of an MLP and the geometric structure of the given simple cube-curve. It will be applied in Section 3.4.1.

11. LEMMA. *Let $f(x)$ be a continuous function defined on an interval $[a, b]$, and assume $f(\xi) = 0$ for some $\xi \in (a, b)$. Then, for every $\varepsilon > 0$, there exist a' and b' such that, for every $x \in [a', b']$, we have $|f(x)| < \varepsilon$.*

Proof. Since $f(x)$ is continuous at $\xi \in (a, b)$, so $\lim_{n\to\xi} f(x) = f(\xi) = 0$. Then, for every $\varepsilon > 0$, there exists a $\delta > 0$ such that for every $x \in (\xi - \delta, \xi + \delta)$ we have that $|f(x)| < \varepsilon$. Let $a' = \xi - \frac{\delta}{2}$ and $b' = \xi + \frac{\delta}{2}$. Then, for every $x \in [a', b']$ we have that $|f(x)| < \varepsilon$. □

We apply this auxiliary lemma to prove the following lemma; the latter one will be used in Sections 3.3.1 and 3.3.3.

12. LEMMA. *Let $f(x)$ be a continuous function on an interval $[a, b]$, with $f(\xi) = 0$ at $\xi \in (a, b)$. Then, for every $\varepsilon > 0$, there are two integers $n > 0$ and $k > 0$ such that, for every $x \in [\frac{(k-1)(b-a)}{n}, \frac{k(b-a)}{n}]$, we have that $|f(x)| < \varepsilon$.*

Proof. By Lemma 11 it follows that for every $\varepsilon > 0$, there exist a' and b' such that for every $x \in [a', b']$ we have that $|f(x)| < \varepsilon$. Select an integer $n \geq \frac{2(b-a)}{b'-a'}$. Then, $\frac{b-a}{n} \leq \frac{b'-a'}{2} \leq b' - a'$. Thus, there is an integer j (where $j = 1, 2, \ldots, n-1$), such that $a' \leq \frac{j(b-a)}{n} \leq b'$. If $\frac{j(b-a)}{n} \leq \frac{b'-a'}{2}$, then $a' \leq \frac{j(b-a)}{n} \leq \frac{(j+1)(b-a)}{n} \leq b'$. If $\frac{j(b-a)}{n} \geq \frac{b'-a'}{2}$, then $a' \leq \frac{(j-1)(b-a)}{n} \leq \frac{j(b-a)}{n} \leq b'$. □

3.3 Algorithm

This section contains main ideas and steps of our algorithm for computing the MLP of a first-class simple cube-curve which has at least one end angle.

3.3.1 Basic Ideas

Let p_i be a point on e_i, where $i = 0, 1, 2, \ldots, n$. Let the coordinates of p_i be

$$(x_i + k_{x_i} t_i, y_2 + k_{y_i} t_i, z_i + k_{z_i} t_i)$$

where i = 0, 1, …, and n. Then the length of the polygon $p_0 p_1 \ldots p_n$ equals

$$d = d(t_0, t_1, \ldots, t_n) = \sum_{i=0}^{n} d_e(p_i, p_{i+1})$$

If the polygon $p_0 p_1 \ldots p_n$ is the MLP of g, then (by Lemma 4) we have $\frac{\partial d}{\partial t_i} = 0$, where $i = 0, 1, \ldots, n$.

Assume that e_i, e_{i+1}, and e_{i+2} form an end angle, and also e_j, e_{j+1}, and e_{j+2}, and that no other three consecutive critical edges between e_{i+2} and e_j form an end angle, where $i \leq j$ and $i, j = 0, 1, 2, \ldots, n$. By Theorem 8 we have

$$t_{i+3} = f_{i+3}(t_{i+1}, t_{i+2})$$

$$t_{i+4} = f_{i+4}(t_{i+2}, t_{i+3})$$

$$t_{i+5} = f_{i+5}(t_{i+3}, t_{i+4})$$

$$\ldots$$

$$t_j = f_j(t_{j-2}, t_{j-1})$$

and

$$t_{j+1} = f_{j+1}(t_{j-1}, t_j)$$

This shows that $t_{i+3}, t_{i+4}, t_{i+5}, \ldots, t_j$, and t_{j+1} can be represented by t_{i+1}, and t_{i+2}. In particular, we obtain an equation $t_{j+1} = f(t_{i+1}, t_{i+2})$, or

$$g(t_{j+1}, t_{i+1}, t_{i+2}) = 0 \tag{3.8}$$

where t_{j+1}, and t_{i+1} are already known, or

$$g_1(t_{i+2}) = 0 \tag{3.9}$$

Since e_i, e_{i+1}, and e_{i+2} form an end angle it follows that $e_{i+1} \perp e_{i+2}$. By Theorem 8 we can express $\frac{\partial d_2}{\partial t_{i+2}}$ either in the form

$$\frac{t_{i+2} - t_{i+1} - a_1}{\sqrt{(t_{i+2} - t_{i+1} - a_1)^2 + b_1^2}} + \frac{t_{i+2} - a_2}{\sqrt{(t_{i+2} - a_2)^2 + (t_{i+3} - b_2)^2 + c_2^2}} \tag{3.10}$$

or in the form

$$\frac{t_{i+2} - b_1}{\sqrt{(t_{i+1} - a_1)^2 + (t_{i+2} - b_1)^2 + c_1^2}} + \frac{t_{i+2} - a_2}{\sqrt{(t_{i+2} - a_2)^2 + (t_{i+3} - b_2)^2 + c_2^2}} \tag{3.11}$$

If t_{i+2} satisfies Equation (3.10), then

$$\frac{\partial d_2}{\partial t_{i+2}}(a_1') < 0, \quad \text{and} \quad \frac{\partial d_2}{\partial t_{i+2}}(a_2') > 0$$

where $a_1' = \min\{t_{i+1} + a_1, a_2\}$, and $a_2' = \max\{t_{i+1} + a_1, a_2\}$. It follows that Equation (3.10) has a unique real root between a_1' and a_2'. If t_{i+2} satisfies Equation (3.11), then Equation (3.11) has a unique real root between a_2 and b_1. In summary, there are two real numbers a and b such that Equation (3.11) has a unique root in between a and b. If $g_1(a)g_1(b) < 0$, then we can use the bisection method (see [25, page 49]) to find an approximate root of Equation (3.11). Otherwise, by Lemma 12 (see also Appendix A), we can also find an approximate root of Equation (3.11). Therefore, we can always find an approximate root for $\frac{\partial d}{\partial t_k} = 0$, where $k = i + 2, i + 3, \ldots$, and j, and an exact root for $\frac{\partial d}{\partial t_k} = 0$, where $k = i + 1$ and $j + 1$. In this way we will find an approximate or exact root t_{k_0} for $\frac{\partial d}{\partial t_k} = 0$, where k = 1,2, …, and n. Let $t'_{k_0} = 0$ if $t_{k_0} < 0$ and $t'_{k_0} = 1$ if $t_{k_0} > 1$, where $k = 1, 2, \ldots, n$. Then (by Lemma 7) we obtain an approximation of the MLP [its length equals $d(t'_{1_0}, t'_{2_0}, \ldots, t'_{i_0}, \ldots, t'_{n_0})$] of the given first-class simple cube-curve.

3.3.2 Main Steps of the Algorithm

The input is a first-class simple cube-curve g with at least one end angle. The output is an approximation of the MLP and a calculated length value.

2. ALGORITHM.

Step 1. Represent g by the coordinates of the endpoints of its critical edges e_i, where $i = 0, 1, 2, \ldots, n$. Let p_i be a point on e_i, where $i = 0, 1, 2, \ldots, n$. Then, the coordinates of p_i are equal to $(x_i + k_{x_i} t_i, y_2 + k_{y_i} t_i, z_i + k_{z_i} t_i)$, where only one of the parameters k_{x_i}, k_{y_i} and k_{z_i} can be equals 1, and the other two are equal to 0, for $i = 0, 1, \ldots, n$.

Step 2. Find all end angles $\angle(e_j, e_{j+1}, e_{j+2}), \angle(e_k, e_{k+1}, e_{k+2}), \ldots$ of g. For every $i \in \{0, 1, 2, \ldots, n\}$, let $d_{i+1} = d_e(p_i, p_{i+1}) + d_e(p_{i+1}, p_{i+2})$. By Lemma 6, we can find a unique root $t_{(i+1)_0}$ of equation $\frac{\partial d_{i+1}}{\partial t_{i+1}} = 0$ if e_i, e_{i+1} and e_{i+2} form an end angle.

Step 3. For every pair of two consecutive end angles

$$\angle(e_i, e_{i+1}, e_{i+2}) \quad \text{and} \quad \angle(e_j, e_{j+1}, e_{j+2})$$

of g, apply the ideas as described in Section 3.3.1 to find the root of equation $\frac{\partial d_k}{\partial t_k} = 0$, where $k = i + 1, i + 2, \ldots$, and $j + 1$.

Step 4. Repeat Step 3 until we find an approximate or exact root t_{k_0} for $\frac{\partial d}{\partial t_k} = 0$, where $d = d(t_0, t_1, \ldots, t_n) = \sum_{i=1}^{n-1} d_i$, for $k = 0, 1, 2, \ldots, n$. Let $t'_{k_0} = 0$ if $t_{k_0} < 0$, and $t'_{k_0} = 1$ if $t_{k_0} > 1$, for $k = 0, 1, 2, \ldots, n$.

Step 5. The output is a polygonal curve $p_0(t'_{1_0})p_1(t'_{2_0}) \ldots p_n(t'_{n_0})$ of total length

$$d(t'_{1_0}, t'_{2_0}, \ldots, t'_{i_0}, \ldots, t'_{n_0})$$

This curve approximates the MLP of g.

3.3.3 Time Complexity

We give an estimate of the complexity of our algorithm in dependency of the number of end angles m and the accuracy (tolerance ε) of approximation.

Let the accuracy of approximation be upper-limited by $\varepsilon = \frac{1}{2^k}$. By [25, page 49], the bisection method needs to know the initial end points a and b of the search interval $[a, b]$. At best, if we can set $a = 0$ and $b = 1$ to solve all the forms of Equation (3.9) by the bisection method, then the algorithm completes each run in $O(mk)$ time. In the worst case, if we have to find out the values of a and b for every of the forms of Equation (3.9) by the bisection method, then (by Lemma 12 and assume that we need $f(k)$ steps to find out the values of a and b), the algorithm completes each run in $O(mkf(k))$ time.

3.4 Experiments

We provide one example where we compare the results obtained with this new algorithm with those obtained with the original rubberband algorithm as described in [24].

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}
e_0	1	4	7	2	4	7
e_1	2	4	5	2	5	5
e_2	4	5	4	4	5	5
e_3	4	7	4	5	7	4
e_4	5	7	2	5	8	2
e_5	7	8	1	7	8	2
e_6	7	10	2	8	10	2
e_7	8	10	4	8	11	4
e_8	10	10	4	10	10	5
e_9	10	8	5	11	8	5
e_{10}	11	7	7	11	8	7
e_{11}	12	7	7	12	7	8
e_{12}	12	5	7	12	5	8
e_{13}	10	4	8	10	5	8
e_{14}	9	4	10	10	4	10
e_{15}	9	2	10	9	2	11
e_{16}	7	1	10	7	2	10
e_{17}	6	2	8	7	2	8
e_{18}	6	4	7	6	4	8
e_{19}	4	4	7	4	4	8
e_{20}	3	2	7	3	2	8
e_{21}	2	2	7	2	2	8

Table 3.1: Coordinates of endpoints of critical edges in Figure 1.2.

3.4.1 The Example

We approximate the MLP of the first-class simple cube-curve of Figure 1.2.

Step 1. See Table 3.1 which lists the coordinates of the critical edges $e_0, e_1, \ldots, e_{21}$ of g. Let p_i be a point on the critical line of e_i, where $i = 0, 1, \ldots, 21$.

Step 2. We calculate the coordinates of p_i, where $i = 0, 1, \ldots 21$, as follows: $(1 + t_0, 4, 7)$, $(2, 4{+}t_1, 5), (4, 5, 4{+}t_2), (4{+}t_3, 7, 4), (5, 7{+}t_4, 2), (7, 8, 1{+}t_5) \ldots (2, 2, 7{+}t_{21})$.

Step 3. Now let $d = d(t_0, t_1, \ldots, t_{21}) = \sum_{i=0}^{21} d_e(p_i, p_{i+1(\text{mod } 22)})$. Then we obtain

$$\frac{\partial d}{\partial t_0} = \frac{t_0 - 1}{\sqrt{(t_0 - 1)^2 + t_{21}^2 + 4}} + \frac{t_0 - 1}{\sqrt{(t_0 - 1)^2 + t_1^2 + 4}} \tag{3.12}$$

$$\frac{\partial d}{\partial t_1} = \frac{t_1}{\sqrt{(t_0-1)^2+t_1^2+4}} + \frac{t_1-1}{\sqrt{(t_1-1)^2+(t_2-1)^2+4}} \tag{3.13}$$

$$\frac{\partial d}{\partial t_2} = \frac{t_2-1}{\sqrt{(t_1-1)^2+(t_2-1)^2+4}} + \frac{t_2}{\sqrt{t_2^2+t_3^2+4}} \tag{3.14}$$

$$\frac{\partial d}{\partial t_3} = \frac{t_3}{\sqrt{t_2^2+t_3^2+4}} + \frac{t_3-1}{\sqrt{(t_3-1)^2+t_4^2+4}} \tag{3.15}$$

$$\frac{\partial d}{\partial t_4} = \frac{t_4}{\sqrt{(t_3-1)^2+t_4^2+4}} + \frac{t_4-1}{\sqrt{(t_4-1)^2+(t_5-1)^2+4}} \tag{3.16}$$

and

$$\frac{\partial d}{\partial t_5} = \frac{t_5-1}{\sqrt{(t_4-1)^2+(t_5-1)^2+4}} + \frac{t_5-1}{\sqrt{(t_5-1)^2+t_6^2+4}} \tag{3.17}$$

By Equations (3.12) and (3.17) we obtain $t_0 = t_5 = 1$.

Similarly, we have $t_7 = t_{15} = 0$, and $t_{16} = 1$. Therefore, we find all end angles as follows: $\angle(e_{21}, e_0, e_1)$, $\angle(e_4, e_5, e_6)$, $\angle(e_6, e_7, e_8)$, $\angle(e_{14}, e_{15}, e_{16})$, and $\angle(e_{15}, e_{16}, e_{17})$.

By Theorem 8 and Equations (3.13), (3.14), (3.15) it follows that

$$t_2 = 1 - \sqrt{\frac{(t_1-1)^2[(t_0-1)^2+4]}{t_1^2} - 4} \tag{3.18}$$

$$t_3 = \sqrt{\frac{t_2^2[(t_1-1)^2+4]}{(t_2-1)^2} - 4} \tag{3.19}$$

and

$$t_4 = \sqrt{\frac{(t_3-1)^2[t_2^2+4]}{t_3^2} - 4} \tag{3.20}$$

By Equation (3.16) we have

$$t_4^2[(t_5-1)^2+4] = (t_4-1)^2[(t_3-1)^2+4]$$

Let

$$g_1(t_1) = t_4^2[(t_5-1)^2+4] - (t_4-1)^2[(t_3-1)^2+4] \tag{3.21}$$

By Equation (3.18) we have that $t_1 \in (0, 0.5)$, $g_1(0.4924) = 3.72978 > 0$, and also $g_1(0.4999) = -51.2303 < 0$. By Lemmas 4, 6, 10, and the Bisection Method we obtain the following unique roots of Equations (3.21), (3.18), (3.19), and (3.20):

$t_1 = 0.492416$, $t_2 = 0.499769$, $t_3 = 0.499769$, and $t_4 = 0.507584$,

with error $g_1(t_1) = 4.59444 \times 10^{-9}$. These roots correspond to the two consecutive end angles $\angle(e_{21}, e_0, e_1)$ and $\angle(e_4, e_5, e_6)$ of g.

Step 4. Similarly, we find the unique roots of equation $\frac{\partial d}{\partial t_i} = 0$, where $i = 6, 7, \ldots, 21$. At first we have $t_6 = 0.5$, which corresponds to the two consecutive end angles $\angle(e_4, e_5, e_6)$ and $\angle(e_6, e_7, e_8)$; then we also obtain $t_8 = 0.492582$, $t_9 = 0.494543$, $t_{10} = 0.331074$, $t_{11} = 0.205970$, $t_{12} = 0.597034$, $t_{13} = 0.502831$, $t_{14} = 0.492339$, which correspond to the two consecutive end angles $\angle(e_6, e_7, e_8)$ and $\angle(e_{14}, e_{15}, e_{16})$; followed by $t_{15} = 0$, $t_{16} = 1$, which correspond to the two consecutive end angles $\angle(e_{14}, e_{15}, e_{16})$ and $\angle(e_{15}, e_{16}, e_{17})$; and finally $t_{17} = 0.501527$, $t_{18} = 0.77824$, $t_{19} = 0.56314$, $t_{20} = 0.32265$, and $t_{21} = 0.2151$, which correspond to the two consecutive end angles $\angle(e_{15}, e_{16}, e_{17})$ and $\angle(e_{21}, e_0, e_1)$.

Step 5. In summary, we obtain the values shown in the first two columns of Table 3.1. The calculated approximation of the MLP of g is

$$p_0(t'_{1_0})p_1(t'_{2_0}) \ldots p_n(t'_{n_0}),$$

and its length is $d(t'_{1_0}, t'_{2_0}, \ldots, t'_{i_0}, \ldots, t'_{n_0})$ = 43.767726, where $t'_{i_0} = t_{i_0}$ for i limited to the set $\{0, 1, 2, \ldots, 21\}$.

3.4.2 Comparison with the Original Rubberband Algorithm

The original rubberband algorithm [24] calculated the roots of Equations (3.12) through (3.17) as shown in the third column of Table 3.2. Note that there is only a finite number of iterations until the algorithm terminates. No threshold needs to be specified for the chosen input curve. From Table 3.2 we can see that both algorithms converge (within some minor numerical deviations) to identical values.

3.5 Conclusions

We designed an algorithm for the approximative calculation of an MLP for a special class of simple cube-curves (namely, first-class simple cube-curves with at least one end angle). Mathematically, the problem is equivalent to solving equations having one variable each. Applying methods of numerical analysis, we can compute their roots with sufficient accuracy. We illustrated the algorithm by one non-trivial example, illustrating this way that our algorithm found the same approximation of an MLP as the original rubberband algorithm.

Critical points	t_{i_0} (Our algorithm)	t_{i_0} (Rubberband algorithm)
p_0	1	1
p_1	0.492416	0.4924
p_2	0.499769	0.4998
p_3	0.499769	0.4998
p_4	0.507584	0.5076
p_5	1	1
p_6	0.5	0.5
p_7	0	0
p_8	0.492582	0.4926
p_9	0.494543	0.4945
p_{10}	0.331074	0.3311
p_{11}	0.205970	0.2060
p_{12}	0.597034	0.5970
p_{13}	0.502831	0.5028
p_{14}	0.492339	0.4923
p_{15}	0	0
p_{16}	1	1
p_{17}	0.501527	0.5015
p_{18}	0.77824	0.7789
p_{19}	0.56314	0.5641
p_{20}	0.32265	0.3235
p_{21}	0.2151	0.2157

Table 3.2: Comparison of results of both algorithms.

Chapter 4

MLPs of First-Class Simple Cube Curves

This chapter provides two correctness proofs for the original rubberband algorithm assuming that the input is restricted to first-class simple cube-curves only. The first proof is longer, but also allows to prove the uniqueness of the MLP. The second proof is rater short, and not containing such a uniqueness proof. [The developed proof methodology is later (in Chapter 6) generalized to prove theorems about general simple cube-curves.]

4.1 Introduction

Chapter 3 presented an algorithm for the calculation of the correct MLP (with correctness proof) for the special type of first-class cube-curves which have at least one end angle. The main idea is to decompose such a cube-curve into arcs by finding "end angles" (see Definition 16 in Chapter 3).

In Chapter 5 we construct an example of a first-class simple cube-curve whose MLP does not have any vertex at a grid point position. In general, we then show that any cube-curve, satisfying this property, does not have any end angle. This means that we cannot use the (provably correct) MLP algorithm of Chapter 3 for first-class simple cube curves in general. This is the basic importance of the existence result in Chapter 5: the existence of cube-curves which don not have any vertex at a grid point position shows the need of further algorithmic studies for solving the general MLP problem.

Chapters 3 and 5 both focus on first-class simple cube-curves (see Definition 14). This chapter proves that the original rubberband algorithm is correct for the family of all first-class simple cube-curves.

The chapter is organized as follows: Section 4.2 defines notations used in this chapter. Section 4.3 provides results and their proofs. Section 4.4 gives the conclusions.

4.2 Definitions

This section introduces some definitions used in proofs in the next section. Some of them are from mathematical analysis or multivariable calculus and elementary topology textbook.

19. DEFINITION. One iteration *of a rubberband algorithm is a complete pass through the main loop of the algorithm. At the end of iteration $n \geq 1$ we obtain the nth* approximate MLP*, denoted by $AMLP_n(g)$, for a given simple cube-curve g.*

We assume that the sequence of the nth approximate MLPs is converging against a polygonal curve; let

$$AMLP(g) = \lim_{n \to \infty} AMLP_n(g)$$

be this polygonal curve.

Let $p_i(t_{i_0})$ be the i-th vertex of the $AMLP(g)$, where $i = 0, 1, \ldots, m+1$. Parameter $t_{i_0} \in [0, 1]$ identifies the ith vertex of $AMLP(g)$ on the critical line l_i, as specified earlier. Let

$$d_i = d_e(p_{i-1}, p_i) + d_e(p_i, p_{i+1})$$

where i = 1, 2, . . ., or m. Let

$$d(t_0, t_1, \ldots, t_m, t_{m+1}) = \sum_{1}^{m} d_i$$

Obviously, $d(t_0, t_1, \ldots, t_m, t_{m+1})$ is an $(m + 2)$-ary function on the domain $[0, 1]^{m+2}$.

Let e_0, e_1, e_2, . . . e_m and e_{m+1} be all consecutive critical edges of a simple cube-curve g, and $p_i \in e_i$, for $i = 0, 1, 2, \ldots, m + 1$. We call the $m + 2$ tuple (p_0, p_1, p_2, . . ., p_m, p_{m+1}) a *critical point tuple* of g. We call it an *AMLP critical point tuple* of g if it is the set of the vertices of the $AMLP$ of g.

Now let P =(p_0, p_1, p_2, . . ., p_m, p_{m+1}) be a critical point tuple of g. Using P as an initial point set, defining $AMLP_0(g)$, and n iterations of the rubberband algorithm, we get another critical point tuple of g, say $P' = (p'_0, p'_1, p'_2, \ldots, p'_m, p'_{m+1})$, which defines (see above) the nth approximate polygon $AMLP_n(g)$, or $AMLP_n$ for short.

20. DEFINITION. *Let*

$$\frac{\partial d(t_0, t_1, \ldots, t_m, t_{m+1})}{\partial t_i}|_{t_{i0}} = 0$$

where i = 0, 1, . . ., or $m + 1$. Then we say that $(t_{00}, t_{10}, \ldots, t_{m0}, t_{m+10})$ is a critical point *of*

$$d(t_0, t_1, \ldots, t_m, t_{m+1})$$

Let P =(p_0, p_1, p_2, . . ., p_m, p_{m+1}) be a critical point tuple of g. Using P as an initial point set, n iterations of the rubberband algorithm, 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 g, and n is a positive integer.

Let P =(p_0, p_1, p_2, . . ., p_m, p_{m+1}) be a critical point tuple of g. For sufficiently small real $\varepsilon > 0$, the set

$$\{(p'_0, p'_1, p'_2, \ldots, p'_m, p'_{m+1}) : x'_i \in (x_i - \varepsilon, x_i + \varepsilon) \wedge y'_i \in (y_i - \varepsilon, y_i + \varepsilon) \\ \wedge z'_i \in (z_i - \varepsilon, z_i + \varepsilon) \wedge p'_i = (x'_i, y'_i, z'_i) \\ \wedge p_i = (x_i, y_i, z_i) \wedge i = 0, 1, 2, \ldots, m, m+1\}$$

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

The ε-neighborhood of P is an open set in the usual, Euclidean $(m+2)$-dimensional topological space $(\mathbb{R}^{m+2}, T)$; T is the topology, that means the family of all open sets in $\mathbb{R}^{m+2}$. We also use the following definition [see Definition 4.1 in [109]]:

21. DEFINITION. *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 $T|_Y$.*

4.3 Two Proofs

In this section we present two versions of proofs to show that the original rubberband algorithm is correct for any first-class simple cube-curve. The first one is longer but with a stronger result: we not only prove that the algorithm is correct but also show that the MLP is unique. The second one is very short but without proving the uniqueness of the MLP.

4.3.1 A Proof Without Using Convex Analysis

Let e_0, e_1, e_2, . . ., e_m and e_{m+1} be $m + 2$ consecutive critical edges in a simple cube-curve, and let l_0, l_1, l_2, . . ., l_m and l_{m+1} be the corresponding critical lines. We express a point

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

on l_i in general form, with $t_i \in \mathbb{R}$, where i = 0, 1, . . ., or $m+1$. In the following, $p_i(t_i)$ will also be denoted by p_i for short, where i = 0, 1, . . ., or $m + 1$.

The following is a multivariable version of Fermat's Theorem in mathematical analysis (see, e.g., [50], Theorem 8.8.1). We will use it for proving Lemma 13; this lemma is then applied in the proofs of Lemmas 15 and Theorem 11.

9. THEOREM. (Fermat's Theorem) *Let f = f($t_1, t_2, \dots, t_k$) be a real-valued function defined on an open set U in $\mathbb{R}^k$. Let C = ($t_{10}, t_{20}, \dots, t_{k0}$) 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 $AMLP$, where $i = 0, 1, \dots, m+1$. Then we have the following:

13. LEMMA. $(t_{00}, t_{10}, \dots, t_{m0}, t_{m+10})$ *is a critical point of* $d(t_0, t_1, \dots, t_m, t_{m+1})$.

Proof. $d(t_0, t_1, \dots, t_m, t_{m+1})$ is differentiable at each point

$$(t_0, t_1, \dots, t_m, t_{m+1}) \in [0, 1]^{m+2}$$

Because $AMLP_n(g)$ is the nth polygon of g, where $n = 1, 2, \dots$, and

$$AMLP = \lim_{n \to \infty} AMLP_n(g)$$

it follows that $d(t_{0_0}, t_{1_0}, \dots, t_{m_0}, t_{m+1_0})$ is a local minimum of $d(t_0, t_1, \dots, t_m, t_{m+1})$. By Theorem 9,

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

where i = 0, 1, 2, . . ., $m+1$. By Definition 20, $(t_{00}, t_{10}, \dots, t_{m0}, t_{m+10})$ is a critical point of $d(t_0, t_1, \dots, t_m, t_{m+1})$. □

By Lemma 13 and Theorems 2, 3 and 4 of [94], we immediately obtain the following theorem.

10. THEOREM. *If e_{i-1}, e_i and e_{i+1} form an end angle, then t_{i_0} = 0 or 1; otherwise we have $0 < t_{i_0} < 1$, where $i = 1, 2, \dots, m$.*

In other words, vertices of an MLP are located at grid points at end angles, and only there. The following two lemmas indicate the form of partial derivative with respect to parameter t_2. They are later used to prove an important lemma (namely, Lemma 14).

By the proofs of Lemmas 1 and 2 of [94], we also have the following two properties:

If $e_1 \perp e_2$, then

$$\frac{\partial d_e(p_1, p_2)}{\partial t_2} = \frac{t_2 - \alpha}{\sqrt{(t_2 - \alpha)^2 + (t_1 - \beta)^2 + \gamma}} \tag{4.1}$$

for some reals α, β, and γ. If $e_1 \parallel e_2$, then

$$\frac{\partial d_e(p_1,p_2)}{\partial t_2} = \frac{t_2 - t_1}{\sqrt{(t_2-t_1)^2+\alpha}} \tag{4.2}$$

for some real α.

The following lemma is very important for the proof of Lemma 15 which estimates the number of $AMLP$ critical point tuples.

14. LEMMA. *The number of critical points of* $d(t_0, t_1, \ldots, t_m, t_{m+1})$ *in* $[0,1]^{m+2}$ *is finite.*

Proof. Let $d = d(t_0, t_1, \ldots, t_m, t_{m+1})$.

Case 1. The simple cube-curve g has at least one end angle.

Assume that e_i, e_{i+1}, and e_{i+2} form an end angle, and also e_j, e_{j+1}, and e_{j+2}, and no other three consecutive critical edges between e_{i+2} and e_j form an end angle, where $i \leq j$ and $i, j = 0, 1, 2, \ldots, m-2$. By Theorem 4 of [93] and Theorem 10 we have that

$$t_{i+3} = f_{i+3}(t_{i+1}, t_{i+2})$$

$$t_{i+4} = f_{i+4}(t_{i+2}, t_{i+3})$$

$$t_{i+5} = f_{i+5}(t_{i+3}, t_{i+4})$$

$$\ldots\ldots$$

$$t_j = f_j(t_{j-2}, t_{j-1})$$

and

$$t_{j+1} = f_{j+1}(t_{j-1}, t_j)$$

This shows that t_{i+3}, t_{i+4}, t_{i+5}, ..., t_j, and t_{j+1} can be represented by t_{i+1}, and t_{i+2}. In particular, we obtain an equation $t_{j+1} = f(t_{i+1}, t_{i+2})$, or

$$g(t_{j+1}, t_{i+1}, t_{i+2}) = 0$$

where t_{j+1}, and t_{i+1} are already known, or

$$g_1(t_{i+2}) = 0 \tag{4.3}$$

By Equations (4.1) and (4.2), function $g_1(t_{i+2})$ can be decomposed into finitely many monotonous functions. Therefore, Equation (4.3) has only a finite number of solutions. This implies that the system formed by

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

(where $i = 0, 1, \ldots, m + 1$) has a finite number of solutions as well.

Case 2. The simple cube-curve g does not have any end angle.

Analogous to Case 1, the system formed by

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

(where $i = 0, 1, \ldots, m + 1$) implies a two-variable system formed by

$$h_1(t_0, t_1) = 0 \tag{4.4}$$

$$h_2(t_0, t_1) = 0 \tag{4.5}$$

There does not exist an interval $I \subset$ [0, 1] such that t_0 is a function of t_1 over I, denoted by t_0 = $f_1(t_1)$ such that

$$h_1(f_1(t_1), t_1) = 0 \tag{4.6}$$

$$h_2(f_1(t_1), t_1) = 0 \tag{4.7}$$

Otherwise, both Equations (4.6) and (4.7) would have an infinite number of real roots (see Figure 4.1). This is a contradiction. Because by Equations (4.1) and (4.2), either Equation (4.6) or (4.7) can be simplified to a polynomial which can only have

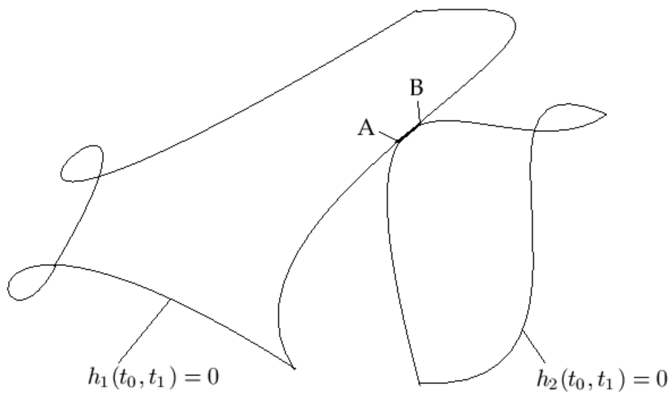


Figure 4.1: Illustration of Case 2 of the proof of Lemma 14, where the arc AB is the intersection of the graph of equation $h_1(t_0, t_1) = 0$ and that of equation $h_2(t_0, t_1) = 0$.

a finite number of real roots. Analogously, there does not exist an interval $I \subset [0,1]$ such that t_1 is a function of t_0 over I, denoted by $t_1 = f_2(t_0)$ such that

$$h_1(t_0, f_2(t_0)) = 0 \tag{4.8}$$

$$h_2(t_0, f_2(t_0)) = 0 \tag{4.9}$$

Again, by Equations (4.1) and (4.2), Equations (4.4) and (4.5) can be decomposed into a finite number of monotonous functions. Thus, the system formed by Equations(4.4) and (4.5) has a finite number of solutions. This implies that the system formed by

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

(where $i = 0, 1, \ldots, m+1$) has finitely many solutions. □

By Lemmas 13 and 14, we have the following:

15. LEMMA. *Any simple cube-curve g has only a finite number of $AMLP$ critical point tuples.*

This is our first important lemma in this section. In the rest of this section, based on Lemma 15, we show a much stronger result: g has actually only one (!) $AMLP$ critical point tuple.

Let e_0, e_1 and e_2 be three consecutive critical edges. Let $p_i = (p_{i_1}, p_{i_2}, p_{i_3})$ be on e_i, for $i = 0, 1, 2$. The proof of the following lemma specifies an explicit expression for the relation between parameter t and the optimum point p_1.

16. LEMMA. *Optimum point $p_1 \in e_1$, defined by*

$$d_e(p_1, p_0) + d_e(p_1, p_2) = \min\{p_1 : d_e(p_1, p_0) + d_e(p_1, p_2) \wedge p_1 \in e_2\}$$

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

Proof. Let the two endpoints of e_1 be $a_1 = (a_{1_1}, a_{1_2}, a_{1_3})$ and $b_1 = (b_{1_1}, b_{1_2}, b_{1_3})$. Let $p_0 = (p_{0_1}, p_{0_2}, p_{0_3})$. Point p_1 can be written as

$$(a_{1_1} + (b_{1_1} - a_{1_1})t, a_{1_2} + (b_{1_2} - a_{1_2})t, a_{1_3} + (b_{1_3} - a_{1_3})t)$$

The formula

$$d_e(p_1, p_0) = \sqrt{\sum_{i=1}^{3} [(a_{1_i} - p_{0_i}) + (b_{1_i} - a_{1_i})t]^2}$$

can be simplified: The straight line $a_1 b_1$ is isothetic (i.e., parallel to one of the three coordinate axes). It follows that only one element of the set

$$\{b_{1_i} - a_{1_i} : i = 1, 2, 3\}$$

is equal to 1, and the other two are equal to 0. Without loss of generality we can assume that

$$d_e(p_1, p_0) = \sqrt{(t + A_1)^2 + B_1}$$

where A_1 and B_1 are functions of a_{1_i}, b_{1_i} and p_{0_i}, for i = 0, 1, 2. – Analogously,

$$d_e(p_1, p_2) = \sqrt{(t + A_2)^2 + B_2}$$

where A_2 and B_2 are functions of a_{1_i}, b_{1_i} and p_{2_i}, for i = 0, 1, 2. In order to find a point $p_1 \in e_1$ such that

$$d_e(p_1, p_0) + d_e(p_1, p_2) = \min\{p_1 : d_e(p_1, p_0) + d_e(p_1, p_2), p_1 \in e_1\}$$

we can solve the equation

$$\frac{\partial(d_e(p_1, p_0) + d_e(p_1, p_2))}{\partial t} = 0$$

The unique solution is

$$t = -(A_1 B_2 + A_2 B_1)/(B_2 + B_1)$$

This proves the lemma. □

By the proof of Lemma 16, assuming the representation

$$p_i = (a_{i_1} + (b_{i_1} - a_{i_1})t_i, a_{i_2} + (b_{i_2} - a_{i_2})t_i, a_{i_3} + (b_{i_3} - a_{i_3})t_i)$$

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

17. LEMMA. *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:

18. LEMMA. *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 $P' \rightarrow_{r-b_1} Q'$ implies $Q' \in U_\varepsilon(Q)$.*

Proof. By Lemma 16 and note that g has $m + 2$ critical edges; thus we use Lemma 17 repeatedly $m + 2$ times, and this proves this lemma. □

By Lemma 18, we have the following:

19. LEMMA. *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 23; the latter one and the following three lemmas are then finally applied to prove the second important lemma (i.e., Lemma 24) in this section. Lemmas 24 and 13 imply then the main theorem (i.e., Theorem 11 below) of this chapter.

By Lemma 15, let Q_1, Q_2, ..., Q_N with $N \geq 1$ be the set of all $AMLP$ critical point tuples of g. 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, ..., N and $i \neq j$. Let

$$D_i = \{P : P \rightarrow Q' \wedge Q' \in U_\varepsilon(Q_i) \wedge P \in [0,1]^{m+2}\}$$

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

The statements in the following two lemmas are obvious:

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

21. LEMMA. $\bigcup_{i=1}^N D_i = [0,1]^{m+2}$

We consider the usual Euclidean topology T on $\mathbb{R}^{m+2}$, and its relativization $T = \mathbb{R}^{m+2}|_{[0,1]^{m+2}}$.

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

Proof. By Lemma 19, 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. □

For the following lemma, which is characterizing open sets in general, see, for example, [150], Proposition 5.1.4.

23. LEMMA. *Let $U \subset R$ 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 section:

24. LEMMA. *g has a unique $AMLP$ critical point tuple.*

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

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

where e_j is a critical edge of g, for $i, j = 1, 2, \ldots, N$. Otherwise we have

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

This is a contradiction to Lemma 20.

Let

$$E = \{e_j : D_i|_{e_j} \subset [0, 1]\}$$

where e_j is any critical edge of g. We can select a critical point tuple of g as follows: go through each $e \in \{e_0, e_1, \ldots, e_m, e_{m+1}\}$. If $e \in E$, by Lemmas 22 and 23, select the minimum left endpoint of the open intervals whose union is $D_i|_e$. Otherwise select the midpoint of e. We denote the resulting critical point tuple as

$$P = (p_0, p_1, p_2, \ldots, p_{m+1})$$

By the selection of P, we know that P is not in D_i. By Lemma 21 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 20. □

Let g be a simple cube-curve. Let $AMLP_n(g)$ be the nth approximate polygon of g, for $n = 1, 2, \ldots$. The section has shown that

$$AMLP = \lim_{n \to \infty} AMLP_n(g)$$

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

11. THEOREM. *The $AMLP$ of g is the MLP of g, or, in short $AMLP = MLP$.*

Proof. By Lemma 24 and the proof of Lemma 13, $d(t_0, t_1, \ldots, t_m, t_{m+1})$ has a unique local minimal value. This implies that the $AMLP$ of g is the MLP of g. □

4.3.2 A Shorter Proof by Using Convex Analysis

This section gives a shorter proof of the correctness of the original rubberband algorithm by applying some basic results from convex analysis (but without obtaining the uniqueness result for the MLP).

The following basic results of convex analysis may be found, for example, in [20, 122, 123]:

1. PROPOSITION. *([20], page 27) Each line segment is a convex set.*

2. PROPOSITION. *([20], page 72) Each norm on $\mathbb{R}^n$ is a convex function.*

3. PROPOSITION. *([20], page 79) A nonnegative weighted sum of convex functions is a convex function.*

12. THEOREM. *([123], Theorem 3.5) 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, $n \in \mathbb{N}$.

4. PROPOSITION. *([123], page 264) 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, the interval [0, 1] is a convex set. By Theorem 12, $[0, 1]^{m+2}$ is a convex set. For any p, $q \in \mathbb{R}^n$, $d_e(p, q)$ is a norm (see, for example, [85], page 78). By Proposition 2 and 3, $d(t_0, t_1, \ldots, t_m, t_{m+1})$ (see Section 4.3.1) is a convex function on $[0, 1]^{m+2}$. Since $d(t_0, t_1, \ldots, t_m, t_{m+1})$ is continuous on $[0, 1]^{m+2}$, so its minimum is attained. It is clear that, for any first-class simple cube-curve, the original rubberband algorithm will always obtain an approximative local minimum of the function $d(t_0, t_1, \ldots, t_m, t_{m+1})$. By Proposition 4, each local minimum of $d(t_0, t_1, \ldots, t_m, t_{m+1})$ is its global minimum. Therefore, we prove Theorem 11 once again.

4.4 Conclusions

We have proved that the original rubberband algorithm is correct for the family of first-class simple cube-curves and that there is unique MLP for a given first-class simple cube-curve.

Although the proof in Section 4.3.1 is much more complicated than the one in Section 4.3.2, we proved a stronger result there, namely, that for each first-class simple cube-curve, the original rubberband algorithm will converge to a unique MLP.

It might be difficult to prove uniqueness simply by using convex analysis, and leave that here as an open problem.

Later we apply the methodology developed in Section 4.3.1 for a correctness proof for general simple cube-curves; see Section 6.3. We further generalize this methodology to prove the correctness of algorithms for some art gallery problems; see Chapter 11.

Chapter 5

Analysis of the Original Algorithm

This chapter proves that the original rubberband algorithm has κ-linear time complexity $\kappa(\varepsilon) \cdot \mathcal{O}(m)$, where $\kappa(\varepsilon)$ is as in Equation (1.1), and m is the number of critical edges of a given simple cube curve. This answers a question for this algorithm, which had been open so far. However, we also find two counterexamples to Option 2 of the original rubberband algorithm (for non-first-class simple cube-curves) which lead to a correction of Option 3 (by adding one missing test) of the original rubberband algorithm.

These counterexamples were found by studying the following open problem, as formulated in [85]: Is there a simple cube-curve such that none of the nodes of its MLP is a grid vertex? This chapter constructs an example of such a simple cube-curve, and we also characterize the class of all of such cube-curves.

5.1 Introduction

An open problem (see [85, page 406]) was stated as follows: Is there a simple cube-curve such that none of the nodes of its MLP is a grid vertex? This chapter constructs an example of such a simple cube-curve, and generalizes this by characterizing the class of all of those cube-curves. Cube-curves in this class do not have any end angle; this means that we cannot use the MLP algorithm proposed in Chapter 2 (see also [93]) which is provably correct. This is the basic importance of the given result: we show the existence of cube-curves which require further algorithmic studies.

In this chapter we also study the original rubberband algorithm carefully and find two counterexamples to its Option 2. This can be compensated by a slight correction of Option 3 in the original rubberband algorithm. – We start with two rather technical definitions used in this chapter:

22. DEFINITION. *Let e be a critical edge of g. Let P_1 and P_2 be the two end points of e. If one of the coordinates of P_1 is less than the corresponding coordinate of P_2, then P_1 is called the* first end point of e, *otherwise P_1 is called the* second end point of e.

Let $e_1, e_2, \ldots, e_m$ be a subsequence of the sequence of all consecutive critical edges $\ldots, e_0, e_1, \ldots, e_m, e_{m+1}, \ldots$ of a cube-curve g. Let $m \geq 2$.

23. DEFINITION. *If $e_0 \perp e_1$, $e_m \perp e_{m+1}$, and $e_i \parallel e_{i+1}$, where $i = 1, 2, \ldots, m-1$, then e_1, $e_2, \ldots, e_m$ is a* maximal run of parallel critical edges *of g, and critical edges e_0 or e_{m+1} are called* adjacent to *this run.*

Figure 1.2 shows a simple cube-curve which has two maximal runs of parallel critical edges: e_{11}, e_{12} and $e_{18}, e_{19}, e_{20}, e_{21}$. The two adjacent critical edges of run e_{11}, e_{12} are e_{10} and e_{13}; they are on two different grid planes. The two adjacent critical edges of run $e_{18}, e_{19}, e_{20}, e_{21}$ are e_{17} and e_0; they are also on two different grid planes.

This chapter still focuses on first-class simple cube-curves; general simple cube-curves are discussed in Chapter 6.

The chapter is organized as follows. Section 5.2 describes theoretical fundamentals for constructing our example in Section 5.4 where non of the nodes of the MLP is a grid vertex. Section 5.2 also presents theoretical fundamentals for Section 5.6. Section 5.3 proves that the original rubberband algorithm has linear time complexity $\kappa(\epsilon) \cdot \mathcal{O}(m)$, where $\kappa(\epsilon)$ is as in Equation (1.1), and m is the number of critical edges of a given simple cube curve. Section 5.4 constructs an example to answer the open problem stated in [85, page 406]. Section 5.5 presents two counterexamples to Option 2 of the original rubberband algorithm. Section 5.6 improves the original rubberband algorithm in its Option 3. Section 5.7 gives a few conclusions.

5.2 Theoretical Results

5.2.1 Theorem on End Angles

Let e_0, e_1, e_2, …, e_m and e_{m+1} be $m+2$ consecutive critical edges in a simple cube-curve, and let l_0, l_1, l_2, …, l_m and l_{m+1} be the corresponding critical lines. We express a point $p_i(t_i) = (x_i + k_{x_i} t_i, y_i + k_{y_i} t_i, z_i + k_{z_i} t_i)$ on l_i in general form, with $t_i \in \mathbb{R}$, where i equals 0, 1, …, or $m+1$.

In the following, $p_i(t_i)$ will be denoted by p_i for short, where i equals 0, 1, …, or $m+1$.

25. LEMMA. *If $e_1 \perp e_2$, then $\frac{\partial d_e(p_1, p_2)}{\partial t_2}$ can be written as $(t_2 - \alpha)\beta$, where $\beta > 0$ is a function of t_1 and t_2, and $\alpha = 0$ if e_1 and the first end point of e_2 are on the same grid plane, or $\alpha = 1$ otherwise.*

Proof. Without loss of generality, we can assume that e_2 is parallel to the z-axis. In this case, the parallel projection [denoted by $g'(e_1, e_2)$] of all of g's cubes, contained

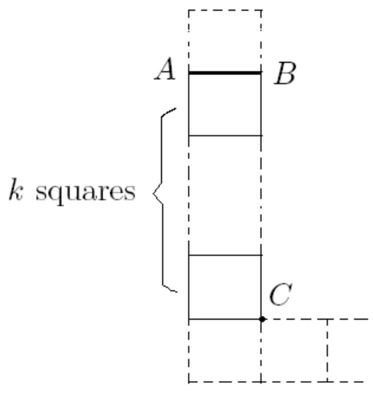


Figure 5.1: Illustration for the proof of Lemma 25.

between e_1 and e_2, is illustrated in Figure 5.1, where AB is the projective image of e_1, and C is that of one of the end points of e_2.

Case 1. e_1 and the first end point of e_2 are on the same grid plane. Let the two end points of e_2 be (a, b, c) and $(a, b, c+1)$. Then the two end points of e_1 are $(a-1, b+k, c)$ and $(a, b + k, c)$. Then the coordinates of p_1 and p_2 are $(a - 1 + t_1, b + k, c)$ and $(a, b, c + t_2)$ respectively, and $d_e(p_1, p_2) = \sqrt{(t_1 - 1)^2 + k^2 + {t_2}^2}$. Therefore,

$$\frac{\partial d_e(p_1, p_2)}{\partial t_2} = \frac{t_2}{\sqrt{(t_1 - 1)^2 + k^2 + {t_2}^2}}$$

Let $\alpha = 0$ and

$$\beta = \frac{1}{\sqrt{(t_1 - 1)^2 + k^2 + {t_2}^2}}$$

This proves the lemma for Case 1.

Case 2. Now assume that e_1 and the first end point of e_2 are on different grid planes (i.e., e_1 and the second end point of e_2 are on the same grid plane). Let the two end points of e_2 be (a, b, c) and $(a, b, c+1)$. Then the two end points of e_1 are $(a-1, b+k, c+1)$ and $(a, b+k, c+1)$. Then the coordinates of p_1 and p_2 are $(a-1+t_1, b+k, c+1)$ and $(a, b, c+t_2)$, respectively, and $d_e(p_1, p_2) = \sqrt{(t_1 - 1)^2 + k^2 + (t_2 - 1)^2}$. Therefore,

$$\frac{\partial d_e(p_1, p_2)}{\partial t_2} = \frac{t_2 - 1}{\sqrt{(t_1 - 1)^2 + k^2 + (t_2 - 1)^2}}$$

Let $\alpha = 1$ and

$$\beta = \frac{1}{\sqrt{(t_1 - 1)^2 + k^2 + (t_2 - 1)^2}}$$

This proves the lemma for Case 2. □

26. LEMMA. *If* $e_1 \parallel e_2$, *then*

$$\frac{\partial d_e(p_1,p_2)}{\partial t_2} = (t_2 - t_1)\beta$$

for some $\beta > 0$, *where* β *is a function in* t_1 *and* t_2.

Proof. Without loss of generality, we can assume that e_2 is parallel to the z-axis. In this case, the parallel projection [denoted by $g'(e_1, e_2)$] of all of g's cubes contained between e_1 and e_2 is illustrated in Figure 5.2, where A is the projective image of one of the end points of e_1, and B is that of one of the end points of e_2.

Case 1. Edges e_1 and e_2 are on the same grid plane. Let the two end points of e_2 be (a, b, c) and $(a, b, c+1)$. Then the two end points of e_1 are $(a, b+k, c)$ and $(a, b+k, c+1)$. Then the coordinates of p_1 and p_2 are $(a, b+k, c+t_1)$ and $(a, b, c+t_2)$, respectively, and $d_e(p_1, p_2) = \sqrt{(t_2-t_1)^2 + k^2}$. Therefore,

$$\frac{\partial d_e(p_1,p_2)}{\partial t_2} = \frac{t_2 - t_1}{\sqrt{(t_2-t_1)^2 + k^2}}$$

Let

$$\beta = \frac{1}{\sqrt{(t_2-t_1)^2 + k^2}}$$

This proves the lemma for Case 1.

Case 2. Now assume that edges e_1 and e_2 are on different grid planes. Let the two end points of e_2 be (a, b, c) and $(a, b, c+1)$. Then the two end points of e_1

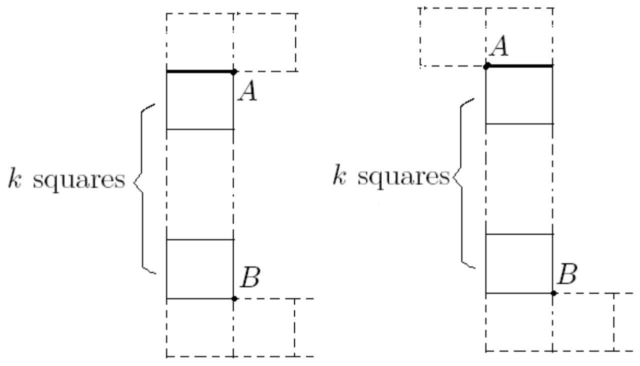


Figure 5.2: Illustration for the proof of Lemma 26. Left: Case 1. Right: Case 2.

are $(a-1, b+k, c)$ and $(a-1, b+k, c+1)$. Then the coordinates of p_1 and p_2 are $(a-1, b+k, c+t_1)$ and $(a, b, c+t_2)$ respectively, and $d_e(p_1, p_2) = \sqrt{(t_2-t_1)^2+k^2+1}$. Therefore,

$$\frac{\partial d_e(p_1, p_2)}{\partial t_2} = \frac{t_2 - t_1}{\sqrt{(t_2-t_1)^2+k^2+1}}$$

Let

$$\beta = \frac{1}{\sqrt{(t_2-t_1)^2+k^2+1}}$$

This proves the lemma for Case 2. □

This lemma will be used later for the proof of Lemma 30. – Let $d_i = d_e(p_{i-1}, p_i) + d_e(p_i, p_{i+1})$, for $i = 1, 2, \ldots, m$.

13. THEOREM. *If $e_i \perp e_j$, where $i, j = 1, 2, 3$ and $i \neq j$, then e_1, e_2 and e_3 form an end angle iff the equation*

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = 0$$

has a unique root 0 or 1.

Proof. Without loss of generality, we can assume that e_2 is parallel to the z-axis.

(A) If e_1, e_2 and e_3 form an end angle, then by Definition 16, the z-coordinates of two end points of e_1 and e_3 are equal.

Case A1. Edges e_1, e_3 and the first end point of e_2 are on the same grid plane. By Lemma 25,

$$\frac{\partial(d_e(p_1, p_2)}{\partial t_2} = (t_2 - \alpha_1)\beta_1$$

where $\alpha_1 = 0$ and $\beta_1 > 0$, and

$$\frac{\partial(d_e(p_2, p_3)}{\partial t_2} = (t_2 - \alpha_2)\beta_2$$

where $\alpha_2 = 0$ and $\beta_2 > 0$. Thus, we have

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = t_2(\beta_1 + \beta_2)$$

Therefore, the equation of the theorem has the unique root $t_2 = 0$.

Case A2. Edges e_1, e_3 and the second end point of e_2 are on the same grid plane. By Lemma 25,

$$\frac{\partial(d_e(p_1, p_2)}{\partial t_2} = (t_2 - \alpha_1)\beta_1$$

where $\alpha_1 = 1$ and $\beta_1 > 0$, and

$$\frac{\partial(d_e(p_2, p_3)}{\partial t_2} = (t_2 - \alpha_2)\beta_2$$

where $\alpha_2 = 1$ and $\beta_2 > 0$. Thus, we have

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = (t_2 - 1)(\beta_1 + \beta_2)$$

Therefore, the equation of the theorem has the unique root $t_2 = 1$.

(B) Conversely, if the equation of the theorem has a unique root 0 or 1, then e_1, e_2 and e_3 form an end angle. Otherwise, e_1, e_2 and e_3 form an inner angle. By Definition 16, the z-coordinates of two end points of e_1 are not equal to z-coordinates of two end points of e_3 (Note: Without loss of generality, we can assume that $e_2 \parallel z$-axis.). So e_1 and e_3 are not on the same grid plane.

Case B1. Let edge e_1 and the first end point of e_2 are on the same grid plane, while e_3 and the second end point of e_2 are on the same grid plane. By Lemma 25,

$$\frac{\partial(d_e(p_1, p_2)}{\partial t_2} = (t_2 - \alpha_1)\beta_1$$

where $\alpha_1 = 0$ and $\beta_1 > 0$, while

$$\frac{\partial(d_e(p_2, p_3)}{\partial t_2} = (t_2 - \alpha_2)\beta_2$$

where $\alpha_2 = 1$ and $\beta_2 > 0$. Thus, we have

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = t_2\beta_1 + (t_2 - 1)\beta_2$$

Therefore $t_2 = 0$ or 1 is not a root of the equation of the theorem. This is a contradiction.

Case B2. Edge e_1 and the second end point of e_2 be on the same grid plane, while e_3 and the first end point of e_2 are on the same grid plane. By Lemma 25,

$$\frac{\partial(d_e(p_1, p_2)}{\partial t_2} = (t_2 - \alpha_1)\beta_1$$

where $\alpha_1 = 1$ and $\beta_1 > 0$, while

$$\frac{\partial(d_e(p_2, p_3)}{\partial t_2} = (t_2 - \alpha_2)\beta_2$$

where $\alpha_2 = 0$ and $\beta_2 > 0$. Thus, we have

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = (t_2 - 1)\beta_1 + t_2\beta_2$$

Therefore, $t_2 = 0$ or 1 is not a root of the equation of the theorem. This is a contradiction as well. □

5.2.2 Theorem on Inner Angles

14. THEOREM. *If $e_i \perp e_j$, where $i, j = 1, 2, 3$ and $i \neq j$, then e_1, e_2 and e_3 form an inner angle iff the equation*

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = 0$$

has a root t_{2_0} such that $0 < t_{2_0} < 1$.

Proof. If edges e_1, e_2 and e_3 form an inner angle, then by Definition 16, e_1, e_2 and e_3 do not form an end angle. By Theorem 13, 0 or 1 is not a root of the equation given in the theorem. By Lemma 25, we have

$$\frac{\partial(d_e(p_1, p_2) + d_e(p_2, p_3))}{\partial t_2} = (t_2 - \alpha_1)\beta_1 + (t_2 - \alpha_2)\beta_2$$

where α_1, α_2 are 0 or 1, $\beta_1 > 0$ is a function of t_1 and t_2, and $\beta_2 > 0$ is a function of t_2 and t_3. Thus, $\alpha_1 \neq \alpha_2$ (i.e., $\alpha_1 = 0$ and $\alpha_2 = 1$ or $\alpha_1 = 1$ and $\alpha_2 = 0$). Therefore, the equation of the theorem has a root t_{2_0} such that $0 < t_{2_0} < 1$.

Conversely, if the equation of the theorem has a root t_{2_0} such that $0 < t_{2_0} < 1$, then by Theorem 13, critical edges e_1, e_2 and e_3 do not form an end angle. By Definition 16, e_1, e_2 and e_3 do form an inner angle. □

5.2.3 Grid Plane Characterization Theorem

Assume that $e_0 \perp e_1$, $e_2 \perp e_3$, and $e_1 \parallel e_2$. Assume that $p(t_{i_0})$ is a vertex of the MLP of g, where $i = 1$ or $i = 2$. Then we have the following:

27. LEMMA. *If e_0, e_3 and the first end point of e_1 are on the same grid plane, and t_{i_0} is a root of*

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

then $t_{i_0} = 0$, where $i = 1$ or $i = 2$.

Proof. From $p_0(t_0)p_1(0) \perp e_1$ it follows that

$$d_e(p_0(t_0)p_1(0)) = \min\{d_e(p_0(t_0), p_1(t_1)) : t_1 \in [0, 1]\}$$

(see Figure 5.3). Analogously, we have

$$d_e(p_2(0)p_3(t_3)) = \min\{d_e(p_2(t_2), p_3(t_3)) : t_2 \in [0, 1]\}$$

and

$$d_e(p_1(0)p_2(0)) = \min\{d_e(p_1(t_1), p_2(t_2)) : t_1, t_2 \in [0, 1]\}$$

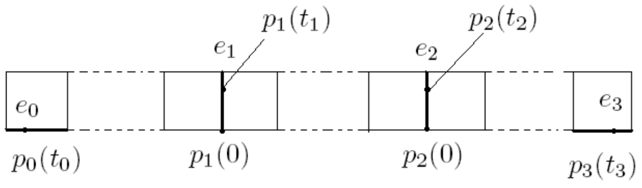


Figure 5.3: Illustration of the proof of Lemma 27.

Therefore we have

$$\min\{d_e(p_0(t_0), p_1(t_1)) + d_e(p_1(t_1), p_2(t_2)) + d_e(p_2(t_2), p_3(t_3)) : t_1, t_2 \in [0, 1]\}$$
$$\geq d_e(p_0(t_0), p_1(0)) + d_e(p_1(0), p_2(0)) + d_e(p_2(0), p_3(t_3))$$

This proves the lemma. □

Assume that we have $e_0 \perp e_1$, $e_m \perp e_{m+1}$, and $e_i \parallel e_{i+1}$, (that is, the set $\{e_1, e_2, \ldots, e_m\}$ is a maximal run of parallel critical edges of g, and e_0 or e_{m+1} are the adjacent critical edges of this set). Furthermore, let $p(t_{i_0})$ be a vertex of the MLP of g, where $i = 1, 2, \ldots, m - 1$. Analogously to the previous lemma, we also have the following two lemmas:

28. LEMMA. *If e_0, e_{m+1} and the first point of e_1 are on the same grid plane, and t_{i_0} is a root of*

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

then $t_{i_0} = 0$, where i = 1, 2, . . ., m.

29. LEMMA. *If e_0, e_{m+1} and the second end point of e_1 are on the same grid plane, and t_{i_0} is a root of*

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

then $t_{i_0} = 1$, where i = 1, 2, . . ., m.

Now we study the case that critical edges are on different grid planes. (Note that even two parallel edges can be on different grid planes.)

30. LEMMA. *If e_0 and e_{m+1} are on different grid planes, and t_{i_0} is a root of*

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

where i = 1, 2, . . ., m, then 0 < t_1 < t_2 < . . . < t_m < 1.

Proof. Assume that e_0 and the first end point of e_1 are on the same grid plane, and e_{m+1} and the second end point of e_1 are on the same grid plane. Then (by Lemmas 25 and 26), the derivatives $\frac{\partial d_i}{\partial t_i}$, where i = 1, 2, ..., m, have the following forms:

$$\begin{aligned}
\frac{\partial d_1}{\partial t_1} &= t_1 b_{1_1} + (t_1 - t_2) b_{1_2} \\
\frac{\partial d_2}{\partial t_2} &= (t_2 - t_1) b_{2_1} + (t_2 - t_3) b_{2_2} \\
\frac{\partial d_3}{\partial t_3} &= (t_3 - t_2) b_{3_1} + (t_3 - t_4) b_{3_2} \\
&\cdots \\
\frac{\partial d_{m-1}}{\partial t_{m-1}} &= (t_{m-1} - t_{m-2}) b_{m-1_1} + (t_{m-1} - t_m) b_{m-1_2}, \quad \text{or} \\
\frac{\partial d_m}{\partial t_m} &= (t_m - t_{m-1}) b_{m_1} + (t_m - 1) b_{m_2} \qquad (5.1)
\end{aligned}$$

where $b_{i_1} > 0$, b_{i_1} is a function of t_i and t_{i-1}, $b_{i_2} > 0$, and b_{i_2} is a function of t_i and t_{i+1}, for $i = 1, 2, \ldots, m$.

If $t_{1_0} < 0$, then (by $\frac{\partial d_1}{\partial t_1} = 0$) we have that $t_{1_0} b_{1_1} + (t_{1_0} - t_{2_0}) b_{1_2} = 0$. Since $b_{1_1} > 0$ and $b_{1_2} > 0$, we also have $t_{1_0} - t_{2_0} > 0$ (i.e., $t_{1_0} > t_{2_0}$).

Analogously, because of $\frac{\partial d_2}{\partial t_2} = 0$ we have $(t_{2_0} - t_{1_0}) b_{2_1} + (t_{2_0} - t_{3_0}) b_{2_2} = 0$. This means that we also have $t_{2_0} > t_{3_0}$.

Analogously we can also verify that $t_{3_0} > t_{4_0}$, ..., and $t_{m-1_0} > t_{m_0}$. Therefore, by Equation (5.1) we have $t_{m_0} - 1 > 0$. Altogether we have $0 > t_{1_0} > t_{2_0} > t_{3_0} > \ldots > t_{m_0} > 1$. This is an obvious contradiction.

If $t_{1_0} = 0$, then (by $\frac{\partial d_1}{\partial t_1} = 0$) we have that $t_{2_0} = 0$. Analogously, $\frac{\partial d_2}{\partial t_2} = 0$ implies $t_{3_0} = 0$, and we also have $t_{4_0} = 0$, ..., $t_{m_0} = 0$ due to the same argument. But, by Equation (5.1) we have

$$\frac{\partial d_m}{\partial t_m} = (t_m - 1) b_{m_2} = -b_{m_2} < 0$$

This contradicts $\frac{\partial d_m}{\partial t_m} = 0$.

If $t_{1_0} \geq 1$, then (by $\frac{\partial d_1}{\partial t_1} = 0$) we have $t_{1_0} b_{1_1} + (t_{1_0} - t_{2_0}) b_{1_2} = 0$. Due to $b_{1_1} > 0$ and $b_{1_2} > 0$ we have $t_{1_0} - t_{2_0} < 0$ (i.e., $t_{1_0} < t_{2_0}$). Analogously, by $\frac{\partial d_2}{\partial t_2} = 0$ it follows that $(t_{2_0} - t_{1_0}) b_{2_1} + (t_{2_0} - t_{3_0}) b_{2_2} = 0$. Then we have $t_{2_0} < t_{3_0}$, and we also have $t_{3_0} < t_{4_0}$, ..., $t_{m-1_0} < t_{m_0}$. Therefore, by Equation (5.1) we have $t_{m_0} - 1 < 0$. Altogether we have $1 \leq t_{1_0} < t_{2_0} < t_{3_0} < \ldots < t_{m_0} < 1$, which is again an obvious contradiction. □

Let t_{i_0} be a root of $\frac{\partial d_i}{\partial t_i} = 0$, where i = 1, 2, ..., m. We apply Lemmas 28, 29 and 30 and obtain

15. THEOREM. *Edges e_0 and e_{m+1} are on different grid planes iff $0 < t_{1_0} < t_{2_0} < \ldots < t_{m_0} < 1$.*

5.2.4 Basics for a Necessary Correction of Option 3

The following two Lemmas will be used in Section 5.6 when revising Option 3 of the original rubberband algorithm. Let p_1, p_2 be points on a critical edge e_i of curve g, and p a point on a critical edge e_j of g.

31. LEMMA. *If the line segments pp_1, pp_2 are contained and complete in tube* **g**, *then the triangular region $\triangle(p_1, p_2, p)$ is also contained and complete in* **g**.

Proof. Without loss generality, we can assume that $i < j$. Let $a(e_i, e_j)$ be the arc from the first cube which contains the critical edge e_i to the last cube which contains the critical edge e_j. (Note that a set of consecutive critical edges will uniquely define a cube-curve.) If line segments pp_1, pp_2 are contained and complete in **g**, then the xy- (yz- and xz-) projection of $\triangle(p_1, p_2, p)$ is contained and complete in the xy- (yz- and xz-) projection of $a(e_i, e_j)$. Therefore, the triangular region $\triangle(p_1, p_2, p)$ is contained and complete in the tube of $a(e_i, e_j)$. □

32. LEMMA. *Let $d_2(t_1, t_2, t_3) = d_e(p_1, p_2) + d_e(p_2, p_3)$. It follows that $d_2(t_1, t_2, t_3)$ is increasing with respect to t_2.*

Proof. Let the coordinates of p_i be $(x_i + k_{x_i}t_i, y_i + k_{y_i}t_i, z_i + k_{z_i}t_i)$, where i equals 1 or 3. Since $p_i \in e_i \subset l_i$, and e_i is a critical edge which is an edge of an orthogonal grid, only one of the values k_{x_i}, k_{y_i} and k_{z_i} can be 1 and the other two must be zero. We consider one of these cases where the coordinates of p_1 are $(x_1 + t_1, y_1, z_1)$, the coordinates of p_2 are $(x_2, y_2 + t_2, z_2)$, and the coordinates of p_3 are $(x_3, y_3, z_3 + t_3)$. Then

$$\begin{aligned} d_2 &= d_e(p_1, p_2) + d_e(p_2, p_3) \\ &= \sqrt{(t_2 - (y_1 - y_2))^2 + (x_1 + t_1 - x_2)^2 + (z_1 - z_2)^2} \\ &\quad + \sqrt{(t_2 - (y_3 - y_2))^2 + (x_3 - x_2)^2 + (z_3 + t_3 - z_2)^2} \end{aligned}$$

This can be rewritten as $d_2 = \sqrt{(t_2 - a_1)^2 + b_1^2} + \sqrt{(t_2 - a_2)^2 + b_2^2}$, where b_1 and b_2 are functions of t_1 and t_3. Then we have

$$\frac{\partial d_2}{\partial t_2} = \frac{t_2 - a_1}{\sqrt{(t_2 - a_1)^2 + b_1^2}} + \frac{t_2 - a_2}{\sqrt{(t_2 - a_2)^2 + b_2^2}} \tag{5.2}$$

and

$$\frac{\partial^2 d_2}{\partial {t_2}^2} = \frac{1}{\sqrt{(t_2-a_1)^2+b_1^2}} - \frac{(t_2-a_1)^2}{[(t_2-a_1)^2+b_1^2]^{3/2}} + \frac{1}{\sqrt{(t_2-a_2)^2+b_2^2}} - \frac{(t_2-a_2)^2}{[(t_2-a_2)^2+b_2^2]^{3/2}}$$

This simplifies to

$$\frac{\partial^2 d_2}{\partial {t_2}^2} = \frac{b_1^2}{[(t_2-a_1)^2+b_1^2]^{3/2}} + \frac{b_2^2}{[(t_2-a_2)^2+b_2^2]^{3/2}} > 0 \tag{5.3}$$

This implies that $d_2(t_1,t_2,t_3)$ is increasing with respect to t_2. All other cases follow analogously. □

5.3 Theorem about Linear Time Complexity

Let $Q_i(x_i,y_i,0)$ be the projection of $P_i(x_i,y_i,z_i)$ onto the $xy-plane$, where i = 1, 2 (see Figure 5.4).

33. LEMMA. *If Q_2 is on the left of OQ_1 then P_2 is on the left of OP_1.*

Proof. This follows because (see Figure 5.4) $\triangle OP_1P_2$ can be obtained by continuously moving Q_i towards P_i, where i = 1, 2. □

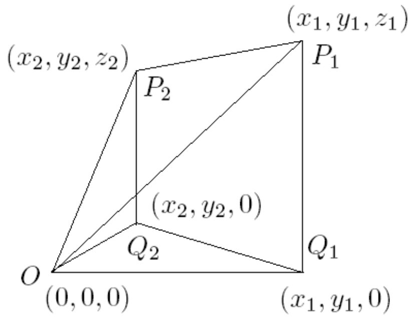


Figure 5.4: Illustration for Lemma 33.

34. LEMMA. *Option 2 of the original rubberband algorithm (see Section 1.3) can be computed in $\mathcal{O}(m)$ time, where m is the number of critical edges intersected by the polygonal path between p_{i-1} and p_{i+1}.*

Proof. We start with vertices of the initial polygon at center points of each critical edge of the given cube-curve.

It follows that the vertices of a resulting polygon, using only Option 1 of the rubberband algorithm, are still at the center points of critical edges.

Option 2 of the algorithm can now be implemented as follows: Let $\mathcal{A}$ be the cube-arc starting at the first cube which contains critical edge e_{i-1}, to the last cube which contains critical edge e_{i+1}. Proceed as follows:

1. Compute all the intersection points, denoted by S_I, of the closed triangular region $\triangle(p_{i-1}, p_i, p_{i+1})$ with consecutive critical edges from e_{i-1} to e_{i+1} (note: they are between both endpoints of a critical edge). This can be computed in $\mathcal{O}(m_1)$ time, where $m_1 = |S_I| \leq$ is the number of critical edges in $\mathcal{A}$.

2. Let S_P be the set of three planes: xy-plane, yz-plane, and zx-plane. Select a plane $\alpha \in S_P$, such that α is not perpendicular to $\triangle(p_{i-1}, p_i, p_{i+1})$. This can be computed in $\mathcal{O}(1)$ time,

3. Project S_I onto α. Let the resulting set be S'_I.

4. Apply the Melkman algorithm [103] (which has linear time complexity, see [132]) to find the convex arc, denoted by $\mathcal{A}'$ in α.

5. By Lemma 33 (the projection of the convex hull of S_I onto α is the convex hull of S'_I); compute a convex arc, denoted by $\mathcal{A}''$, in $\triangle(p_{i-1}, p_i, p_{i+1})$ such that $\mathcal{A}'$ is the projection of $\mathcal{A}''$ onto α .

6. If each edge uw of $\mathcal{A}''$ is fully contained in the tube g, then $\mathcal{A}''$ is the required shortest convex arc from p_{i-1} to p_{i+1}. Otherwise, do not replace the arc from p_{i-1} to p_{i+1}.

All together, it follows that the convex arc can be computed in $\mathcal{O}(m)$ time, where m is the number of critical edges intersecting the arc between p_{i-1} and p_{i+1}. □

The following lemma (Lemma 35) and Procedure 1 are used to analyze the time complexity of Option 3 of the original rubberband algorithm (Lemma 36).

Let e_1, e_2, . . ., e_m be all the consecutive critical edges of g. Let

$$p_i(x_i, y_i, z_i) \in \partial e_i$$

$$d_e(i) = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}$$

where $i = 1, 2, ..., m$,

$$d_e(m) = \sqrt{(x_1 - x_m)^2 + (y_1 - y_m)^2 + (z_1 - z_m)^2}$$

and

$$d = \sum_{i=0}^{m} d_e(i)$$

Let

$$g_f = \min\{y : (x, y, z) \in g\}$$
$$g_b = \max\{y : (x, y, z) \in g\}$$
$$g_l = \min\{x : (x, y, z) \in g\}$$
$$g_r = \max\{x : (x, y, z) \in g\}$$
$$g_u = \max\{z : (x, y, z) \in g\}$$
$$g_d = \min\{z : (x, y, z) \in g\}$$

and

$$M = \sqrt{(g_f - g_b)^2 + (g_l - g_r)^2 + (g_u - g_d)^2}$$

Then we have the following

35. LEMMA. $d \leq mM$

Proof. Obviously, g is fully contained in a cuboid such that the length, depth and height of this cuboid are equal to $|g_l - g_r|$, $|g_f - g_b|$, and $|g_u - g_d|$, respectively. Thus, $d_e(i) \leq M$, where $i = 1, 2, ..., m$. Therefore, $d \leq mM$. □

Now we are ready to discuss the following revised version of Option 3 of the original rubberband algorithm:

1. PROCEDURE. *(Revised Option 3)*

```
1. Let $\epsilon = 10^{-10}$ (the accuracy).
2. Compute the length $l_1$ of the polygonal curve
       $\rho = \langle p_1, p_2, \ldots, p_m, p_1 \rangle$
3. Let $q_1 = p_1$ and $i$ = 1.
4. While $i < m$ do
4.1 Let $q_3 = p_{i+1}$.
4.2 Compute a point $q_2 \in \partial e_i$ such that
       $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \partial e_i\}$
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 $q_3 = p_1$.
5.2 Compute $q_2 \in \partial e_m$ such that
       $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \partial e_m\}$
```

5.3 Update ρ by replacing p_m by q_2.
6. Compute the length l_2 of the updated polygonal curve

$$\rho = \langle p_1, p_2, \ldots, p_m, p_1 \rangle$$

7. Let $\delta = l_1$ - l_2.
8. If $\delta > \epsilon$, then let $l_1 = l_2$ and go to Step 3; otherwise stop.

36. LEMMA. *The revised Option 3 (i.e., Procedure 1) of the original rubberband algorithm can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where $\kappa(\varepsilon)$ is as in Equation (1.1), and m is the number of critical edges of the tube* g.

Proof. We consider Procedure 1 where ε is the accuracy. Let L be the true length of the MLP of g, L_0 that of an initial polygon, and L_n that of the polygonal curve after n iterations. We slightly modify the original rubberband algorithm as follows:[1]

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, . . ., p_{m-1}, then the following vertices p_2, p_4, p_6, . . ., p_m.

Thus, $\{L_n\}_{n\to\infty}$ is a strict decreasing sequence with lower bound 0. By Lemma 35, $L_0 - L$ can be written as am + b (i.e., it is a linear function of m), where a, b are constants such that $a \neq 0$. Because the algorithm will not stop if $L_n - L_{n+1} > \varepsilon$ (see Step 8, Procedure 1), it follows that $L_n - L_{n+1}$ will also depend on m. Again, by Lemma 35, $L_n - L_{n+1}$ can be written as cm + d, where c and d are constants such that $c \neq 0$. Then we have

$$\lim_{m\to\infty} \frac{am+b}{cm+d} = \frac{a}{c}$$

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

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

iterations.)

Thus, by Lemma 16, the time complexity of the original rubberband algorithm equals $\lceil (L_0 - L)/(\varepsilon)\rceil \cdot \mathcal{O}(m)$ = $\kappa(\varepsilon) \cdot \mathcal{O}(m)$, where $\kappa(\varepsilon)$ is as in Equation (1.1), and m is the number of critical edges of the tube g. □

16. THEOREM. *The original rubberband algorithm (with its original Option 3, or the revised Option 3 as given above) is κ-linear, i.e., time complexity $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where $\kappa(\varepsilon)$ is as in Equation (1.1), and m is the number of critical edges of the given simple cube curve.*

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

Proof. Lemma 34 implies that all operations in Option 2 of the original rubberband algorithm can be computed in $\mathcal{O}(m)$ time. This, and Lemma 36 proves the theorem. □

5.4 Example of a "Difficult" Simple Cube Curve

We provide an example to show (in generalization of the example) that there are simple cube-curves such that none of the vertices of their MLPs is a grid vertex. See Figure 5.5 and Table 5.1 for an example of such a cube-curve, which lists the coordinates of the critical edges $e_0, e_1, \ldots, e_{19}$ of g. Let $v(t_0), v(t_1), \ldots, v(t_{19})$ be the vertex of the MLP of g such that $v(t_i)$ is on e_i and t_i is in [0, 1], where $i = 0, 1, 2, \ldots, 19$.

See Appendix B for a complete list of all $\frac{\partial d_i}{\partial t_i}$ (for $i = 0, 1, \ldots, 19$) for this cube-curve g. It follows that there is no end angle in g, but we have six inner angles, namely:

$\angle(e_2, e_3, e_4))$, $\angle(e_3, e_4, e_5))$, $\angle(e_6, e_7, e_8))$, $\angle(e_9, e_{10}, e_{11}))$, $\angle(e_{10}, e_{11}, e_{12}))$, and $\angle(e_{13}, e_{14}, e_{15}))$.

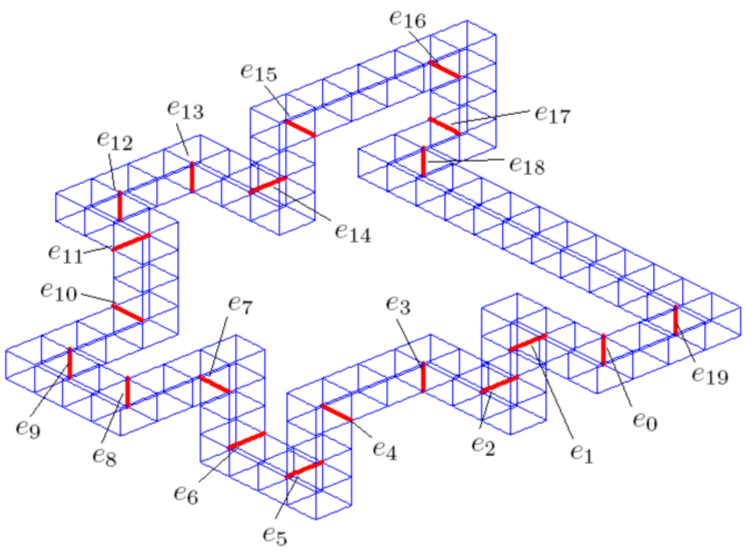


Figure 5.5: A simple cube-curve such that none of the vertices of its MLP is a grid vertex.

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}
e_0	-1	4	7	-1	4	8
e_1	1	4	7	1	5	7
e_2	2	4	5	2	5	5
e_3	4	5	4	4	5	5
e_4	4	7	4	5	7	4
e_5	5	7	2	5	8	2
e_6	7	7	2	7	8	2
e_7	7	8	4	8	8	4
e_8	8	10	4	8	10	5
e_9	10	10	4	10	10	5
e_{10}	10	8	5	11	8	5
e_{11}	11	7	7	11	8	7
e_{12}	12	7	7	12	7	8
e_{13}	12	5	7	12	5	8
e_{14}	10	4	8	10	5	8
e_{15}	9	4	10	10	4	10
e_{16}	9	0	10	10	0	10
e_{17}	9	0	8	10	0	8
e_{18}	9	1	7	9	1	8
e_{19}	-1	2	7	-1	2	8

Table 5.1: Coordinates of endpoints of critical edges of the curve of Figure 5.5.

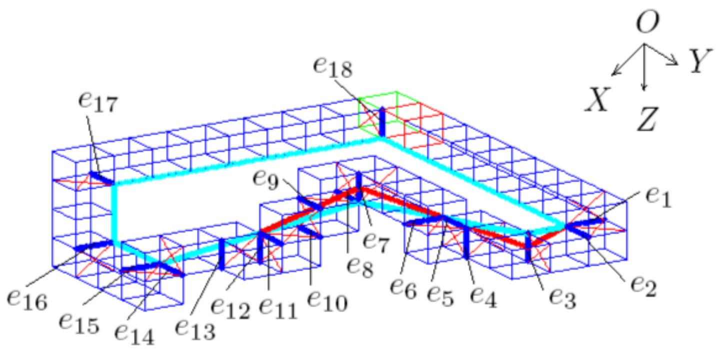


Figure 5.6: Example 1: All critical edges of a simple cube-curve, used for testing Option 2 of the original rubberband algorithm.

By Theorem 14 we have that $t_3, t_4, t_7, t_{10}, t_{11}$ and t_{14} are all in the open interval (0, 1). — Figure 5.5 shows that $e_1 \parallel e_2$, and e_0 and e_3 are on different grid planes. By Theorem 15 it follows that t_1 and t_2 are in (0, 1), too. Analogously we have that t_5 and t_6 are in (0, 1), t_8 and t_9 are in (0, 1), t_{12} and t_{13} are in (0, 1), t_{15}, t_{16} and t_{17} are in (0, 1), and t_{18}, t_{19} and t_0 are in (0, 1). Therefore, each t_i is in the open interval (0, 1), where $i = 0, 1, \ldots, 19$, which proves that g is a simple cube-curve such that none of the vertices of its MLP is a grid vertex.

5.5 Two Counterexamples to Option 2

5.5.1 Counterexample 1 to Option 2

We discuss a first counterexample (a simple cube-curve) to Option 2 of the original rubberband algorithm; see Figure 5.6. Critical edges and centers of the cubes of this simple cube-curve, denoted by g, are shown in Tables 5.2 and 5.3, respectively.

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}
e_1	-0.5	1	0.5	0.5	1	0.5
e_2	-0.5	1	0.5	-0.5	2	0.5
e_3	-1.5	1	0.5	-1.5	1	1.5
e_4	-2.5	0	0.5	-2.5	0	1.5
e_5	-2.5	-1	0.5	-2.5	0	0.5
e_6	-3.5	-1	0.5	-2.5	-1	0.5
e_7	-3.5	-3	-0.5	-3.5	-3	0.5
e_8	-3.5	-4	0.5	-3.5	-3	0.5
e_9	-4.5	-4	0.5	-4.5	-3	0.5
e_{10}	-4.5	-4	1.5	-4.5	-3	1.5
e_{11}	-5.5	-4	1.5	-5.5	-3	1.5
e_{12}	-5.5	-4	1.5	-5.5	-4	2.5
e_{13}	-6.5	-4	1.5	-6.5	-4	2.5
e_{14}	-7.5	-5	2.5	-7.5	-4	2.5
e_{15}	-8.5	-5	2.5	-7.5	-5	2.5
e_{16}	-8.5	-7	2.5	-7.5	-7	2.5
e_{17}	-7.5	-8	0.5	-7.5	-7	0.5
e_{18}	-0.5	-7	-0.5	-0.5	-7	0.5

Table 5.2: Coordinates of endpoints of critical edges of the curve of Figure 5.6.

i	(x_i, y_i, z_i)	i	(x_i, y_i, z_i)	i	(x_i, y_i, z_i)	i	(x_i, y_i, z_i)
1	(0, -7.5, 0)	12	(-1, 1.5, 1)	23	(-5, -3.5, 1)	34	(-8, -7.5, 1)
2	(0, -6.5, 0)	13	(-2, 1.5, 1)	24	(-5, -3.5, 2)	35	(-8, -7.5, 0)
3	(0, -5.5, 0)	14	(-2, 0.5, 1)	25	(-6, -3.5, 2)	36	(-7, -7.5, 0)
4	(0, -4.5, 0)	15	(-2, -0.5, 1)	26	(-6, -4.5, 2)	37	(-6, -7.5, 0)
5	(0, -3.5, 0)	16	(-3, -0.5, 1)	27	(-7, -4.5, 2)	38	(-5, -7.5, 0)
6	(0, -2.5, 0)	17	(-3, -0.5, 0)	28	(-8, -4.5, 2)	39	(-4, -7.5, 0)
7	(0, -1.5, 0)	18	(-3, -1.5, 0)	29	(-8, -4.5, 3)	40	(-3, -7.5, 0)
8	(0, -0.5, 0)	19	(-3, -2.5, 0)	30	(-8, -5.5, 3)	41	(-2, -7.5, 0)
9	(0, 0.5, 0)	20	(-3, -3.5, 0)	31	(-8, -6.5, 3)	42	(-1, -7.5, 0)
10	(0, 1.5, 0)	21	(-4, -3.5, 0)	32	(-8, -7.5, 3)	43	(0, -7.5, 0)
11	(0, 1.5, 1)	22	(-4, -3.5, 1)	33	(-8, -7.5, 2)	44	(0, -6.5, 0)

Table 5.3: Centers of the cubes in the simple cube curve shown in Figure 5.6.

Table 5.4 shows the vertices of the final polygon obtained by original rubberband algorithm.

We discuss why the vertices of the final polygon are not those of the MLP, for curve g as shown in Figure 5.7:

The cyan polyline $q_3q_5q_7q_{11}$ is shorter than the red polyline $p_3p_7p_{11}$. However, according to Option 2, $p_3p_7p_{11}$ can not be improved. In other words, limited by (the

i	$p_i(x_i, y_i, z_i)$
1	(-0.5, 1, 0.5)
3	(-1.5, 1, 1)
7	(-3.5, -3, 0)
11	(-5.5, -4, 1.5)
15	(-7.5, -5, 2.5)
16	(-7.5, -7, 2.5)
17	(-7.5, -7, 0.5)
18	(-0.5, -7, 0.5)

Table 5.4: An example output of the original rubberband algorithm. i is the index of the critical edge e_i. Point p_i is a vertex (on e_i) of a polygon contained in the simple cube curve shown in Figure 5.6.

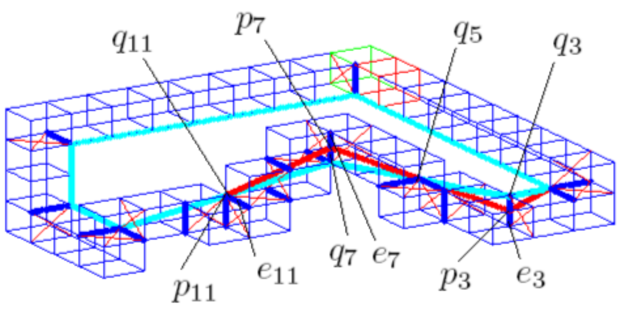


Figure 5.7: Example 1: Three critical edges of the simple cube-curve used for testing Option 2 of the original rubberband algorithm.

original) Option 2, we can not use the shorter polyline $q_3q_5q_7q_{11}$ instead of polyline $p_3p_7p_{11}$. This is because the vertices q_3 and q_7 are not in the set of the intersection points between any critical edge in the set $\{e_i : i = 4, 5, 6, 7, 8, 9, 10\}$ and the closed triangular region $\triangle(p_3, p_7, p_{11})$.

5.5.2 Counterexample 2 to Option 2

We also discuss a second counterexample (see Figure 5.8) to Option 2 of the original rubberband algorithm. The critical edges and centers of the cubes of this simple cube-curve, again denoted by g, are shown in Tables 5.5 and 5.6, respectively. Table 5.7 shows the vertices of the final polygon obtained when applying the original rubberband algorithm.

We discuss why the vertices of the final polygon are not those of the MLP of curve g shown in Figure 5.8:

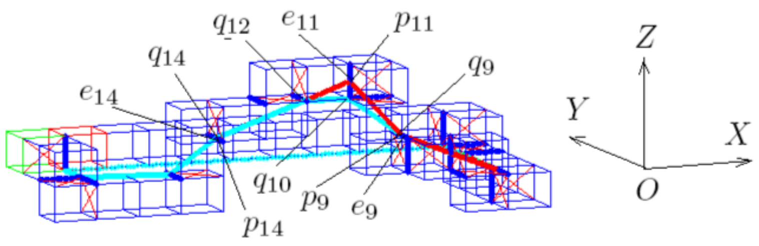


Figure 5.8: Example 2: Three critical edges of a simple cube-curve, also used for testing Option 2 of the original rubberband algorithm.

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}
e_1	0.5	1	-0.5	0.5	1	-0.5
e_2	-0.5	1	-0.5	0.5	1	-0.5
e_3	-0.5	2	-0.5	0.5	2	-0.5
e_4	0.5	4	-1.5	0.5	4	-0.5
e_5	0.5	4	-0.5	0.5	5	-0.5
e_6	1.5	4	-0.5	1.5	5	-0.5
e_7	1.5	4	-0.5	1.5	4	0.5
e_8	2.5	4	-0.5	2.5	4	0.5
e_9	2.5	3	0.5	2.5	4	0.5
e_{10}	2.5	3	1.5	3.5	3	1.5
e_{11}	3.5	3	1.5	3.5	3	2.5
e_{12}	4.5	2	1.5	4.5	3	1.5
e_{13}	5.5	2	1.5	5.5	3	1.5
e_{14}	6.5	2	0.5	6.5	3	0.5
e_{15}	7.5	2	-0.5	7.5	3	-0.5
e_{16}	9.5	2	-0.5	9.5	3	-0.5
e_{17}	9.5	2	-0.5	10.5	2	-0.5
e_{18}	9.5	1	-0.5	9.5	1	0.5

Table 5.5: Coordinates of endpoints of critical edges of the curve of Figure 5.8.

i	(x_i, y_i, z_i)	i	(x_i, y_i, z_i)	i	(x_i, y_i, z_i)	i	(x_i, y_i, z_i)
1	(10, 0.5, 0)	11	(0, 0.5, 0)	21	(3, 3.5, 0)	31	(7, 2.5, -1)
2	(9, 0.5, 0)	12	(0, 1.5, 0)	22	(3, 3.5, 1)	32	(8, 2.5, -1)
3	(8, 0.5, 0)	13	(0, 1.5, -1)	23	(3, 3.5, 2)	33	(9, 2.5, -1)
4	(7, 0.5, 0)	14	(0, 2.5, -1)	24	(3, 2.5, 2)	34	(10, 2.5, -1)
5	(6, 0.5, 0)	15	(0, 3.5, -1)	25	(4, 2.5, 2)	35	(10, 2.5, 0)
6	(5, 0.5, 0)	16	(0, 4.5, -1)	26	(5, 2.5, 2)	36	(10, 1.5, 0)
7	(4, 0.5, 0)	17	(1, 4.5, -1)	27	(5, 2.5, 1)	37	(10, 0.5, 0)
8	(3, 0.5, 0)	18	(1, 4.5, 0)	28	(6, 2.5, 1)	38	(9, 0.5, 0)
9	(2, 0.5, 0)	19	(2, 4.5, 0)	29	(7, 2.5, 1)		
10	(1, 0.5, 0)	20	(2, 3.5, 0)	30	(7, 2.5, 0)		

Table 5.6: Centers of the cubes in the simple cube-curve shown in Figure 5.8.

i	$p_i(x_i, y_i, z_i)$
1	(0.5, 1, -0.5)
5	(0.5, 4.3626, -0.5)
9	(2.5, 3.6374, 0.5)
11	(3.5, 3, 2)
14	(6.5, 2.5044, 0.5)
15	(7.5, 2.2955, -0.5)
16	(9.5, 2, -0.5)
18	(9.5, 1, -0.5)

Table 5.7: An example output of the original rubberband algorithm. i is the index of the critical edge e_i. Point p_i is a vertex (on e_i) of a polygon contained in the simple cube-curve shown in Figure 5.8.

The cyan polyline $q_9q_{10}q_{12}q_{14}$ is shorter than the red polyline $p_9p_{11}p_{14}$. However, according to Option 2, $p_9p_{11}p_{14}$ can not be improved. In other words, by Option 2 we can not identify the actual shorter polyline $q_9q_{10}q_{12}q_{14}$ when considering the polyline $p_9p_{11}p_{14}$. This is because the vertices q_{10} are not in the set of the intersection points between any critical edge in the set $\{e_i : i = 10, 11, 12, 13\}$ and the closed triangular region $\triangle(p_9, p_{11}, p_{14})$.

5.6 Corrected Option 3

The original rubberband algorithm was published in [24] and [85], and the iteration steps of this original algorithm were given in Chapter 1.

Figure 5.9 shows a non-first-class simple cube-curve (see Table 5.8 for the data of this curve). The figure also shows the resulting polygons when using the three options of the original rubberband algorithm.

We start with the polygonal curve L_1. After applying Option 1 we obtain the curve L_2. Then we apply Option 2 and obtain the curve L_3. Finally, we apply Option 3 as given in the original rubberband algorithm, and we obtain curve L_4 as the final result.

For the resulting polygon L_4 note that edge $p(t_{9_0})p(t_{13_0})$ is not contained in the tube **g**. This means that the final polygon is not contained in the tube g! This is because Option 3 of the original algorithm did not check whether $p_{i-1}p_{new}$ and $p_{new}p_{i+1}$ are both contained in the tube **g**. A minor but essential correction is required to fix this problem.

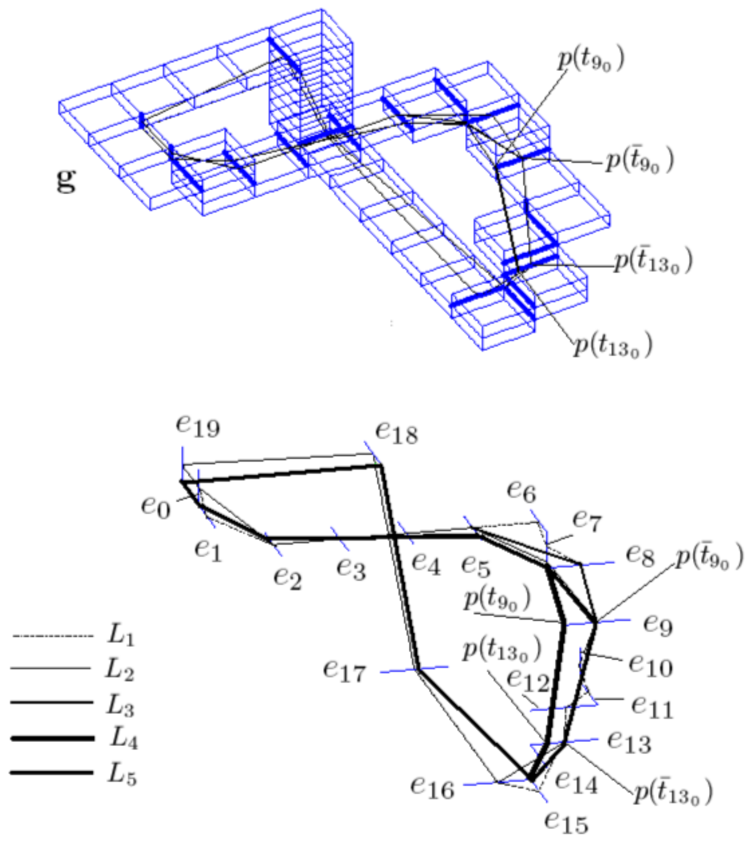


Figure 5.9: An Example of a non-first-class simple cube-curve. The figure shows resulting polygons when applying options of the original or of the corrected rubberband algorithm, respectively. Top: edge $p(t_{9_0})p(t_{13_0})$ is not contained in the tube **g** while $p(\bar{t}_{9_0})p(\bar{t}_{13_0})$ is contained in it. Bottom: the same polygons as on the top, but with all the cubes removed.

The figure also shows the corrected polygon L_5. Note that edge $p(\bar{t}_{9_0})p(\bar{t}_{13_0})$ is now contained in the tube **g**.

Figure 5.10 also shows that there are cases where non of the two endpoints of an edge of the polygonal curve (resulting from Option 2) is allowed to do any move along a critical edge [This allows a further modification of the original Option 3, denoted by (O_3).]:

We consider cubes c_1 and c_2, and two different critical edges e_1 and e_2. Line p_1p_2

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}	t_{i_0}	$\bar{t}_{i_0}$
e_0	0.5	1	-0.5	0.5	1	0.5	1	1
e_1	-0.5	1	0.5	0.5	1	0.5	-	-
e_2	-0.5	2	1.5	0.5	2	1.5	0.7574	0.7561
e_3	-0.5	3	1.5	0.5	3	1.5	0.5858	0.5837
e_4	-0.5	4	1.5	0.5	4	1.5	0.4142	0.4113
e_5	-0.5	5	1.5	0.5	5	1.5	0.2426	0.2388
e_6	-0.5	6	1.5	0.5	6	1.5	-	-
e_7	-0.5	6	1.5	-0.5	6	2.5	1	0.9581
e_8	-0.5	6	2.5	-0.5	7	2.5	-	-
e_9	-1.5	6	3.5	-1.5	7	3.5	0	0.5
e_{10}	-2.5	6	3.5	-2.5	6	4.5	-	-
e_{11}	-3.5	6	4.5	-2.5	6	4.5	-	-
e_{12}	-3.5	5	4.5	-3.5	6	4.5	-	-
e_{13}	-3.5	5	5.5	-3.5	6	5.5	0.2612	0.5
e_{14}	-4.5	5	5.5	-3.5	5	5.5	-	-
e_{15}	-4.5	5	6.5	-3.5	5	6.5	-	-
e_{16}	-3.5	4	6.5	-3.5	5	6.5	1	1
e_{17}	1.5	4	6.5	1.5	5	6.5	0.5455	0.5455
e_{18}	1.5	4	0.5	2.5	4	0.5	0	0
e_{19}	1.5	1	-0.5	1.5	1	0.5	1	1

Table 5.8: Coordinates of endpoints of critical edges in Figure 5.9 and final t values obtained from the original or the revised rubberband algorithm. $p(t_{9_0})p(t_{13_0})$ is not contained in tube **g** (see also Figure 5.9). $p(\bar{t}_{9_0})p(\bar{t}_{13_0})$ is contained in the curve.

is contained and complete in the arc from the cube which contains e_1 to the cube which contains e_2. p_1p_2 intersects with c_1 and c_2 only at a single point each. If p_1 moves to the left along e_1, then p_1p_2 will not intersect with c_2 anymore. If p_1 moves to the right along e_1, then p_1p_2 will not intersect with c_1 anymore. If p_2 moves up along e_2, then p_1p_2 will not intersect with c_2 anymore. If p_2 moves down along e_2, then p_1p_2 will not intersect with c_1 anymore.

The following (revised) Option 3 ensures that the final polygon is always contained and complete in the tube **g**. We use Figure 5.11 for an illustration of the revised Option 3:

Let $p_i = p_i(t_i)$ and $p_{new} = p_i(t_{i_0})$. By (O_3), $t_i, t_{i_0} \in [0, 1]$. Let ε be a sufficiently small positive real number.

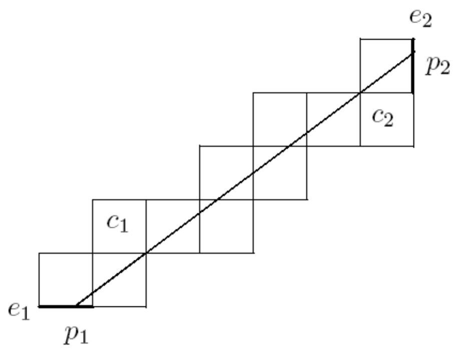


Figure 5.10: An example where any move of one of the two end points of a line segment along critical edges is impossible.

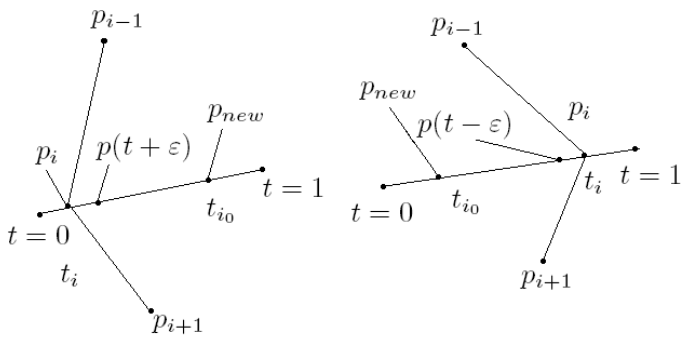


Figure 5.11: Illustration for corrected Option 3. Left: Case 1. Right: Case 2.

(Case 1) $t_i < t_{i0}$ (see Figure 5.11 on the left):

(Case 1.1) both $p_{i-1}p(t_i+\varepsilon)$ and $p_{i+1}p(t_i+\varepsilon)$ are inside the arc from p_{i-1} to p_{i+1}: If both $p_{i-1}p_{new}$ and $p_{i+1}p_{new}$ are inside the arc from e_{i-1} to e_{i+1}, then $\bar{p}_{new} = p_{new}$. Otherwise, by Lemmas 31 and 32, use binary search to find a value $\bar{t}_{i_0} \in (t_i, t_{i_0})$, and then let $\bar{p}_{new} = p(\bar{t}_{i_0})$.

(Case 1.2) either $p_{i-1}p(t_i+\varepsilon)$ or $p_{i+1}p(t_i+\varepsilon)$ are outside the arc from e_{i-1} to e_{i+1}: Then let $\bar{p}_{new} = p_i(t_i) = p_i$.

(Case 2) $t_{i0} < t_i$ (see Figure 5.11 on the right):

(Case 2.1) both $p_{i-1}p(t_i-\varepsilon)$ and $p_{i+1}p(t_i-\varepsilon)$ are inside the arc from p_{i-1} to p_{i+1}: If both $p_{i-1}p_{new}$ and $p_{i+1}p_{new}$ are inside the arc from e_{i-1} to e_{i+1}, then $\bar{p}_{new} = p_{new}$.

Otherwise, (again by Lemmas 31 and 32) use binary search to find a value $\bar{t}_{i_0} \in (t_{i_0}, t_i)$, and then let $\bar{p}_{new} = p(\bar{t}_{i_0})$.

(Case 2.2) either $p_{i-1}p(t_i - \varepsilon)$ or $p_{i+1}p(t_i - \varepsilon)$ are outside the arc from e_{i-1} to e_{i+1}: Then let $\bar{p}_{new} = p_i(t_i) = p_i$.

This revised Option 3 now contains the test of inclusion (which was missing in the original algorithm), and it details the steps for minimizing the length of the calculated polygonal curve, providing a more specific description of Option 3 compared to the original presentation of the rubberband algorithm.

5.7 Conclusions

We constructed a non-trivial simple cube-curve such that none of the vertices of its MLP is a grid vertex. Indeed, Theorems 13 and 15, and Lemmas 29 and 30 allow the conclusion that given a simple first-class cube-curve g, none of the vertices of its MLP is at a grid point position iff g has not any end angle, and for every maximal run of parallel edges of g, its two adjacent critical edges are not on the same grid plane.

It follows that the (provably correct) MLP algorithm proposed in Chapter 2 (see also [93]) cannot be applied to this curve, because this algorithm requires at least one end angle for decomposing a given cube-curve into arcs. Of course, the rubberband algorithm is applicable, and will produce a result (i.e., a polygonal curve).

We also proved that the (original or corrected) rubberband algorithm has κ-linear time complexity $\kappa(\varepsilon) \times \mathcal{O}(m)$, where m is the number of critical edges of a given simple cube-curve, and ε the desired accuracy. We presented two counterexamples to Option 2 of the original rubberband algorithm.

We already corrected the (minor) bug in Option 3 of the original rubberband algorithm. In Chapter 6, we will present a provably correct version of a rubberband algorithm, which is replacing the original rubberband algorithm (for the general case of simple cube-curves). (Possibly. the original Option 2 can also be fixed somehow, however, we have to leave that as an open problem.)

Chapter 6

MLPs of Simple Cube-Curves

This chapter finally answers a main open problem. By a further modification of the corrected (i.e., in its Option 3) rubberband algorithm we obtain a provably correct and provably linear-time algorithm for calculating the MLP of any simple cube-curve.

The chapter also presents an alternative (provably correct and provably linear-time) algorithm for the same task, which is based on moving vertices within faces of cubes rather than moving vertices along critical edges.

6.1 Introduction

Since 1987 it is known that the Euclidean shortest path problem is NP-hard. However, if the 3D world is subdivided into cubes, all of the same size, defining obstacles or possible spaces to move in, then the Euclidean shortest path problem has a κ-linear-time solution!

At the time when starting this PhD project, only one general and linear (only with respect to measured run times) algorithm, called the (original) rubberband algorithm, was known for an approximative calculation of an MLP; a proof, that this algorithm always converges to an MLP, and if so, then always (provably) in linear time, was still an open problem. In this report, we already successfully treated a special case of simple cube-curves. In the previous chapter, we also showed that the original rubberband algorithm required a correction. Instead of modifying Option 2 (which was addressed by the given counterexamples), we corrected the algorithm by modifying Option 3 (only). [To achieve a correction, Option 2 could be modified as well, but it would be more complicated to do so.]

This chapter finally answers the above cited open problem completely: by a further modification of this corrected rubberband algorithm, it turns into a provably correct and provably linear-time algorithm for computing the MLP of any simple cube-curve.

This *edge-based rubberband algorithm* uses Option 2 of the original rubberband algorithm, which is of benefit with respect to practical computation speed (and cases like the provided counterexamples are prevented by the modified Option 3). In contrast, this chapter also provides one *face-based rubberband algorithm* (which moves vertices of constructed curves within faces of cubes, rather than along critical edges) which does not use Option 2 at all, but which is conceptually somehow more complicated and practically slower than the edge-based algorithm. However, with respect to asymptotic, both the edge-based and the face-based rubberband algorithm own a time complexity in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$, where $\kappa(\varepsilon)$ is as in Equation (1.1), and m is the number of critical edges of the given simple cube-curve.

The chapter is organized as follows: Section 6.2 describes the concepts used in this chapter. Section 6.3 provides mathematical fundamentals for our two algorithms. Section 6.4 describes the edge-based and face-based rubberband algorithm, and discusses their time complexity. Section 6.5 presents an example illustrating how the edge-based and face-based rubberband algorithms are converging to identical results (i.e., to the MLP). Section 6.6 gives our conclusions.

6.2 Definitions

We start with defining a simple but very important notion:

24. DEFINITION. *If f is a face of a cube in g and one of f's edges is a critical edge e in g then f is called a* critical face of e in g, *or (for short) a* critical face.

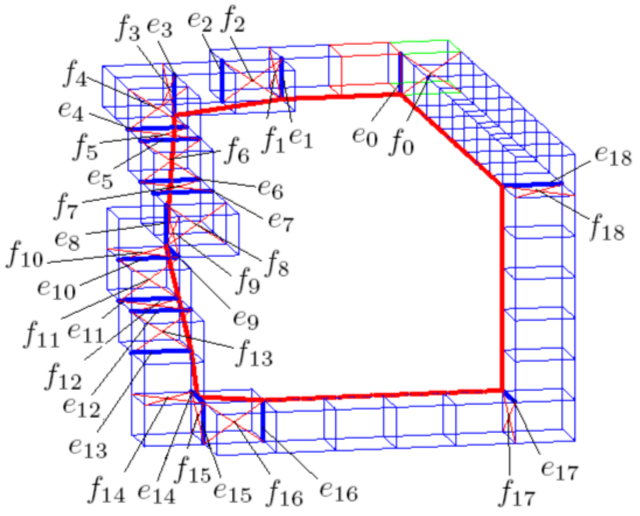


Figure 6.1: A simple cube-curve and its MLP (see also Table 6.1).

Critical edge	x_{i1}	y_{i1}	z_{i1}	x_{i2}	y_{i2}	z_{i2}
e_0	-0.5	1	-0.5	-0.5	1	0.5
e_1	-0.5	2	-0.5	-0.5	2	0.5
e_2	-1.5	3	-0.5	-1.5	3	0.5
e_3	-2.5	3	-0.5	-2.5	4	-0.5
e_4	-3.5	3	-0.5	-3.5	4	-0.5
e_5	-3.5	3	-1.5	-3.5	4	-1.5
e_6	-4.5	3	-1.5	-4.5	4	-1.5
e_7	-5.5	4	-2.5	-5.5	4	-1.5
e_8	-6.5	4	-2.5	-5.5	4	-2.5
e_9	-6.5	4	-2.5	-6.5	5	-2.5
e_{10}	-6.5	4	-3.5	-6.5	5	-3.5
e_{11}	-7.5	4	-3.5	-7.5	5	-3.5
e_{12}	-7.5	4	-4.5	-7.5	5	-4.5
e_{13}	-8.5	4	-5.5	-7.5	4	-5.5
e_{14}	-8.5	4	-6.5	-8.5	4	-5.5
e_{15}	-8.5	3	-6.5	-8.5	3	-5.5
e_{16}	-9.5	-1	-5.5	-8.5	-1	-5.5
e_{17}	-8.5	-2	-0.5	-8.5	-1	-0.5
e_{18}	-0.5	-1	-0.5	-0.5	-1	0.5

Table 6.1: Coordinates of endpoints of critical edges shown in Figure 6.1: these data are used later in an experiment.

Figure 6.1 shows 19 critical faces f_i, where $i = 0, 1, \ldots, 18$. This cube-curve is not first-class because there are no vertices of the MLP on the following critical edges: e_1, e_4, e_5, e_6, e_8, e_9, e_{10}, e_{11} and e_{14}. (If not yet clear at this point then we refer to experiments which are later reported in this report for the shown curve.)

As discussed in [106] and [107], the difficulty of ESP problems can also be verified by combinatorial considerations. For the MLP problem, a combinatorial difficulty can be identified by the complexity of computing the set of critical edges such that each critical edge in this set contains (exactly) one vertex of the MLP.

The basic computational task of Algorithm 1 in Chapter 1 (addressed by Step 1 to Step 4) consists in selecting this set, but the provided algorithm is not time-efficient for large inputs. Option 2 of the original rubberband algorithm was also designed having this goal in mind, but (we discussed that) it is flawed. In this chapter we solve this difficult problem, and started with introducing the notion of a critical face.

Unfortunately, we need to have to undertake a very close observation of the geometric structure of a simple cube-curve, and introduce for this purpose a few rather technical definitions:

25. DEFINITION. *Let e be a critical edge of a simple cube-curve g and f_1, f_2 be two critical faces of e in g. Let c_1, c_2 be the centers of f_1, f_2 respectively. Then a polygonal curve can go in the direction from c_1 to c_2, or from c_2 to c_1, to visit all cubes in g such that each cube is visited exactly once. If e is on the left of line segment c_1c_2, then the orientation from c_1 to c_2 is called* counter-clockwise orientation *of g. f_1 is called* the first *critical face of e in g. If e is on the right of line segment c_1c_2, then the direction from c_1 to c_2 is called* clockwise orientation *of g.*

Figure 6.1 shows all critical edges (e_0, e_1, e_2, . . ., e_{18}) and their first critical faces (f_0, f_1, f_2, . . ., f_{18}) of a simple cube-curve, denoted by g_{19}.

26. DEFINITION. *A* minimum-length pseudo polygon *of a simple cube-curve g, denoted by $MLPP$, is a shortest curve P which is contained and complete in tube* **g** *such that each vertex of P is on the first critical face of a critical edge in g.*

We will show that the $MLPP$ of a simple cube-curve g is unique (see Lemma 40 in Section 6.3). The number of vertices of an $MLPP$ is the number of all critical edges of g. Curve $p_{4_0}p_{4_1} \cdots p_{4_{18}}$ (see Table 6.3) is the $MLPP$ of g as shown in Figure 6.1.

Let f_{i1} and f_{i2} be two critical faces of e_i in g, for $i = 1, 2$. Let c_{i1} and c_{i2} be the centers of f_{i1} and f_{i2}, respectively, for $i = 1, 2$. Obviously, the counter-clockwise orientation of g defined by c_{11} and c_{12} is identical to the one defined by c_{21} and c_{22}.

27. DEFINITION. *Let e_0, e_1, e_2, . . . e_m and e_{m+1} be all consecutive critical edges of g in counter-clockwise orientation of g. Let f_i be the first critical face of e_i in g, and p_i be a point on f_i, for $i = 0, 1, 2, \ldots, m, m + 1$. Then the polygonal curve $p_0p_1 \cdots p_mp_{m+1}$ is called an* approximate minimum-length pseudo polygon *of g, denoted by $AMLPP$.*

The polygonal curve $p_{1_0}p_{1_1} \cdots p_{1_{18}}$ (see Table 6.2) is an $AMLPP$ of g_{18} shown in Figure 6.1.

28. DEFINITION. *Let p_1, p_2 and p_3 be three consecutive vertices of an $AMLPP$ of a simple cube-curve g. If p_1, p_2 and p_3 are colinear, then p_2 is called a* trivial *vertex of the $AMLPP$ of g. Point p_2 is called a* non-trivial *vertex of the $AMLPP$ of g if it is not a trivial vertex of that $AMLPP$ of g.*

We recall that a simple cube-arc is an alternating sequence $\rho = (f_0, c_0, f_1, c_1, \ldots, f_k, c_k, f_{k+1})$ of faces f_i and cubes c_i with $f_{k+1} \neq f_0$, denoted by $\rho = (f_0, f_1, \ldots, f_{k+1})$

or $\rho(f_0, f_{k+1})$ for short; it is a connected part of a simple cube-curve. A *subarc* of an arc $\rho(f_0, f_{k+1})$ is an arc $\rho(f_i, f_j)$, where $0 \le i \le j \le k$.

29. DEFINITION. *Let a polygonal curve P = $p_0p_1 \cdots p_mp_{m+1}$ be an $AMLPP$ of g and p_i $\in f_i$, where f_i is a critical face of g, for $i = 0, 1, 2, \ldots, m+1$. A cube-arc $\rho(f_i, f_j)$ is called*

- *a* (2,3)-cube-arc *with respect to P if each vertex p_k is identical to p_{k-1} or p_{k+1}, where k = i +1, . . ., j - 1,*[1]
- *a* maximal (2,3)-cube-arc *with respect to P if it is a (2,3)-cube-arc and p_i is not identical to p_{i+1} and p_{i-1}, and p_j is not identical to p_{j-1} and p_{j+1},*
- *a* 3-cube-arc unit *with respect to P if it is a (2,3)-cube-arc such that j = i + 4 (mod m + 2) and p_{i+1}, p_{i+2}, p_{i+3} are identical.*
- *a* 2-cube-arc *with respect to P if it is a (2,3)-cube-arc and no three consecutive vertices of P on a are identical,*
- *a* maximal 2-cube-arc *with respect to P if it is both a maximal (2,3)-cube-arc and a 2-cube-arc as well,*
- *a* 2-cube-arc unit *with respect to P if it is a 2-cube-arc such that j = i + 3 (mod m + 2) and p_{i+1} is identical to p_{i+2},*
- *a* regular cube-arc unit *with respect to P if $a = (f_i, f_{i+1}, f_j)$ such that p_i is not identical to p_{i+1} and p_j is not identical to p_{i+1},*
- *a* cube-arc unit *with respect to P if a is a regular cube-arc unit, 2-cube-arc unit or 3-cube-arc unit, or*
- *a* regular cube-arc *with respect to P if no two consecutive vertices of P on a are identical.*

Let P_{18_i} = $p_{i_0}p_{1_1} \cdots p_{1_{18}}$ (see Table 6.2), where i = 1, 2, 3, 4. Then there are four maximal 2-cube-arcs with respect to P_{18_i}: ($p_{i_{18}}$, p_{i_0}, p_{i_1}, p_{i_2}), (p_{i_2}, p_{i_3}, p_{i_4}, p_{i_5}), (p_{i_7}, p_{i_8}, p_{i_9}, $p_{i_{10}}$) and ($p_{i_{12}}$, $p_{i_{13}}$, $p_{i_{14}}$, $p_{i_{15}}$) in total, where i = 1, 2, 3. They are also maximal 2-cube-arcs and 2-cube-arc units with respect to P_{18_i}, where i = 1, 2, 3. There are no 3-cube-arc units with respect to P_{18_i}, where i = 1, 2, 3. (p_{i_1}, p_{i_2}, p_{i_3}) is a regular cube-arc unit with respect to P_{18_i} and (p_{i_4}, p_{i_5}, p_{i_6}, p_{i_7}, p_{i_8}) is a regular cube-arc with respect to P_{18_i}, where i = 1, 2, 3.

There are three maximal 2-cube-arcs with respect to P_{18_4}: ($p_{4_{18}}$, p_{4_0}, p_{4_1}, p_{4_2}), (p_{4_2}, p_{4_3}, p_{4_4}, p_{4_5}), and ($p_{4_{12}}$, $p_{4_{13}}$, $p_{4_{14}}$, $p_{4_{15}}$) in total. They are also maximal 2-cube-arcs

[1]Note that it is impossible that four consecutive vertices of P on ρ are identical.

and 2-cube-arc units with respect to P_{18_4}. ($p_{4_6}, p_{4_7}, p_{4_8}, p_{4_9}, p_{4_{10}}, p_{4_{11}}, p_{4_{12}}$) is a (2,3)-cube-arc with respect to P_{18_4}. ($p_{4_6}, p_{4_7}, p_{4_8}, p_{4_9}, p_{4_{10}}$) is a unique 3-cube-arc unit with respect to P_{18_4}.

30. DEFINITION. *Let $\rho = (f_i, f_{i+1}, \ldots, f_j)$ be a simple cube-arc and $p_k \in f_k$, where $k = i, j$. A* minimum-length arc *with respect to p_i and p_j of ρ, denoted by $MLA(p_i, p_j)$, is a shortest arc (from p_i to p_j) which is contained and complete in ρ such that each vertex of $MLA(p_i, p_j)$ is on the first critical face of a critical edge in ρ.*

6.3 Basics

We provide mathematical fundamentals to be used in the following, and start with a result in elementary geometry; see Figure 6.2.

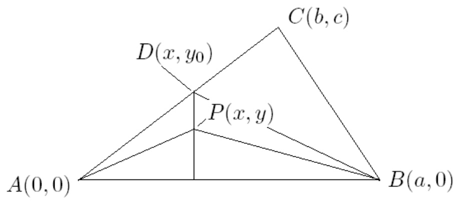


Figure 6.2: Illustration for Lemma 37.

37. LEMMA. *Let P be a point in $\triangle ABC$ such that P is not on any of the three line segments AB, BC and CA. Then $d_e(P, A) + d_e(P, B) < d_e(C, A) + d_e(C, B)$.*

The following Lemma is obvious; we will use it (in Section 6.4) for the description of the edge-based rubberband algorithm. Let $p_i \in f_i$, where f_i is the first critical face of e_i in g, for $i = 0, 1, 2, \ldots, m + 1$. We consider a polygonal curve $P = p_0 p_1 \cdots p_m p_{m+1}$.

38. LEMMA. *Let p_i and p_{i+1} be two consecutive vertices of an $AMLPP$ of g. If p_i is identical to p_{i+1} then p_i and p_{i+1} are on a critical edge of g.*

By Lemma 14 we have the following:

39. LEMMA. *The number of $MLPPs$ of a first-class simple cube-curve g is finite.*

2. COROLLARY. *The number of $MLPPs$ of a simple cube-curve g is finite.*

Proof. If there is a vertex $p_i \in f_i$ such that p_i is not on an edge of f_i (i.e, p_i is a trivial vertex of $MLPP$), then p_{i-1}, p_i and p_{i+1} are colinear. In this case, p_i can be ignored because it is defined by p_{i-1} and p_{i+1}. Therefore, without loss generality, we can assume that each p_i is on one edge of f_i, for $i = 0, 1, 2, \ldots, m+1$. However, for this case the proof of the lemma is exactly the same as that of Lemma 14. □

Analogous to the proof of Lemma 24, we also have the following:

40. LEMMA. *Each simple cube-curve g has a unique $MLPP$.*

17. THEOREM. *P is an $MLPP$ of g iff for each cube-arc unit $\rho(f_i, f_j)$ with respect to P, the arc $(p_i, p_{i+1}, \ldots, p_j)$ is equal to $MLA(p_i, p_j)$.*

Proof. The necessity is straightforward. The sufficiency is by Lemma 40. □

Analogously to the proof of Theorem 3, we also obtain (see Definition 28) the following:

41. LEMMA. *If a vertex p of an $AMLPP$ of g is on a first critical face f but not on any edge of it, then p is a trivial vertex of the $AMLPP$.*

6.4 Algorithms

We present two algorithms which are both linear-time and provably correct (i.e., the calculated curves are converging to the MLP of a simple cube-curve). We start with describing some useful procedures which will be part of those two algorithms (in the sense of subroutines).

6.4.1 Procedures

Given a critical e in g, and two points p_1 and p_3 in g, by Procedure 2, we can find a unique point p_2 in f such that $d_{p_1p_2} + d_{p_3p_2} = \min\{d_{p_1p} + d_{p_3p} : p \in e\}$.

2. PROCEDURE.

Let a and b be the two endpoints of e. Then, by Lemma 6 of [95], $p_2 = a + t*(b-a)$, where $t = -(A_1B_2 + A_2B_1)/(B_2 + B_1)$; A_1, A_2, B_1 and B_2 are functions of the coordinates of p_1, p_3, a and b. [Further details of this procedure are obvious.]

For the next procedure, assume a critical face f of a critical edge in g, and two points p_1 and p_3 in the tube g; by the following Procedure 3 we can find a point p_2 in f such that $d_{p_1p_2} + d_{p_3p_2} = \min\{d_{p_1p} + d_{p_3p} : p \in f\}$.

3. PROCEDURE.

Case 1. p_1p_3 and f are on the same plane.

Case 1.1. Let $p_1p_3 \cap f \neq \phi$.

In this case, $p_1p_3 \cap f$ is a line segment. Let p_2 be that end point of this segment which is closer to p_1; see Figure 6.3.

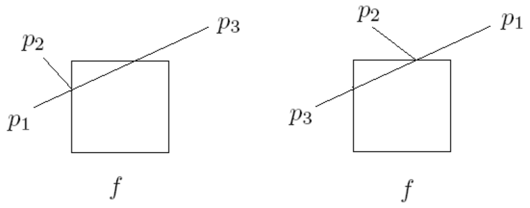


Figure 6.3: Position of point p_2.

Case 1.2. Let $p_1p_3 \cap f = \phi$.

By Lemma 37, p_2 must be on the edges of f. By Lemma 5, p_2 must be uniquely on one of the edges of f. Apply Procedure 2 on the four edges of f, denoted by e_1, e_2, e_3 and e_4; we obtain p_{2_i} such that $d_{p_1p_{2_i}}$ + $d_{p_3p_{2_i}} = \min\{d_{p_1p} + d_{p_3p} : p \in e_i\}$, where i = 1, 2, 3, 4. This allows to calculate a point p_2 such that $d_{p_1p_2}$ + $d_{p_3p_2} = \min\{d_{p_1p_{2_i}} + d_{p_3p_{2_i}} : i = 1, 2, 3, 4.\}$.

Case 2. p_1p_3 and f are not on the same plane.

Case 2.1. Let $p_1p_3 \cap f \neq \phi$.

It follows that $p_1p_3 \cap f$ is a unique point. Let p_2 be this point.

Case 2.2. Let $p_1p_3 \cap f = \phi$.

In this case, p_2 can be found exactly the same way as in Case 1.2.

This concludes Procedure 3. – The following procedure is used to convert an $MLPP$ into an MLP.

4. PROCEDURE.

Given a polygonal curve $p_0p_1 \cdots p_mp_{m+1}$ and three pointers addressing vertices at positions i - 1, i, and i + 1 in this curve. Delete p_i if p_{i-1}, p_i and p_{i+1} are colinear. Next, the subsequence (p_{i-1}, p_i, p_{i+1}) is replaced in the curve by (p_{i-1}, p_{i+1}). Then, continue with vertices $(p_{i-1}, p_{i+1}, p_{i+2})$ until $i + 2$ equals $m + 1$.

Let $p_i \in l_i \subset f_i$,. . ., $p_j \in l_j \subset f_j$ be a sequence of some consecutive vertices of the $AMLPP$ of g, where f_i, . . ., f_j are some consecutive critical faces of g, and l_k is a line segment on f_k, k = i, $i+1$, . . ., j. Let ε = 10^{-10} (this value defines the accuracy of the output of this algorithm; we propose it as an example). We can apply the method of Option 3 of the original rubberband algorithm (see Section 1.3), also including its correction in Section 5.6, for a cube-arc $\rho(f_i, f_j)$ and for finding an approximate $MLA(p_i, p_j)$. This application is as follows:

5. PROCEDURE.

1. Calculate the length of arc $p_i p_{i+1} \cdots p_{j-1} p_j$, denoted by L_1.
2. Let k = i+1.
3. Take two points $p_{k-1} \in f_{k-1}$ and $p_{k+1} \in f_{k+1}$.
4. For line segment l_k on a critical face f_k in g, and points p_{k-1} and p_{k+1} on l_{k-1} and l_{k+1}, respectively, apply Procedure 2 to find a point $q_k \in l_k$ such that

$$d_{p_{k-1}q_k} + d_{p_{k+1}q_k} = \min\{d_{p_{k-1}p} + d_{p_{k+1}p} : p \in l_k\}$$

Let p_k = q_k.
5. Let k = k + 1.
6. If k = j, calculate the length of arc $p_i p_{i+1} \cdots p_{j-1} p_j$, denoted by L_2.
7. If L_1 - L_2 $> \varepsilon$, let L_1 = L_2 and go to Step 2. Otherwise, output the arc $p_i p_{i+1} \cdots p_{j-1} p_j$.

Let e_0, e_1, e_2, . . . e_m and e_{m+1} be all consecutive critical edges of g in the counter-clockwise orientation of g. Let f_i be the first critical face of e_i in g, and c_i be the center of f_i, for $i = 0, 1, 2, \ldots, m+1$. All indices of points, edges and faces are taken $mod(m+2)$. Let ε = 10^{-10}. Using the following Procedure 6, we can compute an $AMLPP$ of g and its length.

6. PROCEDURE.

1. Let P be a polygonal curve $p_0 p_1 \cdots p_m p_{m+1}$.
2. Calculate the length of P, denoted by L_1.
3. Let i = 0.
4. Take two points $p_{i-1} \in f_{i-1}$ and $p_{i+1} \in f_{i+1}$.
5. For the critical face f_i of an critical edge e_i in g, and points p_{i-1} and p_{i+1} in f_{i-1} and f_{i+1}, respectively, apply Procedure 3 to find a point q_i in f_i such that

$$d_{p_{i-1}q_i} + d_{p_{i+1}q_i} = \min\{d_{p_{i-1}p} + d_{p_{i+1}p} : p \in f_i\}$$

Let p_i = q_i.

6. Let i = i + 1.

7. If i = $m + 3$, calculate the length of the polygonal curve $p_0p_1 \cdots p_mp_{m+1}$, denoted by L_2.

8. If L_1 - $L_2 > \varepsilon$, let L_1 = L_2 and go to Step 2. Otherwise, output the polygonal curve $p_0p_1 \cdots p_mp_{m+1}$ as an $AMLPP$ of g and its length L_2.

Given an n-cube-arc unit (f_i, ..., f_j) with respect to a polygonal curve P of g, where n = 2 or 3. Let $p_i \in f_i$ and $p_j \in f_j$. We can calculate an $MLA(p_i, p_j)$ by applying the following procedure.

7. PROCEDURE.

1. Compute the set E = {e: e is an edge of f_k $\wedge$ k = i +1, ..., j - 1}.

2. Let I = 1 and L = 100 (i.e., a sufficiently big positive real number).

3. Compute the set $SE = \{S : S \subseteq E \wedge |S| = I\}$.

4. Go through each $S \in SE$, input p_i, e_1, ..., e_l, p_j to Procedure 5 to compute an approximate $MLA(p_i, p_j)$ such that it has minimal length with respect to all sets $S \in SE$, denoted by $AMLA(I, SE)$, where $e_k \in S$, for k = 1, 2 , ..., l and l = $|S|$. If the length of $AMLA(I, SE)$ is smaller than L, let $MLA(p_i, p_j)$ = $AMLA(I, SE)$ and L equal to the length of $AMLA(I, SE)$.

5. Let I = I +1.

6. If $I < n$ then go to Step 3. Otherwise, stop.

42. LEMMA. *For each cube-arc unit $\rho(f_i, f_j)$ with respect to P, $MLA(p_i, p_j)$ can be computed in $\mathcal{O}(1)$ time.*

Proof. If ρ is a regular cube-arc unit, then $MLA(p_i, p_j)$ can be found by Procedure 3 which has complexity $\mathcal{O}(1)$. Otherwise, ρ is an n-cube-arc unit, where n = 2 or 3. Then, by Lemma 41, $MLA(p_i, p_j)$ can be found by Procedure 7, which can be computed in $\mathcal{O}(1)$ because n = 2 or 3. □

6.4.2 Algorithms

We now extend the corrected rubberband algorithm into the following (provably correct) algorithm.

The Edge-Based Rubberband Algorithm

1. Let P_0 be the polygon obtained by the corrected rubberband algorithm (i.e., original rubberband algorithm whose Option 3 was corrected).

2. Find a point $p_i \in f_i$ such that p_i is the intersection point of an edge of P_0 with f_i, for $i = 0, 1, 2, \ldots, m+1$. Let P be a polygonal curve $p_0 p_1 \cdots p_m p_{m+1}$.

3. Apply Procedure 7 to all cube-arc units of P. If for each cube-arc unit $\rho = (f_i, f_{i+1}, \ldots, f_j)$ with respect to P, the arc $(p_i, p_{i+1}, \ldots, p_j) = MLA(p_i, p_j)$, then P is the $MLPP$ of g (by Theorem 17); go to Step 4. – Otherwise, go to Step 3.

4. Apply Procedure 4 to obtain the final MLP.

The Face-Based Rubberband Algorithm

1. Take a point $p_i \in f_i$, for $i = 0, 1, 2, \ldots, m+1$.

2. Apply Procedure 6 to find an $AMLPP$ of g, denoted by P.

3. Find all maximal 2-cube-arcs with respect to P, apply Procedure 5 to update the vertices of the $AMLPP$, which are on one of the 2-cube-arcs.

(By Lemma 38, the input line segments of Procedure 5 are critical edges.) Repeat this step until the length of the updated $AMLPP$ is sufficiently accurate (i.e., the previous length minus the current length is smaller than ε).

4. Apply Procedure 6 to update the current $AMLPP$.

5. Find all maximal (2,3)-cube-arcs with respect to the current P; apply Procedure 5 to update those vertices of the current $AMLPP$ which are on one of the (2,3)-cube-arcs. The input line segments of Procedure 5 can be found such that they are on the critical face and parallel or perpendicular to the critical edge of the face. Repeat this step until the length of the updated $AMLPP$ is sufficiently accurate.

6. Apply Procedure 6 to update the current $AMLPP$.

7. Apply Procedure 7 to all cube-arc units of P. If the arc $(p_i, p_{i+1}, \ldots, p_j)$ is equal to $MLA(p_i, p_j)$ for each cube-arc unit $\rho = (f_i, f_j)$ with respect to P, then P is the $MLPP$ of g (by Theorem 17); go to Step 8. – Otherwise, go to Step 3.

8. Apply Procedure 4 to obtain the final MLP.

Note that, by Theorem 17, both (Steps 3 and 7 of) the edge-based rubberband algorithm and face-based rubberband algorithm produce the $MLPP$ of g; therefore, by Definition 26, both algorithms are correct.

6.4.3 Computational Complexity

We discuss the time complexity of the edge-based and face-base rubberband algorithm. For a start, it is obvious that Procedures 2 and Procedure 3 can be computed in $\mathcal{O}(1)$, and Procedure 4 can be computed in $\mathcal{O}(m)$, where m is the number of critical edges of g.

p_{1i}	x_{1i}	y_{1i}	z_{1i}	p_{2i}	x_{2i}	y_{2i}	z_{2i}
p_{1_0}	-0.5	1	0	p_{2_0}	-0.5	1	-0.21
p_{1_1}	-0.5	1	0	p_{2_1}	-0.5	1	-0.21
p_{1_2}	-1.5	3	-0.34	p_{2_2}	-1.5	3	-0.34
p_{1_3}	-2.5	3.29	-0.5	p_{2_3}	-2.5	3.23	-0.5
p_{1_4}	-2.5	3.29	-0.5	p_{2_4}	-2.5	3.23	-0.5
p_{1_5}	-3.5	3.5	-1.11	p_{2_5}	-3.5	3.45	-1.11
p_{1_6}	-4.15	3.64	-1.5	p_{2_6}	-4.15	3.64	-1.5
p_{1_7}	-5.5	3.94	-2.32	p_{2_7}	-5.5	3.94	-2.32
p_{1_8}	-5.8	4	-2.5	p_{2_8}	-5.69	4	-2.5
p_{1_9}	-5.8	4	-2.5	p_{2_9}	-5.69	4	-2.5
$p_{1_{10}}$	-6.5	4	-3.32	$p_{2_{10}}$	-6.5	4	-3.32
$p_{1_{11}}$	-6.65	4	-3.5	$p_{2_{11}}$	-6.65	4	-3.5
$p_{1_{12}}$	-7.5	4	-4.5	$p_{2_{12}}$	-7.5	4	-4.5
$p_{1_{13}}$	-7.95	4	-5.5	$p_{2_{13}}$	-8	4	-5.5
$p_{1_{14}}$	-7.95	4	-5.5	$p_{2_{14}}$	-8	4	-5.5
$p_{1_{15}}$	-8.5	3	-5.5	$p_{2_{15}}$	-8.5	3	-5.5
$p_{1_{16}}$	-8.5	-1	-5.5	$p_{2_{16}}$	-8.5	-1	-5.5
$p_{1_{17}}$	-8.5	-1	-0.5	$p_{2_{17}}$	-8.5	-1	-0.5
$p_{1_{18}}$	-0.5	-1	-0.1	$p_{2_{18}}$	-0.5	-1	-0.1

Table 6.2: Comparison of results of steps of the face-based rubberband algorithm. Points p_{1_0}, p_{1_1}, …, $p_{1_{18}}$ are the results of Step 2; points p_{2_0}, p_{2_1}, …, $p_{2_{18}}$ are the results of Step 3.

By Lemma 36, Procedure 5 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ time, where $\kappa(\varepsilon)$ is here (and throughout this subsection) as in Equation (1.1), and n is the number of points of the arc. Analogously, Procedure 6 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where m is the number of points of the polygonal curve.

By Theorem 16, the original rubberband algorithm can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where m is again the number of critical edges of g. The main additional operation of the edge-based rubberband algorithm is Step 3 which can be computed in $\mathcal{O}(m)$, where m is the number of critical edges of g (by Lemma 42). It follows that the edge-based rubberband algorithm can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time.

For the face-based rubberband algorithm, Step 1 requires constant time. Steps 2, 4, and 6 have the same time complexity as Procedure 6. Again, by Lemma 36, Step 3 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where m is the number of points of the polygonal curve. Analogously, Step 5 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time. (Note that there is a constant number of different combinations of input line segments of

p_{3i}	x_{3i}	y_{3i}	z_{3i}	p_{4i}	x_{4i}	y_{4i}	z_{4i}
p_{3_0}	-0.5	1	-0.21	p_{4_0}	-0.5	1	-0.5
p_{3_1}	-0.5	1	-0.21	p_{4_1}	-0.5	1	-0.5
p_{3_2}	-1.5	3	-0.41	p_{4_2}	-1.5	3	-0.5
p_{3_3}	-2.5	3.23	-0.5	p_{4_3}	-2.5	3.22	-0.5
p_{3_4}	-2.5	3.23	-0.5	p_{4_4}	-2.5	3.22	-0.5
p_{3_5}	-3.5	3.47	-1.13	p_{4_5}	-3.5	3.48	-1.17
p_{3_6}	-4.09	3.62	-1.5	p_{4_6}	-4	3.61	-1.5
p_{3_7}	-5.5	3.95	-2.38	p_{4_7}	-5.5	4	-2.5
p_{3_8}	-5.69	4	-2.5	p_{4_8}	-5.5	4	-2.5
p_{3_9}	-5.69	4	-2.5	p_{4_9}	-5.5	4	-2.5
$p_{3_{10}}$	-6.5	4	-3.4	$p_{4_{10}}$	-6.5	4	-3.5
$p_{3_{11}}$	-6.59	4	-3.5	$p_{4_{11}}$	-6.5	4	-3.5
$p_{3_{12}}$	-7.5	4	-4.5	$p_{4_{12}}$	-7.5	4	-4.5
$p_{3_{13}}$	-8	4	-5.5	$p_{4_{13}}$	-8	4	-5.5
$p_{3_{14}}$	-8	4	-5.5	$p_{4_{14}}$	-8	4	-5.5
$p_{3_{15}}$	-8.5	3	-5.5	$p_{4_{15}}$	-8.5	3	-5.5
$p_{3_{16}}$	-8.5	-1	-5.5	$p_{4_{16}}$	-8.5	-1	-5.5
$p_{3_{17}}$	-8.5	-1	-0.5	$p_{4_{17}}$	-8.5	-1	-0.5
$p_{3_{18}}$	-0.5	-1	-0.27	$p_{4_{18}}$	-0.5	-1	-0.5

Table 6.3: Comparison of results of steps of the face-based rubberband algorithm. Points $p_{3_0}, p_{3_1}, \ldots, p_{3_{18}}$ are the results of Step 4; points $p_{4_0}, p_{4_1}, \ldots, p_{4_{18}}$ are the results of Step 7.

Procedure 5). By Lemma 42, Step 7 can be computed in $\mathcal{O}(m)$ time. Therefore, the face-based rubberband algorithm can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time.

6.5 An Example

We approximate the MLP of the simple cube-curve g_{19}, shown in Figure 6.1. Table 6.1 lists all coordinates of critical edges of g_{19}. We take the centers of the first critical faces of g_{19} to produce an initial polygonal curve for the face-based rubberband algorithm. The updated polygonal curves are shown in Tables [2] 6.2 and 6.3. We take the centers of each critical edge of g_{19} for the initialization of the polygonal curve of the (Option 3 corrected) rubberband algorithm. The resulting polygon is

[2]Two digits are used only for displaying coordinates. Obviously, in the calculations it is necessary to use higher precision.

$final_{p_i}$	x_i	y_i	z_i
p_{4_0}	-0.5	1	-0.5
p_{4_2}	-1.5	3	-0.5
p_{4_3}	-2.5	3.22	-0.5
p_{4_7}	-5.5	4	-2.5
$p_{4_{12}}$	-7.5	4	-4.5
$p_{4_{13}}$	-8	4	-5.5
$p_{4_{15}}$	-8.5	3	-5.5
$p_{4_{16}}$	-8.5	-1	-5.5
$p_{4_{17}}$	-8.5	-1	-0.5
$p_{4_{18}}$	-0.5	-1	-0.5

Table 6.4: Results of the edge-based rubberband algorithm. p_{4_0}, p_{4_1}, ..., $p_{4_{18}}$ are the vertices of the MLP of the simple cube-curve shown in Figure 6.1.

shown in Table 6.4. Table 6.5 illustrates that the edge-based and face-based rubberband algorithms converge to the same MLP of g_{19}.

6.6 Conclusions

We presented an edge-based and a face-based rubberband algorithm and have shown that both are provably correct for any simple cube-curve. We also have proved that their time complexity is $\kappa(\varepsilon) \cdot \mathcal{O}(m)$ time, where $\kappa(\varepsilon)$ is as in Equation (1.1), and m is the number of critical edges of g. The presented algorithms followed the basic outline of the original rubberband algorithm [24] (see also Section 1.3).

In this chapter we finally identified one criterion (namely, Theorem 17) for testing whether a polygonal curve inside of a simple cube-curve is actually the MLP of this curve, or not. The main idea of this criterion is implemented by Procedure 7.

This chapter introduced the concept of a *critical face* to deal with the "combinatorial hardness" described in [106], page 666, or in [107]. It also solves the difficulty in

step	initial	2	3	4	8	EBRA
length	35.22	31.11	31.08	31.06	31.01	31.01

Table 6.5: Lengths of calculated curves at different steps of the face-based rubberband algorithm, compared with the length calculated by the edge-based rubberband algorithm (column EBRA).

redesigning Option 2 of the original rubberband algorithm (This option attempted to solve the "combinatorial hardness", but it is flawed in its original design; see Section 5.5).

Referring to Chapter 1 and Section 5.5, we can see that the most powerful operation of the original rubberband algorithm is summarized in its Option 3. All the algorithms, appearing in the following chapters for various applications in computational geometry, are based on this. Option 2 is still very useful even if it may not detect the correct subset of critical edges which contain the vertices of the MLP of a given simple cube-curve. This is because Option 2 is (in general) significantly speeding up and simplifying the algorithm when comparing the edge-based with the face-based rubberband algorithm.

Now we have two provably correct and linear-time rubberband algorithms: the edge-based and the face-based algorithm. We will discuss further variants of rubberband algorithms in the next chapters. We apply the term *rubberband algorithm* for any algorithm which applies the main idea of Option 3 of the original rubberband algorithm.

Chapter 7

ESPs in Simple Cube-Arcs

This chapter proves that there does not exist any exact algorithm for the general MLP problem (and, in conclusion, also not for the ESP problem).

In preparation for applications of rubberband algorithms in the remaining chapters, we also discuss the degenerate case where at least two vertices of an updated polygonal path are identical (calculated by a rubberband algorithm, which applies the original Option 3). We show how to overcome such degenerate cases when applying a rubberband algorithm.

7.1 Introduction

Obviously, the MLP problem (for simple cube-curves) is a special case of an ESP (i.e., Euclidean shortest-path) problem. Before we apply the basic ideas of the discussed versions of MLP algorithms for solving some special cases of ESP problems in the next chapters, we present two issues of ESP or rubberband algorithms in this chapter. At first we apply Galois theory to prove the following fundamental result:

18. THEOREM. *The MLP problem is in general not solvable by radicals over the field of rationals.*

In particular, this theorem allows the following direct conclusions (note: Corollary 3 is Theorem 9 of [15]):

3. COROLLARY. *The ESP problem is in general not solvable by radicals over the field of rationals.*

4. COROLLARY. *There does not exist an exact algorithm for calculating the MLP of any simple cube-curve.*

5. COROLLARY. *There does not exist an exact algorithm for calculating the 3D ESP in any 3D manifold.*

As a second issue, we consider in this chapter a degenerate case (as stated already above) of the rubberband algorithm and how to deal with this problem.

The chapter is organized as follows: Section 7.2 lists some known results used in Section 7.4. Section 7.3 reviews the methodology of testing the irreducibility of a given polynomial. Section 7.4 gives a proof of our main theorem (Theorem 18) in this chapter. Section 7.5 discusses a degenerate case of the rubberband algorithm and how to avoid its occurrence. Section 7.6 concludes this chapter.

7.2 Known Results

In this chapter, all polynomials are monic and with integer coefficients. Let p be a prime. We recall that the set of integers $\{0, 1, 2, \ldots, p-1\}$, with operations

$$a \oplus b = a + b \; mod \; p \quad \text{and} \quad a \odot b = ab \; mod \; p$$

forms a field, denoted by $\mathbb{Z}_p$.

We list theorems used in the proof of the main (i.e., in this chapter) theorem; this theorem is given in Section 7.4.

Let $deg(p(x))$ be the degree of the polynomial $p(x)$. (The following results are well known in mathematical algebra; proofs can be found, for example, in [16, 70].) Let

$$p(x) = x^n + a_{n-1}x^{n-1} + +a_{n-2}x^{n-2} + \ldots + a_1x + a_0$$

The discriminant of $p(x)$ is defined as the $(2n-1) \times (2n-1)$ determinant shown in Figure 7.1. In the case of n = 5, this discriminant is as follows:

$$\begin{vmatrix} 1 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 \\ 0 & 1 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 \\ 0 & 0 & 1 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\ 0 & 0 & 0 & 1 & a_4 & a_3 & a_2 & a_1 & a_0 \\ 5 & 4a_4 & 3a_3 & 2a_2 & 1a_1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 4a_4 & 3a_3 & 2a_2 & 1a_1 & 0 & 0 & 0 \\ 0 & 0 & 5 & 4a_4 & 3a_3 & 2a_2 & 1a_1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 4a_4 & 3a_3 & 2a_2 & 1a_1 & 0 \\ 0 & 0 & 0 & 0 & 5 & 4a_4 & 3a_3 & 2a_2 & 1a_1 \end{vmatrix}$$

Following [16], a *good prime* [1] for a given polynomial $p(x)$ is a prime which does not divide the discriminant of $p(x)$. The following is the first part (i.e., in the case that n is even.) of Lemma 8 in [16]. Let $\mathbb{Q}$ be the set of all rational numbers.

[1]Note that this notion is not uniformly defined in literature; for a different use, see [152], for example.

$$\begin{vmatrix}
1 & a_{n-1} & a_{n-2} & \cdot & \cdot & \cdot & a_0 & 0 & \cdot & \cdot & \cdot & 0 \\
0 & 1 & a_{n-1} & a_{n-2} & \cdot & \cdot & \cdot & a_0 & 0 & \cdot & \cdot & 0 \\
0 & 0 & 1 & a_{n-1} & a_{n-2} & \cdot & \cdot & \cdot & a_0 & 0 & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & \\
0 & 0 & 0 & 0 & 0 & 1 & a_{n-1} & a_{n-2} & \cdot & \cdot & \cdot & a_0 \\
n & (n-1)a_{n-1} & (n-2)a_{n-2} & \cdot & \cdot & 1a_1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\
0 & n & (n-1)a_{n-1} & (n-2)a_{n-2} & \cdot & \cdot & 1a_1 & 0 & 0 & \cdot & \cdot & 0 \\
0 & 0 & n & (n-1)a_{n-1} & (n-2)a_{n-2} & \cdot & \cdot & 1a_1 & 0 & 0 & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & \\
0 & 0 & 0 & 0 & 0 & n & (n-1)a_{n-1} & (n-2)a_{n-2} & \cdot & \cdot & 1a_1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & n & (n-1)a_{n-1} & (n-2)a_{n-2} & \cdot & \cdot & 1a_1
\end{vmatrix}$$

Figure 7.1: Discriminant of a monic polynomial (general case).

43. LEMMA. *Let n = $deg(p(x))$. If n = 0 (mod 2) and $n > 2$, then the joint occurrence of (i) an (n - 1)-cycle, (ii) an n-cycle, and (iii) a permutation of the type 2 + (n - 3) on factoring the polynomial $p(x)$ modulo good primes, allows to conclude that the Galois group of $p(x)$ over $\mathbb{Q}$ is the symmetric group S_n.*

According to [16], an *(n - 1)-cycle occurs* if there exists a good prime p_1 such that

$$p(x)\ mod\ p_1 = f_{1_1}(x) f_{1_2}(x)$$

where $deg(f_{1_1}(x))$ = 1 and $deg(f_{1_2}(x))$ = n - 1, and $f_{1_2}(x)$ is irreducible over $\mathbb{Z}_{p_1}$; an *n-cycle occurs* if there exists a good prime p_2 such that

$$p(x)\ mod\ p_2 = f(x)$$

where $deg(f(x))$ = n is irreducible over $\mathbb{Z}_{p_2}$; and a *permutation of the type 2 + (n - 3) occurs* if there exists a good prime p_3 such that

$$p(x)\ mod\ p_3 = f_{3_1}(x) f_{3_2}(x) f_{3_3}(x)$$

where $deg(f_{3_1}(x))$ = 1, $deg(f_{3_2}(x))$ = 2, $deg(f_{3_3}(x))$ = n - 3, and $f_{3_2}(x)$, $f_{3_3}(x)$ are irreducible over $\mathbb{Z}_{p_3}$.

We recall that the finite symmetric group S_n is the group of all permutations of n elements; it has order n! and it is not abelian. Every element of S_n can be written as a product of cycles. A k-cycle is a permutation of the given n elements such that a k-times repeated application of this permutation maps at least one of the n elements onto itself.

A subgroup H of G is said to be a *normal subgroup* of G iff $ghg^{-1} \in H$, for each $g \in G$ and $h \in H$. A group G is said to be *solvable* iff there exist subgroups $G = H_0 \supset H_1 \supset H_2 \supset \cdots \supset H_r$ such that H_r is a singleton (i.e., it only contains the identity element of G), H_i is normal in H_{i-1}, and H_{i-1}/H_i is abelian.

Assume that $f(x) \in \mathbb{Q}[x]$; then a finite extension E of $\mathbb{Q}$ is said to be a *splitting field* over $\mathbb{Q}$ for $f(x)$ iff $f(x)$ can be factored over E into a product of linear factors, but not over any proper subfield of E.

The *Galois group over $\mathbb{Q}$ of $p(x)$* is defined as a certain group of automorphisms of the splitting field over $\mathbb{Q}$ for $f(x)$; see [70].

The following two propositions are Theorems 5.7.1 and 5.7.2 in [70] (in this order).

44. LEMMA. *The symmetric group S_n is not solvable for $n > 5$.*

45. LEMMA. *If $p(x) \in \mathbb{Q}[x]$ is solvable by radicals over $\mathbb{Q}$, then the Galois group over $\mathbb{Q}$ of $p(x)$ is a solvable group.*

7.3 Irreducibility Test of a Polynomial

The following methodology (which is used in the next section) is a simplification of a 1967 algorithm by Elwyn R. Berlekamp. This algorithm was designed for the factorization of polynomials; see, for example, [86].

Let p be a prime number. In this Section, all arithmetics on polynomials is modulo p. Let

$$u(x) = x^n + u_{n-1}x^{n-1} + \ldots + u_1 x + u_0$$

for $u_i \in \mathbb{Z}_p$ and $i = 0, 1, \ldots, n$. Let

$$\alpha_0 = (0\ 0\ \ldots\ 0\ 1)$$

and

$$\alpha = (a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0)$$

be two n-dimensional row vectors, where $a_i \in \mathbb{Z}_p$ and $i = 0, 1, \ldots, n-1$. Then we may update α by the following procedure:

8. PROCEDURE.

1. Let k = 1 and $\beta = (u_{n-1}\ u_{n-2}\ \ldots\ u_1\ u_0)$.
2. Let α = α_0.
3. Let $t = a_{n-1}$,

$$a_{n-1} = (a_{n-2} - tu_{n-1})\ mod\ p$$

$$\dots\dots\dots\dots$$

$$a_1 = (a_0 - tu_1) \, mod \, p$$

and

$$a_0 = (-tu_0) \, mod \, p$$

4. Let $\alpha_0 = \alpha$.
5. Let $k = k + 1$.
6. If $k < p$, go to Step 2, otherwise output α.

We illustrate Procedure 8 by the following example.

4. EXAMPLE.

Let $p = 19$. Consider the input

$$u(x) = x^6 + 12x^3 + 12x^2 + 13x + 15$$

also expressed by

$$\beta = (1 \;\; 0 \;\; 12 \;\; 12 \;\; 13 \;\; 15)$$

Table 7.1 shows the updated row-vectors α_k, each corresponding to a number k of iterations.

The following procedure computes a matrix Q which is used (for example, see Section 7.4.3) for testing the irreducibility of a polynomial:

9. PROCEDURE.

1. Let $k = 1$ and Q be an $n \times n$ zero matrix.
2. Let be given two n-dimensional row vectors α_0 and β as input; apply Procedure 8 to update α.
3. Update Q by replacing its k-th row by the reversed α.
4. Let $\alpha_0 = \alpha$.
5. Let $k = k + 1$.
6. If $k < n$, go to Step 2, otherwise output Q.

We provide the following example for illustrating this procedure:

5. EXAMPLE.

We start with $u(x) = x^6 + 12x^3 + 12x^2 + 13x + 15$, as obtained in the example before; reverse the last row in Table 7.1. We have

$$\beta = (6 \;\; 13 \;\; 15 \;\; 1 \;\; 15 \;\; 5)$$

k	$a_{k,5}$	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	0	0	0	0	0	1
1	0	0	0	0	1	0
2	0	0	0	1	0	0
3	0	0	1	0	0	0
4	0	1	0	0	0	0
5	1	0	0	0	0	0
6	0	0	7	7	6	4
7	0	7	7	6	4	0
8	7	7	6	4	0	0
9	7	6	15	11	4	9
10	6	15	3	15	13	9
11	15	3	0	17	7	5
12	3	0	8	17	0	3
13	0	8	0	2	2	12
14	8	0	2	2	12	0
15	0	2	1	11	10	13
16	2	1	11	10	13	0
17	1	11	5	8	12	8
18	11	5	15	0	14	4
19	5	15	1	15	13	6

Table 7.1: Illustration of Procedure 8; see Example 4.

as in input for Procedure 9. Table 7.2 shows the updated vector α corresponding to the number k = 3 of iterations. Thus, the third row of Q is equal to

$$(2\ \ 5\ \ 6\ \ 2\ \ 13\ \ 0)$$

Let I be the $n \times n$ identity matrix and $deg(u(x))$ = n. The following proposition by Elwyn R. Berlekamp (see [86], page 441) gives a sufficient and necessary condition for testing the irreducibility of a given polynomial; this condition is later used in Section 7.4.

5. PROPOSITION. *$u(x)$ is irreducible iff the rank of matrix $Q - I$ equals n - 1.*

7.4 Proof of Theorem 18

For this theorem, we consider the simple cube-arc ρ between e_0 and e_3 in Figure 1.2 (see also Table 3.1). The coordinates of $P_0(t_0)$ and $P_3(t_3)$ are equal to (1, 4, 7) and (4,

k	$a_{k,5}$	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	5	15	1	15	13	6
1	15	1	12	10	17	1
2	1	12	1	8	15	3
3	12	1	15	3	9	4
4	1	15	11	17	0	10
5	15	11	5	7	16	4
6	11	5	17	7	18	3
7	5	17	8	0	12	6
8	17	8	16	9	17	1
9	8	16	14	3	8	11
10	16	14	2	7	2	13
11	14	2	5	0	14	7
12	2	5	3	17	15	18
13	5	3	12	10	11	8
14	3	12	7	8	0	1
15	12	7	10	2	0	12
16	7	10	10	8	8	10
17	10	10	0	0	14	9
18	10	0	13	8	12	2
19	0	13	2	6	5	2

Table 7.2: Illustration of Procedure 9: calculation of the third row of Q (before reversing).

7, 4), respectively, where $t_0 = t_3 = 0$.

We want to detect t_1, $t_2 \in [0, 1]$ such that a polyline $P_0(t_0)P_1(t_1)P_2(t_2)P_3(t_3)$ is fully contained in ρ.

By Equations (3.13) and (3.14), we have

$$0 = \frac{t_1}{\sqrt{t_1^2+5}} + \frac{t_1-1}{\sqrt{(t_1-1)^2+(t_2-1)^2+4}} \tag{7.1}$$

$$0 = \frac{t_2-1}{\sqrt{(t_1-1)^2+(t_2-1)^2+4}} + \frac{t_2}{\sqrt{t_2^2+4}} \tag{7.2}$$

We show that the problem is reduced to finding the roots of the polynomial

$$p(x) = 84x^6 - 228x^5 + 361x^4 + 20x^3 + 210x^2 - 200x + 25$$

7.4.1 Computation of the Polynomial $p(x)$

This subsection shows the steps for computing the polynomial $p(x)$.

Let $t_1 = x$ and $t_2 = y$ in Equations (7.1), (7.2). Equations (7.3), (7.5), ..., (7.11) show, step by step, the simplification of Equation (7.1); Equations (7.4), (7.6), ..., (7.12) show the steps of simplifying Equation (7.2):

$$0 = \frac{x}{\sqrt{x^2+5}} + \frac{x-1}{\sqrt{(x-1)^2+(y-1)^2+4}} \tag{7.3}$$

$$0 = \frac{y-1}{\sqrt{(x-1)^2+(y-1)^2+4}} + \frac{y}{\sqrt{y^2+4}} \tag{7.4}$$

$$\frac{x}{\sqrt{x^2+5}} = -\frac{x-1}{\sqrt{(x-1)^2+(y-1)^2+4}} \tag{7.5}$$

$$\frac{y-1}{\sqrt{(x-1)^2+(y-1)^2+4}} = -\frac{y}{\sqrt{y^2+4}} \tag{7.6}$$

$$x\sqrt{(x-1)^2+(y-1)^2+4} = -(x-1)\sqrt{x^2+5} \tag{7.7}$$

$$(y-1)\sqrt{y^2+4} = -y\sqrt{(x-1)^2+(y-1)^2+4} \tag{7.8}$$

$$x^2[(x-1)^2+(y-1)^2+4] = (x-1)^2(x^2+5) \tag{7.9}$$

$$(y-1)^2(y^2+4) = y^2[(x-1)^2+(y-1)^2+4] \tag{7.10}$$

$$x^2[(y-1)^2+4] = 5(x-1)^2 \tag{7.11}$$

$$4(y-1)^2 = y^2[(x-1)^2+4] \tag{7.12}$$

Equations (7.13), (7.14) and (7.15) show the steps for representing y in terms of x, based on Equation (7.11):

$$\begin{aligned}(y-1)^2 &= \frac{5(x-1)^2}{x^2} - 4 \\ &= \frac{5(x-1)^2 - 4x^2}{x^2} \\ &= \frac{5(x^2-2x+1) - 4x^2}{x^2} \\ &= \frac{x^2-10x+5}{x^2}\end{aligned} \tag{7.13}$$

$$y - 1 = -\frac{\sqrt{x^2 - 10x + 5}}{x} \tag{7.14}$$

$$y = 1 - \frac{\sqrt{x^2 - 10x + 5}}{x} \tag{7.15}$$

Substituting y in Equation (7.12) by the right hand side of Equation (7.15), we have the following:

$$4(1 - \frac{\sqrt{x^2 - 10x + 5}}{x} - 1)^2 = (1 - \frac{\sqrt{x^2 - 10x + 5}}{x})^2[(x-1)^2 + 4] \tag{7.16}$$

Equations (7.17), (7.18), . . ., (7.22) show the steps of simplifying Equation (7.16):

$$4\frac{x^2 - 10x + 5}{x^2} = (\frac{x - \sqrt{x^2 - 10x + 5}}{x})^2[(x-1)^2 + 4] \tag{7.17}$$

$$\begin{aligned} \frac{4(x^2 - 10x + 5)}{(x-1)^2 + 4} &= (x - \sqrt{x^2 - 10x + 5})^2 \\ &= x^2 + x^2 - 10x + 5 - 2x\sqrt{x^2 - 10x + 5} \\ &= 2x^2 - 10x + 5 - 2x\sqrt{x^2 - 10x + 5} \end{aligned} \tag{7.18}$$

$$\frac{4(x^2 - 10x + 5)}{(x-1)^2 + 4} - (2x^2 - 10x + 5) = -2x\sqrt{x^2 - 10x + 5} \tag{7.19}$$

$$\frac{4(x^2 - 10x + 5) - (2x^2 - 10x + 5)(x^2 - 10x + 5)}{x^2 - 2x + 5} = -2x\sqrt{x^2 - 10x + 5} \tag{7.20}$$

$$\frac{[4(x^2 - 10x + 5) - (2x^2 - 10x + 5)(x^2 - 10x + 5)]^2}{(x^2 - 2x + 5)^2} = 4x^2(x^2 - 10x + 5) \tag{7.21}$$

$$[4(x^2-10x+5)-(2x^2-10x+5)(x^2-2x+5)]^2 = 4x^2(x^2 - 10x + 5)(x^2-2x+5)^2 \tag{7.22}$$

The left hand side of Equation (7.22) can be further simplified as follows:

$$([4(x^2 - 10x + 5) - (2x^2 - 10x + 5)(x^2 - 2x + 5)]^2$$

$$[4x^2 - 40x + 20 - (2x^4 - 14x^3 + 35x^2 - 60x + 25)]^2$$
$$[-2x^4 + 14x^3 - 31x^2 + 20x - 5]^2$$
$$4x^8 - 56x^7 + 320x^6 - 948x^5 + 1541x^4 - 1380x^3 + 710x^2 - 200x + 25$$

The right hand side of Equation (7.22) can be further simplified as follows:

$$4x^2(x^2 - 10x + 5)(x^2 - 2x + 5)^2$$
$$4x^2(x^2 - 10x + 5)(x^4 - 4x^3 + 14x^2 - 20x + 25)$$
$$4x^2(x^6 - 14x^5 + 59x^4 - 180x^3 + 295x^2 - 350x + 125)$$
$$4x^8 - 56x^7 + 236x^6 - 720x^5 + 1180x^4 - 1400x^3 + 500x^2$$

Altogether, we obtain the following polynomial

$$84x^6 - 228x^5 + 361x^4 + 20x^3 + 210x^2 - 200x + 25 = 0$$

as a simplification of Equation (7.22). In the following, we consider the polynomial $p(x) = 84x^6 - 228x^5 + 361x^4 + 20x^3 + 210x^2 - 200x + 25$.

7.4.2 Application of Bajaj's Method

To prove Theorem 18, by Lemmas 43 to 45 it suffices to find three good primes, p_1, p_2 and p_3 for $p(x)$, such that:

1. $p(x)$ mod p_1 can be factorized as one irreducible polynomial $f_{1_1}(x)$ over the field $\mathbb{Z}_{p_1}$ such that
$$deg(f_{1_1}(x)) = deg(p(x))$$
2. $p(x)$ mod p_2 can be factorized as two irreducible polynomials $f_{2_1}(x)$ and $f_{2_2}(x)$ over the field $\mathbb{Z}_{p_2}$ such that
$$deg(f_{2_1}(x)) = 1$$
and
$$deg(f_{2_2}(x)) = deg(p(x)) - 1$$
and
3. $p(x)$ mod p_3 can be factorized as three irreducible polynomials $f_{3_1}(x)$, $f_{3_2}(x)$ and $f_{3_3}(x)$ over the field $\mathbb{Z}_{p_3}$ such that
$$deg(f_{3_1}(x)) = 1$$
$$deg(f_{3_2}(x)) = 2$$
and
$$deg(f_{3_3}(x)) = deg(g(x)) - 3$$

Indeed, using mathematical software GAP ([55]), we identified three good primes as desired (for further details, see Appendix C), namely:

1. p_1 = 19,

$$f_{1_1}(x) = Z(19)^3 * x^6 + x^3 + x^2 + Z(19)^8 * x + Z(19)^{14}$$

2. p_2 = 37,

$$\begin{aligned} f_{2_1}(x) &= Z(37)^{24} * x + Z(37)^0 \\ f_{2_2}(x) &= x^5 + Z(37)^{26} * x^4 + Z(37)^{22} * x^3 \\ &\quad +Z(37)^{30} * x^2 + Z(37)^9 * x + Z(37)^{10} \end{aligned}$$

3. p_3 = 13,

$$\begin{aligned} f_{3_1}(x) &= Z(13)^5 * x + Z(13)^{10} \\ f_{3_2}(x) &= x^2 + Z(13)^5 \\ f_{3_3}(x) &= x^3 + Z(13)^3 * x^2 + Z(13)^2 * x + Z(13)^3 \end{aligned}$$

7.4.3 $f_{1_1}(x)$ Is Irreducible

To prove that $f_{1_1}(x)$ is irreducible, we proceed as follows:

Step 1. We simplify $f_{1_1}(x)$ to a monic polynomial. Since 2 is a primitive element of Z_{19}, [2] we have

$$\begin{aligned} f_{1_1}(x) &= Z(19)^3 * x^6 + x^3 + x^2 + Z(19)^8 * x + Z(19)^{14} \\ &= 2^3 * x^6 + x^3 + x^2 + 2^8 * x + 2^{14} \\ &= 8x^6 + x^3 + x^2 + 9x + 6 \end{aligned}$$

Since 8 $\times$ 12 $\equiv$ 1 *mod* 19,

$$\begin{aligned} f_{1_1}(x) &= 8x^6 + x^3 + x^2 + 9x + 6 \\ &= 12(8x^6 + x^3 + x^2 + 9x + 6) \\ &= x^6 + 12x^3 + 12x^2 + 13x + 15 \end{aligned}$$

The final row is the desired monic representation.

Step 2. We apply Proposition 5 to prove that $f_{1_1}(x)$ is irreducible. We summarize the computation here; for details see Tables 7.1 and 7.2, and Tables D.1 to D.3 in

[2]For each $a \in \mathbb{Z}_{19}$, there exists a number $j \in \{0, 1, \ldots, 18\}$ such that $2^j \equiv a \; mod \; 19$.

Appendix D. Note that the rows computed in these tables have to be reversed to be the rows of the following matrix Q. We obtain that

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 13 & 15 & 1 & 15 & 5 \\ 2 & 5 & 6 & 2 & 13 & 0 \\ 2 & 10 & 13 & 13 & 0 & 14 \\ 17 & 10 & 10 & 1 & 2 & 7 \\ 4 & 16 & 8 & 10 & 18 & 3 \end{bmatrix}$$

Therefore,

$$Q - I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 12 & 15 & 1 & 15 & 5 \\ 2 & 5 & 5 & 2 & 13 & 0 \\ 2 & 10 & 13 & 12 & 0 & 14 \\ 17 & 10 & 10 & 1 & 1 & 7 \\ 4 & 16 & 8 & 10 & 18 & 2 \end{bmatrix}$$

Because of

$$\begin{vmatrix} 6 & 12 & 15 & 1 & 15 \\ 2 & 5 & 5 & 2 & 13 \\ 2 & 10 & 13 & 12 & 0 \\ 17 & 10 & 10 & 1 & 1 \\ 4 & 16 & 8 & 10 & 18 \end{vmatrix} = -160520 \equiv 11 \; mod \; 19$$

we finally obtain that

$$rank(Q - I) = 6$$

By Proposition 5, we can conclude that $f_{1_1}(x)$ is irreducible.

7.4.4 $f_{2_2}(x)$ Is Irreducible

To prove that $f_{2_2}(x)$ is irreducible, we proceed as follows:

Step 1. We simplify $f_{2_2}(x)$. Because 2 is a primitive element of $\mathbb{Z}_{37}$, we have the following:

$$\begin{aligned} f_{2_2}(x) &= x^5 + Z(37)^{26} * x^4 + Z(37)^{22} * x^3 \\ &\quad + Z(37)^{30} * x^2 + Z(37)^9 * x + Z(37)^{10} \\ &= x^5 + 2^{26} * x^4 + 2^{22} * x^3 + 2^{30} * x^2 + 2^9 * x + 2^{10} \\ &= x^5 + 3x^4 + 21x^3 + 11x^2 + 31x + 25. \end{aligned}$$

Step 2. Now we apply Proposition 5 to prove that $f_{2_2}(x)$ is irreducible. We summarize the computation here; for details see Tables D.4 and D.7 in Appendix D. We obtain that

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 3 & 1 & 11 & 21 & 36 \\ 28 & 29 & 22 & 34 & 33 \\ 23 & 7 & 1 & 22 & 31 \\ 32 & 35 & 24 & 35 & 28 \end{bmatrix}$$

Therefore,

$$Q - I = \begin{bmatrix} 3 & 1 & 11 & 21 \\ 28 & 29 & 22 & 34 \\ 23 & 7 & 1 & 22 \\ 32 & 35 & 24 & 35 \end{bmatrix}$$

Because of

$$\begin{vmatrix} 3 & 1 & 11 & 21 \\ 28 & 29 & 22 & 34 \\ 23 & 7 & 1 & 22 \\ 32 & 35 & 24 & 35 \end{vmatrix} = 9835 \equiv 30 \; mod \, 37$$

we finally have

$$rank(Q - I) = 4$$

By Proposition 5, this proves that $f_{2_2}(x)$ is irreducible.

7.4.5 $f_{3_2}(x)$ Is Irreducible

To prove that $f_{3_2}(x)$ is irreducible, we proceed as follows:

Step 1. We simplify $f_{3_2}(x)$. Because 2 is a primitive element of $\mathbb{Z}_{13}$, we have the following:

$$\begin{aligned} f_{3_2}(x) &= x^2 + Z(13)^5 \\ &= x^2 + 6 \end{aligned}$$

Step 2. We apply Proposition 5 to prove that $f_{3_2}(x)$ is irreducible. For details see Table D.8. We obtain that

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 12 \end{bmatrix}$$

Therefore,

$$Q - I = \begin{bmatrix} 0 & 0 \\ 0 & 11 \end{bmatrix}$$

Thus, we have that

$$rank(Q - I) = 1$$

By Proposition 5, this shows that also $f_{3_2}(x)$ is irreducible.

7.4.6 $f_{3_3}(x)$ Is Irreducible

To prove that $f_{3_3}(x)$ is irreducible, we proceed as follows:

Step 1. We simplify $f_{3_3}(x)$. Since 2 is a primitive element of Z_{13}, we have the following:

$$\begin{aligned} f_{3_3}(x) &= x^3 + Z(13)^3 * x^2 + Z(13)^2 * x + Z(13)^3 \\ &= x^3 + 8x^2 + 4x + 8 \end{aligned}$$

Step 2. We apply Proposition 5 to prove that $f_{3_3}(x)$ is irreducible. For details see Tables D.9 and D.10 in Appendix D. We obtain that

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 3 & 0 \\ 1 & 6 & 9 \end{bmatrix}$$

Therefore,

$$Q - I = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 6 & 8 \end{bmatrix}$$

From this we see that

$$rank(Q - I) = 2$$

By Proposition 5, we know that $f_{3_3}(x)$ is irreducible.

7.5 A Degenerate Case of the Rubberband Algorithm

This section studies a degenerate case of a (simplified) rubberband algorithm, assuming that we apply Option 3 of the original rubberband algorithm: at least two vertices of the updated polygonal path are identical. This section shows how to handle such a degenerate case.

Let s_1, s_2, … s_k be a sequence of segments. Let p, $q \notin \partial s_i$, where i = 1, 2, …, k such that $p \neq q$. If we want to find a point $q_i \in \partial s_i$, where i = 1, 2, …, k, such that the length of the simple polyline $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$ is minimized, then we may slightly modify Procedure 1 into an "arc version" of the rubberband algorithm as follows:

3. ALGORITHM.

```
1. Let $\varepsilon = 10^{-10}$ (the chosen accuracy).
2. Compute the length $l_1$ of the 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.
```

The accuracy parameter in Step 1 can be chosen such that maximum possible numerical accuracy is guaranteed on the given computer. Apart from the examples given below, more examples follow in Section 8.3.2.

In the rest of this report, we call

$$\{s_1, s_2, \ldots, s_k\}$$

the *step set* of the rubberband algorithm, and each s_i a *step element* of the rubberband algorithm, where $i = 1, 2, \ldots, k$.

6. EXAMPLE.

Let the input for Algorithm 3 be as follows (see also Figure 7.2):

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

To initialize, let p_1 and p_2 be 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. Algorithm 3 finds the 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 7.3, which lists resulting δs for the number I of iterations).

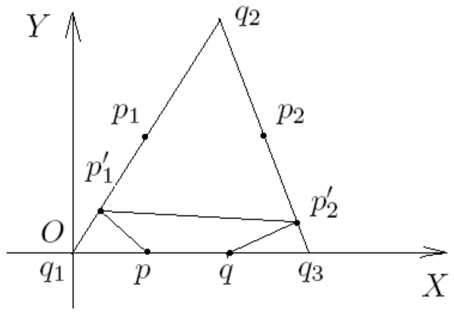


Figure 7.2: Illustration of a degenerate case of the rubberband algorithm.

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

Table 7.3: Number I of iterations and resulting δs, for Example 6 illustrated by Figure 7.2, with $p_1 = (1, 2)$ and $p_2 = (2.5, 2)$ as initialization points.

7. EXAMPLE.

Now we modify Example 6 such that $p_1 = p_2 = q_2$; in this case, the output of Step 4.2 in Algorithm 3 will be wrong: 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, we see that $p_1 \neq p_0$ and p_2 in this example.

We call a situation as in the previous example a *degenerate case* of the rubberband algorithm. In general, it is defined by the occurrence of at least two identical vertices of the updated polygonal path. Such a degenerate case causes Step 4.2 in Algorithm 3 to fail.

A degenerate case can be solved approximately: we will not allow $p_2 = q_2$. To do so, we can remove sufficiently small segments from both segments s_1 and s_2. The following example shows how to handle such a degenerate case.

8. EXAMPLE.

We modify the initialization step of Example 7 as follows: Let the accuracy be

$$\varepsilon = 1.0 \times 10^{-100}$$

and let

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

The length of the initialized polyline $\rho = \langle p, p_1, p_2, q\rangle$ is equal to 8.1231. Algorithm 3 will calculate the 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 7.4 for resulting δs in dependence of the number I of iterations).

Of course, if we leave the accuracy to be equals $\varepsilon = 1.0 \times 10^{-10}$, then the algorithm will stop sooner, after less iterations. The algorithm was implemented on a Pentium 4 PC using Matlab 7.04. If we change the value of δ' into

$$\delta' = 2.22 \times 10^{-16}$$

then we obtain the same wrong result as that of Example 6. This is because the computer is 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.

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

Table 7.4: Number I of iterations and resulting δs, for the example shown in Figure 7.2, with $p_1 = (2-\delta', 2(2-\delta'))$ and $p_2 = (2+\delta', -4((2+\delta')-3))$ as initialization points and δ' = 2.221e-16.

7.6 Conclusions

The chapter showed that there does not exist an exact algorithm for solving the general MLP problem in 3D space. This implies that there does not exist an exact algorithm for the general 3D ESP problem as well.

Interestingly, this result is not true for the 2D case. There exists an exact algorithm for the MLP problem in 2D space (see, for example, [82]) and there are also exact algorithms for the ESP problem in 2D space (see, for example, [106]).

By Example 7 we introduced a degenerate case of the rubberband algorithm: there are at least two identical step elements. Such a degenerate case can be solved approximately with high accuracy by transforming the step set into a set of mutually different elements (e.g., by removing sufficiently small segments). It is recommended to handle such degenerate cases properly by such an operation when applying a rubberband algorithm.

Chapter 8

ESPs in Simple Polygons

Let p and q be two points in a simple polygon Π. An open problem in computational geometry asks to devise a simple linear-time algorithm for computing a shortest path between p and q, which is contained in Π, such that the algorithm does not depend on a (complicated) linear-time triangulation algorithm. This chapter provides an algorithm towards a solution of this problem by applying the rubberband algorithm. The obtained solution has $\kappa(\varepsilon) \cdot \mathcal{O}(n \log n)$ time complexity (where the super-linear time complexity is only due to preprocessing, i.e. for the calculation of critical segments) and is, altogether, considerably simpler than the triangulation algorithm. It has applications in 2D pattern recognition, picture analysis, robotics, and so forth.

8.1 Introduction

So far, methods for computing the Euclidean shortest path (ESP) between two points in a simple polygon, as in [62, 63, 71, 106], all rely on starting with a rather complicated, but linear-time triangulation [35] of a simple polygon. In this chapter, we apply a version of a rubberband algorithm to devise a "methodically simple" $\kappa(\varepsilon)\cdot \mathcal{O}(n^2)$ algorithm for computing a shortest path between p and q, which is contained in Π, where $\kappa(\varepsilon)$ is as in Equation (1.1), and n is the number of vertices of Π. Our algorithm starts with a special (say, horizontal) trapezoidal segmentation of the polygon, which is computationally not very different to a triangulation, and thus our segmentation can also be seen as a possible contribution to simplify the triangulation procedure.

The rubberband algorithm, as applied in this chapter, is a simplified version of the edge-based rubberband algorithm, and it is presented in Section 8.3.1.

There is a general option provided by the studied rubberband algorithms: the basic approach for minimizing a path does not depend upon the specific geometric shape of grid cubes; it can be applied to a wide variety of 2D or 3D path minimization problems where segmentations into convex subsets are appropriate.

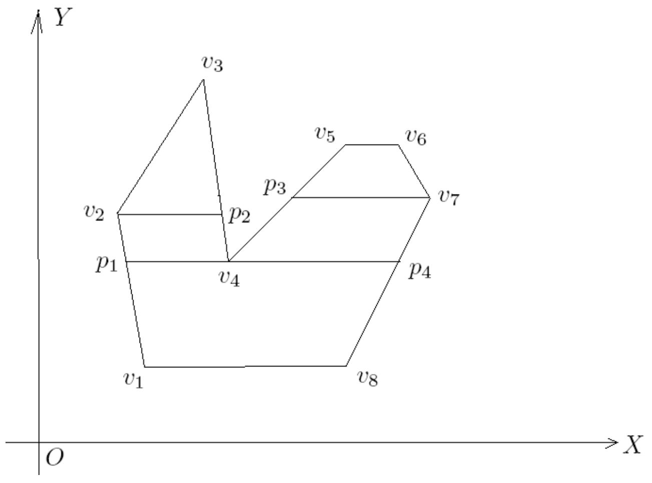


Figure 8.1: Critical segments of a simple polygon (see also Table 8.1).

In the rest of this chapter, Section 8.2 provides necessary definitions and theorems. Section 8.3 presents not only our algorithm but also examples and analysis of time complexity. Section 8.4 concludes the chapter.

8.2 Basics

We denote by $\Pi = \langle v_1, v_2, \ldots, v_n \rangle$ a simple polygon (i.e., a compact polygonal region) in the 2D Euclidean plane (which is equipped with an xy Cartesian coordinate system). $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices of Π, and $\vartheta\Pi = \cup_{i=1}^{n-1}\{v_i v_{i+1}\}$ $\cup\{v_n v_1\}$ is a simple polygonal curve specifying edges forming the frontier of Π.

For $p \in \mathbb{R}^2$, let p_x be the x-coordinate of p. Let $s = p_1 p_2$, with $p_1, p_2 \in \vartheta\Pi$ and $p_{1_x} \leq p_{2_x}$. Furthermore, assume that s is parallel to the x-axis, $s \subset \Pi$, and there is no $v \in V \setminus \{p_1, p_2\}$ such that v is between p_1 and p_2.

31. DEFINITION. *If $p_1 \in V$ ($p_2 \in V$) then we say that s is the* right (left) critical segment *(of Π) with respect to p_1 (p_2). If $p_1 = p_2$ then we say that s is* degenerate. *A* critical segment *is either a left or a right critical segment.*

See Figure 8.1 and Table 8.1 for examples. – For a critical segment s of Π, let s_y be the y-coordinate of points on s. For a given y, let $\{s_1, s_2, \ldots, s_m\}$ be the maximal set of critical segments of Π such that $s_{i_y} = s_{i+1_y}$ and $s_i \cap s_{i+1} \neq \phi$, where i = 1, 2, . . ., m - 1.

Vertex	Left critical segment	Right critical segment
v_1	degenerate	v_1v_8
v_2	degenerate	v_2p_2
v_3	degenerate	degenerate
v_4	p_1v_4	v_4p_4
v_5	degenerate	v_5v_6
v_6	v_5v_6	degenerate
v_7	p_3v_7	degenerate
v_8	v_1v_8	degenerate

Table 8.1: All critical segments of vertices of the simple polygon of Figure 8.1.

32. DEFINITION. *The segment* $\cup_{i=1}^{m}\{s_i\}$ *is a* maximal critical segment *of* Π*. If* $m > 1$ *(*m *= 1) then we say that the segment* $\cup_{i=1}^{m}\{s_i\}$ *is a* non-trivial *(*trivial*)* maximal critical segment *of* Π*.*

In Figure 8.1, p_1p_4 is the only non-trivial maximal critical segment of the shown simple polygon.

33. DEFINITION. *Two maximal critical segments* s_1 *and* s_2 *are called* adjacent *iff there is no maximal critical segment* s_3 *such that* $s_{1y} < s_{3y} < s_{2y}$ *or* $s_{2y} < s_{3y} < s_{1y}$*.*

In Figure 8.1, the trivial maximal critical segment v_5v_6 is adjacent to the trivial maximal critical segment p_3v_7, but not adjacent to the non-trivial maximal critical segment p_1p_4.

Let $\{s_1, s_2, \ldots, s_k\}$ be the set of maximal critical segments of Π. Construct a weighted tree T as follows: Let $T = [V, E]$, where $V = \{u_1, u_2, \ldots, u_k\}$, $E = \{u_iu_j :$ s_i and s_j are adjacent $\}$, and each $e \in E$ has a weight equal to 1.

34. DEFINITION. *We say that* T *is a* 1-tree *of* Π *(with respect to the given Cartesian coordinate system).*

Figure 8.2 shows a 1-tree of a simple polygon, and Figure 8.4 that of the "non-trivial" simple polygon which is shown in Figure 8.3. –

Let $S = \{s_1, s_2, \ldots, s_k\}$ be a subset of the set of all maximal critical segments of Π.

35. DEFINITION. S *is called a* sequence *of maximal critical segments of* Π *iff, for each* $i \in \{1, 2, \ldots, k-1\}$*,* s_i *is adjacent to* s_{i+1}*.*

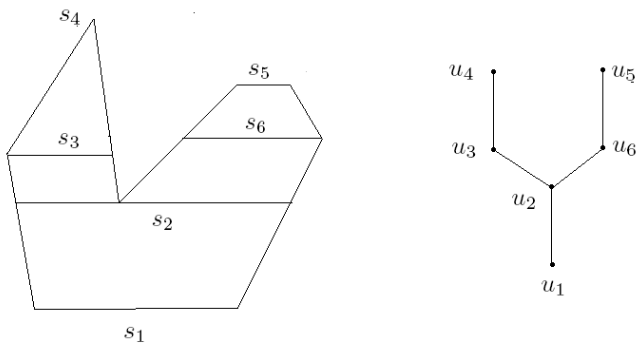


Figure 8.2: Left: simple polygon with six maximal critical segments. Right: its 1-tree.

If, for each $i \in \{1, 2, \ldots, k-1\}$, $s_{iy} < s_{i+1_y}$ ($s_{iy} > s_{i+1_y}$) then S is called an *increasing (decreasing)* sequence of maximal critical segments of Π. S is called a *monotonous* sequence of maximal critical segments of Π iff it is either an increasing or a decreasing sequence of maximal critical segments of it. Finally, S is called an *alternate* monotonous sequence of maximal critical segments of Π iff it is a sequence of maximal critical segments of Π, and it is the union of a finite number of monotonous sequences of maximal critical segments of Π.

In Figure 8.2 (left), $\{s_1, s_2, s_3, s_4\}$ is an increasing sequence of maximal critical segments of Π while $\{s_5, s_6, s_2, s_1\}$ is a decreasing sequence of maximal critical segments. $\{s_3, s_2, s_6\}$ is an alternate monotonous sequence of maximal critical segments of Π.

Let $S = \{s_1, s_2, s_3\}$ be a sequence of maximal critical segments of Π with $s_1 \neq s_3$ such that there is no maximal critical segment between s_1 and s_2 or s_3 and s_2, and there exist critical segments s_1^* and s_2^* such that $s_1^* \subset s_2$ and $s_2^* \subset s_2$, and s_1^* and s_1 are two edges of a quadrilateral, as well as s_2^* and s_3. (For example, in Figure 8.3, $\{s_{28}, v_{10}v_{13}, s_{31}\}$ is such a sequence of maximal critical segments of Π, with $v_{10}v_{11}$, $v_{12}v_{13} \subset v_{10}v_{13}$; segments $v_{10}v_{11}$ and s_{28} are two edges of a quadrilateral, as well as $v_{12}v_{13}$ and s_{31}.) For such a sequence $S = \{s_1, s_2, s_3\}$ we define the following:

36. Definition. *If $s_{1_y} < s_{2_y}$ and $s_{2_y} > s_{3_y}$ ($s_{1_y} > s_{2_y}$ and $s_{2_y} < s_{3_y}$) then s_2 is called an* upward *(*downward*) maximal critical segment of* Π.

Furthermore, s_2 is also called a *stable* maximal critical segment of Π if it is an upward or downward maximal critical segment of Π; both s_1^* and s_2^* are called the *good* critical segments of s_2 with respect to Π.

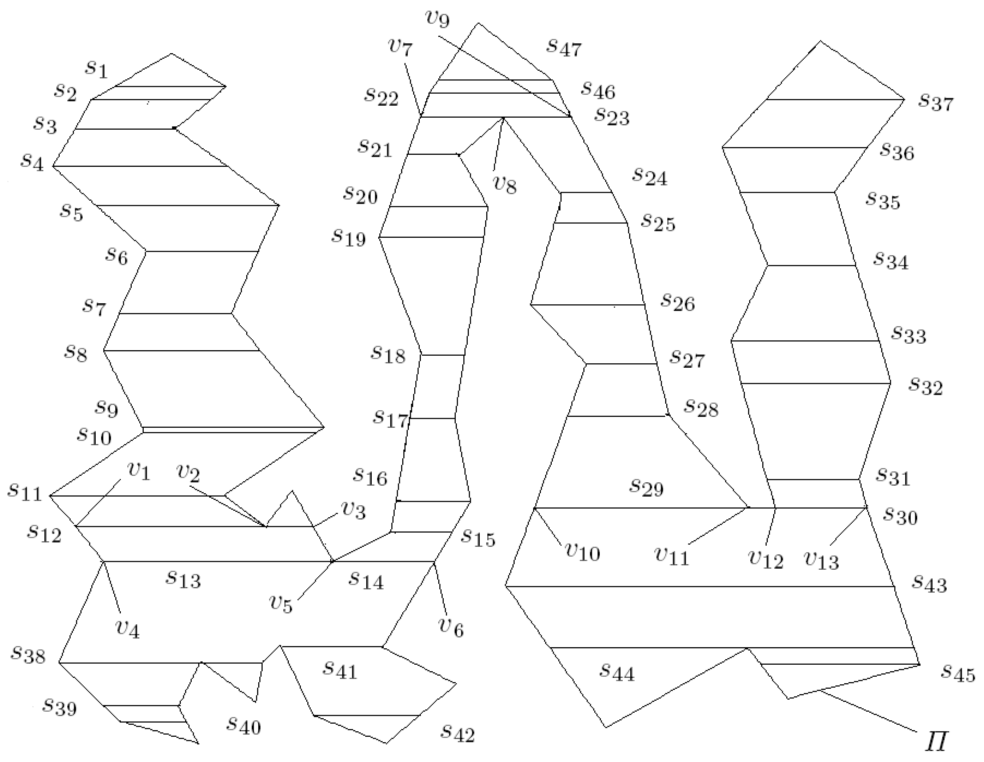


Figure 8.3: A "non-trivial" simple polygon Π with its critical segments.

In Figure 8.2, s_2 is the unique downward maximal critical segment of the shown simple polygon. There is no upward maximal critical segment.

Let $p, q \in \mathbb{R}^2$ be two points in the simple polygon Π. Let $\rho(p, q)$ be a shortest path from p to q. Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of maximal critical segments of Π such that, for each $i \in \{1, 2, \ldots, k\}$, $s_i \cap \rho(p, q) \neq \phi$.

We modify S as follows: if s_i is a stable maximal critical segment, then replace s_i by its good critical segments.

In Figure 8.3, v_4v_6 is a stable maximal critical segment. It has two good critical segments v_4v_5 and v_5v_6. Segment $v_{10}v_{13}$ is another stable maximal critical segment. It has two good critical segments $v_{10}v_{11}$ and $v_{12}v_{13}$.

37. DEFINITION. *The modified set S is called a* step set *of maximal critical segments of* Π *with respect to the shortest path $\rho(p, q)$.*

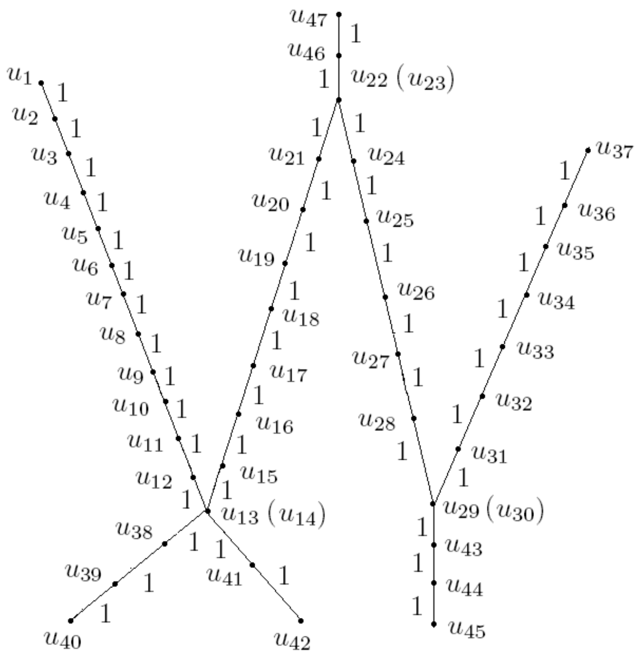


Figure 8.4: 1-tree of the simple polygon shown in Figure 8.3.

For example, in Figure 8.1, $\{v_2p_2, p_1v_4, v_4p_4\}$ (obtained by modifying the set $\{v_2p_2, p_1p_4\}$) is a step set of maximal critical segments of the simple polygon Π with respect to the shortest path $\rho(v_3, v_7)$. As another example, in Figure 8.3,

$$(\cup_{i=1}^{12}\{s_i\}) \cup \{v_4v_6\} \cup (\cup_{i=15}^{21}\{s_i\}) \cup \{v_7v_9\} \cup (\cup_{i=24}^{28}\{s_i\}) \cup \{v_{10}v_{13}\} \cup (\cup_{i=31}^{37}\{s_i\})$$

is a set of maximal critical segments of Π. It can be modified into a step set

$$(\cup_{i=1}^{12}\{s_i\}) \cup \{s_{13}(= v_4v_5), s_{14}(= v_5v_6)\} \cup (\cup_{i=15}^{21}\{s_i\}) \cup$$
$$\{s_{22}(= v_7v_8), s_{23}(= v_8v_9)\} \cup (\cup_{i=24}^{28}\{s_i\}) \cup$$
$$\{s_{29}(= v_{10}v_{11}), s_{30}(= v_{12}v_{13})\} \cup (\cup_{i=31}^{37}\{s_i\})$$

Let S be a step set of maximal critical segments of Π with respect to the shortest path $\rho(p, q)$, and $s_1 \in S$. Let $d_e(p, q)$ be the Euclidean distance between points p and q.

46. LEMMA. *If s_1 is a downward (upward) maximal critical segment of Π and s_2 is a maximal critical segments of Π such that $s_{1_y} > s_{2_y}$ ($s_{1_y} < s_{2_y}$), then $s_2 \notin S$.*

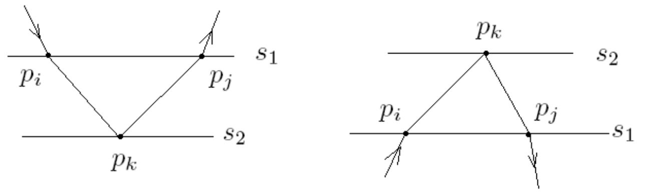


Figure 8.5: Illustration for the proof of Lemma 46. Left: *Case 1.* s_1 is a downward maximal critical segment. Right: *Case 2.* s_1 is an upward maximal critical segment.

Proof. The proof is by contradiction. Let $\{p_i, p_j\} = s_1 \cap \rho(p,q)$. Suppose that $p_k \in s_2 \cap \rho(p,q)$, then $d_e(p_i,p_j) < d_e(p_i,p_k) + d_e(p_k,p_j)$. Therefore, $\rho(p,q)$ is not a shortest path. (See Figure 8.5 for an illustration.) □

By Lemma 46 we have the following theorem:

19. THEOREM. *If S is a step set of maximal critical segments of Π with respect to the shortest path $\rho(p,q)$ then S is an alternate monotonous sequence of maximal critical segments of Π.*

This theorem and the following, previously known result are important for proving that our ESP Algorithm (described in Section 8.3.4) requires only linear computation time.

20. THEOREM. ([146], Theorem 37) *There is a deterministic linear time and linear space algorithm for the single source shortest path problem for undirected graphs with positive integer weights.*

Let T be a tree and p, q vertices of T.

6. COROLLARY. *There is a deterministic linear time and linear space algorithm to find the unique path $\rho(p,q) \subset T$.*

The following definitions will be used in Subsection 8.3.3.

Let $\Pi^\bullet$ be the (topological) closure of a simple polygon Π, $V(\Pi)$ the set of vertices of Π, and $E(\Pi)$ the set of edges of Π. For $v \in \partial\Pi$, v_x is the x-coordinates of v. Let v_1, v_2, $v_3 \in \partial\Pi$. Vertices are ordered either by a clockwise or counter-clockwise scan through $\partial\Pi$. Let $\rho(v_1, \Pi, v_2)$ be the polygonal path from v_1 to v_2, contained in $\partial\Pi$. Let $\rho(v_1, \Pi, v_2, \Pi, v_3)$ be the polygonal path from v_1 to v_2 and then to v_3 around $\partial\Pi$.

Let $u, v, w \in \partial\Pi$ such that $u_y = v_y = w_y$; consider $\rho(u, \Pi, v, \Pi, w) \subset \partial\Pi$. Let Π' be a simple polygon obtained by adding a 'closing edge' uw to $\rho(u, \Pi, v, \Pi, w)$, denoted

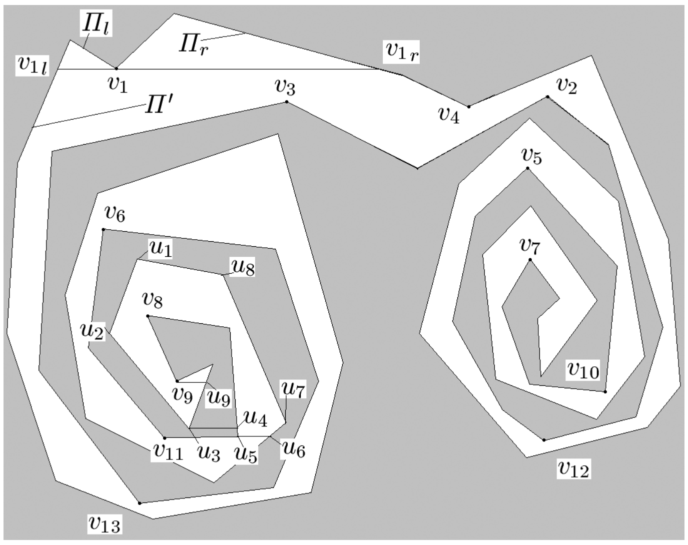


Figure 8.6: Illustration of the given definitions used in Subsection 8.3.3.

by $\Pi' = uw + \rho(u, \Pi, v, \Pi, w)$. Π' is called an *up- (down-) polygon* with respect to v if, for each $p \in \Pi'^\bullet$, $p_y \geq (\leq)\ v_y$.

For example, in Figure 8.6, Π_l and Π_r are up-polygons with respect to v_1. Π' is a down-polygon with respect to v_1.

Let Π, Π' be two simple polygons with $\Pi'^\bullet \subset \Pi^\bullet$; let $\{v_0, v_1, v_2, \ldots, v_{n-1}\}$ and $\{v_0', v_1', v_2', \ldots, v_{n-1}'\}$ be such orders of the vertices of Π and Π', respectively, that the following becomes true: For a sufficiently small $\varepsilon > 0$, we have $d_e(v_0', v_0) = \varepsilon$ for the Euclidean distance between v_0' and v_0, edge $v_i'v_{i+1}'$ is parallel to edge v_iv_{i+1}, and $v_i'v_i$ bisects the angle $\angle v_{i-1}v_iv_{i+1}$, where $i = 0, 1, 2, \ldots, n$, and the addition or subtraction of indices is taken *mod* n. In this case, Π' is called an *inner simple polygon of* Π, denoted by $\Pi(v_0, \epsilon)$. – In Figure 8.7, $u_0u_1u_2u_3u_4u_5u_0$ is an inner polygon of $v_0v_1v_2v_3v_4v_5v_0$.

Let v_{i-1}, v_i, v_{i+1} and v_{i+2} be four consecutive vertices of Π. If $(v_{i-1})_y < (v_i)_y$, $(v_i)_y = (v_{i+1})_y$ and $(v_i)_y > (v_{i+2})_y$ [or $(v_{i-1})_y > (v_i)_y$, $(v_i)_y = (v_{i+1})_y$ and $(v_i)_y < (v_{i+2})_y$], and the point $(0.5 \cdot (v_{ix} + v_{i+1x}), v_{iy} + \varepsilon)$ [or $(0.5 \cdot (v_{ix} + v_{i+1x}), v_{iy} - \varepsilon)$] is inside Π, then v_iv_{i+1} is called an *up- (down-) stable edge* of Π with respect to the xy-coordinate system used for representing Π.

If $(v_{i-1})_y < (v_i)_y$, $(v_i)_y = (v_{i+1})_y$ and $(v_i)_y > (v_{i+2})_y$, and the point$(0.5 \cdot (v_{ix} + v_{i+1x}), v_{iy} - \varepsilon)$ is in Π, then v_iv_{i+1} is called a *maximal edge* of Π with respect to the chosen xy-coordinate system.

Let v_{i-1}, v_i and v_{i+1} be three consecutive vertices of Π. If $(v_{i-1})_y < (v_i)_y$, $(v_i)_y >$

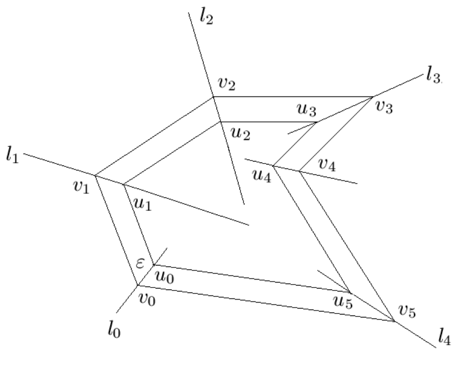


Figure 8.7: Example of an inner polygon.

$(v_{i+1})_y$ [or $(v_{i-1})_y > (v_i)_y$, $(v_i)_y < (v_{i+1})_y$], and there exist points $u_i \in V(\Pi(v_0, \varepsilon))$ [this is the set of vertices of $\Pi(v_0, \varepsilon)$] such that $\triangle v_{i-1}v_iv_{i+1}$ does not (!) contain u_i, then v_i is called an *up- (down-) stable point* of Π with respect to the xy-coordinate system used for representing Π.

In Figure 8.6, v_2, v_3, v_5, v_6, v_7 and v_8 are up-stable points. v_1, v_4, v_9, v_{10}, v_{11}, v_{12} and v_{13} are down-stable points.

If $(v_{i-1})_y < (v_i)_y$, $(v_i)_y > (v_{i+1})_y$ [or $(v_{i-1})_y > (v_i)_y$, $(v_i)_y < (v_{i+1})_y$] and there exist points $u_i \in V(\Pi(v_0, \varepsilon))$ [this is the set of vertices of $\Pi(v_0, \varepsilon)$] such that $\triangle v_{i-1}v_iv_{i+1}$ does (!) contain u_i, then v_i is called an *maximal (minimal) point* of Π with respect to the chosen xy-coordinate system.

In Figure 8.6, u_1 is a maximal point and u_3 is a minimal point. – As common, a simple polygon is called *monotonous* if it has both a unique up- and down-stable point or edge. In Figure 8.6, $u_1u_2u_3u_4u_5u_6u_7u_8u_1$ is a monotonous simple polygon.

Since the discussion of our algorithm in the case of an up- or down-stable edge is analogous to that of an up- or down-stable point, we will just detail the case of up- or down-stable points.

Let v be an up-stable point of Π. Let S_v be the set of minimal points u of Π such that there exists a polygonal path from v to u around $\partial\Pi$ and $u_y < v_y$. Let $u' \in S_v$ such that $u'_y = \max\{u_y : u \in S_v\}$. If there exists a point $w \in \partial\Pi$ such that the segment $u'w \subset \Pi^\bullet$ and $w_y = u'_y$, then $u'w$ is called a *cut edge* of Π. The polygonal path $\rho(v, \Pi, u')$ is called a *decreasing polygonal path from v to $u'w$*. v is called an up-stable point *with respect to* the cut edge $u'w$, and $u'w$ is called a cut edge *with respect to* the up-stable point v. u' is called a *closest* minimal point with respect to v around the frontier of Π.

In Figure 8.6, $v_8v_9u_9u_3$ is a decreasing polygonal path from v_8 to u_3u_4, and u_3 is a closest minimal point with respect to v_8.

If u, $w \in \partial\Pi$ such that $u_y = v_y = w_y$, $u_x < v_x < w_x$, and uv, $vw \in \Pi^\bullet$, then u and w are called the *left and right intersection points* of v, respectively. – In Figure 8.6, v_{1_l} and v_{1_r} are the left and right intersection points of v_1, respectively.

Let v be a maximal point of Π. Let S_v be the set of down-stable points u of Π such that there exists a polygonal path from v to u around $\partial\Pi$ and $u_y < v_y$. Let u' $\in S_v$ such that $u'_y = \max\{u_y : u \in S_v\}$. u' is called a *closest* down-stable point with respect to v around the frontier of Π.

In Figure 8.6, v_9 is a closest down-stable point with respect to the maximal point above the segment v_9u_9.

8.3 The Algorithms

A simplified 2D rubberband algorithm will be used in the main algorithm described in Section 8.3.4.

8.3.1 A Simplified Rubberband Algorithm

Let $p, q \in \mathbb{R}^2$, $S = \{s_1, s_2, \ldots, s_k\}$ be a set of consecutive, pairwise disjoint non-degenerate critical segments, $P = \{p_1, p_2, \ldots, p_k\}$ such that $p_i \in s_i$, where i = 1, 2, $\ldots$, k ($k \geq 3$). Let $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$ be a polygonal arc that starts at p, then visits p_1, p_2, $\ldots$, p_k, and finally ends at q.

Rubberband Algorithm

See Algorithm 3 in Section 7.5.

8.3.2 Examples

In Figure 8.8, (upper row, left) shows start and destination points p and q, and three critical segments s_1, s_2 and s_3. (Upper row, middle) shows the initial points p_1, p_2 and p_3 which are the centers of s_1, s_2 and s_3, respectively. (Upper row, right), (lower row, left) and (lower row, middle) show updated points p_1 (by step 4.3), p_2 (by step 4.3) and p_3 (by step 5.3), respectively. (Lower row, right) shows the updated p_1.

In Figure 8.9, (left) shows the initial path ρ_0, and the updated paths ρ_1 to rho_4 of four iterations of the Rubberband Algorithm. (Right) shows the initial path ρ_0, and the updated paths ρ_1 to ρ_{18} of eighteen iterations of this algorithm.

We can see that different initial points may lead to different numbers of iterations of the algorithm until it terminates (with respect to the chosen accuracy parameter).

From Figure 8.8, we can see that the algorithm only needs two iterations to terminate. The results of the first iteration are shown in (upper row, right), (lower row, left) and (lower row, middle). (Lower row, right) shows the result of the second iteration. Figure 8.9 (left) shows that the algorithm has to run for at least 4 iterations in this case.

8.3.3 Decomposing a Simple Polygon into Trapezoids

This subsection describes the decomposition algorithm and its three subroutines, which are Procedures 10– 12. Procedure 10 is used to update the left or right intersection points of up-stable points. It will be applied by Procedure 11 to remove up-stable points. Procedure 12 applies Procedures 10 and 11 to compute the left and right intersection points of all up-stable points, from bottom to top. This procedure will be called in Step 3 of the main algorithm which processes both up- and down-stable points, from top to bottom.

In Procedure 10, let Π be a simple polygon such that $V(\Pi)$ does not have any up-stable point.

10. PROCEDURE.

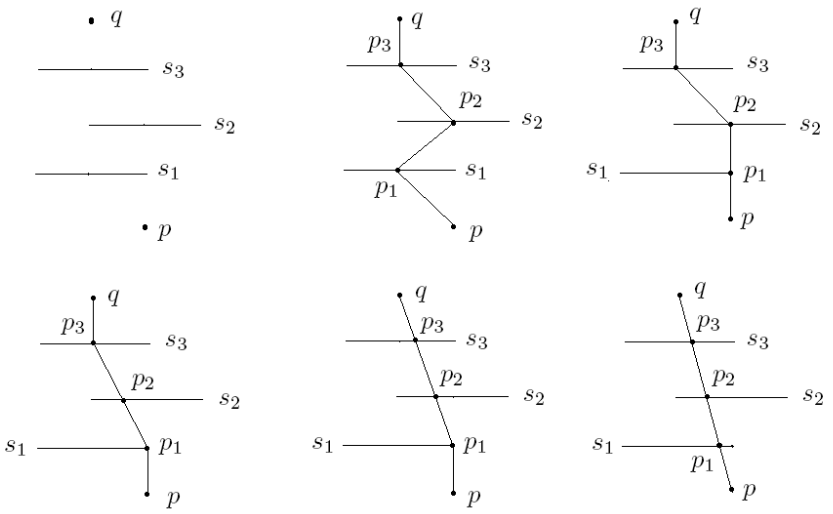


Figure 8.8: Illustration for the Rubberband Algorithm when the initial points are selected as centers of critical segments.

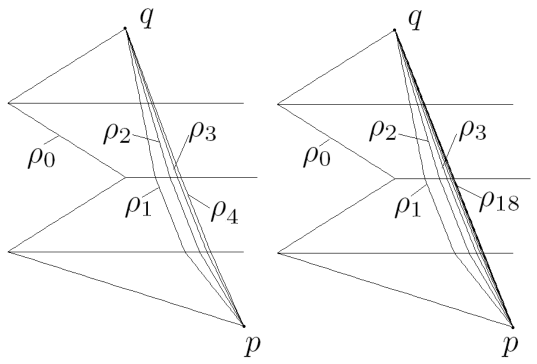


Figure 8.9: Illustration for the Rubberband Algorithm when the initial points are the left end points of critical segments.

This procedure updates left or right intersection points.

1. Decompose Π into a set of trapezoids, denoted by T_Π (note: straightforward, because there is no up-stable point).

2. For each edge $e \in E(\Pi)$.

2.1. If e is a cut edge, then do the following:

2.1.1. Let v_e be the up-stable point with respect to e.

2.1.2. Find a trapezoid $t \in T_\Pi$ such that the bottom edge of t and edge e have a common end point in $V(\Pi)$.

2.1.3. Let ρ_e be the decreasing polygonal path from v_e to e.

2.1.4. Find a stack of trapezoids, denoted by T_e ($\subseteq T_\Pi$), such that for each $t \in T_e$, $t^\bullet \cap \rho_e \neq \emptyset$.

2.1.5. Let t_e be the topmost trapezoid in T_e.

2.1.6. Compute the intersection points between $y = v_{ey}$ with those two edges on the left and right side of t_e.

2.1.7. Update the left and right intersection points of v_e by comparing the results of Step 2.1.6 with the initial left and right intersection points of v_e.

Let I be an interval of real numbers, and M_I be a subset of maximal points of Π such that for each element $v \in M_I$, $v_y \in I$. Suppose M_I is sorted according to y-coordinate decreasingly.

11. PROCEDURE.

This procedure removes up stable points.

1. Let $M_I = \{v_1, v_2, \ldots, v_k\}$.

2. For each $i \in \{1, 2, \ldots, k\}$, do the following:

2.1. Find a closest down-stable point with respect to v_i around the frontier of Π, denoted by u_i.

2.2. Find a point $w_i \in \partial\Pi$ such that $\rho(u_i, \Pi, v_i, \Pi, w_i)$ is the shortest polygonal path in $\partial\Pi$ such that $u_{iy} = w_{iy}$.

2.3. Update Π by replacing $\rho(u_i, \Pi, v_i, \Pi, w_i)$ by the edge u_iw_i.

2.4. Let $\Pi_i = u_iw_i + \rho(u_i, \Pi, v_i, \Pi, w_i)$.

2.5. Now let Π_i be the input of Procedure 10 and update the left and right intersection points of all possible up-stable points.

12. PROCEDURE.

This procedure compute all left or right intersection points.

1. Let $U = \{v_1, v_2, \ldots, v_k\}$ be the sorted set of up-stable points of Π such that $v_{1y} < v_{2y} < \cdots < v_{ky}$.

2. For each $i \in \{1, 2, \ldots, k\}$, do the following:

2.1. Find a closest minimal point with respect to v_i by following the frontier of Π, denoted by u_i.

2.2. Let $I = [a, b]$ where $a = u_{iy}$ and $b = v_{iy}$.

2.3. Compute M_I.

2.4. If $M_I \neq \emptyset$, then apply Procedure 11 and go to Step 2.1.

2.5. Otherwise, find a point w_i such that $\rho(u_i, \Pi, v_i, \Pi, w_i)$ is the shortest polygonal path of $\partial\Pi$ with $u_{iy} = w_{iy}$.

2.6. Set initial left and right intersection points of v_i as follows:

2.6.1. If $v_{i-1_y} = v_{iy} = v_{i+1_y}$ then let the initial left and right intersection points of v_i be v_{i-1} and v_{i+1}, respectively.

2.6.2. Otherwise, find two points w_{il}, w_{ir} such that $\rho(w_{il}, \Pi, u_i, \Pi, v_i, \Pi, w_{ir})$ is the shortest polygonal path of $\partial\Pi$ with $w_{il_y} = v_{iy} = w_{iry}$.

2.6.3. Let the initial left and right intersection points of v_i be w_{il} and w_{ir}, respectively.

2.7. Update Π by replacing $\rho(u_i, \Pi, v_i, \Pi, w_i)$ by the edge u_iw_i.

2.8. Now let Π be the input for Procedure 10; update the left and right intersection points of all possible up stable points.

Decomposition Algorithm

1. Let $S = \emptyset$ and $T = \emptyset$.

2. Compute the set of up- and down-stable points, denoted by V.

3. Apply Procedure 12 to compute the left and right intersection points, for all up-stable points in V.

4. Sort V for decreasing y-coordinates.

5. For each $i \in \{1, 2, \ldots, k\}$, with $k = |V|$, do the following:

5.1. *Case 1.* v_i is a down-stable point.

5.1.1. Find two points v_{il}, $v_{ir} \in \partial\Pi$ such that $v_{il_y} = v_{iy} = v_{ir_y}$ and $v_{il_x} < v_{ix} < v_{ir_x}$.

5.1.2a. Let $\Pi_l = v_{il}v_i + \rho(v_{il}, \Pi, v_i)$ (up-polygon).

5.1.2b. Let $\Pi_r = v_iv_{ir} + \rho(v_i, \Pi, v_{ir})$ (up-polygon).

5.1.2c. Let $\Pi' = v_{il}v_{ir} + \rho(v_{il}, \Pi, v_{ir})$ (down-polygon).

5.1.3. Let $S = S \cup \{\Pi_l, \Pi_r\}$.

5.1.4. Let $\Pi = \Pi'$.

5.1.5. Let $i = i + 1$ and go to Step 5.

5.2. *Case 2.* v_i is an up-stable point.

5.2.1a. Let v_{il}, v_{ir} be the left and right intersection points of v_i, respectively.

5.2.2b. Let $\Pi_l = v_{il}v_i + \rho(v_{il}, \Pi, v_i)$ (down-polygon).

5.2.2c. Let $\Pi_r = v_iv_{ir} + \rho(v_i, \Pi, v_{ir})$ (down-polygon).

5.2.2d. Let $\Pi' = v_{il}v_{ir} + \rho(v_{il}, \Pi, v_{ir})$ (up-polygon).

5.2.3. Let $S = S \cup \{\Pi'\}$.

5.2.4. Let $\Pi = \{\Pi_l, \Pi_r\}$.

5.2.5. Let $i = i + 1$ and go to Step 5.

6. For each $j \in \{1, 2, \ldots, n\}$, with $n = |S|$, do the following:

6.1. For each (monotonous) polygon $\Pi_j \in S$, decompose it into a stack of trapezoids, denoted by T_j.

6.2. Let $T = T \cup T_j$.

7. Output T.

Figure 8.10 shows an example output of this decomposition algorithm.

Time Complexity of Decomposition Algorithm

47. LEMMA. *The set of up- (or down-, maximal) stable points of* Π *can be computed in* $\mathcal{O}(n)$*, where* $n = |V(\Pi)|$.

Proof. For a sufficiently small number $\epsilon > 0$, the "start" vertex v'_0 of an inner polygon $\Pi(v_0, \epsilon)$ can be computed in $\mathcal{O}(n)$, where $n = |V(\Pi)|$ (see [131]). For three consecutive vertices u, v, $w \in V(\Pi)$ such that $u_x < v_x < w_x$, if $u_y < v_y$, $v_y > w_y$ and $v'_y > v_y$, then v is a up stable point (if $u_y < v_y$, $v_y > w_y$ and $v'_y < v_y$, it follows that v is a maximal point; if $u_y > v_y$, $v_y < w_y$ and $v'_y < v_y$, then v is a down-stable point). □

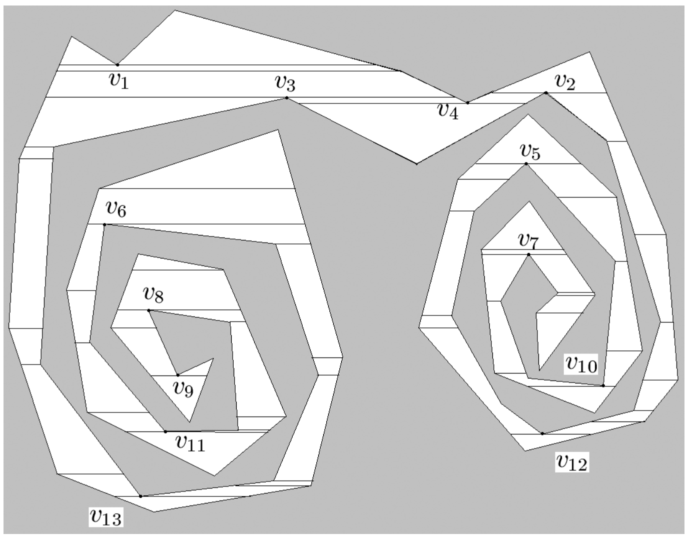


Figure 8.10: Output of the decomposition algorithm for the polygon of Figure 8.6.

48. LEMMA. *Procedure 10 can be computed in $\mathcal{O}(n\ log\ n)$, where $n = |V(\Pi)|$.*

Proof. If Π is monotonous, then (obviously) it can be decomposed into a stack of trapezoids in $\mathcal{O}(|V(\Pi)|)$. Otherwise, by assumption, Π can only have a finite number of down-stable points. Analogous to Step 5.1 in the decomposition algorithm, Π can be decomposed into a set of trapezoids in $\mathcal{O}(|V(\Pi)|)$. Thus, Step 1 can be computed in $\mathcal{O}(|V(\Pi)|)$.

All steps following Step 2, except Step 2.1.4, can be computed in $\mathcal{O}(1)$.

Step 2.1.4 can be computed in $\mathcal{O}(nlogn)$, where $n = |V(\Pi)|$:

Let S_d be the set of all down-stable points of Π.

A1. Sort S_d according to y-coordinates; we obtain S_d = $\{v_1, v_2, \ldots, v_k\}$ such that $v_{1y} \leq v_{2y} \leq v_{3y} \leq \ldots \leq v_{ky}$.

A2. Let m = $\min\{|v_{i-1_y} - v_{iy}| : |v_{i-1_y} - v_{iy}| > 0, i = 2, 3, \ldots, k\}$, and M = $\max\{|v_{i-1_x} - v_{ix}| : |v_{i-1_x} - v_{ix}| > 0, i = 2, 3, \ldots, k\}$.

A3. Transform Π by rotating it by angle θ anticlockwise about the origin (i.e., for each point $p = (x, y) \in \Pi^{\bullet}$, update it by point (x', y'), where $x' = x\cos\theta - y\sin\theta$, $y' = x\sin\theta + y\cos\theta$).

Without loss of generality, assume that m = $|v_{1y} - v_{2y}| > 0$. We have that

$$|v'_{1y} - v'_{2y}| = |(v_{1x} - v_{2x})\sin\theta + (v_{1y} - v_{2y})\cos\theta|$$

$$\geq |(v_{1y} - v_{2y})\cos\theta| - |(v_{1x} - v_{2x})\sin\theta|$$
$$\geq m\cos\theta - M\sin\theta \to m(\theta \to 0)$$

Therefore, there exists a sufficiently small angle $\theta > 0$ such that, after rotating, each down-stable point is still a down-stable point, and all down-stable points have unique y-coordinates. This implies that, for each trapezoid $t \in T_\Pi$, there exist at most two trapezoids t_1, $t_2 \in T_\Pi$ such that t_1 and t_2 has a common vertex with t, respectively. (In this case, t_1 and t_2 have a common vertex which is a down-stable point. Since Step A2 can be computed in $\mathcal{O}(|S_d|)$ and Step A3 can be computed in $\mathcal{O}(1)$, it follows that Step 2.1.4 can be computed in $\mathcal{O}(k)$, where k is the number of vertices of the decreasing polygonal path from v_e to e, after sorting (Step A1) in $\mathcal{O}(|S_d|log|S_d|)$. □

49. LEMMA. *Procedure 11 can be computed in $\mathcal{O}(n\ log\ n)$ time, where n is the number of vertices of the original simple polygon Π.*

Proof. By Lemma 47, Step 1 can be computed in $\mathcal{O}(n\ log\ n)$, where n is the number of vertices of the original simple polygon Π. Step 2.1 can be computed in $\mathcal{O}(n_u)$, where n_u is the number of vertices of $\rho(u_i, \Pi, v_i)$. Step 2.2 can be computed in $\mathcal{O}(n_w)$, where n_w is the number of vertices of $\rho(v_i, \Pi, w_i)$. Steps 2.3 and 2.4 can be computed in $\mathcal{O}(1)$. By Lemma 48, Step 2.5 can be computed in $\mathcal{O}(n_i\ log\ n_i)$, where $n_i = |V(\Pi_i)|$. Thus, Step 2 can be computed in $\mathcal{O}(n\ log\ n)$ altogether, where n is the number of vertices of Π. □

50. LEMMA. *Procedure 12 can be computed in $\mathcal{O}(n\ log\ n)$ time, where n is the number of vertices of the original simple polygon Π.*

Proof. By Lemma 47, Step 1 can be computed in $\mathcal{O}(n\ log\ n)$, where n is the number of vertices of the original simple polygon Π. Step 2.1 can be computed in $\mathcal{O}(n_u)$, where n_u is the number of vertices of $\rho(u_i, \Pi, v_i)$. Step 2.2 can be computed in $\mathcal{O}(1)$. Step 2.3 can be computed in $\mathcal{O}(|M_I|)$. By Lemma 50, Step 2.4 can be computed in $\mathcal{O}(n_u\ log\ n_u)$, where $n_u = |V(\rho(u_i, \Pi, v_i))|$. Step 2.5 can be computed in $\mathcal{O}(n_w)$, where n_w is the number of vertices of $\rho(u_i, \Pi, w_i)$. Step 2.6.1 can be computed in $\mathcal{O}(1)$. Step 2.6.2 can be computed in $\mathcal{O}(n_i)$, where $n_i = |V(\rho(u_i, \Pi, w_{il}))| + |V(\rho(w_i, \Pi, w_{ir}))|$. Step 2.6.3 can be computed in $\mathcal{O}(1)$. Step 2.7 can be computed in $\mathcal{O}(1)$. By Lemma 48, Step 2.8 can be computed in $\mathcal{O}(n\ log\ n)$, where n is the number of the vertices of the updated Π. Therefore, Procedure 3 can be computed in $\mathcal{O}(n\ log\ n)$, where n is the number of the vertices of the original simple polygon Π. □

21. THEOREM. *The given decomposition algorithm has time complexity $\mathcal{O}(n\ log\ n)$, where n is the number of vertices of the original simple polygon Π.*

Proof. Step 1 can be computed in $\mathcal{O}(1)$. By Lemma 47, Step 2 can be computed in $\mathcal{O}(n \ log \ n)$, where n is the number of vertices of the original simple polygon Π. By Lemma 50, Step 3 can be computed in $\mathcal{O}(n \ log \ n)$, where n is the number of the vertices of the original simple polygon Π. Step 4 can be computed in $\mathcal{O}(|V| \ log \ |V|)$. Step 5.1.1 can be computed in $\mathcal{O}(n_i)$, where $n_i = |V(\rho(v_{il}, \Pi, v_i))| + |V(\rho(v_i, \Pi, v_{ir}))|$. Steps 5.1.2a – 5.1.5 can be computed in $\mathcal{O}(1)$. Thus, Step 5.1 can be computed in $\mathcal{O}(n_i)$, where $n_i = |V(\rho(v_{il}, \Pi, v_i))| + |V(\rho(v_i, \Pi, v_{ir}))|$. Step 5.2 can be computed in $\mathcal{O}(1)$. By Lemma 49, Step 6.1 can be computed in $\mathcal{O}(|\Pi_j|)$. Thus, Step 6 can be computed in $\mathcal{O}(n)$, where n is the number of the vertices of the original simple polygon Π. Step 7 can be computed in $\mathcal{O}(1)$. Therefore, the decomposition algorithm can be computed in $\mathcal{O}(n \ log \ n)$, where n is the number of the vertices of Π. □

8.3.4 New Algorithm

Let p, q be the start and destination point inside of a simple polygon Π, respectively. Let V be the set of vertices of Π. Let E be the set of edges of Π.

2D ESP Algorithm

1. Apply the decomposition algorithm to compute the set of maximal critical segments of Π, denoted by S.

2. Construct a 1-tree T.

3. Apply the algorithm of [146] to find the unique path $\rho(p, q) \subset T$.

4. Compute the step set of maximal critical segments of Π with respect to the shortest path $\rho(p, q)$, denoted by S_{step} (see description before Definition 37).

5. Let $P = \{p\}$.

6. For each $s_i \in S_{step}$,

6.1 let v_i be the center point of s_i;

6.2 let $P = P \cup \{v_i\}$;

6.3 let $P = P \cup \{q\}$.

7. Apply the Rubberband Algorithm on S_{step} and P to compute the shortest path $\rho(p, q)$ inside of Π.

8. We finally convert $\rho(p, q)$ into the standard form of the shortest path by deleting all vertices which are not vertices of Π.[1]

We show that Step 8 is always correct. First, let p be a point in the set of vertices of the shortest path $\rho(p, q)$ obtained by Step 7. p must be deleted even if it is close to some vertex of Π. To see this, note that each vertex of the polygon is an endpoint of

[1]It is well known that each vertex ($\neq p$, q) of the shortest path is a vertex of Π ([91]).

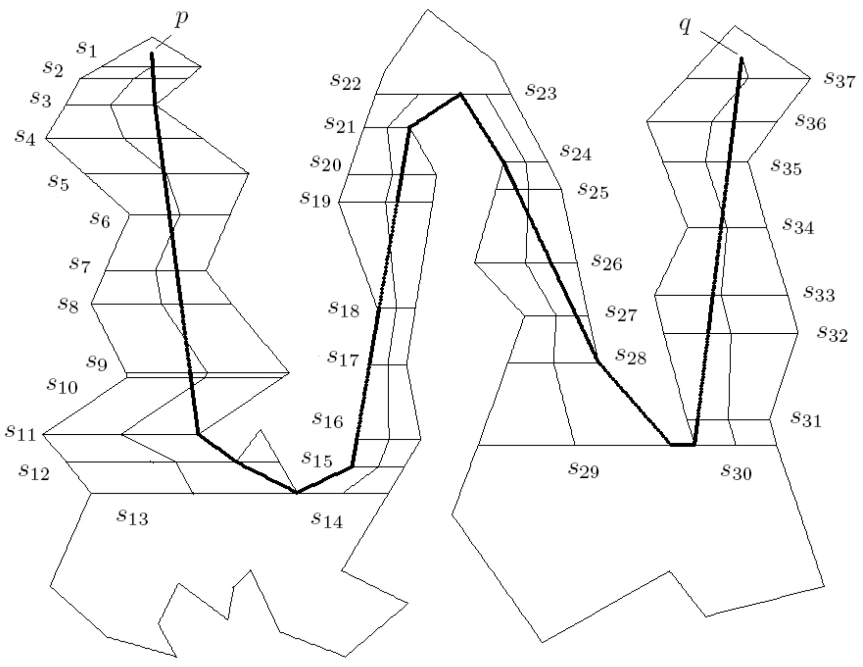


Figure 8.11: The shortest path from p to q inside the simple polygon shown in Figure 8.3.

a critical segment. When we update a point on a critical segment, we search for the new position on the whole segment including its two endpoints. Any really "good" endpoint will be selected quickly. This is illustrated by the example output in Tables 8.2 and 8.3. For a point on the shortest path, if it is really "good" then it must be exactly a vertex of the polygon, not just "close" to some vertex. Secondly, let p_j, p_{j+1}, p_{j+2} be three consecutive points in the set of vertices of the shortest path $\rho(p,q)$ obtained by Step 7. If p_{j+1} must be deleted, then p_jp_{j+2} must be contained inside Π. This is because, if p_{j+1} is not a vertex of the polygon, then p_j, p_{j+1} and p_{j+2} must be colinear. Otherwise, there is a point p'_{j+1}in a sufficiently small neighborhood of p_{j+1} such that p'_{j+1} is contained in the polygon and

$$d_e(p_j, p'_{j+1}) + d_e(p'_{j+1}, p_{j+2}) < d_e(p_j, p_{j+1}) + d_e(p_{j+1}, p_{j+2})$$

(This contradicts that $p_jp_{j+1}p_{j+2}$ is the shortest subpath of the shortest path). Since p_j, p_{j+1}, p_{j+2} are colinear, so p_jp_{j+2} is contained in the polygon. Therefore, if a point in the set of vertices of $\rho(p,q)$ is not a vertex of the polygon, then it must be

i	(x_i, y_i)	i	(x_i, y_i)	i	(x_i, y_i)	i	(x_i, y_i)
1	(281.0, 734.0)	11	(342.0, 284.0)	21	(625.0, 659.0)	31	(1010.1, 302.0)
2	(281.9, 719.0)	12	(392.0, 250.0)	22	(693.0, 700.0)	32	(1024.2, 408.0)
3	(284.0, 687.0)	13	(474.0, 212.0)	23	(693.0, 700.0)	33	(1030.4, 454.0)
4	(289.9, 646.0)	14	(474.0, 212.0)	24	(750.0, 617.0)	34	(1041.4, 537.0)
5	(296.1, 603.0)	15	(548.0, 244.0)	25	(767.1, 584.0)	35	(1052.2, 618.0)
6	(303.3, 553.0)	16	(554.3, 278.0)	26	(813.8, 494.0)	36	(1058.7, 667.0)
7	(313.2, 484.0)	17	(571.2, 369.0)	27	(847.5, 429.0)	37	(1065.8, 720.0)
8	(319.0, 444.0)	18	(584.2, 439.0)	28	(877.0, 372.0)		
9	(331.2, 359.0)	19	(608.1, 568.0)	29	(974.0, 271.0)		
10	(331.9, 354.0)	20	(614.4, 602.0)	30	(1006.0, 271.0)		

Table 8.2: Vertices (not including p and q) of the shortest path $\rho(p, q)$ obtained by Step 7 in the ESP Algorithm, for the example shown in Figure 8.3.

redundant and must be deleted.

Figure 8.11 shows the initial path and the shortest path (inside the simple polygon shown in Figure 8.3) computed by the ESP Algorithm. Table 8.4 illustrates (for the same input example) the relationship between number of iterations and length differences, for the last two updated paths. Initial points are the center points of the segments.

8.3.5 Time Complexity of the ESP Algorithm

By Theorem 21, Step 1 can be computed in $\mathcal{O}(n\ log\ n)$. Based on Step 1, Step 2 can be computed in $\mathcal{O}(n)$. By [146], Step 2 can be computed in $\mathcal{O}(n)$. Based on Step 3, Step 4 can be computed in $\mathcal{O}(n)$. Step 5 can be computed in $\mathcal{O}(1)$. Step 6 can be

i	(x_i, y_i)	i	(x_i, y_i)	i	(x_i, y_i)	i	(x_i, y_i)
3	(284, 687)	14	(474, 212)	23	(693, 700)	30	(1006, 271)
11	(342, 284)	15	(548, 244)	24	(750, 617)		
12	(392, 250)	21	(625, 659)	28	(877, 372)		
13	(474, 212)	22	(693, 700)	29	(974, 271)		

Table 8.3: Vertices (not including p and q) of the standard form of the shortest path $\rho(p, q)$ obtained by Step 8 in the ESP Algorithm, for the example shown in Figure 8.3.

I	δ	I	δ	I	δ	I	δ
1	90.8685	6	0.3787	11	0.0170	16	0.0009
2	34.1894	7	0.2328	12	0.0078	17	0.0006
3	6.9828	8	0.1547	13	0.0043	18	0.0004
4	2.3697	9	0.0992	14	0.0025	19	0.0003
5	0.8061	10	0.0384	15	0.0015	20	0.0002

Table 8.4: Number of iterations (I) and resulting δ, for the example shown in Figure 8.3.

computed in $\mathcal{O}(n)$. By Lemma 36, Step 7 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$. Step 8 can be computed in $\mathcal{O}(n)$. Therefore, the total time complexity of the ESP Algorithm is $\kappa(\varepsilon) \cdot \mathcal{O}(n\ log\ n)$.

8.4 Conclusion

We have described a simple $\kappa(\varepsilon) \cdot \mathcal{O}(n\ log\ n)$ algorithm for computing a shortest path between p and q, which is contained in a simple polygon Π, where $\kappa(\varepsilon)$ is as in Equation (1.1), and n is the number of vertices of Π. The maximum number of iterations is a constant defined by the selected accuracy parameter. The algorithm is easy to implement.

Obviously, our method can also be generalized to deal with special cases of 3D Euclidean shortest paths. However, this short chapter is only an initial illustration how versions of a rubberband algorithm apply to shortest path problems.

Chapter 9

ESPs on Surfaces of Convex Polytopes

This chapter presents a simple $\kappa(\varepsilon) \cdot \mathcal{O}(kn\ log\ n)$ algorithm to compute a restricted solution for the surface ESP problem, considering the surface of a convex polytope Π (i.e., Π is the convex hull of a finite number of 3D points), where k is the number of critical polygons (see Definition 38 below) between source and target point, and n is the number of edges of Π. The best algorithm known so far for solving this problem is of time complexity $\mathcal{O}(n\ log^2 n)$, when there are $\mathcal{O}(n)$ vertices and edges on Π.

9.1 Introduction and Related Work

Let Π be a connected polyhedral domain such that its frontier is a union of a finite number of triangles. An *obstacle* is a connected polyhedral component in the complement $\mathbb{R}^3 \backslash \Pi$ of Π. Let $p, q \in \Pi$ such that $p \neq q$. The general Euclidean shortest-path problem (ESP) asks to find the shortest polygonal path $\rho(p, q)$ which is either completely contained in Π, or just not intersecting with interior points of a finite number of given obstacles. The problem is a special case of the problem of planning optimal collision-free paths for a robot system. As a first result, [125] presented in 1984 a doubly exponential time algorithm for solving the general obstacle avoidance problem. [121] improved this by providing a singly exponential time algorithm. The result was further improved by a PSPACE algorithm in [28].

Since the general ESP problem is known to be NP-hard [27], special cases of the problem have been studied afterwards. [126] gave a polynomial time algorithm for ESP calculations for cases where all obstacles are convex and the number of obstacles is small. [58] solved the ESP problem with an $\mathcal{O}(n^{6k-1})$ algorithm assuming that all obstacles are vertical buildings with k different heights.

[125] is the first publication considering the special case that the shortest polygonal path $\rho(p, q)$ is constrained to stay on the surface of Π. [125] presented an $\mathcal{O}(n^3 logn)$ algorithm where Π was assumed to be convex. [104] improved this result by providing an $\mathcal{O}(n^2 logn)$ algorithm for the surface of any Π. The time complexity was even reduced to $\mathcal{O}(n^2)$ [37].

So far, the best known result for the surface ESP problem is due to [79]; this paper improved in 1999 the time complexity to $\mathcal{O}(n\ log^2 n)$, assuming that there are $\mathcal{O}(n)$ vertices and edges on Π.

Let Π be a convex polytope. Let S be a set of edges sequences (called *step set*) corresponding (by incidence of vertices of the path) to a shortest path on the surface of Π. [110] shows that the cardinality $|S|$ can be calculated in $\mathcal{O}(n^4)$, where n is the number of vertices of Π. [110] also constructs an example such that the lower bound of $|S|$ is n^4.

[1] gives an $\mathcal{O}(n^6 \beta(n)\ logn)$ algorithm to compute S, where $\beta(n)$ is an extremely slowly growing function. [124] proves that the lower bound of the number of maximal edge sequences of shortest paths is n^3 by using the notion *star unfolding* [9]. [42] solves the two-point queries (i.e., "given two points p and q on the surface, find the shortest path from p to q) ESP problem on a (not necessarily convex) polyhedral surface in $\mathcal{O}(log\ n)$ but with higher complexities of preprocessing and space than for the convex case. [18] focuses on terrain surfaces with various optimal path problems.

There are also already a few approximation algorithms for solving the surface ESP problem. Let n be the number of edges of a convex polytope. [2] presents an algorithm for computing an $(1+\varepsilon)$-shortest path on the surface of a convex polytope in time $\mathcal{O}(n + 1/\varepsilon^3)$. [69] improves this result by an $\mathcal{O}((log\ n)/\varepsilon^{1.5} + 1/\varepsilon^3)$ algorithm with a preprocessing time of $\mathcal{O}(n)$. [3, 4, 88, 100] also discuss approximation algorithms for weighted surface ESP problems.

For calculating an ESP on the surface of a convex polytope (in $\mathbb{R}^3$), [106] states on page 667 the following *open problem*:

> Can one compute shortest paths on the surface of a convex polytope in $\mathbb{R}^3$ in subquadratic time? In $\mathcal{O}(n\ log\ n)$?

In this chapter, we apply the rubberband algorithm to find an approximate and restricted ESP in $\kappa(\varepsilon) \cdot \mathcal{O}(n\ log\ n)$ by the algorithm described in Section 9.3.2, where n is the number of vertices of the convex polytope.

In the rest of this chapter, Section 9.2 provides necessary definitions and theorems. Section 9.3 presents our new ESP algorithm. Section 9.4 analyses the time complexity of this algorithm. Section 9.5 concludes the chapter.

9.2 Basics

This section defines a few terms and an important convex function as used later in this chapter.

We denote by $\Pi = \langle v_1, v_2, \ldots, v_n \rangle$ a convex polytope (i.e., a compact convex polyhedral region) in the 3D Euclidean space. $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices of Π. Let $S = \{\triangle_1, \triangle_2, \ldots, \triangle_m\}$ be the set of faces of $\vartheta\Pi$. Let $E = \{e_1, e_2, \ldots, e_{m'}\}$ be the set of edges of the faces of $\vartheta\Pi$. For $v \in V$, let f_v be that plane which contains v and is parallel to the xoy-plane. Let $P_v = f_v \cap \vartheta\Pi$.

38. DEFINITION. *We say that P_v is the* critical polygon *(of* Π*) with respect to v.*

If $u_z = v_z$ and u_z is a vertex of P_v then $P_u = P_v$. Two critical polygons P_1 and P_2 are called *adjacent* iff there are two vertices $v_i \in P_i$, for $i = 1, 2$, such that $v_1v_2 \in E$. Let u, $v \in V$ such that $u_z < v_z$, and the critical polygons P_u and P_v are adjacent. Let

$$S_u = \{w : w \in \partial P_u \wedge uw \subset \text{ some edge } e \in E\}$$

$$S_v = \{w : w \in \partial P_v \wedge vw \subset \text{ some edge } e \in E\}$$

39. DEFINITION. *S_u (S_v) is called the* downward (upward) visible polyline *of vertex v (u).*

P is called a *sequence* of critical polygons of Π iff P_i is adjacent to P_{i+1}, for each $i \in \{1, 2, \ldots, k-1\}$. Let p, $q \in \vartheta\Pi$ such that $p_z < q_z$. Let $\{P_1, P_2, \ldots, P_k\}$ be a sequence of critical polygons of Π such that $p_z < P_{1z} < P_{iz} < P_{i+1z} < P_{kz} < q_z$, where i = 2, 3, ..., k - 1. We construct a path P as follows: Let $P = [V, E]$, where $V = \{P_1, P_2, \ldots, P_k\}$ and $E = \{P_iP_j : P_i$ and P_j are adjacent $\}$. We say that P is the *step path* of Π from p to q (with respect to the given Cartesian coordinate system).

9.3 The Algorithms

At first we present a simplified 3D rubberband algorithm, which is later used within our main algorithm in Section 9.3.2.

9.3.1 A Simplified Rubberband Algorithm

Let p (q) be the start (destination) point on the surface of a convex polytope Π.

13. PROCEDURE.

Input: E, V, and a vertex $u \in V$.
Output: the set of vertices of the downward (upward) visible polylines of u.

```
1. Let $v = \max\{w : w_z < u_z \wedge w \in V\}$.
2. Let $V_u = \{w : \exists e\,(e \in E \wedge w = e \cap [\text{plane } z = v_z])\}$.
```

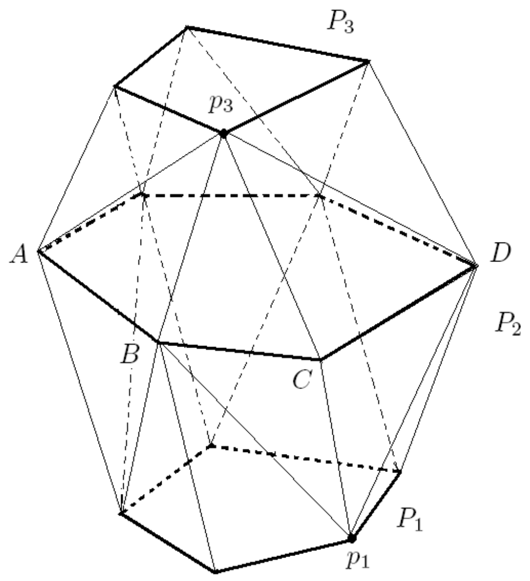


Figure 9.1: Illustration for the output of Procedure 14: $P_2^* = \rho(B, C, D)$, because $\rho(A, B, C, D)$ is the polyline of maximal length which can be seen by p_3, and $\rho(B, C, D)$ is the polyline of maximal length which can be seen by p_1.

3. Apply Graham's Scan algorithm (see, e.g., Algorithm 1.1 in [85], page 25) to compute the set of vertices of the downward visible polyline of u.

4. Analogously, compute the set of vertices of the upward visible polyline of u.

This simple procedure is applied in Case 3 of the following procedure.

14. PROCEDURE.

Input: Three consecutive critical polygons P_1, P_2 and P_3 (i.e., $\{P_1, P_2, P_3\}$ is a sequence of critical polygons) of a convex polytope Π, and two points $p_i \in \partial P_i$, where $i = 1, 3$.

Output: The set of vertices of a maximal polyline $P_2^* \subset P_2$ such that for every point $p \in \partial P_2^*$, there is a $\triangle_i \in S$ such that the segment $p_i p \subset \partial \triangle_i$, where $i = 1, 3$. In other words, P_2^* is the polyline of maximal length between p_1 and p_3 (with respect to the z-coordinate) such that each vertex of it is visible both from p_1 and p_3 (see Figure 9.1).

1. Let $P_i' = \emptyset$, where i= 1, 3 (Note: the final P_1' will be the downward visible polyline of p_1, and the final P_3' will be the upward visible polyline of p_3).

2. Update P_1' as follows:

Case 1. p_1 is not a vertex of P_1.

Find the unique edge e in P_2 such that p_1 and e are contained in a face of Π. Let $P_1' = \{e\}$.

Case 2. p_1 is a vertex of P_1 but not a vertex of Π.

Find the two edges e_1, e_2 in P_2 such that p_1 and e_i are contained in a face of Π, where i = 1, 2.
Let $P_1' = \{e_1, e_2\}$.

Case 3. p_1 is a vertex of Π.

Use E, V, and p_1 as input for Procedure 13; use the result for updating $V(P_1')$.

3. Update $V(P_3')$ analogously.

4. Let $V(P_j') = \{w_{j_1}, w_{j_2}, \ldots, w_{j_{k_j}}\}$, where j = 1, 3.

5. Let $2_1 = \max\{1_1, 3_1\}$ and $2_{k_2} = \min\{1_{k_1}, 3_{k_1}\}$.

6. Output $V(P_2^*) = \{w_{2_1}, w_{2_2}, \ldots, w_{2_{k_2}}\}$.

This procedure is frequently applied in the following main algorithm; it is a simplified version of a rubberband algorithm and is used in Step 9 of the main algorithm.

Let $p, q \in \mathbb{R}^3$, $P = \{P_1, P_2, \ldots, P_k\}$ be a sequence of (pairwise disjoint) critical polygons, and $P' = \{p_1, p_2, \ldots, p_k\}$ such that $p_i \in \partial P_i$, for $i = 1, 2, \ldots, k$ and $k \geq 3$. Let $\rho = (p, p_1, p_2, \ldots, p_k, q)$ be a polygonal arc (i.e., simple polyline) that starts at p, then visits p_1, p_2, ..., p_k, and finally ends at q.

4. ALGORITHM.

```
1. Let $\varepsilon = 10^{-10}$ (the accuracy).
2. Compute the length $l_1$ of the path $\rho = (p, p_1, p_2, \ldots, p_k, q)$.
3. Let $q_1 = p$ and $i$ = 1.
4. While $i < k - 1$ do
4.1. Let $q_3 = p_{i+1}$.
4.1a. Apply Procedure 14 to compute a polyline $P_i^* \subset P_i$.
```

```
4.2. Compute a point $q_2 \in \partial P_i^*$ such that
```

$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \partial P_i^*\}$$

```
4.3. Update $P$ by replacing $p_i$ by $q_2$.
4.4. Let $q_1 = p_i$ and $i = i + 1$.
5.1. $q_3 = q$.
5.1a. Apply Procedure 14 to compute a polyline $P_k^* \subset P_k$.
5.2. Compute $q_2 \in \partial P_k$ such that
```

$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \partial P_k^*\}$$

```
5.3. Update $P$ by replacing $p_k$ by $q_2$.
6. Compute the length $l_2$ of the updated path $\rho = (p, p_1, p_2, \ldots, p_k, q)$.
7. Let $\delta = l_1 - l_2$.
8. If $\delta > \epsilon$, then let $l_1 = l_2$ and go to Step 3. Otherwise, Stop.
```

The accuracy parameter in Step 1 can be chosen such that maximum possible numerical accuracy is guaranteed on a given computer.

We note that the point q_2, computed in Step 4.2 in Algorithm 4, may not be unique. It is this non-uniqueness that leads our Algorithm 5 in Section 9.3.2 to produce a restricted ESP.

9.3.2 Surface ESP Algorithm

Finally, this subsection presents the main algorithm of this chapter. Let p, $q \in \vartheta\Pi$ such that $p_z < q_z$.

5. ALGORITHM.

```
1. Compute $V_1 = \{v : p_z < v_z < q_z \wedge v \in V\}$.
2. Sort $V_1$ according to the $z$-coordinates. As a result, we have
```

$$V_1 = \{u_1, u_2, \ldots, u_{k'}\}$$

with

$$u_{1z} \leq u_{2z} \leq \ldots \leq u_{k'z}$$

```
3. Construct a 1-weighted graph $G = [V, E]$ (i.e., such that each $e \in E$ has weight
   equal to 1).
4. Apply the algorithm of [146] to find the shortest path $\rho(u_1, u_{k'}) \subset G$.
5. Let $P = \{p\}$.
6. For each vertex $v \in V_1$,
```

6.1. compute the polygon P_v;
6.2. find an edge $e = u_i u_{i+1}$ of $\rho(u_1, u_{k'})$ such that

$$u_{i_z} \leq v_z \leq u_{i+1_z}$$

where $1 \leq i < k'$.
6.3. Compute the point where P_v and e intersect, denoted by v'.
6.4. Let $P = P \cup \{v'\}$.
7. Let $P = P \cup \{q\}$.
8. Let $S_{step} = \{P_v : v \in V_1\}$, which is the step path of Π from p to q.
9. Apply Algorithm 4 on S_{step} and P to compute the shortest path $\rho(p, q)$ on the surface of Π.

9.4 Time Complexity

This section analyses (step by step) procedures and the algorithm proposed in this Chapter.

51. LEMMA. *Procedure 13 has time complexity $\mathcal{O}(|E|log|E|)$, where E is the set of edges of* Π.

Proof. Step 1 can be computed in $\mathcal{O}(|V|)$, where V is the set of vertices of Π. Step 2 can be computed in $\mathcal{O}(|E|)$, where E is the set of edges of Π. Step 3 can be computed in $\mathcal{O}(|V_u|log|V_u|)$. Note that $|V_u| < |V| \leq |E|$.

It follows that Procedure 13 can be computed in $\mathcal{O}(|E|log|E|)$. This proves the lemma. □

52. LEMMA. *Procedure 14 has time complexity $\mathcal{O}(|E|log|E|)$, where E is the set of edges of* Π.

Proof. Step 1 requires only constant time. For Step 2, Case 1 can be computed in $\mathcal{O}(|V(P_1)|)$. Case 2 can be computed in $\mathcal{O}(|V|)$, where V is the set of vertices of Π. Case 3 can be computed in $\mathcal{O}(|E|log|E|)$, where E is the set of edges of Π. Thus, Step 2 can be computed in $\mathcal{O}(|E|log|E|)$, where E is the set of edges of Π because of $|V(P_1)| < |V| \leq |E|$.

Analogously, Step 3 has the same time complexity as Step 2. Steps 4-6 require constant time only. This proves the lemma. □

53. LEMMA. *The time complexity of Algorithm 4 is in $\kappa(\varepsilon) \cdot \mathcal{O}(k|E|log|E|)$, where E is the set of edges of* Π, *and k is the number of the critical polygons between p and q.*

Proof. The difference between Algorithm 3 and Algorithm 4 is defined by Steps 4.1a, 4.2, 5.1a, and 5.2.

By Lemma 52, Steps 4.1a and 5.1a can be computed in $\mathcal{O}(|E|log|E|)$, where E is the set of edges of Π. Steps 4.2 and 5.2 can be computed in in $\mathcal{O}(|V(P_j^*)|)$, where $j = i, k$.

Because $|V(P_j^*)| \leq |V(P_j)| \leq |V| < |E|$, it follows that Algorithm 4 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(k|E|log|E|)$ time, where E is the set of edges of Π, and k is the number of critical polygons between p and q. This proves the lemma. □

22. THEOREM. *Algorithm 5 has a time complexity in $\kappa(\varepsilon) \cdot \mathcal{O}(k|E|log|E|)$, where E is the set of edges of* Π, *and k is the number of critical polygons between p and q.*

Proof. Step 1 can be computed in $\mathcal{O}(|V|)$, where V is the set of vertices of Π. Step 2 can be computed in $\mathcal{O}(|V|log|V|)$. Step 3 can be computed in $\mathcal{O}(|E|)$, where E is the set of edges of Π. Step 4 can be computed in $\mathcal{O}(|V|)$. Step 5 is trivial. By Lemma 51, Step 6.1 can be computed in $\mathcal{O}(|E|log|E|)$. Step 6.2 can be computed in $\mathcal{O}(|E|)$. Step 6.3 can be computed in $\mathcal{O}(|V(P_v)|)$. Steps 6.4 and 7 can be computed in $\mathcal{O}(|P|)$. Because of

$$\sum_{i=1}^{|V_1|}(|E|log|E| + |V(P_{u_i})|) = |V_1|(|E|log|E|) + \sum_{i=1}^{|V_1|}(|V(P_{u_i})|) \leq |V_1|(|E|log|E|) + |V|$$

and $|V(P_v)| \leq |V| \leq |E|$, it follows that Step 6 can be computed in $\mathcal{O}(|E|log|E|)$.

Step 8 can be computed in $\mathcal{O}(|V_1||E|log|E|)$. By Lemma 53, Step 9 can be computed in time $\kappa(\varepsilon) \cdot \mathcal{O}(|V_1||E|log|E|)$, where E is the set of edges of Π, and $|V_1|$ is the number of the critical polygons between p and q. Step 10 can be computed in $\mathcal{O}(|V_1|)$. All together, this proves the theorem. □

9.5 Conclusions

We described a simple $\kappa(\varepsilon) \cdot \mathcal{O}(kn\ log\ n)$ algorithm which contributes a "partial, restricted and approximate answer" to the cited open problem, stated in [106]. Note that our algorithm requires convexity of critical polygons. By experience, the algorithm is easy to implement.

This short chapter is also a further (initial) illustration, how the rubberband algorithm applies to shortest path problems in computational geometry.

Chapter 10

ESPs in Simple Polyhedrons

This chapter considers the 3D ESP problem where the domain Π of possible moves is a simple polyhedron. We propose an $\kappa(\varepsilon) \cdot \mathcal{O}(M|V|)$ algorithm (including preprocessing, with total time complexity $\mathcal{O}((M\ logM + |E|)|V|)$) for solving a special case of the 3D ESP problem (namely, if the domain Π is a type 2 simply connected polyhedron but not necessarily convex). As always, V and E are the set of vertices and edges of Π, respectively; M is the maximal number of vertices of the critical polygon (see Definition 40 below).

The given algorithm solves approximately three NP-complete or NP-hard 3D ESP problems in time $\kappa(\varepsilon) \cdot \mathcal{O}(k)$, where k is the number of layers in a stack, which is introduced as the problem environment.

10.1 Introduction and Related Work

Basically, we continue in this chapter with the 3D ESP problem, already discussed in the previous chapter. There exist already several approximation algorithms for 3D ESP, which we briefly recall below.

Pioneering the field, [119] presents an $\mathcal{O}(n^4(L + log(n/\varepsilon))^2/\varepsilon^2)$ algorithm (see Example 3 in Section 1.1) for the general 3D ESP problem. This was followed by [45], which presents an approximation algorithm for computing an $(1 + \varepsilon)$-shortest path from p to q in time

$$\mathcal{O}(n^2\lambda(n)log(n/\varepsilon)/\varepsilon^4 + n^2\ log\ nr\ log(n\ logr))$$

where r is the ratio of the Euclidean distance $d_e(p, q)$ to the length of the longest edge of any given obstacle, and

$$\lambda(n) = \alpha(n)^{\mathcal{O}(\alpha(n)^{\mathcal{O}(1)})}$$

where $\alpha(n) = A^{-1}(n, n)$ is an inverse Ackermann function [98], which grows very slowly (because A grows very rapidly).

Let there be a finite set of polyhedral obstacles in $\mathbb{R}^3$. Let p, q be two points outside of the union of all obstacles. Assume that $0 < \varepsilon < 1$; [68] gives an $\mathcal{O}(log(n/\varepsilon))$ algorithm to compute an $(1 + \varepsilon)$-shortest path from p to q such that it avoids the interior of any obstacle. The algorithm is based on a subdivision of $\mathbb{R}^3$ which is computed in $\mathcal{O}(n^4/\varepsilon^6)$.

For some special cases of 3D ESP problems, [106] states on page 666 the following:

> The problem is difficult even in the most basic Euclidean shortest-path problem (ESP) in a three-dimensional polyhedral domain P, and even if the obstacles are convex, or the domain P is simply connected.

In this chapter, we apply a simplified version of a rubberband algorithm to present a $\kappa(\varepsilon) \cdot \mathcal{O}(n \ log \ n)$ algorithm (described in Section 10.3) for ESP calculation when P is a (type 1, see Definition 41 below) simply connected polyhedron (which is not necessarily convex).

Section 10.2 provides necessary definitions and theorems. Section 10.3 presents our algorithms. Section 10.4 analyses the time complexity of these algorithms. Section 10.5 illustrates these algorithms by some examples. We generalize the main algorithm to type 2 simple polyhedra in Section 10.6. Section 10.7 concludes the chapter.

10.2 Basics

We denote by Π a *simple polyhedron* (i.e., a compact polyhedral region which is homeomorphic to a unit ball) in the 3D Euclidean space, which is equipped with an xyz Cartesian coordinate system. Let E the set of edges of Π; $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices of Π.

For $p \in \Pi$, let π_p be the plane which is incident with p and parallel to the xoy-plane. The intersection $\pi_p \cap \Pi$ is a finite set of simple polygons; a singleton (i.e., a single point) is considered to be a degenerate polygon. Let P be one of those simple polygons, defined by p and Π.

40. DEFINITION. *Any simple polygon P, being one connected component of $\pi_p \cap \Pi$, is a* critical polygon *(of Π, and with respect to p).*

Any vertex p defines in general a finite set of critical polygons. We start with cases where any vertex p only defines one critical polygon, and this is even required to be convex:

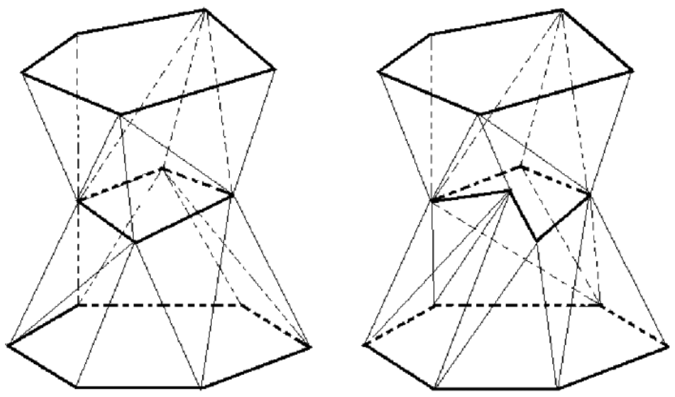


Figure 10.1: Left: a type 1 polyhedron. Right: type 2 polyhedron.

41. DEFINITION. *We say that a simple polyhedron* Π *is a* type 1 *polyhedron iff any vertex* p *defines exactly one convex critical polygon. We say that a simple polyhedron* Π *is a* type 2 *polyhedron iff any vertex* p *defines exactly one simple critical polygon.*

Figure 10.1 shows a type 1 polyhedron on the left, and a type 2 polyhedron on the right. First, this chapter focuses on type 1 simple polyhedrons Π. Later on (in Section 10.6), we generalize the main algorithm also to type 2 simple polyhedrons.

For a convex critical polygon P of Π, let P_z be the z-coordinate of all points in P. Let q_1q_2 be a segment such that ${q_1}_z = {q_2}_z$. Let p_1 and p_2 be two points such that ${p_1}_z < {q_1}_z < {p_2}_z$ and $d_e(p_1, q_1) + d_e(p_2, q_1) = d_e(p_1, q_2) + d_e(p_2, q_2)$. Let p be a point on the line q_1q_2 such that

$$d_e(p_1, p) + d_e(p_2, p) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in q_1q_2\}$$

Then we have the following

54. LEMMA. *p is in between q_1 and q_2.*

Proof. Without loss of generality, suppose that q_1q_2 is parallel to the x-axis. Let the coordinates of p_i be (a_i, b_i, c_i), where i = 1, 2. Let $p = (x, b, c)$ be a point on the line q_1q_2. Then,

$$d_e(p_i, p) = \sqrt{(x - a_i)^2 + (b - b_i)^2 + (c - c_i)^2}$$

for $i = 1, 2$. Let $f(x) = d_e(p_1, p) + d_e(p_2, p)$. Then we have that

$$f'(x) = \frac{x - a_1}{\sqrt{(x - a_1)^2 + (b - b_1)^2 + (c - c_1)^2}} + \frac{x - a_2}{\sqrt{(x - a_2)^2 + (b - b_2)^2 + (c - c_2)^2}}$$

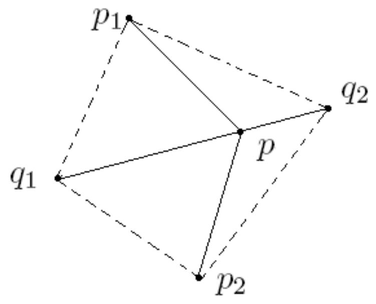


Figure 10.2: Illustration for the proof of Lemma 54.

By setting $f'(x) = 0$, we can find a unique critical point x_p of function $f(x)$; that means, the coordinates of p are equals (x_p, b, c). Let the coordinates of q_i be equals (a_{q_i}, b, c), where i = 1, 2. Since x_p is the unique critical point of the function $f(x)$, it follows that $f(x)$ is decreasing in the interval $(-\infty, x_p)$ and increasing in the interval (x_p, ∞). Because

$$d_e(p_1, q_1) + d_e(p_2, q_1) = d_e(p_1, q_2) + d_e(p_2, q_2)$$

implies $f(a_{q_1}) = f(a_{q_2})$, we have that $a_{q_1} \in (-\infty, x_p)$ and $a_{q_2} \in (x_p, \infty)$. Thus, x_p is located between a_{q_1} and a_{q_2}. This proves the lemma. □

This lemma is used in the following theorem (Theorem 23) which will be frequently used for justifying the main algorithm (Algorithm 7) of this chapter.

Let P be a convex critical polygon of Π. Let e_1 and e_2 be two edges of P. Let p_1 and p_2 be two points such that ${p_1}_z < P_z < {p_2}_z$. Let $P^\bullet$ be the closure of P. Then we have the following

23. THEOREM. *There is a unique point* $p \in P^\bullet$ *such that*

$$d_e(p_1, p) + d_e(p_2, p) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in P^\bullet\}$$

Proof. Let q be the intersection point between the plane $z = P_z$ and the segment p_1p_2.

Case 1. $q \in P^\bullet$.

Let p = q.

Case 2. $q \notin P^\bullet$.

Let e_1, e_2, ..., e_m be all edges of P. By Lemma 16, there is a unique point $q_i \in e_i$ such that $d_e(p_1, q_i) + d_e(p_2, q_i) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in e_i\}$, where $i = 1, 2, \ldots, m$. Let $d_i = d_e(p_1, q_i) + d_e(p_2, q_i)$, where $i = 1, 2, \ldots, m$. Let $d = \min\{d_i : i = 1, 2, \ldots, m\}$. Then there are no two different numbers $j \neq k \in \{1, 2, \ldots, m\}$ such that $d_i = d_k = d$. Otherwise, by Lemma 54, there would be a point p between q_j and q_k such that

$$d_e(p_1, p) + d_e(p_2, p) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in P^\bullet\}$$

Because q_j, $q_k \in \partial P$, and P is convex, p must be in the interior P° of P (with $P^\circ = P^\bullet \setminus \partial P$). This contradicts Lemma 37.

Thus, there is a unique point $p \in \{q_i : i = 1, 2, \ldots, m\}$ such that

$$d_e(p_1, p) + d_e(p_2, p) = \min\{d_e(p_1, q_i) + d_e(p_2, q_i) : i = 1, 2, \ldots, m\}$$

This proves the theorem. □

Analogous to the convex function $d(\Pi, p, q)$ in the previous chapter, we introduce again a convex function which can be used to prove the correctness of the main algorithm.

Let p, $q \in \Pi$ such that $p_z < q_z$. Let $V' = \{v : p_z < v_z < q_z \wedge v \in V\}$. Sort V' according to the z-coordinate. The result is

$$V' = \{v_1, v_2, \ldots, v'_k\} \quad \text{where} \quad v_{1z} \leq v_{2z} \leq \ldots \leq v'_{kz}$$

We partition V' into pairwise disjoint subsets V_1, V_2, ..., and V_k such that $V_i = \{v_{i1}, v_{i2}, \ldots, v_{in_i}\}$, with $v_{ij_y} = v_{ij+1_y}$, for $j = 1, 2, \ldots, n_i - 1$, and $v_{i1y} < v_{i+11_y}$, for $i = 1, 2, \ldots, k - 1$.

Let $u_i = v_{i1}$, for i = 1, 2, ..., k. Let

$$V'' = \{u_1, u_2, \ldots, u_k\}$$

Then we have that

$$u_{1z} < u_{2z} < \ldots < u_{kz}$$

Now we define a function (in $p_0, \ldots, p_{k+1}$)

$$d = \sum_{i=1}^{k+1} d_e(p_{i-1}, p_i)$$

where $p_j \in P^\bullet_{u_j}$ and P_{u_j} is the critical polygon of Π with respect to u_j, where j = 1, 2, ..., k.

It follows that d is a distance function (metric) on Π with respect to the given Cartesian coordinate system, also denoted by $d(\Pi, p, q)$.

The domain of $d(\Pi, p, q)$ is equal to

$$\prod_{i=0}^{k+1} CH(P_{u_i})$$

with convex hulls $CH(P_{u_i})$, for $i = 0, 1, \ldots, k+1$. By Theorem 12, $d(\Pi, p, q)$ is defined on a convex set. By Proposition 2, each $d_e(p_{i-1}, p_i)$ is a convex function defined on $CH(P_{u_{i-1}}) \times CH(P_{u_i})$, where $i = 1, 2, \ldots, k+1$. By Proposition 3, we have

24. THEOREM. *$d(\Pi, p, q)$ is a convex function over*

$$\prod_{i=0}^{k+1} CH(P_{u_i})$$

This theorem is of importance for proving the correctness of the main algorithm (i.e., Algorithm 7) described in Section 10.3.

10.3 The Algorithms

The following small procedure is a simplified version of Procedure 13. It will be used in both the simplified rubberband algorithm (Algorithm 6) and the main algorithm (Algorithm 7) below.

15. PROCEDURE.

Input: E and a point $p \in \Pi$.
Output: V_p (the set of vertices of the critical polygon P_p).

1. Let $E_p = \{e : e = uv \in E \wedge (u_z \leq p_z \leq v_z \vee v_z \leq p_z \leq u_z)\}$.
2. Let $V_1 = \{w : \exists e\,(e \in E_p \wedge w \in e \cap [\text{plane } z = p_z])\}$. In other words, V_1 is the set of intersection points between the plane $z = p_z$ and edges in E_p.
3. Apply Graham's Scan algorithm (see, e.g., Algorithm 1.1 in [85], page 25) to convert V_1 into V_p.
4. Output V_p.

Let $p, q \in \Pi$, $P = \{P_1, P_2, \ldots, P_k\}$ be a sequence of (pairwise disjoint) critical polygons, $P' = \{p_1, p_2, \ldots, p_k\}$ such that $p_i \in P_i^\bullet$, where $i = 1, 2, \ldots, k$ and $k \geq 3$. Let $\rho = (p, p_1, p_2, \ldots, p_k, q)$ be a simple polygonal arc (i.e., a simple polyline) that starts at p, then visits $p_1, p_2, \ldots, p_k$, and finally ends at q (without any self-intersection).

The following algorithm is a simplified version of a rubberband algorithm and a modification of Algorithm 3. (The difference between Algorithm 3 and Algorithm 6 is defined by Steps 2.1a, 4.2 and 5.2.)

6. ALGORITHM.

1. Let $\varepsilon = 10^{-10}$ (the chosen accuracy).
2. Compute the length l_1 of the path $\rho = (p, p_1, p_2, \ldots, p_k, q)$.
2.1a For each $i \in \{1, 2, \ldots, k\}$, apply Procedure 15 for computing V_{p_i} (the set of vertices of the critical polygon P_{p_i}).
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 $q_2 \in P^{\bullet}_{p_i}$ such that

$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in cl(P_{p_i})\}$$

4.3. Update P by replacing p_i by q_2.
4.4. Let $q_1 = p_i$ and $i = i$ + 1.
5.1. $q_3 = q$.
5.2. Compute $q_2 \in P^{\bullet}_{p_k}$ such that

$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in cl(P_{p_k})\}$$

5.3. Update P by replacing p_k by q_2.
6. Compute the length l_2 of the updated path $\rho = (p, p_1, p_2, \ldots, p_k, q)$.
7. Let $\delta = l_1 - l_2$.
8. If $\delta > \epsilon$, then let $l_1 = l_2$ and go to Step 3. Otherwise, Stop.

The accuracy parameter in Step 1 can be chosen such that maximum possible numerical accuracy is guaranteed on the given computer.

The following is the main algorithm of this chapter. – Let p, $q \in \Pi$ such that $p_z < q_z$.

7. ALGORITHM.

1. Compute V' = $\{v : p_z < v_z < q_z \wedge v \in V\}$.
2. Do the following substeps:
2.1. Sort V' according to the z-coordinate. We obtain

$$V' = \{v_1, v_2, \ldots, v_{k'}\} \quad \text{with} \quad {v_1}_z \leq {v_2}_z \leq \ldots \leq {v_{k'}}_z$$

2.2. Partition V' into pairwise disjoint subsets V_1, V_2, ..., and V_k such that V_i = $\{v_{i1}, v_{i2}, \ldots, v_{in_i}\}$, with v_{ij_y} = v_{ij+1_y}, for $j = 1, 2, \ldots, n_i - 1$, and $v_{i1_y} < v_{i+1 1_y}$, for $i = 1, 2, \ldots, k-1$.

2.3. Let $u_i = v_{i1}$, where i = 1, 2, ..., k. Let

$$V'' = \{u_1, u_2, \ldots, u_k\}$$

(then we have that $u_{1z} < u_{2z} < \ldots < u_{kz}$).

3. For each $u_i \in V''$, apply Procedure 15 for computing V_{u_i} (i.e., the set of vertices of the critical polygon P_{u_i}).

4. Let $S_{step} = \{P^{\bullet}_{u_1}, P^{\bullet}_{u_2}, \ldots, P^{\bullet}_{u_k}\}$.

5. Let $P = \{p\} \cup V'' \cup \{q\}$.

6. Apply Algorithm 6 on inputs S_{step} and P, for computing the shortest path $\rho(p, q)$ inside of Π.

7. Convert $\rho(p, q)$ into the standard form of a shortest path by deleting all vertices which are not on any edge of Π (i.e., delete p_i if $p_i \notin \partial P_{u_i}$). [1]

By Theorem 24, we have the following

25. THEOREM. *The solution obtained by Algorithm 7 is an approximate global solution to the 3D ESP problem.*

10.4 Time Complexity

This section analyses time complexities of procedures and algorithms presented above in this chapter.

55. LEMMA. *Procedure 15 has a time complexity in* $\mathcal{O}(|V_p| log |V_p| + |E|)$.

Proof. Step 1 can be computed in $\mathcal{O}(|E|)$. Step 2 can be computed in $\mathcal{O}(|E_p|)$. Step 3 can be computed in $\mathcal{O}(|V_p| log |V_p|)$. Thus, Procedure 15 can be computed in $\mathcal{O}(|V_p| log |V_p| + |E|)$. □

56. LEMMA. *Algorithm 6 can be computed in*

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{p_j}|)$$

[1]It is well known (see, e.g., [106], page 666) that each vertex ($\neq p, q$) of the shortest path is on an edge of Π.

time; the time for preprocessing (i.e., Step 2.1a) equals

$$\mathcal{O}(\sum_{j=1}^{k} |V_{p_j}| log |V_{p_j}| + k|E|)$$

Proof. The difference between Algorithm 3 and Algorithm 6 is defined by Steps 2.1a, 4.2, and 5.2. By Lemma 15, Step 2.1a can be computed in

$$\mathcal{O}(\sum_{j=1}^{k} |V_{p_j}| log |V_{p_j}| + k|E|)$$

time. By Lemma 16 and the proof of Theorem 23, Steps 4.2 and 5.2 can be computed in $\mathcal{O}(|V_{p_j}|)$ time, where V_{p_j} is as in Algorithm 6, for $j = i, k$. Thus, each iteration of Algorithm 6 can be computed in

$$\mathcal{O}(\sum_{j=1}^{k} |V_{p_j}|)$$

time. Therefore, Algorithm 6 can be computed in

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{p_j}|)$$

time; (by applying Lemma 16 k times) the preprocessing step (i.e., Step 2.1a) requires time

$$\mathcal{O}(\sum_{j=1}^{k} |V_{p_j}| log |V_{p_j}| + k|E|)$$

This proves the lemma. □

26. THEOREM. *Algorithm 7 has a time complexity in*

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|)$$

The time needed for the preprocessing steps (i.e., Step 3 and the preprocessing step in Step 6) equals

$$\mathcal{O}(\sum_{j=1}^{k} |V_{u_j}| log |V_{u_j}| + k|E|)$$

Proof. Step 1 can be computed in $\mathcal{O}(|V|)$ time. Step 2.1 can be computed in $\mathcal{O}(|V'|log|V'|)$ time. Step 2.2 can be computed in $\mathcal{O}(|V'|)$ time. Step 2.3 can be computed in $\mathcal{O}(|V''|)$ = $\mathcal{O}(k)$ time. By Lemma 55, Step 3 requires a computation time in

$$\mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|log|V_{u_j}| + k|E|)$$

Steps 4 and 5 can be computed in $\mathcal{O}(|V''|)$ = $\mathcal{O}(k)$. By Lemma 56, Step 6 has a time complexity in

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|)$$

if excluding to count the preprocessing time

$$\mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|log|V_{u_j}| + k|E|)$$

Step 7 can be computed in $\mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|)$ time. Therefore, Algorithm 7 can be computed in time

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|)$$

if excluding to count the preprocessing time

$$\mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|log|V_{u_j}| + k|E|)$$

This proves the theorem. □

7. COROLLARY. *Algorithm 7 can be computed in time*

$$\kappa(\varepsilon) \cdot \mathcal{O}(M|V|) + \mathcal{O}((M\ logM + |E|)|V|)$$

where

$$\mathcal{O}((M\ logM + |E|)|V|)$$

is the time for preprocessing, and $M = \max\{|V_{u_j}| : j = 1, 2, \ldots, k\}$.

10.5 Examples

In this section we apply Algorithm 7 to two special cases of ESP problems which can be solved efficiently. We also point out that the "complements" of these two problems are both NP-complete or NP-hard.

In this section, we generalize the notion of a critical polygon, as given in Definition 40 of Section 10.2. We assume that a generalized Π is a simply connected (possibly unbounded) polyhedron, and we allow that the resulting (generalized) critical polygons are unbounded. For example, a generalized critical polygon may have a vertex at infinity, or it can be the complement of a critical polygon (as specified in Definition 40).

We recall some concepts introduced in [107]. Let (x_0, y_0, z_0) be a point in 3D space. Let

$$\begin{aligned} S_1 &= \{(x, y, z_0) : x_0 \leq x < \infty \wedge y_0 \leq y < \infty\} \\ S_2 &= \{(x, y, z_0) : -\infty < x \leq x_0 \wedge y_0 \leq y < \infty\} \\ S_3 &= \{(x, y, z_0) : -\infty < x \leq x_0 \wedge -\infty < y \leq y_0\} \\ S_3 &= \{(x, y, z_0) : x_0 \leq x < \infty \wedge -\infty < y \leq y_0\} \end{aligned}$$

S_i is called a *q-rectangle of type i*, where i = 1, 2, 3, 4. Furthermore, let (x_1, y_1, z_0) be a point in 3D space such that $x_1 > x_0$ and $y_1 > y_0$. Let

$$\begin{aligned} S_h &= \{(x, y, z_0) : -\infty < x < \infty \wedge y_0 \leq y \leq y_1\} \\ S_v &= \{(x, y, z_0) : x_0 \leq x \leq x_1 \wedge -\infty < y < \infty\} \end{aligned}$$

S_h (S_v) is called a *horizontal (vertical) strip*. Finally, let

$$\begin{aligned} S_{h_1} &= \{(x, y, z_0) : x_0 \leq x < \infty \wedge y_0 \leq y \leq y_1\} \\ S_{h_2} &= \{(x, y, z_0) : -\infty < x \leq x_0 \wedge y_0 \leq y \leq y_1\} \\ S_{v_1} &= \{(x, y, z_0) : x_0 \leq x \leq x_1 \wedge y_0 \leq y < \infty\} \\ S_{v_2} &= \{(x, y, z_0) : x_0 \leq x \leq x_1 \wedge -\infty < y \leq y_0\} \end{aligned}$$

According to their geometric shape, we notice that

S_1 [S_2, S_3, S_4] is unbounded in direction $(+x, +y)$ [$(-x, +y)$, $(-x, -y)$, $(+x, -y)$];

S_h [S_v] is unbounded in direction $\pm x$ [$\pm y$];

S_{h_1} [S_{h_2}, S_{v_1}, S_{v_2}] is unbounded in direction $+x$ [$-x$, $+y$, $-y$].

S_i, S_h, S_v, S_{h_j}, and S_{v_j} are axis-aligned rectangles (see Figure 10.3), where i = 1, 2, 3, 4, and j = 1, 2. The stack $\mathcal{S}$ of axis-aligned rectangles is called *terrain-like* if, for at least one of the four directions $-x$, $+x$, $-y$, or $+y$, each rectangle in $\mathcal{S}$ is unbounded.

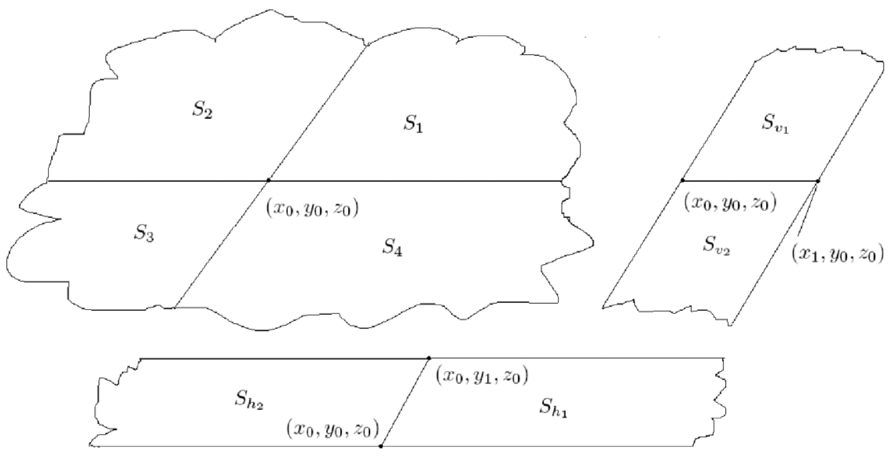


Figure 10.3: Axis-aligned rectangles.

9. EXAMPLE.

Let Π be a simple polyhedron such that each generalized critical polygon is an axis-aligned rectangle. Let $p, q \in \Pi$ such that $p_z < q_z$. Let $V_{pq} = \{v : p_z < v_z < q_z \land v \in V\}$, where V is the set of vertices of Π. By Theorem 26, the Euclidean shortest path between p and q inside of Π can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$. Therefore, the 3D ESP problem can be solved efficiently in such a special case.

However, if we modify Π such that each generalized critical polygon is the complement of an axis-aligned rectangle, then the problem of finding the Euclidean shortest path between p and q inside of Π is NP-complete (!) because of the following

27. THEOREM. ([107], Theorem 4*) It is NP-complete to decide wether there exists an obstacle-avoiding path of Euclidean length at most L among a set of stacked axis-aligned rectangles. The problem is (already) NP-complete for the special case that the axis-aligned rectangles are all q-rectangles of types 1 or 3.*

10. EXAMPLE.

We slightly modify Π as considered in Example 9: Let Π be a simply connected polyhedron such that each critical polygon is a triangle. By Theorem 26, the Euclidean shortest path between p and q inside of Π can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$ time. Therefore, the 3D ESP problem can be solved efficiently in such a special case.

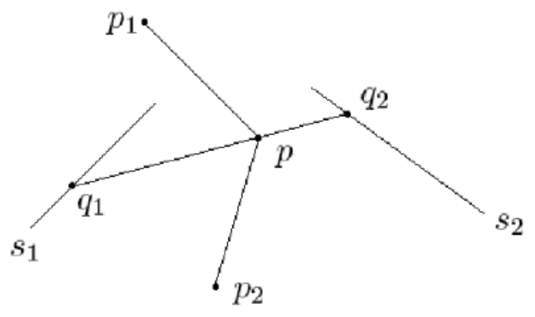


Figure 10.4: Illustration for the proof of Lemma 57.

However, if we modify Π such that each generalized critical polygon is the complement of a triangle, then the problem of finding the Euclidean shortest path between p and q inside of Π is NP-hard (!) because of the following

28. THEOREM. *([27]) It is NP-hard to decide whether there exists an obstacle-avoiding path of Euclidean length at most L among a set of stacked triangles.*

We approximately solve these two very difficult problems, addressed in Theorems 27 and 28, in Section 10.6.4.

10.6 Simple Polyhedrons of Type 2

This section generalize the given algorithms to type 2, or (more general) to "type 2-like" simple polyhedrons.

10.6.1 Basics

Let $\mathcal{S}_r$ or $\mathcal{S}_t$ be finite stacks of axis-aligned rectangles or triangles (there may be several, pairwise disjoint triangles on the same plane), respectively.

In this section, let Π be a type 2 simple polyhedron. We consider the closed sets $\overline{\mathcal{S}}_r^\bullet$ or $\overline{\mathcal{S}}_t^\bullet$, where $\overline{\mathcal{S}}_r$ or $\overline{\mathcal{S}}_t$ are the complementary sets of $\mathcal{S}_r$ or $\mathcal{S}_t$, respectively.

We aim at generalizing Theorem 23 for the case of possibly also nonconvex critical polygons. Let q_1, $q_2 \in \mathbb{R}^3$ such that $q_1 \neq q_2$. Let the coordinates of q_i be equals (x_i, y_i, z_i), where i = 1, 2. We use (standard) lexicographic order: q_1 *is before* q_2 iff $q_{1x} < q_{2x}$, or $q_{1x} = q_{2x}$ and $q_{1y} < q_{2y}$, or $q_{1x} = q_{2x}$ and $q_{1y} = q_{2y}$ and $q_{1z} < q_{2z}$. If q_1 is before q_2, then we write $\min\{q_1, q_2\} = q_1$, otherwise $\min\{q_1, q_2\} = q_2$.

Let s_1 and s_2 be two segments. Let p_1 and p_2 be two points such that $p_i \notin \partial s_j$, where i, j = 1, 2. Let $S = \{p : p \in \partial s_1 \vee p \in \partial s_2\}$. Then we have the following

57. LEMMA. *There is a point $p \in S$ such that* [2]

$$d_e(p_1, p) + d_e(p_2, p) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in S\}$$

Proof. By Lemma 16, there is a unique point $q_i \in \partial s_i$ such that

$$d_e(p_1, q_i) + d_e(p_2, q_i) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in \partial s_i\}$$

where i = 1, 2. Let $s = q_1q_2$ (see Figure 10.4). We can find the required point p as follows:

Case 1. $d_e(p_1, q_1) + d_e(p_2, q_1) < d_e(p_1, q_2) + d_e(p_2, q_2)$

Let $p = q_1$.

Case 2. $d_e(p_1, q_1) + d_e(p_2, q_1) > d_e(p_1, q_2) + d_e(p_2, q_2)$

Let $p = q_2$.

Case 3. $d_e(p_1, q_1) + d_e(p_2, q_1) = d_e(p_1, q_2) + d_e(p_2, q_2)$

Let $p = \min\{q_1, q_2\}$. – This proves the lemma. □

Let $\{s_1, s_2, \ldots, s_m\}$ be a set of segments. Let p_1 and p_2 be two points such that $p_i \notin \partial s_j$, where i = 1, 2, j = 1, 2, …, m. Let $S = \{p : p \in \partial s_j \wedge j = 1, 2, \ldots, m\}$. By Lemma 57, we have the following generalized version of Theorem 23:

29. THEOREM. *There is a point $p \in S$ such that*

$$d_e(p_1, p) + d_e(p_2, p) = \min\{d_e(p_1, q) + d_e(p_2, q) : q \in S\}$$

Note that the specified point p in Theorem 29 is not unique in general. However, using the recalled alphabetic order, we can select a unique minimum with respect to this order (see also proof of Lemma 57). This uniqueness is very important (!) for the proof of the correctness of the Algorithm 9. (This methodology will also be used for Algorithm 10, when discussing its Steps 4.2 and 5.2 in the next chapter.)

It is easy to see that Theorem 29 is still correct if s_j is a straight line or a ray, where j = 1, 2, …, m.

[2] As we can see in the proof of this lemma below, this point p is uniquely specified if it is obtained by comparing it with other candidate points not only by length but also by coordinates.

10.6.2 The Algorithms

The following procedure is a generalized version of Procedure 15 for nonconvex critical polygons. – Let $S = \{\triangle_1, \triangle_2, \ldots, \triangle_m\}$ be the set of all faces of Π.

16. PROCEDURE.

Input: set S and a vertex $v \in V$.
Output: set V_v of all vertices of the critical polygon P_v (which may be nonconvex).

```
1. Let $S_v = \{\triangle : \triangle \in S \wedge e = uw \in E(\triangle) \wedge (u_z \leq p_z \leq w_z \vee w_z \leq p_z \leq u_z)\}$, and
   $E(S_v) = \{e : \exists\triangle \in S \wedge e \in E(\triangle)\}$.
2. Let $V_v = \emptyset$, and $v_1 = v$.
3. Let $V_v = V_v \cup \{v_1\}$.
4. Find a face $\triangle_1 \in S_v$ such that $v_1 \in V(\triangle_1)$.
5. Find an edge $e \in E(\triangle_1)$ such that $v_1 \notin \partial e$ and $\pi_{v_1} \cap e \neq \emptyset$.
6. Do the following substeps:
Case 1. $|\pi_{v_1} \cap e| = 1$. (note: $e$ is not parallel to the plane $\pi_{v_1}$)
6.1. Let $\pi_{v_1} \cap e = v_2$, and $V_v = V_v \cup \{v_2\}$.
Case 2. $|\pi_{v_1} \cap e| > 1$. (note: $e$ is parallel to the plane $\pi_{v_1}$)
Let $e = w_1w_2$.
Case 2A. $vw_i \in E(S_v)$, for $i$ = 1 and 2. (note: the critical polygon $P_{v_1}$ is a triangle)
6.2A.1. Let $e = w_1w_2$, and $V_v = V_v \cup \{w_1, w_2\}$.
6.2A.2. Output $V_v$, and Stop.
Case 2B. $vw_1 \in E(S_v)$ and $vw_2 \notin E(S_v)$.
6.2B.1. Find a face $\triangle_2 \in S_v$ such that $w_2 \in V(\triangle_2)$ and $w_1w_2 \notin E(\triangle_2)$.
6.2B.2. Let $\triangle_1 = \triangle_2$ and $v_1 = w_2$, and go to Step 5.
Case 2C. $vw_2 \in E(S_v)$ and $vw_1 \notin E(S_v)$.
6.2C.1. Find a face $\triangle_2 \in S_v$ such that $w_1 \in V(\triangle_2)$ and $w_1w_2 \notin E(\triangle_2)$.
6.2C.2. Let $\triangle_1 = \triangle_2$ and $v_1 = w_1$, and go to Step 5.
   (note: Case 2C is a modified Case 2B, simply by swapping $w_1$ and $w_2$)
7. Let $\triangle_2 \in S_v$ be that face which shares edge $e$ with $\triangle_1$.
8. If $v_2 \neq v$, let $\triangle_1 = \triangle_2$ and $v_1 = v_2$, and go to Step 4.
   Otherwise output $V_v$, and Stop.
```

The following algorithms are generalized versions of Algorithms 6 and 7, respectively.

8. ALGORITHM.

Simply replace "Procedure 15" by "Procedure 16" in Step 2.1a of Algorithm 6.

9. ALGORITHM.

Simply replace "Procedure 15" by "Procedure 16" in Step 3, and "Algorithm 6" by "Algorithm 8" in Step 6 of Algorithm 7.

The correctness of this algorithm follows by

30. THEOREM. *The solution obtained by Algorithm 9 is an approximate global solution to the 3D ESP problem.*

Proof. The main idea is by applying a basic property of a compact space: each open cover for a compact space has already a finite subcover for this space. For further details of the proof, see the discussions at the end of Section 11.6.3 in Chapter 11. □

10.6.3 Time Complexity

We analyze the time complexity of the generalized algorithms, and start with the used procedure:

58. LEMMA. *Procedure 16 can be computed in $\mathcal{O}(|V_v||E(S_v)|)$ time.*

Proof. Step 1 can be computed in $\mathcal{O}(|S|)$ time. Step 2 only requires constant time. Step 3 can be computed in time $\mathcal{O}(1)$. Step 4 can be computed in time $\mathcal{O}(|S_v|)$. Step 5 can be computed in time $\mathcal{O}(1)$.

Step 6.1 (Case 1) can be computed in $\mathcal{O}(1)$ time. In Step 6, Case 2A can be computed in $\mathcal{O}(|E(S_v)|)$ time. Step 6.2A.1 can be computed in $\mathcal{O}(1)$ time. Step 6.2A.2 only needs constant time. Case 2B (Case 2C) can be computed in time $\mathcal{O}(|E(S_v)|)$. Step 6.2B.1 (6.2C.1) can be computed in time $\mathcal{O}(|S_v|)$. Step 6.2B.2 (6.2C.2) only needs constant time. It follows that Step 6 can be computed in time $\mathcal{O}(|E(S_v)|)$ because of $|S_v| \leq |E(S_v)|$.

Step 7 can be computed in time $\mathcal{O}(|S_v|)$. Step 8 can be computed in time $\mathcal{O}(1)$. Thus, each iteration can be computed in time $\mathcal{O}(|E(S_v)|)$ because of $|S_v| \leq |E(S_v)|$.

Altogether, Procedure 16 has a time complexity in $\mathcal{O}(|V_v||E(S_v)|)$. □

59. LEMMA. *Algorithm 8 can be computed in*

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{p_j}| + \mathcal{O}(\sum_{j=1}^{k} |V_{v_j}||E(S_{v_j})|))$$

where

$$\mathcal{O}(\sum_{j=1}^{k} |V_{v_j}||E(S_{v_j})|)$$

is the time for preprocessing.

31. THEOREM. *Algorithm 7 can be computed in*

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V_{u_j}|) + \mathcal{O}(\sum_{j=1}^{k} |V_{u_j}||E(S_{u_j})|)$$

where

$$\mathcal{O}(\sum_{j=1}^{k} |V_{u_j}||E(S_{u_j})|)$$

is the time for preprocessing.

8. COROLLARY. *Algorithm 7 can be computed in*

$$\kappa(\varepsilon) \cdot \mathcal{O}(M|V|) + \mathcal{O}(M|E|)$$

where

$$\mathcal{O}(M|E|)$$

is the time for preprocessing, and $M = \max\{|V_{u_j}| : j = 1, 2, \ldots, k\}$.

10.6.4 Three NP-complete or NP-hard Problems

As in Section 10.5, we again generalize the notion of a critical polygon, also allowing unbounded polygons. We now apply the generalized Algorithms 8 and 9 to approximately solve hard problems, characterized in Section 10.5 as being NP-complete or NP-hard.

11. EXAMPLE.

We modify Example 9 as follows: Let Π be a simply connected polyhedron such that each critical polygon is the complement of an axis-aligned rectangle. Let p, $q \in \Pi$ such that $p_z < q_z$. Let $V_{pq} = \{v : p_z < v_z < q_z \wedge v \in V\}$, where V is the set of all vertices of Π. By Theorem 31, the Euclidean shortest path between p and q inside of Π can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$ time. Therefore, the 3D ESP problem can be approximately solved efficiently in such a special case.

12. EXAMPLE.

We modify Example 10 as follows (also just a slight modification of Π in Example 11): Let Π be a simply connected polyhedron such that each critical polygon is the complement of a triangle (or of a finite number of pairwise disjoint triangles). By Theorem 31, the Euclidean shortest path between p and q inside of Π can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(|V_{pq}|)$ time.

13. EXAMPLE.

Let $\mathcal{S}$ be a stack of k horizontal or vertical strips. The Euclidean shortest path among $\mathcal{S}$ can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(k)$ time. Finding the exact solution is very hard (NP-complete!) because of the following

32. THEOREM. ([107], Theorem 5*) It is NP-complete to decide whether there exists an obstacle-avoiding path of Euclidean length at most L among a finite number of stacked horizontal and vertical strips.*

14. EXAMPLE.

Let $\mathcal{S}$ be a stack of k terrain-like axis-parallel rectangles. The Euclidean shortest path among $\mathcal{S}$ can be approximately computed in $\kappa(\varepsilon) \cdot \mathcal{O}(k)$ time. The best known algorithm for finding the exact solution has a time complexity in $\mathcal{O}(k^4)$ due to the following

33. THEOREM. ([107], Theorem 6*) Let $\mathcal{S}$ be a stack of k terrain-like axis-parallel rectangles. The Euclidean shortest path among $\mathcal{S}$ can be computed in $\mathcal{O}(k^4)$ time.*

10.7 Conclusions

This chapter described at first an algorithm for solving the 3D ESP problem when the domain Π is a type 1 simply connected polyhedron (note: Π itself is not necessary convex). The time complexity of this algorithm is in

$$\kappa(\varepsilon) \cdot \mathcal{O}(M|V|) + \mathcal{O}((M\ logM + |E|)|V|)$$

(where $\mathcal{O}((M\ logM + |E|)|V|)$ is the time for preprocessing). M is the maximal number of vertices of one of the critical polygons. We also presented two examples for illustrating that the algorithm can approximately solve efficiently some special cases of 3D ESP problems where the "complement" of these problems is known to be NP-complete or NP-hard.

Most importantly, we further generalized this algorithm to an algorithm for solving the 3D ESP problem when the domain Π is a type 2 simply connected polyhedron. This generalized algorithm has a time complexity in

$$\kappa(\varepsilon) \cdot \mathcal{O}(M|V|) + \mathcal{O}(M|E|)$$

(where $\mathcal{O}(M|E|)$ is the time for preprocessing). It was also shown that the algorithm approximately solves three NP-complete or NP-hard problems in time $\kappa(\varepsilon) \cdot \mathcal{O}(k)$, where k is the number of layers in the given stack of polygons.

Chapter 11

Art Gallery Problems

This chapter applies the basic idea of the original rubberband algorithm for deriving solutions for a few classic problems such as the safari, zookeeper, or watchman route problem. [1] *It contains several important results. For example, it answers the open problem "What is the complexity of the touring polygons problem for pairwise disjoint nonconvex simple polygons?" by providing a κ-linear approximate algorithm for solving this problem. The chapter also significantly improves the results for the watchman route problem. As a further example, this chapter finds an approximate solution to the unconstrained touring polygons problem which is known to be NP-hard.*

11.1 Introduction

There are a number of computational geometry problems which involve finding a shortest path [106], for example, the safari problem, zookeeper problem, or watchman route problem. This chapter presents algorithms for solving the touring polygons problem (TPP), parts cutting problem, safari problem, zookeeper problem, and watchman route problem. These problems are closely related to one-another. A solution to the first problem implies solutions to the other four problems. The safari, zookeeper, and watchman route problems belong to the class of art gallery problems [147].

11.1.1 Touring Polygons Problem

We recall some notations from [52], which introduced the touring polygons problem. Let π be a plane, which is identified with $\mathbb{R}^2$. Consider polygons $P_i \subset \pi$, where $i = 1, 2, \ldots, k$, and two points p, $q \in \pi$. Let p_0 = p and p_{k+1} = q. Let $p_i \in \mathbb{R}^2$, where $i = 1, 2, \ldots, k$. Let $\rho(p, p_1, p_2, , \ldots, p_k, q)$ denote the path $pp_1p_2 \ldots p_kq \subset \mathbb{R}^2$. Let

[1]So far, our algorithms can compute an approximate restricted solution for both safari and zookeeper problems; and an approximate general solution for the watchman route problem.

$\rho(p, q) = \rho(p, p_1, p_2, , \ldots, p_k, q)$ if this does not cause any confusion. If $p_i \in P_i$ such that p_i is the first (i.e., along the path) point in $\partial P_i \cap \rho(p, p_i)$, then we say that path $\rho(p, q)$ *visits* P_i at p_i, where $i = 1, 2, \ldots, k$.

The *unconstrained* TPP is defined as follows:

How to find a shortest path $\rho(p, p_1, p_2, , \ldots, p_k, q)$ such that it visits each of the polygons P_i in the given order $i = 1, 2, \ldots, k$?

Let $F_i \subset \mathbb{R}^2$ be a simple polygon such that $(P_i^\bullet \cup P_{i+1}^\bullet) \subset F_i^\bullet$; then we say that F_i is a *fence* [with respect to P_i and P_{i+1} (mod $k+1$)], where $i = 0, 1, 2, \ldots, k+1$. Now assume that we have a fence F_i for any pair of polygons P_i and P_{i+1}, for $i = 0, 1, \ldots, k+1$.

The *constrained* TPP is defined as follows:

How to find a shortest path $\rho(p, p_1, p_2, , \ldots, p_k, q)$ such that it visits each of the polygons P_i in the given order, also satisfying $p_i p_{i+1}(mod\ k+1) \subset F_i^\bullet$, for $i = 1, 2, \ldots, k$?

Assume that for any $i, j \in \{1, 2, \ldots, k\}$, $\partial P_i \cap \partial P_j = \emptyset$, and each P_i is convex; this special case is dealt with in [52]. The given algorithm runs in $\mathcal{O}(kn\ log(n/k))$ time, where n is the total number of all vertices of all polygons $P_i \subset \pi$, for $i = 1, 2, \ldots, k$.

According to [52], "one of the most intriguing open problems" identified by their results "is to determine the complexity of the TPP for (pairwise) disjoint nonconvex simple polygons".

Algorithm 10 in Section 11.4.1 answers this problem by providing an approximate algorithm running in time $\kappa(\varepsilon) \cdot \mathcal{O}(n)$, where n is the total number of vertices of all polygons.

11.1.2 The Parts Cutting Problem

In a variety of industries, such as clothing, window manufacturing, or mechanic, it is necessary to cut a set of parts (modeled by polygons) from large sheets of paper, cloth, glass, metal, and so forth. Motivated by such applications, [75] introduced the following three restrictions for cutting scenarios:

1. *The continuous cutting problem*: here the path of the cutting tool visits each object (i.e., polygon) to be cut just once. The tool can engage the object at any point on its frontier, but must cut the entire object before it travels to the next object. Ac-

cordingly, the same frontier point must be used for entry and departure from the object.

2. *The endpoint cutting problem*: here the tool can enter and exit the object only at some predefined frontier points; however, it may cut the object in sections (i.e., it may visit an object repeatedly).

3. *The intermittent cutting problem*: this is the most general version of the problem in which the object can be cut in sections and there is no restriction on the points that can be used for entry or exit.

[75] focused on the continuous cutting problem where each object is a polygon. They also called this problem the *plate-cutting traveling salesman problem* (P-TSP).

It is a generalization of the well-known traveling salesman problem (TSP) [90]. The P-TSP further generalizes the generalized TSP (GTSP) [89, 114]. If each polygon degenerates into a single vertex, then P-TSP becomes the TSP which is known to be NP-hard [56]. It follows that P-TSP is NP-hard as well.

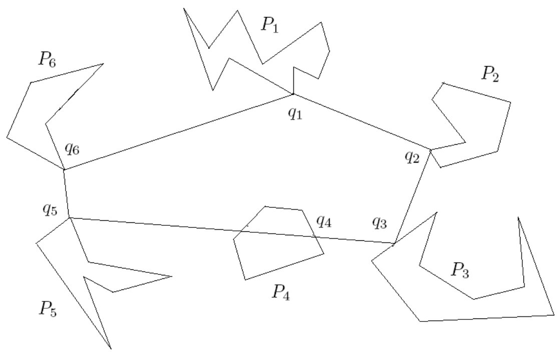


Figure 11.1: Illustration for the simplified P-TSP in [75] where polygons are assumed to be given in a particular order.

The difficulty of the P-TSP caused [75] to consider the problem with an additional condition, assuming now that all polygons are given in a particular order (see Figure 11.1). [75] then solved this simplified P-TSP by a heuristic approach based on a Lagrange relaxation method [57, 64], without providing a complexity analysis for this proposed approach. As a follow-up of the work in [75], [51] then proved that a further simplified P-TSP (i.e., only convex polygons, and a particular order of those polygons) is solvable in polynomial time (see Figure 11.2). If, additionally, the start point is also given (see Figure 11.3), then [52] claim that they can solve the same problem in time $\mathcal{O}(kn\ log(n/k))$, where n is the total number of vertices of all

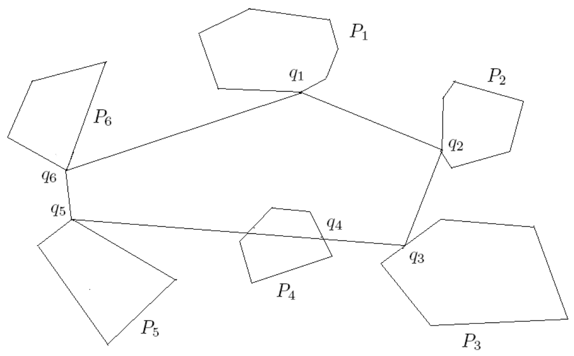


Figure 11.2: Illustration for the further simplified P-TSP in [51] also assuming convex polygons.

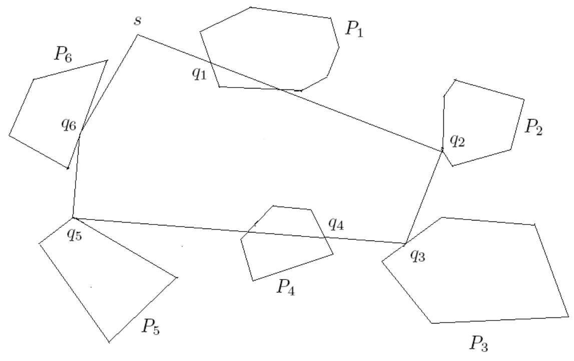


Figure 11.3: Illustration for the P-TSP as considered in [52], now also with a given start point s.

polygons, $\partial P_i \subset \pi$, and $i = 1, 2, \ldots, k$.

11.1.3 The Safari and Zookeeper Problems

Let π be a plane. Let Π be a simple polygon which contains k pairwise disjoint simple polygons P_i ($i = 1, 2, \ldots, k$) such that exactly one edge of each of these polygons P_i is incident with the frontier of Π. We are interested in a cycle with vertices $p_i \in \partial P_i$, for $i = 1, 2, \ldots, k$. We consider indices modulo k, and identify index k with index 0 this way. Start point p and end point q are assumed to be identical; that means we have $p = q = p_0 \in \partial P_k$. Let P_i° be the interior of P_i. The *safari* problem, introduced in [115], is defined as follows:

How to find a shortest tour (i.e., a closed path) $\rho(p, q)$ inside of Π such that $\rho(p, q)$ visits each P_i at a point p_i on the frontier of P_i, allowing that straight segments $p_i p_{i+1}$ (mod k) intersect P_i° in one or several segments, for $i = 1, 2, \ldots, k$?

[115] was historically (in 1992) the first publication which studied the safari problem. It claimed to have an $\mathcal{O}(kn^2)$ time algorithm for solving this problem, where n is the total number of vertices of polygon Π and all polygons P_i, for $i = 1, 2, \ldots, k$. In 1994, [138] improved the result to an $\mathcal{O}(n^2)$ time algorithm for the general case, not using anymore the restriction used in [115] of forcing the route through a specific point. In 2003, [143] showed that there is an error in the algorithm proposed in [115], and presented an $\mathcal{O}(n^3)$ time algorithm for the case that a starting point is given for the route to be constructed, where n is again the total number of vertices of Π and all polygons P_i, for $i = 1, 2, \ldots, k$. The algorithm runs in $\mathcal{O}(n^4)$ time if not using the restriction of a given start point. In the same year, the result was improved by [52] with an algorithm running in time $\mathcal{O}(kn\ log(n/k))$ assuming a given start point.

The *zookeeper* problem was introduced in [40]; it is defined as follows (informally speaking, "the zookeeper is not supposed to enter any of the cages on his path, but to visit all"):

How to find a shortest tour (i.e., a closed path) $\rho(p, q)$ inside of Π such that $\rho(p, q)$ visits each "cage" P_i at a point p_i on its frontier, and such that this path is not intersecting any of the interiors P_i°, for $i = 1, 2, \ldots, k$?

Both [38] (in 1987) and [40] (in 1992) present an $\mathcal{O}(n^2)$ algorithm, where n is the total number of vertices of Π and P_i, and $i = 1, 2, \ldots, k$. In 1994, [72] improved this to $\mathcal{O}(n\ log^2 n)$. In 2002, [19] improved this further to $\mathcal{O}(n\ log\ n)$; the algorithm starts at a given point, and n is the input size as defined above. [141] (in 2001) gave an $\mathcal{O}(n^2)$ algorithm without specifying a starting point.

– [106] states on page 685 the following *open problem*:

Can the zookeeper problem be solved in time $\mathcal{O}(n)$?

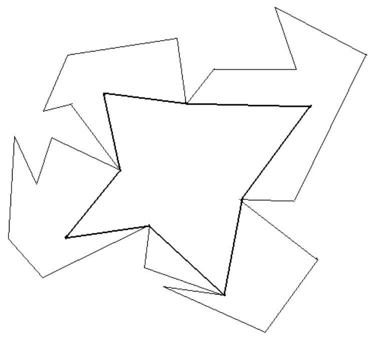


Figure 11.4: A watchman route.

11.1.4 The Watchman Route Problem

Let Π be a simple polygon with n vertices. The *watchman route problem* (WRP) is discussed in [10], and it is defined as follows:

How to calculate a shortest route $\rho \subset \Pi^\bullet$ such that any point $p \in \Pi^\bullet$ is visible from at least one point on the path?

This is actually equivalent to the requirement, that all points $p \in \Pi^\bullet$ are visible just from the vertices of the path ρ, that means, for any $p \in \Pi^\bullet$ there is a vertex q of ρ such that $pq \subset \Pi^\bullet$; see Figure 11.4. If the start point of the route is given, then this refined problem is known as the *fixed* WRP.

A simplified WRP was first solved in 1988 in [39] by presenting an $\mathcal{O}(n\ loglog\ n)$ algorithm to find a shortest route in a simple isothetic polygon. In 1991, [41] claimed to have presented an $\mathcal{O}(n^4)$ algorithm, solving the fixed WRP. In 1993, [136] obtained an $\mathcal{O}(n^3)$ solution for the fixed WRP. In the same year, this was further improved to a quadratic time algorithm [137]. However, four years later, in 1997, [65] pointed out that the algorithms in both [41] and [136] were flawed, but presented a solution for fixing those errors. Interestingly, two years later, in 1999, [139] found that the solution given by [65] was also flawed! By modifying the (flawed) algorithm presented in [136]. [139] gave an $\mathcal{O}(n^4)$ runtime algorithm for the fixed WRP.

In 1995,[32] proposed an $\mathcal{O}(n^{12})$ runtime algorithm for the WRP. In the same year, [112] gave an $\mathcal{O}(n^6)$ algorithm for the WRP. This was improved in 2001 by an $\mathcal{O}(n^5)$ algorithm in [140]; this paper also proved the following

34. THEOREM. *There is a unique watchman route in a simple polygon, except for those* *cases where there is an infinite number of different shortest routes, all of equal length.*

So far the best known result for the WRP is due to [52] which gave in 2003 an $\mathcal{O}(n^3 \ log \ n)$ runtime algorithm.

Given the time complexity of those algorithms for solving the WRP, finding efficient approximation algorithm became an interesting subject. In 1995, [100] gave an $\mathcal{O}(log \ n)$-approximation algorithm for solving the WRP. In 1997, [33] gave a 99.98-approximation algorithm with time complexity $\mathcal{O}(n \ log \ n)$ for the WRP. In 2001, [142] presented a linear-time algorithm for an approximative solution of the fixed WRP such that the length of the calculated watchman route is at most twice of that of the shortest watchman route. The coefficient of accuracy was improved to $\sqrt{2}$ in [144] in 2004. Most recently, [145] presented a linear-time algorithm for the WRP for calculating an approximative watchman route of length at most twice of that of the shortest watchman route.

There are several generalizations and variations of watchman route problems; see, for example, [30, 29, 34, 48, 59, 60, 61, 87, 105, 111, 112, 116, 117, 118]. [5, 6, 8] show that some of these problems are NP-hard and solve them by approximation algorithms.

The rest of this chapter is organized as follows: Section 11.2 recalls and introduces useful notions. Section 11.3 lists some known results which are applied later in this chapter. Section 11.4 describes the new algorithms. Section 11.5 analyzes the time complexity of these algorithms. Section 11.6 proves the correctness of the algorithms. Section 11.7 concludes the chapter.

11.2 Definitions

We have to state a few more definitions before we are able to describe algorithms for solving the safari, zookeeper, watchman route, or touring polygons problem.

Let Π be a simple polygon. Let $q_1, q_2 \in \partial\Pi$ with $q_1 \neq q_2$. We denote by $\rho(q_1, \Pi, q_2)$ [$\rho^{-1}(q_1, \Pi, q_2)$] a simple polyline which starts at q_1 and goes then anti-clockwise [clockwise] around $\partial\Pi$; finally it ends at q_2. (Note that "anti-clockwise" means that when tracing an edge of $\partial\Pi$, a point $p \in \Pi^\circ$, which is sufficiently close to this edge, is always on the left-hand side of this edge.)

For discussing the safari or zookeeper problem, let $S_\Pi = (P_1, P_2, \ldots, P_k)$ be a sequence of all given polygons in Π, listing them in anti-clockwise order when tracing $\partial\Pi$. Each polygon P_i has exactly one edge $e_i = v_{i_1}v_{i_2}$ which is incident with the frontier of Π, for $i = 1, 2, \ldots, k$. The following definition is independent of the specific choice of S_Π (i.e., which of the k polygons takes the leading position P_1).

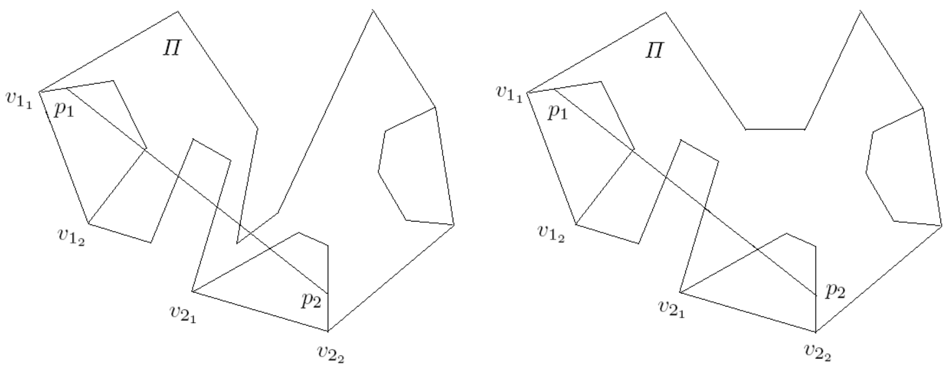


Figure 11.5: Left: This is not a simplified safari problem. Right: A simplified safari problem.

42. DEFINITION. *The polygons* Π *and* P_i *(*$i = 1, 2, \ldots, k$*) define a* simplified safari or zookeeper problem *iff, for each* $i \in \{1, 2, \ldots, k\}$*, it is true that* $P_i \cap P_{i+1} = \emptyset$*, and for all pairs of vertices* $p_i \in P_i$ *and* $p_{i+1} \in P_{i+1}$*,* [2]

the straight segment $p_i p_{i+1}$ *does not intersect the path* $\rho(v_{i+1_2}, \Pi, v_{i_1})$*.*

See Figure 11.5 for an illustration of this definition.

Now we consider the constrained TPP. The given set of polygons P_i, for $i = 0, 1, \ldots, k$, is assumed to be given in form of a sorted sequence $S_\Pi = (P_0, P_1, \ldots, P_{k-1})$. Let F_i be the fence of P_i and P_{i+1}, where $i = 0, 1, \ldots, k-1$ (mod k).

43. DEFINITION. *These polygons* P_i *and fences* F_i *define a* simplified constrained TPP *iff, or each* $i \in \{0, 1, \ldots, k-1\}$ *and for each* $p_i \in P_i$ *and* $p_{i+1} \in P_{i+1}$*, we have that* $p_i p_{i+1} \cap \rho(q_1, F_i, q_2) = \emptyset$ *or* $p_i p_{i+1} \cap \rho(q_2, F_i, q_1) = \emptyset$*, where* $q_j \in \partial F_i$*, for* $j = 1, 2$*, are the intersection points between line segment* $p_i p_{i+1}$ *and* ∂F_i*.*

We recall some definitions from [145]. Let Π be a simple polygon. The vertex v of Π is called *reflex* if v's internal angle is greater than $180°$. Let u be a vertex of Π such that it is adjacent to a reflex vertex v. Let the straight line uv intersect an edge of Π at v'. Then the segment $C = vv'$ partitions Π into two parts. C is called a *cut* of Π, and v is called a *defining vertex* of C. That part of Π is called an *essential* part of C if it does not contain u; it is denoted by $\Pi(C)$. A cut C *dominates* a cut C' if $\Pi(C)$ contains $\Pi(C')$. A cut is called *essential* if it is not dominated by another cut. In Figure 11.6 (which is Figure 1 in [145]), the cuts xx' and yy' are dominated by C_2

[2]If $i = k + 1$, then let $i = i \ mod \ k$.

and C_5, respectively; the cuts C_1, C_2, C_3, C_5 and C_4 are essential. Let $\mathcal{C}$ be the set of all essential cuts. The WRP is reduced to find the shortest route ρ such that ρ visits in order each cut in $\mathcal{C}$ (see Lemma 62, and also see [6] or [29]).

Let $S_C = \{C_1, C_2, \ldots, C_k\}$ be the sorted set $\mathcal{C}$ such that v_i is a defining vertex of C_i, and vertices v_i are located in anti-clockwise order around $\partial\Pi$, for $i = 1, 2, \ldots, k$.

44. DEFINITION. *Let Π and S_C be the input of a watchman route problem. This problem is simplified iff, for each $i \in \{1, 2, \ldots, k-1\}$, for each $p_i \in C_i$ and $p_{i+1} \in C_{i+1}$ we have that $p_i p_{i+1} \cap \rho(v_{i+1}, \Pi, v_i) = \emptyset$.*

11.3 Known Results

We lists some useful results used in the rest of this chapter. The first three are about "order", and the remaining four are about time complexity.

60. LEMMA. ([115], Lemma 2) *A solution to the safari problem (shortest tour) must visit the P_i's in the same order as it meets $\partial\Pi$.*

61. LEMMA. ([40], Lemma 2) *A solution to the zookeeper problem (shortest tour) must visit the P_i's in the same order as it meets $\partial\Pi$.*

62. LEMMA. ([39], Lemma 3.3) *A solution to the watchman route problem (shortest tour) must visit the P_i's in the same order as it meets $\partial\Pi$.*

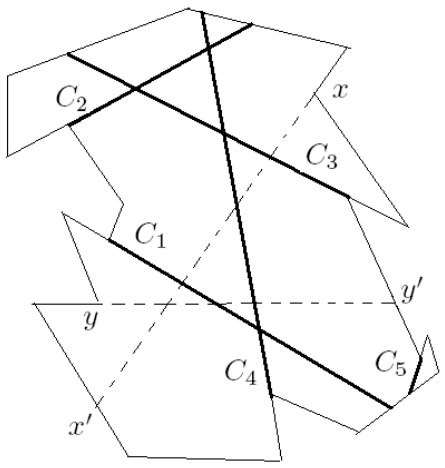


Figure 11.6: Examples for cuts and essential cuts.

63. LEMMA. ([106], pages 639–641) *There exists an $\mathcal{O}(n)$ time algorithm for calculating the shortest path between two points in a simple polygon.*

64. LEMMA. ([132]) *The two straight lines, incident with a given point and being tangents to a given convex polygon, can be computed in $\mathcal{O}(log\ n)$, where n is the number of vertices of the polygon.*

35. THEOREM. *([145], Theorem 1) Given a simple polygon Π, the set C of all essential cuts for the watchman routes in Π can be computed in $\mathcal{O}(n)$ time.*

36. THEOREM. *([52], Theorem 6) The touring polygons problem (TPP) is NP-hard, for any Minkowski metric L_p ($p \geq 1$) in the case of nonconvex polygons P_i, even in the unconstrained ($F_i = \mathbb{R}^2$) case with obstacles bounded by edges having angles 0, 45, or 90 degrees with respect to the x-axis.*

In Section 11.5, we will apply Algorithm 11 to show that the same problem stated in Theorem 36 can be approximately solved in polynomial time.

11.4 The Algorithms

11.4.1 An Algorithm for the Unconstrained TPP

The main algorithm (Algorithm 10) in this subsection answers the open problem in Section 11.1.1 by an approximate algorithm.

Let π be a plane, containing the polygon $V(P_i) \subset \pi$, where $i = 1, 2, \ldots, k$. Suppose that for any i, $j \in \{1, 2, \ldots, k\}$, $\partial P_i \cap \partial P_j = \emptyset$. Let p, $q \in \pi$, $p = p_0$, and $q = p_{k+1}$.

The following algorithm is a simplified version of a rubberband algorithm, actually a modification of Algorithm 3. (The difference between Algorithm 3 and Algorithm 10 is defined by Steps 4.2 and 5.2.)

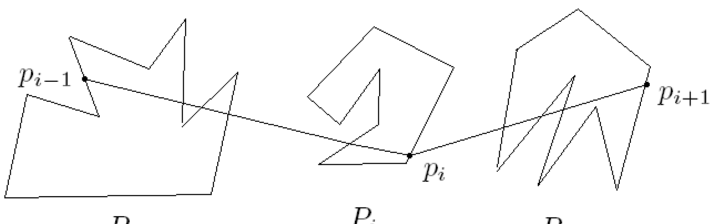


Figure 11.7: Illustration for the initialization for Steps 4.2 and 5.2 in Algorithm 10.

10. ALGORITHM.

```
1. Let $\varepsilon = 10^{-10}$ (the accuracy).
2. Compute the length $l_1$ of the 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 $q_2 \in \partial P_i$ (see the proof of Lemma 57) such that

        $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q') + d_e(q_3, q') : q' \in \partial P_i\}$

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 $q_2 \in \partial P_k$ such that

        $d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q) + d_e(q_3, q) : q \in \partial P_k\}$

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.
```

The algorithms for the safari, zookeeper, constrained TPP, and watchman route problem are also modifications of Algorithm 3.

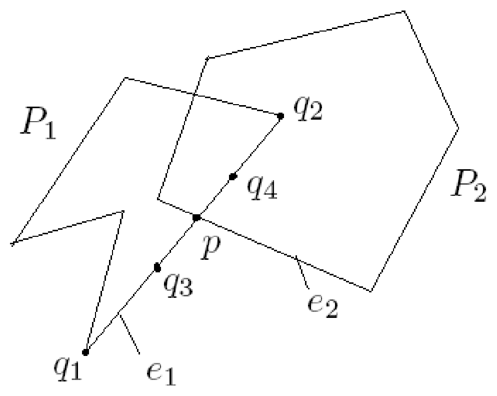


Figure 11.8: Illustration for Procedure 17.

The following procedure handles the degenerate case (see also Section 7.5) of the rubberband algorithm. Such a case may occur and should be dealt with when we apply Algorithm 11 (which is a modified version of Algorithm 10) to the unconstrained TPP when the polygons are not pairwise disjointed.

17. PROCEDURE.

Input: A point p and two polygons P_1 and P_2 such that $p \in \partial P_1 \cap \partial P_2$ (see Figure 11.8).

Output: A point $q \in \partial P_1$ such that $d_e(q, p) \leq \varepsilon$ and $q \notin \partial P_2$.

1. Let $\varepsilon = 10^{-10}$ (the accuracy).
2. Find a point $e_j \in E(P_j)$, where j = 1, 2, such that $p \in e_1 \cap e_2$.
3. Let e_1 = q_1q_2. Let q_3 and q_4 be two points in two segments q_1p and q_2p, respectively (see Figure 11.8) such that $d_e(q_j, p) \leq \varepsilon$ and $q_j \notin \partial P_2$, where j = 3, 4.
4. Let q = $\min\{q_3, q_4\}$ (with respect to lexicographic order).
5. Output q.

If there exist i, $j \in \{1, 2, \ldots, k\}$ such that $i \neq j$ and $\partial P_i \cap \partial P_{i+1} \neq \emptyset$, then we modify Algorithm 10 as follows: Let p_0 = p, p_{k+1} = q, P_0 = p, and P_{k+1} = q. (The difference between Algorithm 11 and Algorithm 10 is defined by Steps 4.1a and 5.1a.) Let $P = \{p, p_1, p_2, \ldots, p_k, q\}$.

11. ALGORITHM.

1. Let $\varepsilon = 10^{-10}$ (the accuracy).

2. Compute the length l_1 of the 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:

4.1a. If $(p_i = p_{i-1} \wedge p_i \neq p_{i+1}) \vee (p_i \neq p_{i-1} \wedge p_i = p_{i+1}) \vee (p_i = p_{i-1} \wedge p_i = p_{i+1})$, then apply Procedure 17 to compute a point p_i such that $p_i \neq p_{i-1}$ and $p_i \neq p_{i+1}$.

4.1. Let $q_3 = p_{i+1}$.

4.2. Compute a point $q_2 \in \partial P_i$ (see the proof of Lemma 57) such that

$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q') + d_e(q_3, q') : q' \in \partial P_i\}$$

4.3. Update P by replacing p_i by q_2.

4.4. Let $q_1 = p_i$ and $i = i$ + 1.

5.1a. If $(p_k = p_{k-1} \wedge p_k \neq p_{k+1}) \vee (p_k \neq p_{k-1} \wedge p_k = p_{k+1}) \vee (p_k = p_{k-1} \wedge p_k = p_{k+1})$, then apply Procedure 17 to compute a point p_k such that $p_k \neq p_{k-1}$ and $p_k \neq p_{k+1}$.

5.1. Let $q_3 = q$.

5.2. Compute $q_2 \in \partial P_k$ such that

$$d_e(q_1, q_2) + d_e(q_3, q_2) = \min\{d_e(q_1, q') + d_e(q_3, q') : q' \in \partial P_k\}$$

5.3. Update P 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.

Section 11.5 applies this algorithm to show that the TPP for possibly non-disjoint nonconvex simple polygons can be approximately computed in polynomial time. By Theorem 36, finding the exact solution of this problem is NP-hard.

11.4.2 Algorithms for Restrictedly Solving the Safari Problem

The following simple procedures will be used in the main algorithm (Algorithm 12 below) of this subsection.

18. PROCEDURE.

Input: Three simple polygons P_1, P_2 and Π such that P_1 and P_2 are contained in Π, and exactly one edge of P_1 and P_2 is incident with the frontier of Π. (They may be non-convex polygons.); two points $p_i \in \partial P_i$, where i = 1, 2.

Output: The set of vertices V of the shortest path from p_1 to p_2, contained in Π.

Case 1. $p_1p_2 \cap \rho(p_1, \Pi, p_2)$ = $\emptyset$. (note: see Figure 11.9 for an example)

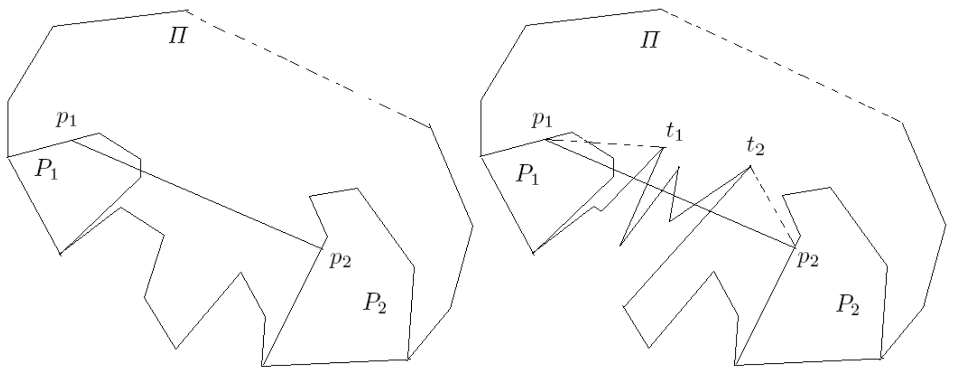


Figure 11.9: Illustration for Procedure 18. Left: Case 1. Right: Case 2.

Output $V = \{p_1, p_2\}$ and Stop.

Case 2. $p_1p_2 \cap \rho(p_1, \Pi, p_2) \neq \emptyset$.

2.1. Compute a vertex t_i in Π such that p_it_i is a tangent of Π, where $i = 1, 2$ (see [132]).
2.2. Apply the Melkman algorithm (see [103]) to compute a convex path $\rho(t_1, t_2)$ from t_1 to t_2, inside of Π.
2.3. Let $V = \{p_1\} \cup V(\rho(t_1, t_2)) \cup \{p_2\}$.
2.4. Output V.

This procedure computes the shortest path between two points p_1 and p_2, located on polygons P_1 and P_2, respectively, inside of the "background" polygon Π. It will be applied in the following procedure.

Obviously, Procedure 18 is still correct if P_1 or P_2 degenerate into a single point (inside of Π).

19. PROCEDURE.

Input: Simple polygons P_1, P_2, P_3, and Π, and three points $p_i \in \partial P_i$, where $i = 1, 2, 3$.
Output: An updated shortest path $\rho(p_1, \ldots, p_2, \ldots, p_3)$, which also contains vertices of polygon Π in general, and which is allowed to intersect P_i°, for $i = 1, 2, 3$.

1. Let P_1, P_2, Π and $p_i \in \partial P_i$, where $i = 1, 2$, be the input for Procedure 18; the output is a set V_{12}.
2. Let P_2, P_3, Π and $p_i \in \partial P_i$, where $i = 2, 3$, be the input for Procedure 18; the output is a set V_{23}.
3. Let $V = V_{12} \cup V_{23}$.
4. Find q_1 and $q_3 \in V$ such that $\{q_1, p_2, q_3\}$ is a subsequence of V (i.e., q_1, p_2, q_3 appear consecutively in V).
5. Find vertex $p_2' \in \partial P_2$ such that

$$d_e(q_1, p_2') + d_e(p_2', q_3) = \min\{d_e(q_1, p') + d_e(p', q_3) : p' \in \partial P_2\}$$

6. Update V by letting $p_2 = p_2'$.

This procedure applies the basic idea of Option 3 of the original rubberband algorithm. It is frequently applied in the following algorithm which approximately solves the simplified safari problem.

Let $p_0 = p = q$ and $P_0 = \{p\} = \{q\}$. (i.e., let P_0 be a polygon which degenerates to a single point.) Let $\varepsilon = 10^{-10}$ (the accuracy).

12. ALGORITHM.

1. For each $i \in \{0, 1, \ldots, k-1\}$, let p_i be a vertex of P_i.
2. Let $V_0 = V = \{p_0, p_1, \ldots, p_{k-1}\}$.
3. Calculate the perimeter L_0 of the polygon, given by the set V_0 of vertices.
4. For each $i \in \{0, 1, \ldots, k-1\}$:

4.1. Let P_{i-1}, P_i, P_{i+1} (mod k) and V_0 be the input for Procedure 19, which updates p_i in V_0.

4.2. Let U_i be the set of vertices of the convex path $\rho(p_i, p_{i+1})$ with respect to P_i and P_{i+1} (inside of Π; it is computed by applying the Melkman algorithm).

4.3. Let $U_i = \{q_1, q_2, \ldots, q_m\}$.

4.4. Insert (after p_i) the points of set U_i (in the given order) into V_0; that means, we have

$$V_1 = \{p_0, p_1, \ldots, p_{i-1}, p_i, q_1, q_2, \ldots, q_m, p_{i+1}, \ldots, p_{k-1}\}$$

(note: set V_1 is the updated set V_0, after inserting U_i)

5. Calculate the perimeter L_1 of the polygon, given by the set V_1 of vertices.
6. If $L_1 - L_0 > \varepsilon$, then let $L_0 = L_1$ and $V_0 = V$ (note: we use the original set V instead of V_1 for the next iteration), and go to Step 4. Otherwise, output set V_1, and the desired length equals L_1.

If the safari problem is not a simplified safari problem, then we can slightly modify Procedure 19: just replace "Procedure 18" in Steps 1 and 2 of Procedure 19 by the algorithm described in [106] on pages 639–641. In this way, Algorithm 12 will still work even for non-simplified safari problems restrictedly.

11.4.3 Algorithms for Restrictedly Solving the Zookeeper Problem

This subsection presents algorithms for restrictedly solving the zookeeper problem by just slightly modifying the algorithms for solving the safari problem, given in the previous subsection. The following Procedure 20 is a modification of Procedure 18.

20. PROCEDURE.

Input: Three simple polygons P_1, P_2 and Π, and two points $p_i \in \partial P_i$, for $i = 1, 2$.

Output: The set V of all vertices of the shortest zookeeper path from p_1 to p_2 inside of Π.

If $p_1p_2 \cap \rho(p_1, \Pi, p_2) = \emptyset$ and $p_1p_2 \cap \partial P_i = \{p_i\}$, for $i = 1$ and $i = 2$,
then output $V = \{p_1, p_2\}$, and Stop.
Otherwise:

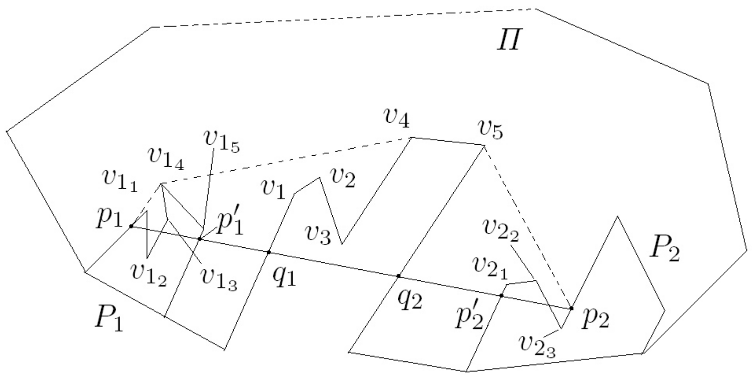


Figure 11.10: Illustration for Procedure 20.

1. Let $p_1p_2 \cap \partial P_i = \{p_i, \ldots, p'_i\}$ (note: p_i and p'_i may be identical), where $i = 1, 2$, and $p_1p_2 \cap \rho(p_1, \Pi, p_2) = \{q_1, \ldots, q_2\}$.

2. Compute $\rho^{-1}(p_1, P_1, p'_1)$, which is the subpath from p_1 to p'_1, clockwise around P_1.

3. Compute $\rho^{-1}(q_1, \Pi, q_2)$, which is the subpath from q_1 to q_2, clockwise around Π.

4. Compute $\rho^{-1}(p'_2, P_2, p_2)$, which is the subpath from p'_2 to p_2, clockwise around P_2.

5. Let $V = V(\rho^{-1}(p_1, P_1, p'_1)) \cup V(\rho^{-1}(q_1, \Pi, q_2)) \cup V(\rho^{-1}(p'_2, P_2, p_2))$.

6. Let V be the input for the Melkman algorithm and compute a convex path $\rho(p_1, p_2)$ from p_1 to p_2 inside of Π.

7. Output V.

Figure 11.10 shows an example: $V(\rho^{-1}(p_1, P_1, p'_1)) = \{p_1, v_{1_1}, v_{1_2}, v_{1_3}, v_{1_4}, v_{1_5}, p'_1\}$, $V(\rho^{-1}(q_1, \Pi, q_2)) = \{q_1, v_1, v_2, v_3, v_4, v_5, q_2\}$, $V(\rho^{-1}(p'_2, P_1, p'_2)) = \{p'_2, v_{2_1}, v_{2_2}, v_{2_3}, p_2\}$, and $V = \{p_1, v_{1_4}, v_4, v_5, q_2\}$. – We also slightly modify Procedure 19 and Algorithm 12 as follows:

21. PROCEDURE.

We modify Procedure 19 at two places:

1. Insert "not" in front of "allow".

2. In Steps 1 and 2, replace "Procedure 18" by "Procedure 20".

13. ALGORITHM.

We modify Algorithm 12 at one place only:

1. Replace "Procedure 19" by "Procedure 21" in Step 4.1.

If the given zookeeper problem is not a simplified zookeeper problem, then we can use a slightly modified Procedure 21: replace "Procedure 20" in Procedure 21 by the algorithm described in [106] on pages 639–641. In this way, Algorithm 13 will still work and solve also a non-simplified zookeeper problem restrictedly.

11.4.4 Algorithms for Restrictedly Solving the Constrained TPP

The following Procedure 22 is obtained by modifying Procedure 18.

22. PROCEDURE.

Input: Two polygons P_1 and P_2 and their fence F; two points $p_i \in \partial P_i$, where i = 1, 2, such that $p_1p_2 \cap \partial F = \{q_1, \ldots, q_2\}$.

Output: The set of all vertices of the shortest path, which starts at p_1, then visits ∂F, and finally ends at p_2 (but not including p_1 and p_2.)

1. Compute $\rho(q_1, F, q_2)$, which is the subpath from q_1 to q_2 inside of F.
2. Apply the Melkman algorithm (see [103]) to compute the convex path from q_1 to q_2 inside of F, denoted by $\rho(q_1, q_2)$.
3. Compute a tangent p_it_i of p_i and $\rho(q_1, q_2)$ such that $p_it_i \cap F = t_i$, where i = 1, 2 (see [132]).
4. Compute $\rho(t_1, F, t_2)$, which is the subpath from t_1 to t_2 inside of F.
5. Output $V(\rho(t_1, F, t_2))$.

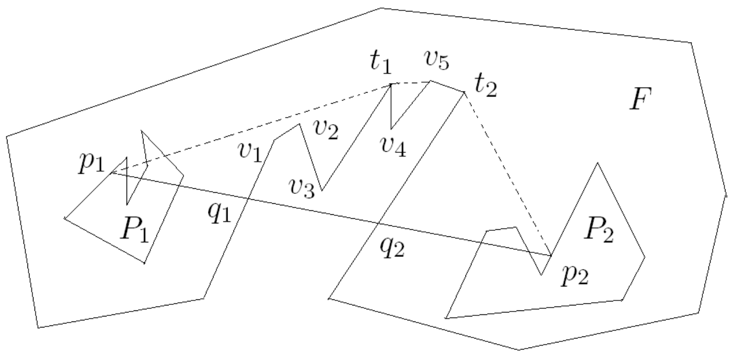


Figure 11.11: Illustration for Procedure 22.

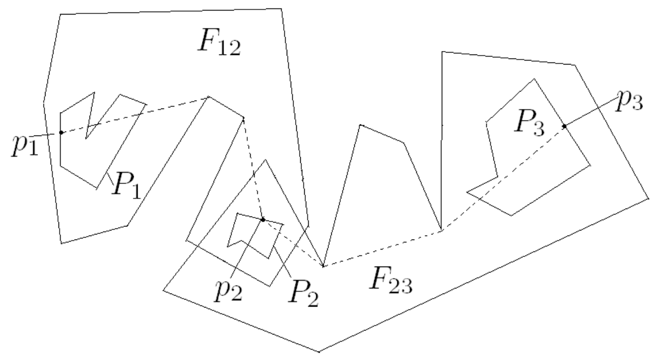


Figure 11.12: Illustration for Procedure 23.

See Figure 11.11 for an example: $V(\rho(q_1, F, q_2)) = \{q_1, v_1, v_2, v_3, t_1, v_4, v_5, t_2, q_2\}$ and $V(\rho(t_1, F, t_2)) = \{t_1, v_4, v_5, t_2\}$. – The following Procedure 23 is a modification of Procedure 19.

23. PROCEDURE.

Input: Three polygons P_1, P_2 and P_3 in order, the fence of P_1 and P_2, denoted by F_{12}, the fence of P_2 and P_3, denoted by F_{23}, and three points $p_i \in \partial P_i$, where i = 1, 2, 3.

Output: The set of all vertices of the shortest path starts at p_1, then visits P_2, and finally ends at p_3 (see Figure 11.12).

1.1. Let $\varepsilon = 10^{-10}$ (the accuracy).

1.2. If $(p_2 = p_1 \wedge p_2 \neq p_3) \vee (p_2 \neq p_1 \wedge p_2 = p_3) \vee (p_2 = p_1 \wedge p_2 = p_3)$, then apply Procedure 17 to compute a point to update p_2 such that $p_2 \neq p_1$ and $p_2 \neq p_3$.

1.3. Compute a point $p_2 \in \partial P_2$ such that

$$d_e(p_1, p_2') + d_e(p_2', p_3) = \min\{d_e(p_1, p') + d_e(p', p_3) : p' \in \partial P_2\}$$

2. Update V by letting $p_2 = p_2'$.
3. If $p_1p_2 \cap F_{12} = \emptyset$, then let $V_{12} = \emptyset$.
4. Otherwise suppose that $p_1p_2 \cap F_{12} = \{q_1, \ldots, q_2\}$.
5. Use P_1, P_2, F_{12}, q_1, and q_2 as input for Procedure 22; the output equals V_{12}.
6. Analogously, compute a set V_{23} from p_2, p_3 and F_{23}, as follows:

6.1. If $p_2p_3 \cap F_{23} = \emptyset$, then let $V_{23} = \emptyset$.

6.2. Otherwise suppose that $p_2p_3 \cap F_{23} = \{q_2', \ldots, q_3'\}$.

6.3. Use P_2, P_3, F_{23}, q'_2, and q'_3 as input for Procedure 22; the output equals V_{23}.
7. Let $V = \{v_1\} \cup V_{12} \cup \{v_2\} \cup V_{23} \cup \{v_3\}$.
8. Find q_1 and $q_3 \in V$ such that $\{q_1, p_2, q_3\}$ is a subsequence of V.
9.1. If $(p_2 = q_1 \wedge p_2 \neq q_3) \vee (p_2 \neq q_1 \wedge p_2 = q_3) \vee (p_2 = q_1 \wedge p_2 = q_3)$, then apply Procedure 17 to compute an update of point p_2 such that $p_2 \neq q_1$ and $p_2 \neq q_3$.
9.2. Find a point $p'_2 \in \partial P_2$ such that

$$d_e(q_1, p'_2) + d_e(p'_2, q_3) = \min\{d_e(q_1, p') + d_e(p', q_3) : p' \in \partial P_2\}$$

10. Update set V by letting $p_2 = p'_2$.
11. Output V.

Note that in Steps 1.2 and 9.1, the updated point p_2 depends on the chosen value of ε. – The following algorithm is a modification of Algorithm 12.

14. ALGORITHM.

1. For each $i \in \{0, 1, \ldots, k-1\}$, let p_i be a point on ∂P_i.
2. Let $V = \{p_0, p_1, \ldots, p_{k-1}\}$.
3. Calculate the perimeter L_0 of the polygon, which has the set V of vertices.
4. For each $i \in \{0, 1, \ldots, k-1\}$, use P_{i-1}, P_i, P_{i+1} (mod k) and F_{i-1}, F_i (mod k) (note: these are the fences of P_{i-1} and P_i, and P_i and P_{i+1} (mod k), respectively) as input for Procedure 23, to update p_i and output a set V_i.
5. Let $V_1 = V$ and update V by replacing $\{p_{i-1}, \ldots, p_i, \ldots, p_{i+1}\}$ by V_i.
6. Let $V = \{q_0, q_1, \ldots, q_m\}$.
7. Calculate the perimeter L_1 of the polygon, which has the set V of vertices,
8. If $L_1 - L_0 > \varepsilon$, then let $L_0 = L_1$, $V = V_1$, and go to Step 4. Otherwise, output the updated set V and (its) calculated length L_1.

If a given constrained TPP is not a simplified constrained TPP, then we can slightly modify Procedure 23: just replace "Procedure 22" in Step 5 of Procedure 23 by the algorithm in [106] on pages 639–641. In this way, Algorithm 14 will still work also for solving a non-simplified constrained TPP restrictedly.

11.4.5 Algorithms for Solving the Watchman Route Problem

The algorithm for solving the simplified constrained TPP (or the "general" constrained TPP) implies the following algorithm for solving the simplified WRP (or the "general" WRP).

15. ALGORITHM.

We modify Algorithm 14 at two places:

1. Replace polygon "P_j" by segment "s_j" in Steps 1 and 4, for $j = i-1, i$, or $i+1$.

2. Replace "F_{i-1}, F_i (mod k)" (the fences of P_{i-1} and P_i, and P_i and P_{i+1}, all mod k, respectively) by the common polygon Π.

11.5 Computational Complexity

This section analyzes (step by step) the time complexity of procedures and algorithms presented above (in this chapter).

11.5.1 Unconstrained TPP

65. LEMMA. *Algorithm 10 can be computed in time*

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V(P_j)|)$$

Proof. The difference between Algorithm 3 and Algorithm 10 is defined by Steps 4.2 and 5.2. By the proof of Lemma 57, Steps 4.2 and 5.2 can be computed in $\mathcal{O}(|V(P_j)|)$ time, where $V(P_j)$ is as in Algorithm 10, for $j = i, k$. Thus, each iteration of Algorithm 10 can be computed in time

$$\mathcal{O}(\sum_{j=1}^{k} |V(P_j)|)$$

Therefore, Algorithm 10 can be computed in

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |V(P_j)|)$$

time. This proves the lemma. □

66. LEMMA. *Procedure 17 can be computed in $\mathcal{O}(|E(P_1)| + |E(P_2)|)$ time.*

Proof. Steps 1 and 5 only need constant time. Step 2 can be computed in time $\mathcal{O}(|E(P_1)| + |E(P_2)|)$, Step 3 in time $\mathcal{O}(|E(P_2)|)$, and Step 4 in time $\mathcal{O}(1)$. Therefore, Procedure 17 can be computed in $\mathcal{O}(|E(P_1)| + |E(P_2)|)$ time. □

67. LEMMA. *Algorithm 11 can be computed in time*

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |E(P_j)|)$$

Proof. The difference between Algorithm 11 and Algorithm 10 is defined by Steps 4.1a and 5.1a. By Lemma 66, Steps 4.1a and 5.1a can be computed in $\mathcal{O}(|E(P_{k-1})| + 2|E(P_k)| + |E(P_{k+1})|)$ time, where j = i, k. Thus, each iteration of Algorithm 11 can be computed in time

$$\mathcal{O}(\sum_{j=1}^{k} |E(P_j)|)$$

Therefore, Algorithm 11 can be computed in

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{j=1}^{k} |E(P_j)|)$$

time. This proves the lemma. □

By Lemmas 65 and 67, we have the following

37. THEOREM. *The unconstrained TPP can be solved in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ time, where n is the total number of vertices of all polygons involved.*

11.5.2 Safari Problem

68. LEMMA. *Procedure 18 can be computed in $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|)$ time.*

Proof. Cases 1 and 2 can be tested in $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|)$ time. By Lemma 64, Step 2.1 can be computed in $\mathcal{O}(log\,|V(\rho(p_1, \Pi, p_2))|)$ time. According to [103], Step 2.2 can be computed in $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|)$ time. Step 2.3 can be computed in time $\mathcal{O}(1)$, and Step 2.4 in time $\mathcal{O}(|V(\rho(t_1, t_2))|)$. Therefore, Procedure 18 can be computed in $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|)$ time. □

69. LEMMA. *Procedure 19 can be computed in time*

$$\mathcal{O}(|V(\rho(p_1, \Pi, p_2))| + |V(\rho(p_2, \Pi, p_3))| + |E(P_2)|)$$

Proof. By Lemma 68, Steps 1 and 2 can be computed in time $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|)$ and $\mathcal{O}(|V(\rho(p_2, \Pi, p_3))|)$, respectively. Step 3 can be computed in time $\mathcal{O}(|V_{12}| + |V_{23}|)$. Since $|V_{12}| \leq |V(\rho(p_1, \Pi, p_2))|$ and $|V_{23}| \leq |V(\rho(p_2, \Pi, p_3))|$, Steps 3 and 4 can be computed in time $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))| + |V(\rho(p_2, \Pi, p_3))|)$. By the proof of Lemma 57, Step 5 can be computed in time $\mathcal{O}(|E(P_2)|)$. Step 6 can be computed in time $\mathcal{O}(1)$. Altogether, Procedure 19 can be computed in

$$\mathcal{O}(|V(\rho(p_1, \Pi, p_2))| + |V(\rho(p_2, \Pi, p_3))| + |E(P_2)|)$$

time. This proves the lemma. □

70. LEMMA. *Algorithm 12 can be computed in time*

$$\kappa(\varepsilon) \cdot \mathcal{O}(|V(\Pi)| + \sum_{i=1}^{k} |E(P_i)|)$$

Proof. Steps 1–3 can be computed in $\mathcal{O}(k)$ time. By Lemma 69, Step 4.1 can be computed in time

$$\mathcal{O}(|V(\rho(p_{i-1}, \Pi, p_i))| + |V(\rho(p_i, \Pi, p_{i+1}))| + |E(P_i)|)$$

According to [103], Step 4.2 can be computed in $\mathcal{O}(|V(\rho(p_i, p_{i+1}))|)$ time. Steps 4.3 and 4.4 can be computed in $\mathcal{O}(|U_i|)$ time. Since $U_i \subset V(\Pi)$, Steps 4.3 and 4.4 can be computed in time $\mathcal{O}(|V(\rho(p_i, p_{i+1}))|)$. Note that

$$V(\rho(p_i, p_{i+1})) \subset V(\rho(p_i, \Pi, p_{i+1}))$$

Thus, each iteration in Step 4 can be computed in time

$$\mathcal{O}(\sum_{i=1}^{k}(|V(\rho(p_i, \Pi, p_{i+1}))| + |E(P_i)|))$$

what is equivalent to time

$$\mathcal{O}(|V(\Pi)| + \sum_{i=1}^{k} |E(P_i)|)$$

Step 5 can be computed in $\mathcal{O}(|V(\Pi)|)$ time. Step 6 can be computed in constant time. Therefore, Algorithm 12 can be computed in

$$\kappa(\varepsilon) \cdot \mathcal{O}(|V(\Pi)| + \sum_{i=1}^{k} |E(P_i)|)$$

time. This proves the lemma. □

Lemma 70 allows to conclude the following

38. THEOREM. *The simplified safari problem can be solved restrictedly in $\kappa(\varepsilon)\cdot\mathcal{O}(n+m_k)$ time, where n is the number of vertices of polygon Π, and m_k is the total number of vertices of all polygons P_i, for $i = 1, 2, \ldots, k$.*

At the end of this subsection, we finally discuss the time complexity of the proposed algorithm for the general safari problem. Consider that we modify Procedure 19 as follows: replace "Procedure 18" in Steps 1 and 2 in Procedure 19 by the algorithm described in [106] on pages 639–641. Then we have the following

71. LEMMA. *The modified Procedure 19 can be computed in time $\mathcal{O}(|V(\Pi)|)$.*

Proof. By Lemma 63, Steps 1 and 2 can be computed in $\mathcal{O}(2|V(\Pi)| + |E(P_2)|)$ time. Steps 3–6 can be analyzed in the same way as in the proof of Lemma 69. Therefore, the modified Procedure 19 can be computed in $\mathcal{O}(2|V(\Pi)| + |E(P_2)|)$ time. □

Therefore, if the safari problem is not necessarily simplified, then we have the following

39. THEOREM. *The general safari problem can be solved restrictedly in $\kappa(\varepsilon)\cdot\mathcal{O}(kn+m_k)$ time, where n is the number of vertices of polygon Π, and m_k is the total number of all vertices of polygons P_i, for $i = 1, 2, \ldots, k$; k is the number of polygons.*

11.5.3 Zookeeper Problem

72. LEMMA. *Procedure 20 can be computed in time*

$$\mathcal{O}(|V(\rho(p_1,\Pi,p_2))|) + \mathcal{O}(|V(P_1)| + |V(P_2)|)$$

Proof. The condition "$p_1p_2 \cap \rho(p_1,\Pi,p_2) = \emptyset$" can be tested in $\mathcal{O}(|V(\rho(p_1,\Pi,p_2))|)$ time. The conditions "$p_1p_2 \cap \partial P_i = \{p_i\}$, where i = 1 and 2" can be tested in time $\mathcal{O}(|V(P_1)| + |V(P_2)|)$. Thus, all these conditions can be tested in time

$$\mathcal{O}(|V(\rho(p_1,\Pi,p_2))|) + \mathcal{O}(|V(P_1)| + |V(P_2)|)$$

Step 1 can be computed in $\mathcal{O}(|V(P_1)| + |V(P_2)|) + \mathcal{O}(|V(\rho(p_1,\Pi,p_2))|)$ time. Steps 2, 3 and 4 can be computed in time $\mathcal{O}(|V(\rho^{-1}(p_1,P_1,p_1'))|)$, $\mathcal{O}(|V(\rho^{-1}(q_1,\Pi,q_2))|)$, and $\mathcal{O}(|V(\rho^{-1}(p_2',P_2,p_2))|)$, respectively. Step 5 can be computed in time

$$\mathcal{O}(|V(\rho^{-1}(p_1,P_1,p_1'))|) + \mathcal{O}(|V(\rho^{-1}(q_1,\Pi,q_2))|) + \mathcal{O}(|V(\rho^{-1}(p_2',P_2,p_2))|)$$

Steps 6 (see [103]) and 7 can be computed in time

$$\mathcal{O}(|V(\rho^{-1}(p_1, P_1, p_1'))|) + \mathcal{O}(|V(\rho^{-1}(q_1, \Pi, q_2))|) + \mathcal{O}(|V(\rho^{-1}(p_2', P_2, p_2))|)$$

Note that

$$\begin{aligned}|V(\rho^{-1}(p_1, P_1, p_1'))| &\le |V(P_1)|\\ |V(\rho^{-1}(p_2', P_2, p_2))| &\le |V(P_2)|\\ |V(\rho^{-1}(q_1, \Pi, q_2))| &\le |V(\rho(p_1, \Pi, p_2))|\end{aligned}$$

Therefore, Procedure 20 can be computed in

$$\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|) + \mathcal{O}(|V(P_1)| + |V(P_2)|)$$

time. This proves the lemma. □

73. LEMMA. *Procedure 21 can be computed in time*

$$\mathcal{O}(|V(\rho(p_1, \Pi, p_2))| + |V(\rho(p_2, \Pi, p_3))| + \mathcal{O}(|V(P_1)| + 2|V(P_2)| + |V(P_3)|)$$

Proof. By Lemma 72, Steps 1 and 2 can be computed in

$$\mathcal{O}(|V(\rho(p_1, \Pi, p_2))|) + \mathcal{O}(|V(P_1)| + |V(P_2)|)$$

and

$$\mathcal{O}(|V(\rho(p_2, \Pi, p_3))|) + \mathcal{O}(|V(P_2)| + |V(P_3)|)$$

time, respectively. – The rest of the proof is exactly the same as that of Lemma 69: Step 3 can be computed in time $\mathcal{O}(|V_{12}| + |V_{23}|)$. Since $|V_{12}| \le |V(\rho(p_1, \Pi, p_2))|$ and $|V_{23}| \le |V(\rho(p_2, \Pi, p_3))|$, Steps 3 and 4 can be computed in time $\mathcal{O}(|V(\rho(p_1, \Pi, p_2))| + |V(\rho(p_2, \Pi, p_3))|)$. By the proof of Lemma 57, Step 5 can be computed in $\mathcal{O}(|E(P_2)|)$ time. Step 6 can be computed in constant time. Since $|V(P_1)| = |E(P_1)|$, $|V(P_2)| = |E(P_2)|$, and $|V(\Pi)| = |E(\Pi)|$, Procedure 21 can be computed in

$$\mathcal{O}(|V(\rho(p_1, \Pi, p_2))| + |V(\rho(p_2, \Pi, p_3))| + \mathcal{O}(|V(P_1)| + 2|V(P_2)| + |V(P_3)|)$$

time. This proves the lemma. □

74. LEMMA. *Algorithm 13 can be computed in time*

$$\kappa(\varepsilon) \cdot \mathcal{O}(|V(\Pi)| + \sum_{i=1}^{k} |E(P_i)|)$$

Proof. Note that only the consideration of Step 4.1 makes a difference to the proof of Lemma 70. [Steps 1–3 can be computed in $\mathcal{O}(k)$.]

By Lemma 73, Step 4.1 can be computed in

$$\mathcal{O}(|V(\rho(p_{i-1}, \Pi, p_i))| + |V(\rho(p_i, \Pi, p_{i+1}))| + \mathcal{O}(|V(P_{i-1})| + 2|V(P_i)| + |V(P_{i+1})|)$$

time. – According to [103], Step 4.2 can be computed in $\mathcal{O}(|V(\rho(p_i, p_{i+1}))|)$ time. Steps 4.3 and 4.4 can be computed in $\mathcal{O}(|U_i|)$ time. Since $U_i \subset V(\Pi)$, Steps 4.3 and 4.4 can be computed in $\mathcal{O}(|V(\rho(p_i, p_{i+1}))|)$ time. Thus, each iteration in Step 4 can be computed in

$$\mathcal{O}(\sum_{i=1}^{k}(|V(\rho(p_i, p_{i+1}))| + |E(P_i)|))$$

time, which is equivalent to

$$\mathcal{O}(|V(\Pi)| + \sum_{i=1}^{k}|E(P_i)|)$$

Step 5 can be computed in $\mathcal{O}(|V(\Pi)|)$, and Step 6 in constant time. Therefore, Algorithm 12 can be computed in

$$\kappa(\varepsilon) \cdot \mathcal{O}(|V(\Pi)| + \sum_{i=1}^{k}|E(P_i)|)$$

time. This proves the lemma. □

Lemma 74 allows to conclude the following:

40. THEOREM. *The simplified zookeeper problem can be solved restrictedly in $\mathcal{O}(n + m_k)$ computation time, where n is the number of vertices of polygon Π, and m_k is the total number of all vertices of polygons P_i, for $i = 1, 2, \ldots, k$.*

Analogous to the proof of Theorem 39, we also have the following

41. THEOREM. *The general Zookeeper problem can be solved restrictedly in time*

$$\kappa(\varepsilon) \cdot \mathcal{O}(kn + m_k)$$

where n is the number of vertices of polygon Π, and m_k is the total number of all vertices of polygons P_i, for $i = 1, 2, \ldots, k$.

These two theorems answer (restrictedly, partially, and approximately) the open problem stated in [106] at page 685, addressing the zookeeper problem. With reference to Example 8, our result is best possible in some sense.

11.5.4 Constrained TPP

75. LEMMA. *Procedure 22 can be computed in $\mathcal{O}(|V(\rho(q_1, F, q_2))|)$ time.*

Proof. Step 1 can be computed in $\mathcal{O}(|V(\rho(q_1, F, q_2))|)$ time. According to [103], Step 2 can be computed in $\mathcal{O}(|V(\rho(q_1, F, q_2))|)$ time. By Lemma 64, Step 3 can be computed in $\mathcal{O}(log\ |V(\rho(q_1, F, q_2))|)$ time. Steps 4 and 5 can be computed in $\mathcal{O}(|V(\rho(t_1, F, t_2))|)$ time. Altogether, the time complexity of Procedure 22 is equal to $\mathcal{O}(|V(\rho(q_1, F, q_2))|)$. □

76. LEMMA. *Procedure 23 can be computed in time*

$$\mathcal{O}(|E(P_1)| + 2|E(P_2)| + |E(P_3)| + |E(F_{12})| + |E(F_{23})|)$$

Proof. Step 1.1 requires only constant time. By Lemma 66, Steps 1.2 and 9.1 can be computed in time $\mathcal{O}(|E(P_1)|+2|E(P_2)|+|E(P_3)|)$. Steps 1.3 and 9.2 can be computed in $\mathcal{O}(|E(P_2)|)$, and Steps 2 and 10 in $\mathcal{O}(1)$) time. Steps 3–4 can be computed in time $\mathcal{O}(|E(F_{12})|)$. By lemma 75, Step 5 can be computed in time $\mathcal{O}(|V(\rho(q_1, F_{12}, q_2))|)$ = $\mathcal{O}(|V_{12}|)$. Since $|V_{12}| \leq |E(F_{12})|$, Steps 3–5 can be computed in $\mathcal{O}(|E(F_{12})|)$ time. Step 6 can be computed in $\mathcal{O}(|E(F_{23})|)$, and Steps 7, 8 and 11 in $\mathcal{O}(|V_{12}|)$ + $\mathcal{O}(|V_{23}|)$ $\leq \mathcal{O}(|E(F_{12})|)$ + $\mathcal{O}(|E(F_{23})|)$ time.

Therefore, Procedure 23 can be computed in

$$\mathcal{O}(|E(P_1)| + 2|E(P_2)| + |E(P_3)| + |E(F_{12})| + |E(F_{23})|)$$

time. This proves the lemma. □

77. LEMMA. *Algorithm 14 can be computed in time $\kappa(\varepsilon)\cdot\mathcal{O}(n)$, where n is the total number of all vertices of the polygons involved.*

Proof. Steps 1–3 can be computed in $\mathcal{O}(k)$ time. By Lemma 76, each iteration in Step 4 can be computed in time

$$\mathcal{O}(\sum_{i=0}^{k-1}(|E(P_{i-1})| + 2|E(P_i)| + |E(P_{i+1})| + |E(F_{i-1})| + |E(F_i)|))$$

Steps 5 and 8 can be computed in $\mathcal{O}(k)$, and Steps 6 and 7 in $\mathcal{O}(|V|)$ time. Note that

$$|V| \leq \sum_{i=0}^{k-1}(|V(P_i)| + |V(F_i)|)$$

$|V(P_i)| = |E(P_i)|$, and $|V(F_i)| = |E(F_i)|$, where i = 0, 1, ..., $k-1$. Thus, each iteration (Steps 4–8) in Algorithm 14 can be computed in time

$$\mathcal{O}(\sum_{i=0}^{k-1}(|E(P_{i-1})| + 2|E(P_i)| + |E(P_{i+1})| + |E(F_{i-1})| + |E(F_i)|))$$

Therefore, Algorithm 14 can be computed in

$$\kappa(\varepsilon) \cdot \mathcal{O}(\sum_{i=0}^{k-1}(|E(P_{i-1})| + 2|E(P_i)| + |E(P_{i+1})| + |E(F_{i-1})| + |E(F_i)|))$$

time, where each index is taken mod k. This time complexity is equivalent to $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ where n is the total number of vertices of all polygons. □

Lemma 77 allows to conclude the following

42. THEOREM. *The simplified constrained TPP can be solved restrictedly and approximately in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ time, where n is the total number of all vertices of involved polygons.*

At the end of this subsection, we finally discuss the case when the constrained TPP is not simplified. In this case, we replaced "Procedure 22" in Step 5 of Procedure 23 by the algorithm in [106] on pages 639–641. Then, by Lemma 63 and the proof of Lemma 76, Steps 3–5 can *still* be computed in $\mathcal{O}(|E(F_{12})|)$ time, and Step 6 can *still* be computed in $\mathcal{O}(|E(F_{23})|)$ time. All the other steps are analyzed in exactly the same way as those in the proof of Lemma 76. Therefore, the modified Procedure 23 has the same time complexity as the original procedure. Thus, we have the following

43. THEOREM. *The constrained TPP can be solved restrictedly and approximately in $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ time, where n is the total number of all vertices of involved polygons.*

According to Theorem 36, Theorem 43 is the best possible result in some sense.

11.5.5 Watchman Route Problem

We consider the time complexity of Algorithm 15 for solving the simplified watchman route problem.

78. LEMMA. *Consider Procedure 23. If P_i is an essential cut, for i = 1, 2, 3, and $F_{12} = F_{23} = \Pi$, then this procedure can be computed in time $\mathcal{O}(|V(\rho(v_1, \Pi, v_3))|)$, where v_i is the defining vertex of the essential cut P_i, for i = 1, 3.*

Proof. Step 1.1 requires constant time only. By Lemma 66, Steps 1.2 and 9.1 can be computed in $\mathcal{O}(|E(P_1)| + 2|E(P_2)| + |E(P_3)|) = \mathcal{O}(1)$ time. Steps 1.3 and 9.2 can be computed in $\mathcal{O}(|E(P_2)|) = \mathcal{O}(1)$ time. Steps 2 and 10 can be computed in constant time. Steps 3–4 can be computed in time $\mathcal{O}(|E(\rho(v_1, \Pi, v_2))|)$, where v_i is the defining vertex of the essential cut P_i, for $i = 1, 2$.

By Lemma 75, Step 5 can be computed in $\mathcal{O}(|V(\rho(v_1, \Pi, v_2))|)$ time. Thus, Steps 3–5 can be computed in $\mathcal{O}(|V(\rho(v_1, \Pi, v_2))|)$ time. Analogously, Step 6 can be computed in time $\mathcal{O}(|V(\rho(v_2, \Pi, v_3))|)$, where v_i is the defining vertex of the essential cut P_i, for $i = 2, 3$.

Steps 7, 8 and 11 can be computed in $\mathcal{O}(|V(\rho(v_1, \Pi, v_2))|) + \mathcal{O}(|V(\rho(v_2, \Pi, v_3))|) = \mathcal{O}(|V(\rho(v_1, \Pi, v_3))|)$ time. Therefore, the time complexity of Procedure 23 is equal to $\mathcal{O}(|V(\rho(v_1, \Pi, v_3))|)$. □

79. LEMMA. *Consider Algorithm 14. If P_i is an essential cut, and $Q_i = \Pi$, for i = 0, 1, ..., $k-1$, then this algorithm can be computed in time $\kappa(\varepsilon) \cdot \mathcal{O}(n+k)$, where n is the total number of vertices of Π, and k the number of essential cuts.*

Proof. Steps 1–3 can be computed in $\mathcal{O}(k)$ time. By Lemma 78, each iteration in Step 4 can be computed in time

$$\mathcal{O}(\sum_{i=0}^{k-1}(|V(\rho(v_{i-1}, \Pi, v_{i+1}))|))$$

where v_i is the defining vertex of the essential cut P_i, for $i = 0, 1, \ldots, k-1$. This time complexity is actually equivalent to $\mathcal{O}(|V(\Pi)|)$. Steps 5 and 8 can be computed in $\mathcal{O}(k)$, and Steps 6 and 7 in $\mathcal{O}(|V|)$ time. Note that $|V| \leq k + |V(\Pi)|$, and $|V(P_i)| = 2$, where i = 0, 1, ..., $k-1$, and $\sum_{i=0}^{k-1}(|V(\rho(v_{i-1}, \Pi, v_{i+1}))|) = 2|E(\Pi)| = 2|V(\Pi)|$. Thus, each iteration (Steps 4–8) in Algorithm 14 can be computed in time $\mathcal{O}(n+k)$. Therefore, Algorithm 14 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(n+k)$ time, where n is the total number of vertices of Π, and k the number of essential cuts. □

Lemma 79 allows to conclude the following

44. THEOREM. *The simplified WRP can be solved approximately in $\kappa(\varepsilon) \cdot \mathcal{O}(n+k)$ time, where n is the number of vertices of polygon Π, and k the number of essential cuts.*

If the WRP is not simplified, then we have the following

80. LEMMA. *Consider Algorithm 14. If P_i is an essential cut, and $F_i = \Pi$, for i = 0, 1, ..., $k-1$, then this algorithm can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(kn)$ time, where n is the total number of vertices of Π, and k the number of essential cuts.*

Proof. Steps 1–3 can be computed in $\mathcal{O}(k)$ time. By Lemma 78, each iteration in Step 4 can be computed in

$$\mathcal{O}(\sum_{i=0}^{k-1}(|E(P_{i-1})| + 2|E(P_i)| + |E(P_{i+1})| + |E(F_{i-1})| + |E(F_i)|))$$

time, what is equivalent to the asymptotic class $\mathcal{O}(k|E(\Pi)|)$, because of $|E(P_{i-1})| = |E(P_i)| = |E(P_{i+1})| = 1$ and $|E(F_{i-1})| = |E(F_i)| = |E(\Pi)| = |V(\Pi)|$. Steps 5 and 8 can be computed in $\mathcal{O}(k)$, and Steps 6 and 7 in $\mathcal{O}(|V|) \leq |V(\Pi)|$ time.

Thus, each iteration (Steps 4–8)) in Algorithm 14 can be computed in $\mathcal{O}(kn)$ time. Therefore, Algorithm 14 can be computed in $\kappa(\varepsilon) \cdot \mathcal{O}(kn)$ time, where n is the total number of vertices of Π, and k the number of essential cuts. □

This lemma and Theorem 43 allow to conclude the following

45. THEOREM. *The general WRP can be solved approximately in $\kappa(\varepsilon) \cdot \mathcal{O}(kn)$ time, where n is the number of vertices of polygon Π, and k the number of essential cuts.*

11.6 Proofs of Correctness

We generalize the methodology developed for Chapter 4 to prove the correctness of the algorithms described above (in this chapter). We select and modify definitions, lemmas and theorems already given in Chapter 4.

11.6.1 Definitions

Algorithm 12 (13, 14, 15) is called *s-rubberband algorithm (z-rubberband algorithm, t-rubberband algorithm, w-rubberband algorithm)*. For the specification of the step sets of the s-rubberband algorithm (z-rubberband algorithm, w-rubberband algorithm), see Lemma 60 (61, 62). The step sets of the t-rubberband algorithm are also defined (in Section 11.6.2). At first, we focus on the w-rubberband algorithm. A *single iteration* of a w-rubberband algorithm is a complete run through the main loop of the algorithm. – Let Π be a simple polygon.

45. DEFINITION. *Let $P = (p_0, p_1, p_2, \ldots, p_m, p_{m+1})$ be a critical point tuple of Π. Using P as an initial point set, and n iterations of the w-rubberband algorithm, we get another critical point tuple of Π, say $P' = (p'_0, p'_1, p'_2, \ldots, p'_m, p'_{m+1})$. The polygon with vertex set $\{p'_0, p'_1, p'_2, \ldots, p'_m, p'_{m+1}\}$ is called an* nth polygon *of Π, denoted by $AESP_n(\Pi)$, or (for short) by $AESP_n$, where n = 1, 2, ...*

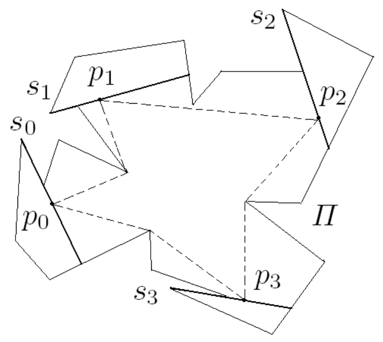


Figure 11.13: Illustration for Definition 46.

Let p, $q \in \Pi^\bullet$. Let $d_{ESP}(\Pi, p, q)$ be the length of the shortest path between p and q inside of Π. Let $AESP_n(\Pi)$ be an nth polygon fully contained inside of Π, where $n \geq 1$. Let

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

Let $p_i(t_{i_0})$ be the i-th vertex of $AESP$, for $i = 0, 1, \ldots$, or $m + 1$. Let

$$d_{ESP_i} = d_{ESP}(\Pi, p_{i-1}, p_i) + d_{ESP}(\Pi, p_i, p_{i+1})$$

where $i = 1, 2, \ldots$, or m. Let

$$d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1}) = \sum_{i=1}^{m} d_{ESP_i}$$

46. DEFINITION. *Let s_0, s_1, s_2, ... s_m and s_{m+1} be a sequence of segments fully contained in Π and located around the frontier of a simple polygon Π, with points $p_i \in \partial s_i$ (see Figure 11.13),* [3] *for $i = 0, 1, 2, \ldots, m$ or $m + 1$. We call the $m + 2$ tuple (p_0, p_1, p_2, ..., p_m, p_{m+1}) a* critical point tuple *of* Π. *We call it an* AESP critical point tuple *of* Π *if it is the set of all vertices of an $AESP$ of* Π.

47. DEFINITION. *Let*

$$\frac{\partial d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1})}{\partial t_i}|_{t_i = t_{i_0}} = 0$$

[3]Possibly, the straight segment $p_i p_{i+1}$ is not fully contained in Π. In this case, replace $p_i p_{i+1}$ by the ESP(Π, p_i, p_{i+1}) between p_i and p_{i+1}, where $i = 1, 2, \ldots$, or m.

for i = 0, 1, …, or $m+1$. Then we say that $(t_{0_0}, t_{1_0}, \ldots, t_{m_0}, t_{m+1_0})$ is a critical point *of*

$$d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1})$$

48. DEFINITION. *Let $P = (p_0, p_1, p_2, \ldots, p_m, p_{m+1})$ be a critical point tuple of Π. Using P as an initial point set and n iterations of the w-rubberband algorithm, we calculate an* n-w-rubberband transform *of P, denoted by $P\overrightarrow{(w-r-b)_n}Q$, or $P \rightarrow Q$ for short, where Q is the resulting critical point tuple of Π, and n is a positive integer.*

11.6.2 A Correctness Proof for the w-Rubberband Algorithm

We provide mathematical fundamentals for our proof that the w-rubberband algorithm is correct for any input (see Theorem 48 below). At first we recall some basic definitions and theorems from topology, using the book [109] as a reference for basic notions such as topology, topologic space, compact, cover, open cover and so forth. The following is the well-known Heine-Borel Theorem for E_n (see [109], Corollary 2.2 on page 128):

46. THEOREM. *A subset S of $\mathbb{R}^n$ is compact iff S is closed and bounded.*

Now let X be a set, and let d be a metric on $X \times X$ defining a metric space (X, d), for example, such as the Euclidean space $(\mathbb{R}^n, d)$. A map f of a metric space (X, d) into a metric space (Y, e) is *uniformly continuous* iff, for each $\varepsilon > 0$, there is a $\delta > 0$ such that $e(f(x), f(y)) < \varepsilon$, for any x, $y \in X$ with $d(x, y) < \delta$. Another well-known theorem (see [109], Theorem 3.2, page 84) is the following:

47. THEOREM. *Let f be a map of a metric space (X, d) into a metric space (Y, e). If f is continuos and X is compact, then f is uniformly continuous.*

Now let s_0, s_1, s_2, …, s_m and s_{m+1} be a sequence of segments fully contained in a simple polygon Π, and located around (see Figure 11.13) the frontier of Π. We express a point $p_i(t_i) = (x_i + k_{x_i}t_i, y_i + k_{y_i}t_i, z_i + k_{z_i}t_i)$ on s_i this way in general form, with $t_i \in \mathbb{R}$, for i = 0, 1, …, or $m+1$. In the following, $p_i(t_i)$ will be denoted by p_i for short, where i = 0, 1, …, or $m+1$.

81. LEMMA. *$(t_{0_0}, t_{1_0}, \ldots, t_{m_0}, t_{m+1_0})$ is a critical point of $d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1})$.*

Proof. $d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1})$ is differentiable at each point

$$(t_0, t_1, \ldots, t_m, t_{m+1}) \in [0, 1]^{m+2}$$

Because $AESP_n(\Pi)$ is an nth polygon of Π, where n = 1, 2, ... and

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

it follows that

$$d_{ESP}(t_{0_0}, t_{1_0}, \ldots, t_{m_0}, t_{m+1_0})$$

is a local minimum of

$$d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1})$$

By Theorem 9, $\frac{\partial d_{ESP}}{\partial t_i} = 0$, for i = 0, 1, 2, . . ., $m + 1$. □

The following lemma is a generalization of Lemma 16. Let s_0, s_1 and s_2 be three segments. Let $p_i(p_{i_1}, p_{i_2}, p_{i_3}) \in s_i$, for i = 0, 1, 2.

82. LEMMA. *There exists a unique point $p_1 \in s_1$ such that*

$$d_e(p_1, p_0) + d_e(p_1, p_2) = \min\{d_e(p', p_0) + d_e(p', p_2) : p' \in \partial s_2\}$$

Proof. We transform the segment from the original 2D coordinate system into a new 2D coordinate system such that s_1 is parallel to one axis, say, the x-axis. Then follow the proof of Lemma 16. □

Lemma 82 and Algorithm 15 define a function f_w, which maps $[0,1]^{m+2}$ into $[0,1]^{m+2}$. A point p_i is represented as follows:

$$(a_{i_1} + (b_{i_1} - a_{i_1})t_i, a_{i_2} + (b_{i_2} - a_{i_2})t_i, a_{i_3} + (b_{i_3} - a_{i_3})t_i)$$

Then, following the proof of Lemma 82, we obtain

83. LEMMA. *Function $t_2 = t_2(t_1, t_3)$ is continuous at each $(t_1, t_3) \in [0,1]^2$.*

84. LEMMA. *If $P\overrightarrow{(w - r - b)_1}Q$, then, for every sufficient small real $\varepsilon > 0$, there is a sufficiently small real $\delta > 0$ such that $P' \in U_\delta(P)$, and $P'\overrightarrow{(r - b)_1}Q'$ implies $Q' \in U_\varepsilon(Q)$.*

Proof. This lemma follows from Lemma 82; also note that Π has $m + 2$ segments, that means we apply Lemma 83 repeatedly, $m + 2$ times. □

By Lemma 84, we have the following

85. LEMMA. *If $P\overrightarrow{(w - r - b)_n}Q$, 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 $P'\overrightarrow{(w - r - b)_{n'}}Q'$ implies that $Q' \in U_\varepsilon(Q)$, where n' is an integer and $n' > N_\varepsilon$.*

86. LEMMA. *Function f_w is uniformly continuous in $[0,1]^{m+2}$.*

Proof. By Lemma 85, f_w is continuous in $[0,1]^{m+2}$. Since $[0,1]^{m+2}$ is a compact and bounded set, by Theorem 47, f_w is uniformly continuous in $[0,1]^{m+2}$. □

By Lemma 86, we are now able to construct an open cover for $[0,1]^{m+2}$ as follows:

1. For each $t_i \in$ [0,1]:

1.1. Let

$$P = (p_0(t_0), p_1(t_1), p_2(t_2), \ldots, p_m(t_m), p_{m+1}(t_{m+1}))$$

be a critical point tuple such that $p(t_i) \in s_i$, for i = 0, 1, 2, . . ., $m + 1$.

1.2. By Lemma 86, there exists a $\delta > 0$ such that, for each $Q \in U_\delta(P) \cap [0,1]^{m+2}$, it is true that $f_w(P) = Q$.

1.3. Let $U'_\delta(P) = U_\delta(P) \cap [0,1]^{m+2}$.

1.4. Let $S_\delta = \{U'_\delta(P) : t_i \in [0,1]\}$.

By Theorem 46, there exists a subset of S_δ, denoted by S'_δ, such that, for each t'_i $\in$ [0,1], and a critical point tuple

$$P' = (p'_0(t'_0), p'_1(t'_1), p'_2(t'_2), \ldots, p'_m(t'_m), p'_{m+1}(t'_{m+1}))$$

there exists $U'_\delta(P) \in S'_\delta$ such that $P' \in U'_\delta(P)$. This proves the following:

87. LEMMA. *The number of critical points of*

$$d_{ESP}(t_0, t_1, \ldots, t_m, t_{m+1})$$

in $[0,1]^{m+2}$ *is finite.*

Furthermore, in analogy to the proof of Lemma 24, we also have the following:

88. LEMMA. Π *has a unique* $AESP$ *critical point tuple.*

Analogously to the proof of Theorem 11, we obtain

48. THEOREM. *The* $AESP$ *of* Π *is the shortest watchman route of* Π.

9. COROLLARY. *For each* $\varepsilon > 0$, *the* w*-rubberband algorithm computes an approximative solution* ρ' *to a WRP such that* $|l(\rho') - l(\rho)| < \varepsilon$, *where* ρ *is the true solution to the same WRP.*

10. COROLLARY. *The WRP has a unique solution.*

Obviously, Corollary 10 is a stronger result than Theorem 34.

11.6.3 A Note for the $t(s, z)$-Rubberband Algorithm

Let $P_0, P_1, \ldots, P_{k-1}$ be a sequence of simple polygons (not necessarily convex).

49. THEOREM. *For any constrained TPP, there exists a finite number of local minima only.*

Proof. Let F_i be a fence of P_i and P_{i+1} (mod k), for i = 0, 1, …, k - 1. For each segment s, let $\partial s = \{p : p \in s\}$ (i.e., $\partial s = s$). For each polygon P, let $\partial P = \{p : p \in e \wedge e \in E(P)\}$, where $E(P)$ is the set of all edges of P. Let (as standard)

$$\prod_{i=0}^{k-1} \partial P_i = \partial P_0 \times \partial P_1 \times \cdots \partial P_{k-1}$$

For each $p_i \in \partial P_i$, there exists a unique constrained ESP

$$\rho(p_i q_{i_1} q_{i_2} \cdots q_{i_{m_i}} p_{i+1}) \subset F_i^\bullet$$

such that $q_{i_j} \in V(F_i)$, where j = 0, 1, …, m_i, $m_i \geq 0$ and i = 0, 1, …, k - 1. For a set S, let $\wp(S)$ be the power set of S.

We define a map f from

$$\prod_{i=0}^{k-1} \partial P_i \qquad \text{to} \qquad \prod_{i=0}^{k-1} \wp(V(F_i))$$

such that

$$f : (p_0, p_1, \ldots, p_{k-1}) \mapsto \prod_{i=0}^{k-1} \{q_{i_1}, q_{i_2}, \ldots, q_{i_{m_i}}\}$$

In general, for a map $g : A \mapsto B$ let

$$Im(g) = \{b : b \in B \wedge \exists\, a\, [a \in A \wedge b = g(a)]\}$$

For each subset $B_1 \subseteq B$, let

$$g^{-1}(B_1) = \{a : a \in A \wedge \exists\, b\, [b \in B \wedge b = g(a)]\}$$

Since $V(F_i)$ is a finite set, $Im(f)$ is a finite set as well. Let

$$Im(f) = \{S_1, S_2, \ldots, S_m\} (m \geq 1)$$

Then we have that

$$\cup_{i=1}^{m} f^{-1}(S_i) = \prod_{i=0}^{k-1} \partial P_i$$

In other words,

$$\prod_{i=0}^{k-1} \partial P_i$$

can be partitioned into m pairwise disjoint subsets

$$f^{-1}(S_1), f^{-1}(S_2), \ldots, f^{-1}(S_m)$$

For each constrained ESP

$$\rho(p_0 q_{0_1} q_{0_2} \cdots q_{0_{m_0}} p_1 \cdots p_i q_{i_1} q_{i_2} \cdots q_{i_{m_i}} p_{i+1} \cdots p_{k-1} q_{k-1_1} q_{k-1_2} \cdots q_{k-1_{m_{k-1}}} p_0)$$

there exists an index $i \in \{1, 2, \ldots, m\}$ such that

$$(p_0, p_1, \ldots, p_{k-1}) \in f^{-1}(S_i)$$

and

$$\prod_{i=0}^{k-1} \{q_{i_1}, q_{i_2}, \ldots, q_{i_{m_i}}\} \subseteq S_i$$

On the other hand, analogously to the proof of Lemma 24, we have that for each $f^{-1}(S_i)$, there exists a unique constrained ESP.

This proves the theorem. □

By Theorems 49 and 34, we have a second and shorter proof for Corollary 10 which implies that the w-rubberband algorithm computes an approximate and general solution to a WRP.

11. COROLLARY. *For any safari problem, there exists a finite number of local minima only.*

Analogously to the proof of Theorem 49, we also have the following

50. THEOREM. *For any zookeeper problem, there exists a finite number of local minima only .*

Finally, in analogy to the discussion at the end of Section 8.3.4, we also obtain

51. THEOREM. *The approximate restricted solution obtained by the z-rubberband algorithm can be converted (in linear time) into an exact restricted solution of the same zookeeper problem.*

Problems (*simplified)*	Known results	Results in this report
Safari (1992) convex polygons	$\mathcal{O}(kn\ log(n/k))$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(n + m_k)$ any simple polygons restricted solution
Zookeeper (1992) convex polygons	$\mathcal{O}(n\ log(n))$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(n + m_k)$ any simple polygons restricted solution
Is zookeeper linear (2000) convex polygons	open general solution	yes any simple polygons restricted solution
WRP (1988) fixed	$\mathcal{O}(n^4)$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(n + k)$ not fixed general solution
WRP not fixed	$\mathcal{O}(n^5)$	
WRP $\sqrt{2}$-approximation	$\mathcal{O}(n)$	

Table 11.1: Summary of results for simplified art gallery problems. Here, n is the number of vertices of the simple polygon Π, k is the number of polygons P'_i, m_k is the total number of all vertices of involved polygons P_i, or the number of essential cuts in the case of the WRP. Note that "fixed" means that a start point is given.

We point out that, for unconstrained TPP, analogously to the proof of Lemma 24, we may again apply Theorem 46 to prove that the solutions, obtained by Algorithms 10 and 11, are lexicographically unique, and, thus, they are correct.

Note that the uniqueness of a global minimum depends upon that of a local minimum. If the lexicographic uniqueness of a local minimum is obtained by applying the proof of Lemma 57, such as in Step 4 in Procedure 17, then, this lexicographical uniqueness is obtained by considering coordinates (i.e., using lexicographic sorting) of points. In other words, this lexicographical uniqueness is obtained not only by comparing the lengths of paths but also by comparing positions of points. Thus, we speak here about "position and length based uniqueness", and not just about "length based uniqueness". This proof methodology can be generalized to proofs of correctness for rubberband algorithms where step sets are closed subsets in a plane, such as the closed regions of critical faces in Chapter 6, or the critical polygons in Chapter 10.

11.7 Conclusions

The chapter started with recalling an open problem published 2003 in [52], that one of the most intriguing open problems ... is to determine the complexity of the TPP for (pairwise) disjoint nonconvex simple polygons. The chapter described a simple algorithm which "approximately" answers this open problem. (Note that the solution in [52] is only valid if the following two requirements are satisfied: the polygons should be convex and pairwise disjoint; the given algorithm has time complexity $\kappa(\epsilon) \cdot \mathcal{O}(kn\ log(n/k))$, where n is the total number of vertices of involved polygons $P_i \subset \pi$, for $i = 1, 2, \ldots, k$.) The algorithm presented in this chapter also works for nonconvex polygons, and has κ-linear time complexity. We also presented algorithms for a few classic problems (see Tables 11.1 and 11.2 for a summary of the results). In particular, Theorem 40 approximately and restrictedly answers the previously raised open problem of the zookeeper problem for the case if it is simplified; we also significantly improved the results on solving the watchman route problem. One of the most important results in this chapter is Theorem 37 which provides an approximate solution to the unconstrained touring polygons problem (TPP) which is known to be NP-hard (see Theorem 36).

Problems	Known Results	Results in this report
TPP (2003) disjoint, convex polygons	$\mathcal{O}(kn\ log(n/k))$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(n)$ general solution any simple polygons
TPP (2003) non convex polygons	open general solution	$\kappa(\epsilon) \cdot \mathcal{O}(n)$ any simple polygons general solution
Parts cutting (1999) disjoint, convex, fixed	$\mathcal{O}(kn\ log(n/k))$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(n)$, not fixed any simple polygons general solution
Safari (1992) convex polygons	$\mathcal{O}(kn\ log(n/k))$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(kn + m_k)$ any simple polygons restricted solution
Zookeeper (1992) convex polygons	$\mathcal{O}(n\ log(n))$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(kn + m_k)$ any simple polygons restricted solution
WRP (1988) fixed	$\mathcal{O}(n^4)$ general solution	$\kappa(\epsilon) \cdot \mathcal{O}(kn)$ not fixed general solution
WRP not fixed	$\mathcal{O}(n^5)$	
WRP $\sqrt{2}$-approximation	$\mathcal{O}(n)$	

Table 11.2: Summary of results for non-simplified (i.e., "general") art gallery problems. For the TPP or parts cutting problems, n is the total number of vertices of involved polygons P_i'. For safari or zookeeper problems, and WRP, n is the number of vertices of the simple polygon Π, k is the number of polygons P_i, m_k is the total number of vertices of all polygons P_i, or the number of essential cuts in the case of the WRP. Note that "fixed" means that the start point is given.

Chapter 12

Concluding Remarks

This chapter summarizes the report. It points to the major issues of rubberband algorithms, which have been considered and solved in this report. It also comes to some important conclusions. It states a conjecture and ends with some open problems for future research.

12.1 Major Issues of Rubberband Algorithms

The original rubberband algorithm was proposed for solving the 3D MLP problem in digital geometry. Its mathematic foundations have been identified in this report as being the well known Monotone Convergence Theorem (see Theorem 2 in Section 1.4) and the Cauchy Convergence Criterion (see Theorem 1 in Section 1.4).

The basic principle of the original rubberband algorithm is as follows: find a global minimum by calculating (repeatedly) local minima. The most important and basic version of the original rubberband algorithms is Algorithm 3 in Section 7.5. It has many applications. For example, it can quickly calculate roots in Step 3 in Algorithm 2, presented in Section 3.3.2. [1] All other rubberband algorithms, presented in the report for solving ESP problems or art gallery problems, are variants of this basic algorithm.

A striking property of rubberband algorithms is that each step element (also called “step set” in the report) does not have to be a convex set. For example, all algorithms presented in Chapter 11 do not require to deal with convex polygons only.

The mathematic reason for this very important property is the well-known “finite subcover principle” in elementary topology or mathematic analysis, and the “position and length based uniqueness” introduced at the end of Chapter 11. Traditional algorithms such as in [21] for minimizing a function of several variables, always assume that the function is positive definite (which implies that the function is convex).

[1]Algorithm 2 is significantly improved by applying Algorithm 3 in Step 3 instead of applying the n-section method (see Appendix A) of numeric analysis.

According to [106, 107], there are two basic difficulties in solving ESP problems: algebraic hardness and combinatorial hardness. Algebraic hardness means that the comparison of the lengths of two paths may require exponentially many bits; combinatorial hardness means that the number of different paths from a source to a target may be exponential with respect to the input size. Rubberband algorithms solve the algebraic hardness by iterating, aiming at converge to *the* true solution, or to one of the true solutions. These algorithms also solve combinatorial hardness by finding step sets (which is usually the main task of each rubberband algorithm).

Chapter 7 highlighted the importance of a degenerate case of rubberband algorithms, also showing that it can be solved approximately with high accuracy.

We conclude that rubberband algorithms have many applications for solving various Euclidean shortest path problems in computational geometry. The basic principle also applies in a more general sense to all minimization problems in various disciplines, including pure and applied mathematics. It can help to solve some "nonconvex" problems without using traditional convex analysis. This might be the most important result of this report at a methodologic level, pointing to the establishment of a new theory of minimization.

12.2 Main Problems Solved in Report

We list the main problems solved in this report, either exactly or approximately (in the later case, typically in $\kappa(\varepsilon)$ linear time), by applying some version a of rubberband algorithm:

- *1*: The example in Section 5.4 answers an open problem in the book [85].
- *2*: The edge-based and face-based rubberband algorithms in Section 6.4.2 approximately answer the first open problem in [22]. This answer is the best possible in some sense because, by Corollary 4 in Section 7.1, there does not exist an exact algorithm for calculating an exact solution.
- *3*: Theorem 16 in Section 5.3 answers the second open problem in [22].
- *4*: Algorithm 5 in Section 9.3.2 restrictedly and approximately answers an open problem stated in [106] which asks for all solutions. Note that Algorithm 5 does not require critical polygons to be convex.
- *5*: Algorithm 8 in Section 10.6 approximately solves two NP-complete problems and one NP-hard problem in κ-linear time.

- *6*: Algorithm 11 in Section 11.4.1 is an approximate solution for an open problem stated in [52]. This algorithm also approximately solves one NP-hard problem.

- *7*: Algorithm 13 in Section 11.4.3 solves an open problem in [106] but only restrictedly for a simplified version of the problem.

- *8*: Algorithm 15 in Section 11.1.4 significantly improves so far existing results on watchman route problems.

Note that many known algorithms for solving art gallery problems depend on the condition that the start point is fixed. They may run slower without assuming such a fixed start point. For example, [140] proved for the presented algorithm that it requires for the "floating" WRP (i.e., without specifying a start point) $\mathcal{O}(n)$ times run time of the time needed for the fixed WRP. However, we showed that this additional input constraint does not make a difference for the rubberband algorithms. Note again that previously known algorithms for solving the touring polygons, parts cutting, safari, or zookeeper problem were formulated under the condition that all polygons are convex. Rubberband algorithms apply to these problems even when the polygons are nonconvex.

With reference to the listed items 5 and 6, we conclude that a few NP-complete or NP-hard problems can be approximately solved very efficiently – in $\kappa(\varepsilon)$ linear time! This means that although exact solutions to these problems are classified to be "very difficult", in the sense of approximation algorithms, they are actually easy! They can be solved efficiently with any desired accuracy. Based on these exciting results, we state a conjecture (where κ-quadratic means $\kappa \cdot \mathcal{O}(n^2)$):

Conjecture

A large subclass of NP-complete or NP-hard problems can be solved by some approximate algorithm in κ-quadratic time.

We say that a problem is solved *very well* by a computer if the computer can find solution(s) in κ-quadratic time (for some function κ) and with arbitrary accuracy. In this sense, we conjecture that a large diversity of NP-complete or NP-hard problems can be solved very well by a computer.

12.3 Open Problems

There exist exact algorithms for solving the 2D ESP problem (see [91]) or the zookeeper problem, but this is impossible for the case of the 3D ESP problem (see Corollary 5). Open problems:

- *1*: Does there exist any exact algorithm for solving the surface ESP problem?
- *2*: Do there exist any exact algorithms for the following problems which are closely related to ESP problems: the touring polygons problem (TPP), the parts cutting problem, safari problem, and watchman route problem?

Appendix A

Algorithm 1 (n-Section Method)

The following C++ code is used to find a root of $f(x) = 0$, where $f(x)$ is a continuous function on $[a, b]$, with $f(a)f(b) > 0$. The input are endpoints a and b, a tolerance TOL, and the maximum number N of iterations. The output is an approximate root p, or a fail-message.

```
int main( void )
{
     long int i=0;
     for(i =0;i <N;i++){
       if(function(i*(b-a)/N) < TOL){
         cout << "approximate root p = " << i*(b-a)/N << endl;
         return 0;
       }
     }
     //fail message
     cout <<"N is too small! N = "<< N << endl;
     cout <<"Try a bigger number!" << endl;

     return 0;
}
```

Appendix B
Data for the Example in Chapter 5

Example 1: Below we list all $\frac{\partial d_i}{\partial t_i}$ (i = 0, 1, ..., 19) for the cube-curve g as shown in Figure 5.5:

$$d_{t_0} = \frac{t_0}{\sqrt{{t_0}^2 + {t_1}^2 + 4}} + \frac{t_0 - t_{19}}{\sqrt{(t_0 - t_{19})^2 + 4}}$$

$$d_{t_1} = \frac{t_1}{\sqrt{{t_0}^2 + {t_1}^2 + 4}} + \frac{t_1 - t_2}{\sqrt{(t_1 - t_2)^2 + 5}}$$

$$d_{t_2} = \frac{t_2 - t_1}{\sqrt{(t_2 - t_1)^2 + 5}} + \frac{t_2 - 1}{\sqrt{(t_2 - 1)^2 + (t_3 - 1)^2 + 4}}$$

$$d_{t_3} = \frac{t_3 - 1}{\sqrt{(t_2 - 1)^2 + (t_3 - 1)^2 + 4}} + \frac{t_3}{\sqrt{{t_3}^2 + {t_4}^2 + 4}}$$

$$d_{t_4} = \frac{t_4}{\sqrt{{t_3}^2 + {t_4}^2 + 4}} + \frac{t_4 - 1}{\sqrt{(t_4 - 1)^2 + {t_5}^2 + 4}}$$

$$d_{t_5} = \frac{t_5}{\sqrt{(t_4 - 1)^2 + {t_5}^2 + 4}} + \frac{t_5 - t_6}{\sqrt{(t_5 - t_6)^2 + 4}}$$

$$d_{t_6} = \frac{t_6 - t_5}{\sqrt{(t_6 - t_5)^2 + 4}} + \frac{t_6 - 1}{\sqrt{(t_6 - 1)^2 + {t_7}^2 + 4}}$$

$$d_{t_7} = \frac{t_7}{\sqrt{(t_6 - 1)^2 + {t_7}^2 + 4}} + \frac{t_7 - 1}{\sqrt{(t_7 - 1)^2 + {t_8}^2 + 4}}$$

$$d_{t_8} = \frac{t_8}{\sqrt{(t_7 - 1)^2 + {t_8}^2 + 4}} + \frac{t_8 - t_9}{\sqrt{(t_8 - t_9)^2 + 4}}$$

$$d_{t_9} = \frac{t_9 - t_8}{\sqrt{(t_9 - t_8)^2 + 4}} + \frac{t_9 - 1}{\sqrt{(t_9 - 1)^2 + {t_{10}}^2 + 4}}$$

$$d_{t_{10}} = \frac{t_{10}}{\sqrt{(t_9 - 1)^2 + {t_{10}}^2 + 4}} + \frac{t_{10} - 1}{\sqrt{(t_{10} - 1)^2 + (t_{11} - 1)^2 + 4}}$$

$$d_{t_{11}} = \frac{t_{11}-1}{\sqrt{(t_{11}-1)^2+(t_{10}-1)^2+4}} + \frac{t_{11}}{\sqrt{{t_{11}}^2+{t_{12}}^2+1}}$$

$$d_{t_{12}} = \frac{t_{12}}{\sqrt{{t_{11}}^2+{t_{12}}^2+1}} + \frac{t_{12}-t_{13}}{\sqrt{(t_{12}-t_{13})^2+4}}$$

$$d_{t_{13}} = \frac{t_{13}-t_{12}}{\sqrt{(t_{13}-t_{12})^2+4}} + \frac{t_{13}-1}{\sqrt{(t_{13}-1)^2+(t_{14}-1)^2+4}}$$

$$d_{t_{14}} = \frac{t_{14}-1}{\sqrt{(t_{13}-1)^2+(t_{14}-1)^2+4}} + \frac{t_{14}}{\sqrt{{t_{14}}^2+(t_{15}-1)^2+4}}$$

$$d_{t_{15}} = \frac{t_{15}-1}{\sqrt{{t_{14}}^2+(t_{15}-1)^2+4}} + \frac{t_{15}-t_{16}}{\sqrt{(t_{15}-t_{16})^2+16}}$$

$$d_{t_{16}} = \frac{t_{16}-t_{15}}{\sqrt{(t_{16}-t_{15})^2+16}} + \frac{t_{16}-t_{17}}{\sqrt{(t_{16}-t_{17})^2+4}}$$

$$d_{t_{17}} = \frac{t_{17}-t_{16}}{\sqrt{(t_{17}-t_{16})^2+4}} + \frac{t_{17}}{\sqrt{{t_{17}}^2+(t_{18}-1)^2+1}}$$

$$d_{t_{18}} = \frac{t_{18}-1}{\sqrt{{t_{17}}^2+(t_{18}-1)^2+1}} + \frac{t_{18}-t_{19}}{\sqrt{(t_{18}-t_{19})^2+101}}$$

$$d_{t_{19}} = \frac{t_{19}-t_{18}}{\sqrt{(t_{19}-t_{18})^2+101}} + \frac{t_{19}-t_{0}}{\sqrt{(t_{19}-t_{0})^2+4}}$$

Appendix C

GAP Inputs and Outputs

```
1. To Compute the Factors of the Determinant of Polynomial
f(x) = 84*x^6-228*x^5+361*x^4+20*x^3+210*x^2-200*x+25.

1.1 Create the rows of a (2n-1)x(2n-1) matrix, where n is 6.

r1:=[1,-228,361,20,210,-200,25,0,0,0,0];
r2:=[0,1,-228,361,20,210,-200,25,0,0,0];
r3:=[0,0,1,-228,361,20,210,-200,25,0,0];
r4:=[0,0,0,1,-228,361,20,210,-200,25,0];
r5:=[0,0,0,0,1,-228,361,20,210,-200,25];

r6:= [6*1,-5*228,4*361,3*20,2*210,-200,0,0,0,0,0];
r7:= [0,6*1,-5*228,4*361,3*20,2*210,-200,0,0,0,0];
r8:= [0,0,6*1,-5*228,4*361,3*20,2*210,-200,0,0,0];
r9:= [0,0,0,6*1,-5*228,4*361,3*20,2*210,-200,0,0];
r10:=[0,0,0,0,6*1,-5*228,4*361,3*20,2*210,-200,0];
r11:=[0,0,0,0,0,6*1,-5*228,4*361,3*20,2*210,-200];


m:=[r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11];

1.2 Compute the Determinant.

gap> d:=DeterminantMatDestructive(m);
31364519252281021125000000

gap> d:=84*d;
2634619617191605774500000000

1.3 Compute the Factors of the Determinant.
```

```
gap> FactorsInt(d);
[ 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 7, 11,
  19249, 204797, 214309 ]
```

2. Factorizing f(x) mod 13.

```
gap> F:=GaloisField(13);
GF(13)
gap> e:=Elements(F);
[ 0*Z(13), Z(13)^0, Z(13), Z(13)^2, Z(13)^3, Z(13)^4, Z(13)^5, Z(13)^6,
  Z(13)^7, Z(13)^8, Z(13)^9, Z(13)^10, Z(13)^11 ]
gap> x:= X(F,"x");
x
gap> f:=84*x^6-228*x^5+361*x^4+20*x^3+210*x^2-200*x+25;;
gap> Factors(f);
[ Z(13)^5*x+Z(13)^10, x^2+Z(13)^5, x^3+Z(13)^3*x^2+Z(13)^2*x+Z(13)^3 ]
```

3. Factorizing f(x) mod 19.

```
gap> F:=GaloisField(19);
GF(19)
gap> e:=Elements(F);
[ 0*Z(19), Z(19)^0, Z(19), Z(19)^2, Z(19)^3, Z(19)^4, Z(19)^5, Z(19)^6,
  Z(19)^7, Z(19)^8, Z(19)^9, Z(19)^10, Z(19)^11, Z(19)^12, Z(19)^13,
  Z(19)^14, Z(19)^15, Z(19)^16, Z(19)^17 ]
gap> x:= X(F,"x");
x
gap> f:=84*x^6-228*x^5+361*x^4+20*x^3+210*x^2-200*x+25;;
gap> Factors(f);
[ Z(19)^3*x^6+x^3+x^2+Z(19)^8*x+Z(19)^14 ]
```

4. Factorizing f(x) mod 37.

```
gap> F:=GaloisField(37);
GF(37)
gap> e:=Elements(F);
[ 0*Z(37), Z(37)^0, Z(37), Z(37)^2, Z(37)^3, Z(37)^4, Z(37)^5, Z(37)^6,
  Z(37)^7, Z(37)^8, Z(37)^9, Z(37)^10, Z(37)^11, Z(37)^12, Z(37)^13,
  Z(37)^14, Z(37)^15, Z(37)^16, Z(37)^17, Z(37)^18, Z(37)^19, Z(37)^20,
  Z(37)^21, Z(37)^22, Z(37)^23, Z(37)^24, Z(37)^25, Z(37)^26, Z(37)^27,
  Z(37)^28, Z(37)^29, Z(37)^30, Z(37)^31, Z(37)^32, Z(37)^33, Z(37)^34,
  Z(37)^35 ]
gap> x:= X(F,"x");
x
gap> f:=84*x^6-228*x^5+361*x^4+20*x^3+210*x^2-200*x+25;;
gap> Factors(f);
[ Z(37)^24*x+Z(37)^0,
  x^5+Z(37)^26*x^4+Z(37)^22*x^3+Z(37)^30*x^2+Z(37)^9*x+Z(37)^10 ]
```

Appendix D

Compution of Matrix Q of Chapter 7

k	$a_{k,5}$	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	0	13	2	6	5	2
1	13	2	6	5	2	0
2	2	6	1	17	2	14
3	6	1	12	16	7	8
4	1	12	1	11	6	5
5	12	1	18	13	11	4
6	1	18	2	0	0	10
7	18	2	7	7	16	4
8	2	7	0	9	17	15
9	7	0	4	12	8	8
10	0	4	4	0	12	9
11	4	4	0	12	9	0
12	4	0	2	18	5	16
13	0	2	8	14	2	16
14	2	8	14	2	16	0
15	8	14	16	11	12	8
16	14	16	10	11	18	13
17	16	10	14	2	2	18
18	10	14	0	0	0	7
19	14	0	13	13	10	2

Table D.1: Compution of the fourth row of matrix Q in Section 7.4.3.

k	$a_{k,5}$	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	14	0	13	13	10	2
1	0	13	16	13	10	18
2	13	16	13	10	18	0
3	16	13	6	14	2	14
4	13	6	12	0	15	7
5	6	12	15	11	9	14
6	12	15	15	13	12	5
7	15	15	2	1	1	10
8	15	2	11	11	5	3
9	2	11	2	15	17	3
10	11	2	10	12	15	8
11	2	10	13	16	17	6
12	10	13	11	12	18	8
13	13	11	6	12	11	2
14	11	6	8	7	4	14
15	6	8	8	5	4	6
16	8	8	9	8	4	5
17	8	9	7	3	15	13
18	9	7	2	14	4	13
19	7	2	1	10	10	17

Table D.2: Compution of the fifth row of matrix Q in Section 7.4.3.

k	$a_{k,5}$	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	7	2	1	10	10	17
1	2	1	2	2	2	9
2	1	2	16	16	2	8
3	2	16	4	9	14	4
4	16	4	4	9	16	8
5	4	4	7	14	9	7
6	4	7	4	18	12	16
7	7	4	8	2	2	16
8	4	8	13	13	1	9
9	8	13	3	10	14	16
10	13	3	9	13	7	13
11	3	9	9	3	15	14
12	9	9	5	17	13	12
13	9	5	4	0	9	17
14	5	4	6	15	14	17
15	4	6	12	11	9	1
16	6	12	1	18	6	16
17	12	1	3	10	14	5
18	1	3	18	3	1	10
19	3	18	10	8	16	4

Table D.3: Compution of the sixth row of matrix Q in Section 7.4.3.

k	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	1	0	0	0	0
5	34	16	26	6	12
6	25	15	2	31	1
7	14	32	15	3	4
8	27	17	34	14	20
9	10	22	13	34	28
10	29	25	35	14	9
11	12	18	28	35	15
12	19	35	14	13	33
13	15	22	26	36	6
14	14	7	19	22	32
15	2	21	16	5	20
16	15	11	20	32	24
17	3	1	15	3	32
18	29	26	7	13	36
19	13	27	27	25	15
20	25	13	30	19	8
21	12	23	3	10	4
22	24	10	26	2	33
23	12	3	34	29	29
24	4	4	8	27	33
25	29	35	20	20	11
26	22	3	34	0	15
27	11	16	17	36	5
28	20	8	26	34	21
29	22	13	36	30	18
30	21	18	10	2	5
31	29	13	30	20	30
32	0	13	34	19	15
33	13	34	19	15	0
34	32	5	20	4	8
35	20	14	22	15	14
36	28	9	17	23	18
37	36	21	11	1	3

Table D.4: Compution of the second row of matrix Q in Section 7.4.4.

k	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	36	21	11	1	3
1	24	32	12	34	25
2	34	26	29	21	29
3	35	18	17	11	1
4	24	22	33	26	13
5	24	10	21	9	29
6	12	35	4	25	29
7	36	11	4	27	33
8	14	25	1	27	25
9	20	3	21	35	20
10	17	8	0	29	18
11	31	13	27	9	19
12	31	5	1	20	2
13	23	16	12	3	2
14	21	10	9	29	17
15	21	12	20	32	30
16	23	23	23	8	30
17	28	21	14	20	17
18	11	18	8	0	3
19	22	36	27	32	21
20	7	9	12	5	5
21	25	13	2	10	10
22	12	32	31	12	4
23	33	1	28	2	33
24	13	1	9	9	26
25	36	32	14	30	8
26	35	35	4	2	25
27	4	9	24	13	13
28	34	14	6	0	11
29	23	32	33	30	1
30	0	31	36	28	17
31	31	36	28	17	0
32	17	6	9	1	2
33	29	22	36	30	19
34	9	19	7	8	15
35	29	3	20	32	34
36	27	3	9	23	15
37	33	34	22	29	28

Table D.5: Compution of the third row of matrix Q in Section 7.4.4.

k	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	33	34	22	29	28
1	9	32	36	4	26
2	5	32	16	6	34
3	17	22	25	27	23
4	8	1	25	14	19
5	14	5	0	30	22
6	0	2	24	32	20
7	2	24	32	20	0
8	18	27	35	12	24
9	10	27	36	21	31
10	34	11	22	17	9
11	20	11	13	28	1
12	25	0	30	10	18
13	36	23	31	20	4
14	26	15	31	35	25
15	11	3	8	33	16
16	7	36	23	8	21
17	15	24	5	26	10
18	16	23	9	26	32
19	12	6	35	17	7
20	7	5	33	5	33
21	21	34	2	1	10
22	8	5	29	25	30
23	18	9	11	4	22
24	29	3	28	19	31
25	27	11	33	20	15
26	4	21	19	29	28
27	9	9	22	15	11
28	19	18	27	28	34
29	35	35	4	0	6
30	4	9	22	31	13
31	34	12	24	0	11
32	21	13	33	30	1
33	24	36	21	16	30
34	1	35	11	26	29
35	32	27	15	35	12
36	5	9	16	19	14
37	31	22	1	7	23

Table D.6: Compution of the fourth row of matrix Q in Section 7.4.4.

k	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	31	22	1	7	23
1	3	16	36	24	2
2	7	10	28	20	36
3	26	29	17	4	10
4	25	26	14	18	16
5	25	7	2	18	4
6	6	32	2	6	4
7	14	24	14	3	35
8	19	16	34	8	20
9	33	5	21	23	6
10	17	31	30	19	26
11	17	6	17	17	19
12	29	30	15	10	19
13	17	35	24	8	15
14	21	0	6	6	19
15	11	9	34	34	30
16	13	25	24	22	21
17	23	10	27	25	8
18	15	25	31	35	17
19	17	12	18	33	32
20	35	31	31	23	19
21	0	36	8	7	13
22	36	8	7	13	0
23	11	28	24	31	25
24	32	15	21	17	21
25	30	15	35	28	14
26	36	34	31	9	27
27	0	15	20	21	25
28	15	20	21	25	0
29	12	2	8	16	32
30	3	15	32	30	33
31	6	6	34	14	36
32	25	19	22	35	35
33	18	15	19	0	4
34	35	11	24	1	31
35	17	29	23	19	13
36	15	36	17	4	19
37	28	35	24	35	32

Table D.7: Compution of the fifth row of matrix Q in Section 7.4.4.

k	$a_{k,1}$	$a_{k,0}$
0	0	1
1	1	0
2	0	7
3	7	0
4	0	10
5	10	0
6	0	5
7	5	0
8	0	9
9	9	0
10	0	11
11	11	0
12	0	12
13	12	0

Table D.8: Compution of the second row of matrix Q in Section 7.4.5.

k	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	0	0	1
1	0	1	0
2	1	0	0
3	5	9	5
4	8	11	12
5	12	6	1
6	1	5	8
7	10	4	5
8	2	4	11
9	1	3	10
10	8	6	5
11	7	12	1
12	8	12	9
13	0	3	1

Table D.9: Compution of the second row of matrix Q in Section 7.4.6.

k	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	0	3	1
1	3	1	0
2	3	1	2
3	3	3	2
4	5	3	2
5	2	8	12
6	5	4	10
7	3	3	12
8	5	0	2
9	12	8	12
10	3	3	8
11	5	9	2
12	8	8	12
13	9	6	1

Table D.10: Compution of the third row of matrix Q in Section 7.4.6.

Bibliography

[1] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. A. Schevon. Star unfolding of a polytope with applications. *SIAM J. Comput.*, **26**:1689–1713, 1987.

[2] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan. Approximate shortest paths on a convex polytope in three dimensions. *J. ACM*, **44**:567–584, 1997.

[3] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ϵ-approximation algorithm for weighted shortest path queries on polyhedral surfaces. In Abstracts *European Workshop Comput. Geom.*, pages 19–21, 1998.

[4] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces. In Proc *Scand. Workshop Algorithm Theory*, LNCS 1432, pages 11–22, Springer, Berlin, 1998.

[5] M. H. Alsuwaiyel and D. T. Lee. Minimal link visibility paths inside a simple polygon. *Comput. Geom.*, **3(1)**: 1–25, 1993.

[6] M. H. Alsuwaiyel and D. T. Lee. Finding an approximate minimum-link visibility path inside a simple polygon. *Information Processing Letters*, **55**:75–79, 1995.

[7] T. M. Amarunnishad and P. P. Das. Estimation of length for digitized straight lines in three dimensions. *Pattern Recognition Letters*, **11**:207–213, 1990.

[8] E. M. Arkin, J. S. B. Mitchell, and C. Piatko. Minimum-link watchman tours. Report, University at Stony Brook, 1994.

[9] B. Aronov and J. O'Rourke. Nonoverlap of the star unfolding. *Discrete Comput. Geom.*, **8**:219–250, 1992.

[10] T. Asano, S. K. Ghosh, and T. C. Shermer. Visibility in the plane. In *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, editors), pages 829–876, Elsevier, 2000.

[11] T. Asano, Y. Kawamura, R. Klette, and K. Obokata. Minimum-length polygons in approximation sausages. In Proc. *Int. Workshop Visual Form*, LNCS 2059, pages 103–112, Springer, Berlin, 2004.

[12] T. Asano, Y. Kawamura, R. Klette, and K. Obokata. Digital curve approximation with length evaluation. *IEICE Trans. Fundamentals*, **E. 86-A**:987–994, 2003.

[13] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. M-S, and M. Protasi. *Complexity and Approximation*. Springer, New York, 1999.

[14] D. Bailey. An efficient Euclidean distance transform. In Proc. *Int. Workshop Combinatorial Image Analysis*, LNCS 3322, pages 394–408, Springer, Berlin, 2004.

[15] C. Bajaj. The algebraic complexity of shortest paths in polyhedral spaces. In Proc. *Allerton Conf. Commum. Control Comput.*, pages 510–517, 1985.

[16] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete Computational Geometry*, **3**:177–191, 1988.

[17] R. G. Bartle and D. Sherbert. *Introduction to Real Analysis*, 3rd edition, Wiley, New York, 2000.

[18] M. de Berg and M. van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, **18**:306–323, 1997.

[19] S. Bespamyatnikh. An $\mathcal{O}(n\,log\,n)$ algorithm for the zoo-keepers problem. *Computational Geometry: Theory and Applications*, **24**:63–74, 2002.

[20] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, UK, 2004.

[21] R. P. Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, Englewwood Cliffs, New Jersey, 1973.

[22] T. Bülow and R. Klette. Rubber band algorithm for estimating the length of digitized space-curves. In Proc. *Intern. Conf. Pattern Recognition*, volume 3, pages 551–555, 2000,

[23] T. Bülow and R. Klette. Approximation of 3D shortest polygons in simple cube curves. In Proc. *Digital and Image Geometry*, LNCS 2243, pages 281–294, Springer, Berlin, 2001.

[24] 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.

[25] R. L. Burden and J. D. Faires. *Numerical Analysis*. 7th edition, Brooks Cole, Pacific Grove, 2000.

[26] R. Busemann and W. Feller. Krümmungseigenschaften konvexer Flächen. *Acta Mathematica*, **66**:27–45, 1935.

[27] J. Canny and J.H. Reif. New lower bound techniques for robot motion planning problems. In Proc. *IEEE Conf. Foundations Computer Science*, pages 49–60, 1987.

[28] J. Canny. Some algebraic and geometric configurations in PSPACE. In Proc. *Annu. ACM Sympos. Theory Computation*, pages 460–467, 1988.

[29] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. In Proc. *Int. Sympos. Algorithms Computation*, pages 58–67, 1993.

[30] S. Carlsson, H. Jonsson, and B. J. Nilsson. Optimum guard covers and m-watchmen routes for restricted polygons. Proc. *Workshop Algorithms Data Struct.*, LNCS 519, pages 367–378, Springer, 1991.

[31] S. Carlsson, H. Jonsson, and B. J. Nilsson. Optimum guard covers and m-watchmen routes for restricted polygons. *Internat J. Comput. Geom. Appl.*, **3**:85–105, 1993.

[32] S. Carlsson and H. Jonsson. Computing a shortest watchman path in a simple polygon in polynomial-time. Proc. *Workshop Algorithms Data Struct.*, LNCS 955, pages 122–134, Springer, 1995.

[33] S. Carlsson, H. Jonsson, and B. J. Nilsson. Approximating the shortest watchman route in a simple polygon. Technical report, Lund University, Sweden, 1997.

[34] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete Computational Geom.*, **22**:377–402, 1999.

[35] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Computational Geom.*, **6**:485–524, 1991.

[36] S. Chattopadhyay and P. P. Das. Estimation of the original length of a straight line segment from its digitization in three dimensions. *Pattern Recognition*, **25**:787–798, 1992.

[37] J. Chen and Y. Han. Shortest paths on a polyhedron. In Proc. *Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1990.

[38] W.-P. Chin and S. Ntafos. Optimum zookeeper routes. Combinatorics, Graph Theory and Computing. Proc. *South-East Conf., Boca Raton.*, Congr. Number **58**, 257–266, 1987.

[39] W. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, **28**:39–44, 1988.

[40] W.-P. Chin and S. Ntafos. The zookeeper route problem. *Information Sciences*, **63**:245–259, 1992.

[41] W.-P. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete Computational Geometry*, **6**:9–31, 1991.

[42] Y.-J. Chiang and J. S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. In Proc. *ACM-SIAM Sympos. Discrete Algorithms*, pages 215–224, 1999.

[43] J. Choi, J. Sellen, and C.-K. Yap. Approximate Euclidean shortest path in 3-space. In Proc. *ACM Conf. Computational Geometry*, ACM Press, pages 41–48, 1994.

[44] 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.

[45] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In Proc. *Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[46] D. Coeurjolly, I. Debled-Rennesson, and O. Teytaud. Segmentation and length estimation of 3D discrete curves. In Proc. *Digital and Image Geometry*, pages 299–317, LNCS 2243, Springer, 2001.

[47] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[48] J. Czyzowicz, P. Egyed, H. Everett, W. Lenhart, K. Lyons, D. Rappaport, T. Shermer, D. Souvaine, G. Toussaint, J. Urrutia, and S. Whitesides. The aquarium keeper's problem. In Proc. *ACM-SIAM Sympos. Data Structures Algorithms*, pages 459–464, 1991.

[49] L. Dorst and A. W. M. Smeulders. Length estimators for digitized contours. *Computer Vision Graphics Image Processing*, **40**:311–333, 1987.

[50] S. A. Douglass. *Introduction to Mathematical Analysis.* Addison-Wesley, 1996.

[51] M. Dror. Polygon plate-cutting with a given order. *IIE Transactions*, **31**:271–274, 1999.

[52] M. Dror, A. Efrat, A. Lubiw, and J. Mitchell. Touring a sequence of polygons. In Proc. *STOC*, pages 473–482, 2003.

[53] T. J. Ellis, D. Proffitt, D. Rosen, and W. Rutkowski. Measurement of the lengths of digitized curved lines. *Computer Graphics Image Processing*, **10**:333–347, 1979.

[54] E. Ficarra, L. Benini, E. Macii, and G. Zuccheri. Automated DNA fragments recognition and sizing through AFM image processing. *IEEE Trans. Inf. Technol. Biomed.*, **9**:508–517, 2005.

[55] GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra. See $http://www-gap.mcs.st-and.ac.uk/gap.html$

[56] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In Proc. *ACM Sympos. Theory Computing*, pages 10–22, 1976.

[57] A. M. Geoffrion Lagrangean relaxation and its uses in integer programming. In *Mathematical Programming Study*, volume 2, pages 82–114. North Holland, Amsterdam, 1974.

[58] L. P. Gewali, S. Ntafos, and I. G. Tollis. Path planning in the presence of vertical obstacles. Technical Report, Computer Science, University of Texas at Dallas, 1989.

[59] L. P. Gewali, A. Meng., J. S. B. Mitchell, and S. Ntafos. Path planning in 0/1/infinity weighted regions with applications. *ORSA J. Comput.*, **2**:253–272, 1990.

[60] L. P. Gewali and R. Lombardo. Watchman routes for a pair of convex polygons. *Lecture Notes in Pure Appl. Math.*, volume 144. 1993.

[61] L. P. Gewali and S. Ntafos. Watchman routes in the presence of a pair of convex polygons. In Proc. *Canad. Conf. Comput. Geom.*, pages 127–132, 1995.

[62] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, **2**:209–233, 1987.

[63] L. Guibas and J. Hershberger Optimal shortest path queries in a simple polygon. *J. Computer System Sciences*, **39**:126–152, 1989.

[64] M. Guignard and S. Kim. Lagrangean decomposition: a model yielding stronger Lagrangean bounds. *Mathematical Programming*, **39**:215–228, 1987.

[65] M. Hammar and B. J. Nilsson. Concerning the time bounds of existing shortest watchman routes. In Proc. *FCT'97*, LNCS 1279, pages 210–221, 1997.

[66] Y. Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters*, **91**:245–250, 2004.

[67] J. Hromkovič. *Algorithms for Hard Problems*. Springer, Berlin, New York, 2001.

[68] S. Har-Peled. Constructing approximate shortest path maps in three dimensions. In Proc. *Annu. ACM Sympos. Comput. Geom.*, pages 125–130, 1998.

[69] S. Har-Peled. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. *Discrete Computational Geometry*, **21**: 217–231, 1999.

[70] I. N. Herstein. *Topics in Algebra*, 2nd edition. Wiley, New York, 1975.

[71] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, **38**:231-235, 1991.

[72] J. Hershberger and J. Snoeyink. An efficient solution to the zookeeper's problem. In Proc. *Canad. Conf. Comput. Geom.*, pages 104–109, 1994.

[73] J. Hershberger and S. Suri. Practical methods for approximating shortest paths on a convex polytope in $\mathbb{R}^3$. In Proc. *ACM-SIAM Sympos. Discrete Algorithms*, pages 447–456, 1995.

[74] D. S. Hochbaum (editor). *Approximation Algorithms for NP-Hard Problems*. PWS Pub. Co.,Boston, 1997.

[75] J. Hoeft and U. S. Palekar. Heuristics for the plate-cutting traveling salesman problem. *IIE Transactions*, **29**:719–731, 1997.

[76] A. Jonas and N. Kiryati. Length estimation in 3-D using cube quantization. In Proc. *Vision Geometry*, SPIE 2356, pages 220–230, 1994.

[77] A. Jonas and N. Kiryati. Length estimation in 3-D using cube quantization, *J. Math. Imaging Vision*, **8**: 215–238, 1998.

[78] M. I. Karavelas and L. J. Guibas. Static and kinetic geometric spanners with applications. In Proc. *ACM-SIAM Symp. Discrete Algorithms*, pages 168–176, 2001.

[79] S. Kapoor. Efficient computation of geodesic shortest paths. In Proc. *Annu. ACM Sympos. Theory Comput.*, pages 770–779, 1999.

[80] N. Kiryati and G. Szekely. Estimating shortest paths and minimal distances on digitized three-dimensional surfaces. *Pattern Recognition*, **26**:1623–1637, 1993.

[81] N. Kiryati and O. Kubler. On chain code probabilities and length estimators for digitized three-dimensional curves. *Pattern Recognition*, **28**:361–372, 1995.

[82] R. Klette, V. Kovalevsky, and B. Yip. Length estimation of digital curves. In Proc. *Vision Geometry*, SPIE 3811, pages 117–129, 1999.

[83] R. Klette and T. Bülow. Critical edges in simple cube-curves. In Proc. *Discrete Geometry Computational Imaging*, LNCS 1953, pages 467–478, Springer, Berlin, 2000.

[84] R. Klette and B. Yip. The length of digital curves. *Machine Graphics & Vision*, **9**:673–703, 2000.

[85] R. Klette and A. Rosenfeld. *Digital Geometry*. Morgan Kaufmann, San Francisco, 2004.

[86] D. E. Knuth. *The Art of Computer Programming: Volume 2*, 3rd edition, Addison-Wesley, 1997.

[87] P. Kumar and C. Veni Madhavan. Shortest watchman tours in weak visibility polygons. In Proc. *Canad. Conf. Comput. Geom.*, pages 91–96, 1995.

[88] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In Proc. *Annu. ACM Sympos. Comput. Geom.*, pages 274–283, 1997.

[89] G. Laporte, H. Mercure, and Y. Nobert. Generalized traveling salesman problem through n clusters. *Discrete Applied Mathematics*, **18**:185–197, 1987.

[90] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*. John Wiley and Sons, New York, 1985.

[91] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, **14**:393–410, 1984.

[92] T.-Y. Li, P.-F. Chen, and P.-Z. Huang. Motion for humanoid walking in a layered environment. In Proc. *Conf. Robotics Automation*, volume 3, pages 3421–3427, 2003.

[93] F. Li and R. Klette. Minimum-length polygon of a simple cube-curve in 3D space. In Proc. *Int. Workshop Combinatorial Image Analysis*, LNCS 3322, pages 502–511, Springer, Berlin, 2004.

[94] F. Li and R. Klette. The class of simple cube-curves whose MLPs cannot have vertices at grid points. In Proc. *Discrete Geometry Computational Imaging*, LNCS 3429, pages 183–194, Springer, Berlin, 2005.

[95] F. Li and R. Klette. Minimum-length polygons of first-class simple cube-curves. In Proc. *Computer Analysis Images Patterns*, LNCS 3691, pages 321–329, Springer, Berlin, 2005.

[96] F. Li and R. Klette. Analysis of the rubberband algorithm. *Image and Vision Computing* (to appear, 2006).

[97] F. Li and R. Klette. Shortest paths in a cuboidal world. In Proc. *Int. Workshop Combinatorial Image Analysis*, LNCS 4040, pages 415–429, Springer, Berlin, 2006.

[98] Y. A. Liu. and S. D. Stoller. Optimizing Ackermann's function by incrementalization. In Proc. *ACM SIGPLAN Sympos. Partial Evaluation Semantics-Based Program Manipulation*, pages 85–91, 2003.

[99] H. Luo and A. Eleftheriadis. Rubberband: an improved graph search algorithm for interactive object segmentation. In Proc. *Int. Conf. Image Processing*, volume 1, pages 101–104, 2002.

[100] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In Proc. *Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1995.

[101] C. Mata and J. S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions. In Proc. *Annu. ACM Sympos. Comput. Geom.*, pages 264–273, 1997.

[102] E. W. Mayr, H. J. Prömel, and A. Steger (editors). *Lectures on Proof Verification and Approximation Algorithms*. Springer, Berlin, New York, 1998.

[103] A. Melkman. On-line construction of the convex hull of a simple polygon. *Information Processing Letters*, **25**:11–12, 1987.

[104] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, **16**:647–668, 1987.

[105] J. S. B. Mitchell and E. L. Wynters. Watchman routes for multiple guards. In Proc. *Canad. Conf. Comput. Geom.*, pages 126–129, 1991.

[106] J. S. B. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, editors). pages 633–701, Elsevier, 2000.

[107] J. S. B. Mitchell and M. Sharir. New results on shortest paths in three dimensions. In Proc. *SCG*, pages 124–133, 2004.

[108] U. Montanari. A note on minimal length polygonal approximations to a digitalized contour. *Comm. ACM*, **13**:41–47, 1970.

[109] T. O. Moore. *Elementary General Topology.* Prentice-Hall, Englewood Cliffs, N.J., 1964.

[110] D. M. Mount. The number of shortest paths on the surface of a polyhedron. *SIAM J. Comput.*, **19**:593–611, 1990.

[111] B. J. Nilsson and D. Wood. Optimum watchmen routes in spiral polygons. In Proc. *Canad. Conf. Comput. Geom.*, pages 269–272, 1990.

[112] B. J. Nilsson. Guarding art galleries; Methods for mobile guards. Ph.D. Thesis, Lund University, Sweden, 1995.

[113] L. Noakes, R. Kozera, and R. Klette. Length estimation for curves with different samplings. *Digital and Image Geometry*, LNCS 2243, pages 334–346, Springer, 2001.

[114] C. E. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, **31**:39–44, 1993.

[115] S. Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, **1**:149–170, 1992.

[116] S. Ntafos. The robber route problem. *Inform. Process. Lett.*, **34**:59–63, 1990.

[117] S. Ntafos. Watchman routes under limited visibility. *Comput. Geom.*, **1**:149–170, 1992.

[118] S. Ntafos and L. Gewali. External watchman routes. *Visual Comput.*, **10**:474–483, 1994.

[119] C. H. Papadimitriou. An algorithm for shortest path motion in three dimensions. *Inform. Process. Lett.*, **20**:259–263, 1985.

[120] Y. Rabani. Approximation algorithms. See $http://www.cs.technion.ac.il/\ rabani/236521.95.wi.html$ (last visit: July 2006).

[121] J. H. Reif and J. A. Storer. A single-exponential upper bound for shortest paths in three dimensions. *J. ACM*, **41**:1013–1019, 1994.

[122] A. W. Roberts and V. D. Varberg. *Convex Functions.* Academic Press, New York, 1973.

[123] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, N.J., 1970.

[124] C. Schevon and J. O'Rourke. The number of maximal edge sequences on a convex polytope. In Proc. *Allerton Conf. Commun. Control Comput.*, pages 49–57, 1988.

[125] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, **15**:193–215, 1986.

[126] M. Sharir. On shortest paths amidst convex polyhedra. *SIAM J. Comput.*, **16**:561–572, 1987.

[127] J. Sklansky and D. F. Kibler. A theory of nonuniformly digitized binary pictures. *IEEE Trans. Systems Man Cybernetics*, **6**:637–647, 1976.

[128] F. Sloboda, B. Zaťko, and R. Klette. On the topology of grid continua. In Proc. *Vision Geometry*, SPIE 3454, pages 52–63, 1998.

[129] F. Sloboda, B. Zaťko, and J. Stoer. On approximation of planar one-dimensional grid continua. In *Advances in Digital and Computational Geometry* (R. Klette, A. Rosenfeld, and F. Sloboda, editors), pages 113–160, Springer, Singapore, 1998.

[130] C. Sun and S. Pallottino. Circular shortest path on regular grids. CSIRO Math. Information Sciences, CMIS Report No. 01/76, Australia, 2001.

[131] D. Sunday. Algorithm 3: Fast winding number inclusion of a point in a polygon. See $www.geometryalgorithms.com/Archive/algorithm_0 103/$ (last visit: April 2007).

[132] D. Sunday. Algorithm 14: Tangents to and between polygons. See $http://softsurfer.com/Archive/algorithm_0 201/$ (last visit: June 2006).

[133] D. Sunday. Algorithm 15: convex hull of a 2D simple polygonal path. See $http://www.softsurfer.com/Archive/algorithm_0 203/$ (last visit: July 2006).

[134] S. Suri and J. Hershberger. Efficient computation of Euclidean shortest paths in the plane. *IEEE Foundations of Computer Science*, pages 508–517, 1993.

[135] M. Talbot. A dynamical programming solution for shortest path itineraries in robotics. *Electr. J. Undergrad. Math.*, **9**:21–35, 2004.

[136] X. Tan, T. Hirata, and Y. Inagaki. An incremental algorithm for constructing shortest watchman route algorithms. *Int. J. Comp. Geom. and Appl.*, **3**:351–365, 1993.

[137] X. Tan and T. Hirata. Constructing shortest watchman routes by divide-and-conquer. In Proc. *ISAAC*, LNCS 762, pages 68–77, 1993.

[138] X. Tan and T. Hirata. Shortest safari routes in simple polygons. LNCS 834, pages 523–531, 1994.

[139] X. Tan, T. Hirata, and Y. Inagaki. Corrigendum to 'An incremental algorithm for constructing shortest watchman routes'. *Int. J. Comp. Geom. App.*, **9**:319–323, 1999.

[140] X. Tan. Fast computation of shortest watchman routes in simple polygons. *Information Processing Letters*, **77**:27–33, 2001.

[141] X. Tan. Shortest zookeepers routes in simple polygons. *Information Processing Letters*, **77**:23–26, 2001.

[142] X. Tan. Approximation algorithms for the watchman route and zookeeper's problems. In Proc. *Computing and Combinatorics*, LNCS 2108, pages 201–206, Springer, Berlin, 2005.

[143] X. Tan and T. Hirata. Finding shortest safari routes in simple polygons. *Information Processing Letters*, **87**:179–186, 2003.

[144] X. Tan. Approximation algorithms for the watchman route and zookeeper's problems. *Discrete Applied Mathematics*, **136**:363–376, 2004.

[145] X. Tan. Linear-time 2-approximation algorithm for the watchman route problem. In Proc. *Theory Applications Models Computation*, LNCS 3959, pages 181–191, Springer, Berlin, 2006.

[146] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, **3**:362–394, 1999.

[147] J. Urrutia. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, editors). pages 973–1027, Elsevier, 2000.

[148] L. G. Valiant and V. V. Vazirani. NP is as Easy as Detecting Unique Solutions. *Theoretical Computer Science*, **47**:85–93, 1986.

[149] V.V. Vazirani. *Approximation Algorithms*. Springer, Berlin, New York, 2001.

[150] B. G. Wachsmuth. *Interactive real analysis*. See
http://www.shu.edu/projects/reals/index.html, 2000.

[151] R. Wolber, F. Stäb, H. Max, A. Wehmeyer, I. Hadshiew, H. Wenck, F. Rippke, and K. Wittern. Alpha-Glucosylrutin: Ein hochwirksams Flavonoid zum Schutz vor oxidativem Stress. *J. German Society Dermatology*, **2**:580–587, 2004.

[152] Wolfram Mathworld. Good Prime. See
$http://mathworld.wolfram.com/GoodPrime.html$ (last visit: October 2006).

[153] C.-K. Yap. Towards exact geometric computation. *Computational Geometry: Theory Applications.*, **7**:3–23, 1997.

Index

Acknowledgments

The authors thank Donald Bailey, Thomas Bülow, Michael J. Dinneen, Ivan Reilly, David Gauld, Joe Mitchell, David Eppstein, Chanderjit Bajaj, and Tetsuo Asano for helpful discussions or comments concerning the problems studied in this report and related problems.

Fajie Li received his PhD in January 2007 from The University of Auckland, New Zealand. He will be on a post-doc position at Rijksuniversiteit Groningen, The Neterlands. His recent email address is still as follows: *fli006@ec.auckland.ac.nz*; comments about subjects related to this report will be very welcomed by Dr. Li.

Reinhard Klette is professor at the computer science department of The University of Auckland, New Zealand; see *www.citr.auckland.ac.nz/ rklette/*.