

Minimal Discrete Curves and Surfaces

A thesis presented

by

Danil Kirsanov

to

The Division of Engineering and Applied Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Applied Mathematics

Harvard University

Cambridge, Massachusetts

September 2004

© 2004 Danil Kirsanov
All rights reserved.

This version of the thesis differs from the original in the following ways:

- Line spacing has been reduced to single-spacing

Minimal Discrete Curves and Surfaces

Abstract

In this thesis, we apply the ideas from combinatorial optimization to find *globally* optimal solutions arising in discrete and continuous minimization problems. Though we limit ourselves with the 1- and 2-dimensional surfaces, our methods can be easily generalized to higher dimensions.

We start with a continuous N -dimensional variational problem and show that under very general conditions it can be approximated with a finite discrete spatial complex. It is possible to construct and refine the complex such that the globally minimal discrete solution on the complex converges to the continuous solution of the initial problem.

We develop polynomial algorithm to find the minimal discrete solution on the complex and show that *embeddability* is a crucial property for such a solution to exist.

Later on, we demonstrate that the spatial complex arising in the minimum weight triangulation problem is not embeddable. Linear programming becomes the main tool to investigate this problem and to develop a practical algorithm for constructing the minimum weight triangulation of a random point set.

Finally, we consider two curve minimization problems arising in computer graphics. Both of them are described by the curves on the piece-wise linear triangulated surface. We revisit the *geodesic* problem, simplify and enhance the well-known exact algorithm and construct the fast approximate algorithm. We also show that the *silhouette simplification* can be done efficiently using the minimization techniques that we develop in the previous chapters.

In many cases, the algorithms described in this thesis can be considered as a generalization of the classical Dijkstra's shortest path algorithm.

Contents

Abstract	iv
Contents	v
List of Figures	viii
List of Tables	xii
Citations to Published Work	xiii
Acknowledgments	xiv
1 Introduction	1
1.1 Overview	3
2 Variational Problems	5
2.1 Contribution	6
2.2 Previous work	7
2.3 Main ideas	10
2.4 Problem definition	13
2.5 Grids	14
2.5.1 Sufficient conditions.	14
2.5.2 Univariate variational problem	15
Regular grids.	16
Random grids.	17
Regular versus random	17
2.5.3 Grids for higher dimensions	18
2.5.4 From grids to discrete problem.	18
2.6 Algorithms	18
2.6.1 Univariate problem	19
Additional constraints	21
Duality for minimum functions	22
Relaxed boundary conditions	24
2.6.2 Minimal cost valid surface	24
2.7 Implementation and results	25
2.7.1 Simple examples	27
2.7.2 Examples with many local minima	28
2.8 Discussion	31

2.9	Proof of the density theorem (2.5.7)	32
2.10	Proof of the density theorem (2.5.8)	34
2.11	Proof of the duality theorem (2.6.4)	35
2.12	Dual graph is bi-chromatic	35
3	Minimum Weight Triangulations	37
3.1	Edge-based and triangle-based definitions of the problem	38
3.2	Previous work on MWT.	39
3.2.1	Previous edge-based linear programming approach.	40
3.3	Minimum weight triangulation problem as a spatial complex	41
3.4	Triangle-based linear program	43
3.4.1	Analysis of the triangle-based integer LP	44
3.4.2	Solving integer LP using continuous LP	45
3.4.3	Double-coverings and non-integer solutions of LP	45
3.4.4	Random experiments	47
	Using previous methods to minimize the number of variables	47
3.5	Discussion	47
3.6	Special cases	49
3.6.1	Proof of the theorem 3.4.4	50
4	Geodesics	52
4.0.2	Related Work	54
4.1	Exact Algorithm	54
4.1.1	Back-tracing the shortest path	55
4.1.2	Intervals of optimality	56
4.1.3	Interval propagation	57
4.1.4	Interval intersection	59
4.1.5	Interval-based algorithm	59
4.1.6	Properties of the interval-based algorithm	60
4.2	Merge simplifications	61
4.2.1	Dependence DAG	62
4.3	Flat-exact algorithm	63
4.3.1	Simplified interval structure and propagation rules	63
4.3.2	Dependance DAG for the simplified algorithm	64
4.4	Results	65
5	Silhouettes	70
5.1	Previous Work	71
5.2	Approach	72
5.2.1	Loop decomposition	72
5.2.2	Loop picking	73
5.2.3	Method B: Loop mapping	74

5.3	Results	76
5.4	Discussion	77
6	Conclusion	80
	Bibliography	82

List of Figures

2.1	Solution of the minimal surface of rotation problem on the regular (a) and random (b) grids. Exact solution is shown as a dashed line. . . .	12
2.2	(a) The graph C_2 can be used to obtain a combinatorial problem that is arbitrarily close to the continuous one. (b) The planar graph C_2^{pl} is constructed by intersection of the set of lines.	16
2.3	Duality for the minimum valid curve problem. (a) A primal graph with boundary conditions. (b) Source and sink faces and edges are added to the primal graph. (c) Dual graph is shown in red. (d) The minimal cut of the dual graph. The edges across the cut are shown in black. (e) A cut of the dual graph correspond to a partition of the primal graph. (f) The boundary of the (R, K) partition is a minimum valid curve on the primal graph. Every edge of the curve corresponds to an edge across the dual cut.	20
2.4	Construction of the dual directed graph. (a) We want the minimal valid curve to be a function $f(x)$. (b) Constraints. If an edge e (top) belongs to the solution, the adjacent upper face belongs to a sink part of the cut and the adjacent lower face belongs to a source part of the cut. We enforce this condition by introducing two directed edges in the dual graph (bottom). The edge with the infinite weight is dashed. (c) The resulting directed dual graph (edges with infinite weights are not shown). Compare it with fig. 2.3c.	22
2.5	Enforcing relaxed boundary conditions. (a) Exterior edges that are within the region allowed by the relaxed boundary condition are shown in green. (b) In the dual graph, the edges that intersect the green edges are removed (removed edges are dashed). (c) Resulting dual graph. .	24
2.6	Results of the plane selection algorithm. Boundary conditions are shown in red. (a) Minimal area surface for the saddle boundary conditions, 150 random planes. (b) Membrane in the gravity field, 200 random planes.	27

2.7	(a) A two-dimensional slice of the density function. Intensity of the color is proportional to the density. (b) Numerical optimization methods converge to the local minimum. (c) The minimal discrete surface using a grid of random planes. (d) Our method followed by numerical relaxation.	30
3.1	$n = 5$. Solution of the edge-based linear program, the edges with non-zero variables x_i are shown. (a) Integer LP gives minimum weight triangulation; $x_i = 1$ for all solid edges. (b) Continuous LP gives non-integer solution which is smaller than (a); $x_i = 1$ for the solid edges, $x_i = 1/3$ for the dashed edges.	41
3.2	A subset of the spatial complex that corresponds to a single triangulation. (a) Triangulation of the set of points on the plane. (b) Random third coordinate is introduced for each vertex.	42
3.3	Single- and double- coverings. (a) Triangulation (single-covering) of 6-point set consists of five triangles. (b) Double-covering of the same set consists of ten triangles. For simplicity, only two triangles are shown, the rest can be obtained by the rotation of this pair around the center point by $4\pi k/5$, where $k = 1, 2, 3, 4$. (c) The same double-covering. All ten triangles are shown (transparent). Note, that this double-covering cannot be split onto two triangulations.	46
3.4	13-point counterexample for Euclidean MWT. (a) Minimum weight triangulation weights 18.798. (b) Solution of the continuous LP corresponds to a double covering with a weight 18.764. Triangles are shown transparent.	46
3.5	Comparison of the linear programming and LMT approach for a random wheel example. (a) Typical random setup for $n = 20$. (b) Experimental probability of success by LMT (green line) and our LP method (blue line) depends on the number of points in the example.	48
3.6	For the convex set S , any non-integer solution \mathbf{x} consists of several triangulations. Here is how we can extract one of them. (a) Fix any triangle with a positive x_i (dark grey) and one of its vertices. Constraint (3.4.4) guarantees that going around this vertex in a clockwise or counter-clockwise direction we can always find a triangle with a positive variable x_j . These triangles are shown in light gray. (b) Fixing the triangles found on the previous step, we can repeat procedure until the whole triangulation is complete.	50
4.1	If the shortest path from the point x_2 of the edge e_2 crosses the edge e_1 , the distance function in this point can be computed by theorem (4.1.1).	55

4.2	When the faces are unfolded on the plane, the shortest path from the the source (or pseudo-source) s to any point on the interval of optimality p is represented as a straight line. c is the distance from the (pseudo-)source to the edge origin, x is a parameter along the edge.	57
4.3	When interval p propagates onto the adjacent edge, it can create up to three new intervals. Central intervals I_c has its (pseudo-)source located in the same point s_p ; left and right intervals I_l and I_r have their (pseudo-)sources located in the points s_l and s_r respectively.	58
4.4	Comparison of the approximate geodesics algorithms for the planar mesh, which is subdivision of a triangle (top). The source is located in the upper corner of the mesh. Kimmel-Sethian algorithm is on the right, our flat-exact one is on the left. Geodesics are shown in green, iso-distant curves are shown in blue.	62
4.5	Edge e_3 with source s is used to compute the propagated distance function in the vertex v_3	63
4.6	Construction of the dependance DAG. The vertex v_3 is updated from the edge e_{12} (red arrow), the blue arrows show the desirable parent-children links. On the right, two possible pseudo-sources for the edge e_{34} are located symmetrically to this edge.	64
4.7	Limitations of our flat-exact algorithms. In the areas where several wavefronts meet together, the one-interval approximation of the edge distance function leads to geodesic distortion. Enlarged area of the bunny head is shown in the middle (flat-exact). Result of the exact algorithm is shown to the right. Geodesics are shown in green, iso-distance curves are shown in blue.	67
4.8	The results of the our exact (top row) and flat-exact (bottom row) algorithms are shown on the cat, bunny, and dragon models (366, 10000, and 50000 vertices respectively). Geodesics are shown in green, iso-distant curves are shown in blue. In the bunny model, the source is on the back side.	68
5.1	Fine silhouette (left) and simple silhouette (right) computed using the loop picking algorithm. The simple silhouette consists of only four loops.	73
5.2	Coarse and fine bands are shown with colored triangles on the bunny modes. In order to show the mapping between models, each triangle on the coarse mesh (left) has the same color as corresponding fine triangles on the fine mesh (right).	74
5.3	One loop of the coarse silhouette of the gargoyle (left) and its mapping on the fine mesh (right).	75

5.4	Primal (left) and dual (right) graphs. Internal boundary (red) on the primal graph corresponds to the source edges on the dual graph. External boundary (blue) on the primal graph corresponds to the sink edges on the dual graph.	77
5.5	Silhouette rendering comparisons of the bunny and gargoyle meshes. Model parameters and benchmarks can be seen in the table 5.1. Front (row 1) and side (row 2) views of bunny mesh silhouette. Front (row 3) and side (row 4) views of gargoyle mesh silhouette.	79

List of Tables

2.1	Comparative properties of the graph-cut functional minimization algorithms.	10
2.2	Experimental time and memory usage by the minimal surface algorithm. N_{planes} is a number of the planes used for the unit cube subdivision. $T_{cell\ tree}$ and $T_{min-cut}$ are the times used for the construction of the cell tree and computation of the min-cut. N_{cells} is the number of the cells in the cell tree.	28
2.3	$2 \cdot 10^6$ possible local minimum surfaces. (a) Local optimization with random starting guess. (b) Global combinatorial optimization with 240 planes. (c) Global optimization enhanced with a plane selection heuristic.	29
2.4	Comparison of the different algorithms for the functional with two level of details.	31
4.1	Simple algorithm for computing minimal distance functions on the edges.	56
4.2	Interval-based algorithm for computing minimal distance functions on the edges. When the interval list is updated, some intervals might be deleted and/or created.	60
4.3	Simplified algorithm for computing minimal distance function.	66
4.4	Time/error statistics for the geodesics algorithms. Time is given in seconds, the relative error is computed by 4.4.1 and shown in percents.	69
5.1	Quantitative results (Pentium 4, 2.0Mhz).	77

Citations to Published Work

The application of the discrete algorithms to the variational problems (chapter 2) first appeared as a technical report and is currently submitted to the ACM Journal of Optimization.

”A Discrete Global Minimization Algorithm for Continuous Variational Problems.”

D. Kirsanov, S. J. Gortler.

Harvard University Computer Science Technical Report: TR-14-04, July 2004.

The geodesic shortest path algorithms (section 4) are described in

”Fast exact and approximate geodesic paths on meshes.”

D. Kirsanov, S. Gortler, H. Hoppe.

Harvard University Computer Science Technical Report: TR-10-04, May 2004.

The silhouette simplification (chapter 5) previously appeared in

”Simple Silhouettes over Complex Surfaces.”

D. Kirsanov, P. V. Sander, S. J. Gortler.

In Proceedings of First Symposium on Geometry Processing 2003.

Acknowledgments

First of all, I would like to thank my advisor Steven Gortler and my fellow students Xianfeng Gu, Pedro Sander, Geetika Tewari, Doug Nachand and Hamilton Chong from Harvard Computer Graphics lab. Without their daily support I would never finish this thesis.

Leonard McMillan provided very interesting references to the pinwheel tiling and Lagrange optimization that otherwise would miss our scope. Hugues Hoppe shared a lot of valuable ideas for the geodesic chapter of this paper.

I would like to thank Chris Buehler for help with development of the discrete duality construction and Tom Duchamp for pointing us to the literature on integral currents. Ray Jones helped us with the directed duality construction and proposed to test our MWT linear program program on L^1 and L^∞ metrics.

I am very grateful to Alex Healy, Erik Demaine, Anton Likhodedov, Ilya and Boris Berdnikov for many helpful discussions.

Danil Kirsanov
Cambridge, MA, September 30, 2004

To my wife Yana.

Introduction

1

The general idea of this thesis is to investigate some problems where the construction of discrete and continuous minimal curves and surfaces is necessary. This statement is very broad, and we limit ourselves with several problems of theoretical and/or practical importance. In particular, we are interested in finding curves and surfaces that are *globally* minimal.

The problems that we examine here come from two different areas. The first area is calculus in variations. We consider a classical formulation of the problem and show that under very general conditions it is possible to find a globally optimal solution. Interesting enough, the discrete formulation of this problem is closely related to the minimal weight triangulation problem.

In the second part of the thesis we consider more practical problems related to computer graphics. Finding the shortest path (geodesic) on the discrete surface is a well-known problem. Though theoretical algorithms are known for a long time, there are a lot of open question on implementation and approximate solutions. Finally, we investigate the problem of silhouette simplification and show that this problem can be successfully addressed by globally minimal approach.

Contribution In this thesis we present the following contributions:

- A novel algorithm for constructing embeddable spatial complexes that approximate continuous minimum of the variational functional.
- A duality algorithm that allows finding the global minimum on the embeddable spatial complex in polynomial time.
- A reduction of the minimum weight triangulation problem to the minimum surface problem on the non-embeddable spatial complex.
- An integer linear program for the general triangle-based minimum weight triangulation problem.
- A continuous linear problem that allows finding the minimal weight triangulations for the random point sets.
- A robust modification of the exact algorithm for finding the minimum length curves (geodesics) on a piece-wise linear surfaces (meshes).
- A fast approximate geodesic algorithm.
- A novel method for mesh silhouette simplification that treats the optimal silhouette as a minimum weight closed curve on a mesh.
- Implementation and testing of all the methods discussed.

1.1 Overview

The thesis consists of four major parts.

Variational Problems. We solve for the best function $f(x) : \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}$ that minimizes some functional

$$\int_{\Omega} \cdots \int G(f(x), \nabla f(x), x) dx. \quad (1.1.1)$$

with boundary conditions $f(\partial\Omega) = \Gamma$.

Under very general conditions this problem can be discretized and related to a problem of finding minimal cost surface in an $(n+1)$ -dimensional spatial complex. The global minimum of the discrete problem can be found with combinatorial optimization techniques (in particular, min-cut).

We pay a special attention to bi-variate variational problems where the solution is a 2D minimal surface in 3D. We also implement the algorithm and compare it to existing numerical techniques.

Minimum Weight Triangulations. Minimal weight triangulation (MWT) problem can also be considered as a globally minimal surface in the discrete face complex. The main difference of this spatial complex from the complex arising in the previous problem is that it is not *embeddable*. It turns out that this property make a dramatic difference.

The complexity of MWT is unknown. We construct the linear programming algorithm and show that integer solution of this problem corresponds to MWT. Though there exist examples where the solution of the linear program is not integer, our approach can be of a practical use for the random point sets.

Geodesics. This chapter describes practical approaches to finding geodesic curves on the triangulated meshes. We modify the classical MMP algorithm to make it robust for the meshes of the large size. Because the exact algorithm is non-linear in time and space, we also develop approximation algorithm that works in close to linear time and space. Both algorithm are implemented and compared with the existing methods.

Silhouettes. Processing silhouettes (edges between front- and back- facing polygons) on meshes is quite a challenging practical problem. It is known that for the complex meshes the silhouettes can be very noisy and complicated.

We develop the method of constructing the approximate silhouette that have the same main "features" as the exact one. It is done by mapping the silhouette from the coarse version of the model to the fine one. The problem can be stated as a

minimal separating curve on the surface of the fine mesh. It can be approached with the combinatorial minimization techniques developed in the previous chapters.

Variational Problems

2

In this chapter we apply the ideas from combinatorial optimization to find globally optimal solutions of the continuous variational problems. More specifically, suppose that one is given a variational problem to solve for the best function $f(x) : \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}$ that minimizes some functional

$$\int_{\Omega} \cdots \int G(f(x), \nabla f(x), x) dx. \quad (2.0.1)$$

with boundary conditions $f(\partial\Omega) = \Gamma$.

Most of the existing numerical methods for this classical problem are based on two approaches: direct methods or Euler-Lagrange differential equation [8, 19]. If we use direct approach, we describe the function f as a linear combination of some set of fixed basis functions $b_i(t)$ with a finite set of coefficients c_i

$$f(x) = \sum_i c_i b_i(x) \quad (2.0.2)$$

Given this finite dimensional problems, one then solves for the best vector of coefficients c_i . This is a standard numerical problem that can be approached with a variety of numerical algorithms [46].

When a finite element set of basis functions is chosen, we are implicitly discretizing the domain $x \in \Omega$. The discretization becomes *explicit* in the important case of the direct methods that is called a finite-difference approximation. In this case we consider the basis functions to be small boxes and directly discretize function f and its gradient. In 1-dimensional case it can be written as

$$f(x_i) \approx c_i, \quad \frac{\partial f}{\partial x}(x_i) \approx \frac{c_{i+1} - c_i}{h} \quad (2.0.3)$$

It can be shown that under certain condition the solution of the variational problem (2.0.1) is also a solution of the differential Euler-Lagrange equation

$$\frac{\partial G}{\partial f} - \sum_{i=1}^N \frac{\partial}{\partial x_i} \frac{\partial G}{\partial f'_{x_i}} = 0. \quad (2.0.4)$$

In general, it is a non-linear (possibly degenerate) equation. Though there exist a lot of methods to solve the differential equation (2.0.4) [46], most of them are based on the discretizations similar to (2.0.2, 2.0.3).

2.1 Contribution

The existing numerical algorithms share the property that (except for simple quadratic problems), one needs to seed the algorithm with some guess c_i^0 which the

algorithm iteratively improves upon. As a result, these methods efficiently converge to some *local* minimum. For some problems this is acceptable, but for many difficult problems, one needs to find the absolute best *global* minimum. The brute force approach to finding the global minimum is to create a large number of starting guesses, and run local optimization from each of these initial states. This is an expensive process, and the problems become even more intractable if one is solving for a higher dimensional problem.

In this chapter we study how combinatorial algorithms (instead of numerical ones) may be applied to these problems to find global minima (instead of local ones). Where numerical algorithms discretize the domain Ω and then use floating point numbers to represent functions $\Omega \subset \mathbb{R}^n \mapsto \mathbb{R}$, we discretize the space $\Omega \times \mathbb{R}$. With this discretization, the optimization becomes completely combinatorial in nature. For example, suppose $\Omega = \mathbb{R}$, then a discretization of $\Omega \times \mathbb{R}$ might be describable as a planar graph (described with a set of vertices and edges) over \mathbb{R}^2 . In this case, a two point boundary value problem reduces to the problem of finding the optimal path in a graph, and it is well known that one can solve for the global discrete minimal path using Dijkstra's algorithm [12]. (see Figure 2.1).

In particular, in this chapter we present the following contributions

- In order to solve the given continuous variational problem, we must be assured that, with enough discretization, the solution to the combinatorial problem will be close to the continuous one. Here we show conditions sufficient to ensure this, and we demonstrate that these conditions can be met with a sequence of deterministic and random grids we construct.
- We prove that, in arbitrary dimension, the resulting combinatorial optimization problem can be reduced to an instance of min-cut, over an appropriate dual graph. In 3D, for example, this gives us a simple algorithm for finding globally minimum discrete surfaces.
- Finally, we describe some important details about the implementation of our algorithm and we show some experimental results demonstrating our method's superiority to traditional numerical approaches.

2.2 Previous work

The literature on finding solutions to variational problems is extensive. For a detailed review, see [13, 19]. Most of the described numerical methods use some form of local descent and can only guarantee convergence to some local minimum. In contrast to all of these methods, our approach has global guarantees even in the presence of local minima.

In the literature, there are a few algorithms that are able to find global minima for certain classes of problems. Here we review and compare them along the following set of axes.

- **Functional:** Some of the methods can only solve for minimal *area* surfaces, where others can minimize energy measured by any first-order functional.
- **Boundary condition:** Some of the methods search for surfaces that have some proscribed boundary, while others look for closed surfaces that surround certain points.
- **Topology:** Some methods solve for surfaces in R^3 , while others solve for scalar functions over the plane (height fields). Surface-based methods typically find surfaces of arbitrary topology. It should be noted that to correctly define the continuous minimal surface problem one needs to employ the mathematical machinery of “integral currents” [41].
- **Discrete solution:** All of these methods solve the continuous problem by discretizing space and solving a combinatorial optimization problem. They show that in the limit, the discrete problem produces a solution to the continuous problem. For some of the methods, the discrete algorithm outputs a specific piecewise linear surface, other methods only output a volumetric “slab” in which the solution must lie.
- **Algorithm** Some of the methods reduce to MIN-CUT which has the complexity of $O(N^2 \log N)$, while some reduce to circulation problems with higher complexity.
- **Experimentation:** Some of the methods are described in publications which include experimental results and implementation details.

Hu et al. describe a method that uses MIN-CUT to compute minimal *area* functions [27, 28]. In particular they find the minimal area a two-dimensional function $f : \Omega \in \mathbb{R}^2 \mapsto [a, b]$ that satisfies boundary conditions $f(\partial\Omega) = \Gamma$. The boundary curve must be “extremal”, which means that it is on the exterior of volume $\Omega \times [a, b]$ in which the surface must lie. In their solution, they discretize space into N volumetric nodes. They then proceed to find the “slab” of nodes of some thickness $d \ll 1$ that satisfies the boundary conditions, separates 3D volume $\Omega \times [a, b]$ into two pieces and has minimal volume. The discretized version of the latter problem can be solved using a MIN-CUT algorithm over an appropriate graph with N nodes and $O(N^2 d^3)$ edges.

Boykov and Kolmogorov have introduced a similar but more sophisticated approach for solving for minimal surfaces using MIN-CUT [6] Similar to Hu, this algorithm has two major degrees of freedom: spatial discretization and connectivity. By

cleverly selecting the weights for the edges in their graph they can solve for a wider class of functionals than simply “area”. In particular, they allow one to choose an arbitrary Riemannian metric over space, and then can solve for the minimal area surface, where area is defined using this new metric. Unfortunately, if one is given a problem in the form of equation (2.0.1) where the function G depends non-quadratically on the surface normal direction, then their algorithm is not applicable.

In their paper, they solve for closed surfaces; to avoid surface collapse, they also impose an energy term which measures which points are inside and outside the closed surface. It seems likely that their approach could be generalized to handle extremal boundary curves like Hu et al.

In both the above algorithms, MIN-CUT is used to divide the volumetric nodes of R^3 into slabs. They show that in the limit, the actual minimal solution lies somewhere in a shrinking volumetric “slab” determined by the cut. These algorithms do not compute a specific discrete surface that minimizes the functional over some discrete space.

In contrast to the above methods, our method is applicable to any first order functional. Our discrete algorithm outputs a specific minimal discrete surface, and can thus be seen as a generalization of the “shortest path problem over a planar graph” to a “smallest surface problem in a spatial CW complex”.

The closest work to ours is the algorithm described in the Ph.D. thesis of John Sullivan [52]. More generally than the above methods, Sullivan solves for the surface that minimizes any arbitrary first order functional. His discrete algorithm outputs a specific discrete surface. In addition he allows his boundary to be any set of loops, including non-extremal loops and knots. Because of his general treatment of boundary conditions his algorithm reduces not to MIN-CUT but to a more complicated network circulation problem. He then solves this problem using an algorithm with time complexity $O(N^2 A \log N)$, where N is the number of nodes in his discrete graph, and A is the complexity of the solution surface. It is unclear if his approach was ever implemented or compared with numerical methods. In addition, due to the amount of difficult machinery employed, this work has remained somewhat inaccessible and unfortunately unappreciated outside of the minimal surface community.

In contrast to the work of Sullivan, by only allowing extremal boundary conditions, we are able to apply a very simple reduction to obtain an instance of MIN-CUT with time complexity $O(N^2 \log N)$. We also describe specific constraints so that we can solve for minimal functions, which is the focus of this thesis, though removal of these constraints would let us solve for minimal surfaces of arbitrary topology as well. In our work, we use random grids for finding approximate solutions which allows us to satisfy complicated boundary conditions and to employ practical local grid refinement techniques. Finally, we describe the details of an implementation and demonstrate how it compares in practice to standard numerical approaches.

To summarize, we compare the relative properties of these approaches in table (2.1).

Algorithm	HKR [28]	BK [6]	Sullivan [52]	Our Method
Continuous functional	Euclidean	Riemannian	general	general
Boundary	external	closed surface	general	external
Boundary condition	exact or free	exact or free	exact	exact or free
Topology	function only	surface only	surface only	function and surface
Discrete solution	volume	volume	piece-wise linear surface	piece-wise linear surface
Method	min-cut	min-cut	min-circulation	min-cut
Experimental justification	no	yes	no	yes
Grids	regular	regular	regular	regular and random

Table 2.1: Comparative properties of the graph-cut functional minimization algorithms.

Our research was specifically inspired by recent success of the combinatorial methods in the discrete energy minimization problems emerged from computer vision [9], [7]. Though the problems that arise in this area are completely discrete from the very beginning, their method of construction of the directed dual graph can be adopted for our purpose.

Duality between the shortest path and min-cut problem for both directed and undirected planar graphs is well known and described in the literature for different applications (see [26] pg.156, [29]). As we will see, this duality can be generalized to higher dimension, and applied to solve variational problems.

2.3 Main ideas

Before we start the formal proofs, we want to illustrate our main ideas on the 1-dimensional case. An advance reader, who immediately wants to see the mathematical definition of the problem, can skip this example and start reading the chapter 2.4.

As an example, we are going to solve the classical minimization problem and find the minimal surface of rotation [8]. In this problem we are looking for the function that minimizes the functional

$$\int_0^1 |f(x)| \sqrt{1 + f_x'^2} dx \quad (2.3.1)$$

given the boundary conditions

$$f(0) = A, f(1) = B. \quad (2.3.2)$$

The analytical solution of this problem is

$$f(x) = A \frac{\cosh(x \cosh(d) + d)}{\cosh(d)},$$

where d is chosen to satisfy the boundary condition (2.3.2).

Our first idea is to discretize *both* the domain x and the range f arriving at a completely combinatorial problem. We will be looking for a solution among the piecewise linear functions described by subsets of a particular grid (Figure 2.1).

In fact, we have to generate a family of the convergent grids. In order for the combinatorial algorithm to be able to give us information about the continuous solution, we must be able to create a grids which, if dense enough, contain discrete paths that are within epsilon of the continuous solution and its derivative. The recipes how to create the convergent grids and the proofs of their correctness can be found in the following chapters.

The integral over the piecewise linear function can be split on the integral over its "edges"

$$\int_0^1 G(f, f', x) dx = \sum_{i=0}^{n-1} \int_{a_i}^{a_{i+1}} G(f, f', x) dx = \sum_{edges} w_i. \quad (2.3.3)$$

For every given edge which is a piece of some line $y(x) = kx + b$ we can compute its weight as

$$w_i = \int_{a_i}^{a_{i+1}} G(kx + b, k, x) dx. \quad (2.3.4)$$

In particular, in the case of the surface of rotation (2.3.1) we have

$$w_i = \sqrt{1 + k^2} \int_{a_i}^{a_{i+1}} |kx + b| dx. \quad (2.3.5)$$

If the entire edge is in the positive or negative semi-plane, this equation can be simplified to

$$w_i = |0.5kx^2 + bx| \sqrt{1 + k^2} \Big|_{a_i}^{a_{i+1}}. \quad (2.3.6)$$

In the cases when variational integral is more complicated and the edge weights (2.3.4) cannot be computed analytically, it can be done numerically.

Now we have a completely discrete problem. We have to find a path with minimum weight from point A to point B on the graph. The weight of the path is a sum of the

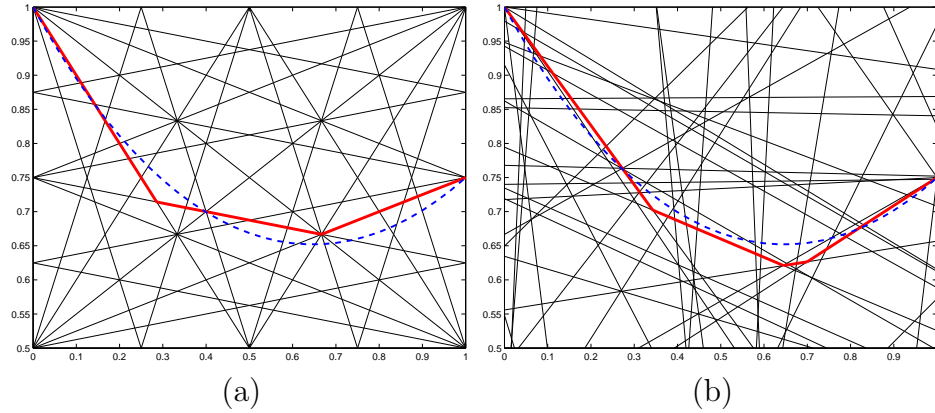


Figure 2.1: Solution of the minimal surface of rotation problem on the regular (a) and random (b) grids. Exact solution is shown as a dashed line.

weights of its edges. All edges are oriented from left to right, because we want the resulting path to be a function, and their weights are known from (2.3.4). Indeed, that is an exact definition of the famous *directed shortest path* problem [12]. There are many efficient algorithms for this problem, best of which have low-order polynomial time complexity [12]. Applying this method we can find the best approximation of the surface of rotation on the given grid (Figure 2.1).

If instead of the boundary conditions (2.3.2) we apply the boundary conditions

$$A_0 \leq f(0) \leq A_1, \quad B_0 \leq f(1) \leq B_1, \quad (2.3.7)$$

the only difference in the algorithm will be in changing the single source shortest path algorithm to the multiple source one. In this case all nodes that are between A_0 and A_1 are considered to be sources, and all nodes between B_0 and B_1 to be destinations.

This type of discretization easily generalized to arbitrary dimensions, where our grid will be constructed by the intersection of the hyper-planes. For example, for variational problems over the plane, $f : \mathbb{R}^2 \mapsto \mathbb{R}$ we will have to construct a minimal surface in 3D. Though minimal path algorithms cannot be generalized for this purpose, as we will describe below, minimal surfaces can be found using duality and combinatorial maxflow/min-cut algorithms. In order for these algorithms to work, we will have to apply some additional restrictions on the grid and functional.

Now we are ready for the formal definition of the variational problem that we are able to solve using our method.

2.4 Problem definition

Our problem is to find the function $f(x) : \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}$ that minimizes some functional

$$F(f) = \int \cdots \int_{\Omega} G(f(x), \nabla f(x), x) dx. \quad (2.4.1)$$

It is common to apply the strict boundary conditions

$$f(\partial\Omega) = \Gamma(\partial\Omega), \quad (2.4.2)$$

but our method will also work with the less restrictive conditions

$$\Gamma_0(\partial\Omega) \leq f(\partial\Omega) \leq \Gamma_1(\partial\Omega). \quad (2.4.3)$$

We require that the domain Ω is a simply-connected polytope, with boundary $\partial\Omega$ homeomorphic to an $n - 1$ -dimensional sphere S^{n-1} . For example, for $n = 2$, Ω is a polygon with no holes and $\partial\Omega$ a closed piece-wise linear curve that is homeomorphic to a circle.¹

In addition, we require our boundary condition(s) Γ to be described as a piece-wise linear function with a finite number of pieces.

We require that the solution of the variational problem $f_s(x)$ belongs to a function space \mathcal{P} of continuous bounded piece-wise twice differentiable functions with bounded first and second derivatives². By "piece-wise" we mean that if $g \in \mathcal{P}$ then there exists a subdivision of the domain Ω into a finite set of non-intersecting polytopes $\Omega = \bigcup \varpi_i^g$, such that on the internal points of every polytope ϖ_i^g the function g is a continuous twice differentiable function with bounded first and second derivatives. Without the loss of generality, we can scale the problem to make sure that

$$0 \leq g(x) \leq 1, \quad \left| \frac{\partial g}{\partial x_i}(x) \right| \leq C_1^g, \quad \left| \frac{\partial^2 g}{\partial x_i \partial x_j}(x) \right| \leq C_2^g. \quad (2.4.4)$$

We also require the kernel of the functional (2.4.1) to be nonnegative³. In other words, for all x, y such that $x \in \Omega$, $0 \leq y \leq 1$, and n -dimensional vectors $p \in \mathbb{R}^n$ we have

¹If we want to be picky, we have to mention that the case $n = 1$ is a minor exception, because Ω is an interval and $\partial\Omega$ is two points.

²Though we will strongly rely on it in the proofs of the correctness of our method, we believe that this assumptions can be weakened in many cases.

³We can weaken the condition (2.4.5), requiring the kernel to be limited from below. Indeed, we know that if the function $f_s(x)$ is a solution of the variational problem with the kernel $G(f, \nabla f, x)$ then it is also a solution of a problem with the kernel $G(f, \nabla f, x) + C$ for any constant C . Therefore, if there exists a constant C such that for all x, y , and p in (2.4.5) we have $G(y, p, x) \geq -C$, then the problem with the kernel $G(f, \nabla f, x) + C$ will satisfy (2.4.5) and have the same solution $f_s(x)$.

$$G(y, p, x) \geq 0. \quad (2.4.5)$$

This ensures that the weights of all edges of graphs that we construct will be non-negative ⁴.

2.5 Grids

Our basic approach is to approximate the continuous variational problem with a discretized combinatorial one which we can globally solve in polynomial time. For this scheme to work, it is essential that the discrete solution gives us meaningful information about the continuous minimum. Here we present sufficient conditions for such an approach to succeed. Then we describe a grid structure that satisfies these conditions.

2.5.1 Sufficient conditions.

Let us assume that our variational problem is to be solved over a space of possible functions \mathcal{P} with a norm $\|\cdot\|$ in it. Our discretized version of the problem will only allow for the representation of some other set $\mathcal{L} \subset \mathcal{P}$. The following lemma states the sufficient conditions to guarantee that our discretized solution will be “close enough” to the actual solution.

Definition 2.5.1. *The functional F is **continuous** in the function space \mathcal{P} , if for any $f \in \mathcal{P}$, and any $\varepsilon > 0$ there exists $\delta > 0$ such that if $\|f - g\| < \delta$ then $|F(f) - F(g)| < \varepsilon$.*

Definition 2.5.2. *The set \mathcal{L} is **dense** in the set \mathcal{P} if for any $f \in \mathcal{P}$, and any $\varepsilon > 0$ there exists $g_\varepsilon \in \mathcal{L}$ such that $\|f - g_\varepsilon\| < \varepsilon$.*

Lemma 2.5.3. *Let \mathcal{P} be a function space, and $\|\cdot\|$ a norm over \mathcal{P} . Let $F(f)$ be a functional that is continuous in \mathcal{P} with respect to $\|\cdot\|$. Suppose that $\mathcal{L} \subset \mathcal{P}$ is dense in \mathcal{P} with respect to the norm. Then for any $\bar{f} \in \mathcal{P}$, any $\varepsilon > 0$, any $\delta > 0$ there exists $g \in \mathcal{L}$ such that $\|g - \bar{f}\| < \delta$ and $|F(g) - F(\bar{f})| < \varepsilon$.*

Proof. Due to the density we can pick a sequence $\{u_n\}_{n=0,1,\dots,\infty} \subset \mathcal{L}$ such that $\lim_{n \rightarrow \infty} \|u_n - \bar{f}\| = 0$. In particular, there exists an integer N_1 such that for all $n > N_1$ we have $\|u_n - \bar{f}\| < \delta$.

Because the functional is continuous, there also exists an integer N_2 such that $|F(u_n) - F(\bar{f})| < \varepsilon$ for all $n > N_2$. Therefore, for any $n > \max(N_1, N_2)$ function $g = u_n$ satisfies the requirements of the lemma. \square

⁴This is not a necessary condition for the shortest path algorithms in 1D, but will be required in higher dimensions.

Corollary 2.5.4. *Let us denote $F_{min} = \inf_{f \in \mathcal{P}} F(f)$. If \mathcal{L} is dense in \mathcal{P} and \mathcal{P} is dense in \mathcal{L} , then $F(g) \geq F_{min}$ for any $g \in \mathcal{L}$, and for any $\varepsilon > 0$ there exists $g \in \mathcal{L}$ such that $|F(g) - F_{min}| < \varepsilon$.*

So, if the functional is continuous and we chose a set of the functions \mathcal{L} that satisfies the conditions of the corollary, we can solve the problem over this set \mathcal{L} and our the energy of our result will be as close to the minimal energy of the original problem as we want. In most cases we can also guarantee that the solution of the discrete problem is close to the solution of the continuous problem under the norm $\|\cdot\|$.

Definition 2.5.5. *Let f_{min} be the global minimum, $F(f_{min}) = F_{min}$. It is called an **isolated global minimum** if for any $\varepsilon > 0$ there exists $\delta > 0$ such that if $\|f - f_{min}\| > \varepsilon$, then $F(f) - F_{min} > \delta$.*

Corollary 2.5.6. *If there exists an isolated global minimum f_{min} , then for any $\varepsilon, \delta > 0$, there exists $g \in \mathcal{L}$ such that $\|g - f_{min}\| < \delta$ and $|F(g) - F_{min}| < \varepsilon$.*

In other words, under very general assumptions we can find “good enough” solution in our numerical subset \mathcal{L} , which can be much simpler then \mathcal{P} . In the next sections we are going to construct such a subset.

2.5.2 Univariate variational problem

Before we go to the higher dimensions, we illustrate our approach for the univariate problems domain.

Here we describe grids for the univariate minimization problems that satisfy conditions (2.4.1 - 2.4.5). Though many constructions are possible, we have chosen this one because it is easily generalizable to arbitrary dimension.

We assume w.l.o.g. that $\Omega = [0, 1]$ and therefore the solution of the univariate variational problem lies in the unit square of the plane $\Phi = \{(x, f) \mid x \in [0, 1], f \in [0, 1]\}$. Define the space \mathcal{P} of the allowable solutions as a set of continuous finite piece-wise twice differentiable functions $f : [0, 1] \mapsto [0, 1]$ with bounded first and second derivatives. We define the integral norm to be

$$\|f\| = \int_0^1 |f(x)| + |f'(x)| \, dx. \quad (2.5.1)$$

This norm includes a term for both the value *and* the derivative of the f . As a result, our functional F , which depends on both value and derivative, is continuous with respect to this norm. Essentially, this means that two “close” functions must have similar energy measurements. This continuity is needed in the preconditions of Lemma 2.5.3.

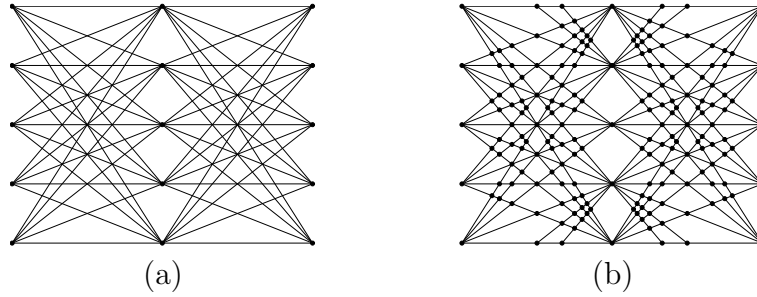


Figure 2.2: (a) The graph C_2 can be used to obtain a combinatorial problem that is arbitrarily close to the continuous one. (b) The planar graph C_2^{pl} is constructed by intersection of the set of lines.

Our strategy is to piecewise linearly embed a planar graph in this square, and then represent possible functions $g(x)$ as piecewise linear paths in this embedded graph. Every path in the graph with monotonically increasing x , from 0 to, 1 corresponds to a piecewise linear function $g : [0, 1] \mapsto [0, 1]$. The set of all possible piecewise linear functions generated by the grid R we denote as $\mathcal{M}(R)$. For a graph with a finite number of vertices the set $\mathcal{M}(R)$ is also finite. Obviously, $\mathcal{M}(R) \subset \mathcal{P}$.

Regular grids.

For univariate minimization problems, we will use a grid such as one shown in Figure (2.2a). This grid, C_n , has $n + 1$ columns and $n^2 + 1$ rows of vertices. The distance between the columns is therefore equal to $h = n^{-1}$ and the distance between the rows is $d = h^2 = n^{-2}$. Edges are included between each pair of vertices in adjacent columns. This graph has about n^3 vertices and n^5 edges. The number of the possible functions generated by this grid $|\mathcal{M}(C_n)| = n^{(2n+2)}$. Fortunately, we will be able to find the global minimum of these possible functions with efficient combinatorial minimization techniques. This grid has been chosen so that it can represent functions with a wide variety of values *and* derivatives. In particular we can state the following:

Theorem 2.5.7. *The set*

$$\mathcal{L} = \bigcup_{n=1}^{\infty} \mathcal{M}(C_n)$$

is dense in \mathcal{P} with respect to our norm.

The proof of the theorem is given in the appendix 2.9.

Combined with the corollary (2.5.6), this theorem states that if the continuous problem has an isolated minimal solution, then it is possible to solve the variational problems on the grids C_n , and the discrete solution will converge to the actual solution of the continuous problem when $n \rightarrow \infty$. Note, that our definition of the

grid does not depend on the variational problem. In fact, any continuous variational problem that has a solution in the set \mathcal{P} can be solved on this grid.

The grid C_n is not planar. As described below, for higher dimensional problems, we will need to have an embeddable grid. The grid C_n can be easily modified to obtain the embeddable grid C_n^{pl} as shown in Figure(b). These planar graphs are also dense in \mathcal{P} because they include all piece-wise linear functions from \mathcal{L} , $\mathcal{M}(C_n^{pl}) \supset \mathcal{M}(C_n)$.

Random grids.

The described regular grid structure can be created as the intersection of sets of lines. As n increases, the lines become more "dense", meaning that in the vicinity of every point of the plane there pass more and more lines with different positions and orientations. Random grids have the same property, and thus can be used for solving our continuous variational problems.

The random grid R_n in the unit square of the plane Φ is created by intersection of n random lines. One vertex is created at each line intersection, and the resulting graph is planar. Each line is constructed by taking the point $p \in \Phi$ and orientation $\leq c \in [-\pi, \pi]$ from some probability distributions⁵. The grid R_{n+1} can be constructed by adding one random line to R_n . Such families of the grids has a nice property that $\mathcal{M}(R_{n+1}) \supset \mathcal{M}(R_n)$. The set of all functions generated by the family we will denote as

$$\mathcal{L}_r = \bigcup_{n=1}^{\infty} \mathcal{M}(R_n).$$

Similar to the regular case, we can state the density theorem

Theorem 2.5.8. *The set \mathcal{L}_r is dense in \mathcal{P} .*

The proof of the theorem is given in the appendix 2.10.

Therefore, we can solve the variational problems in the domain of the random grids.

Regular versus random

We have shown that univariate variational problems can be numerically solved in the domains of the regular and random grids. What domain should be chosen depends on the type of the problem. Here we emphasize the major tradeoffs between these approaches.

It can be clearly seen that regular grids C_n are based on the simple blocks that can be pre-computed for all reasonable n and stored in data files. The graph for the given problem can be constructed by joining the blocks together. Such a structure gives us another advantage when the kernel of the functional $F(f, f', x)$ does not depend

⁵In this thesis we will consider uniform distributions unless the different distribution is mentioned.

on x and the computation of the edge weight is expensive. In this case we have to compute the weights for the small fraction of the edges and then simply copy these values for the whole structure.

Random grids are more flexible. We can experiment with the density distributions of the lines, making them more "dense" in the vicinity of the possible solution. In order to produce an "average" answer and "deviation" we can run the algorithm several times on different random grids. These advantages are often dominant because for big meshes the time for the grid construction and weight computing is usually small comparing to the other parts of the algorithm. In our implementation we pursued the use of random grids.

The solution for the function generating a minimum surface of rotation using both types of grids can be seen on the Figure (2.1).

2.5.3 Grids for higher dimensions

The described grid structure can be created as the intersection of the set of lines. The generalization in higher dimensions is straightforward; the grid is created as the intersection of the set of hyperplanes in the $n + 1$ -dimensional space, $\Omega \times [0, 1]$. With this grid structure, some function space of piecewise linear functions over Ω can be represented using appropriate subsets of facets. The proof of the correctness of these constructions for the higher dimensions is omitted for the sake of space.

Though regular grids have a clear structure and might be easier to build and investigate, random grids are more flexible. Similar to the 1-dimensional case, we can experiment with the density distributions for the hyperplanes, making them more "dense" in the vicinity of the possible solution. It is also much easier to construct random grids that fit the given boundary conditions.

2.5.4 From grids to discrete problem.

The weight of the face can be computed by evaluating the integral (2.4.1) numerically or analytically. For instance, for bivariate variational problem every face of the grid C_n is a convex polygon located on some plane $z(x, y) = Ax + By + C$ with the constant gradient $\nabla z = (A, B)^T$. The cost of the face can be computed as

$$w_i = \iint_{face} G(Ax + By + C, (A, B)^T, x, y) dx dy. \quad (2.5.2)$$

2.6 Algorithms

We now have approximated our continuous variational problem by a series of discrete optimization problems. In the univariate case, the discrete problem is a

shortest path problem, and in the bivariate case is a minimal discrete surface problem. We now wish to find the global optima for these problems. For the shortest path problem, we could simply use a standard shortest path algorithm, such as Dijkstra's algorithm on the grid. Unfortunately, it is not clear how to use these algorithms to solve for minimal surfaces.

In this section we show how both the discrete path and surface problems may be globally solved in polynomial time. The basic approach will be to generate an appropriate dual graph from the grid structure. This dual graph will be constructed so that "cuts" of the graph correspond to paths (surfaces) in the original grid structure. We can then apply well known algorithms to solve for the min-cut of the dual graph. For simplicity, we first demonstrate how these ideas work for the univariate problem. The topological machinery we use is possibly a tad bit heavy for the univariate case, but will be sufficient to generalize to bivariate case. In fact, this machinery can be applied to variational problems of any dimension.

2.6.1 Univariate problem

For the discrete univariate problem, we are given a planar graph (V, E) that is embedded in \mathbb{R}^2 , with positive weights on each of its edges. We will label the axes of \mathbb{R}^2 as x and y . Because the graph is embedded, we have not only vertices and edges, but also 2D faces. This collection of vertices edges and faces defines a complex (formally it is a CW complex [22]) that is homeomorphic to the 2D ball (disk) B^2 .

We wish to compute the minimum weight path that connects two specified vertices that are on the exterior of this complex. In addition we will want to ensure that the our path corresponds to a function in the original variational setting; that is, there should be one y value over each x value. We will deal with this restriction later on in section (2.6.1).

The two boundary vertices A and B partition the exterior edges into two sets which we call upper U and lower L . We introduce two additional faces, f_r and f_k , in the graph. The source face f_r is surrounded by edges in L and one additional outer edge e_r . The sink face f_k is constructed by edges in U and one additional outer face e_k . With these two added faces and edges, the new complex is still homeomorphic to B^2 (see Figure 2.3).

The following are simple definitions from *simplicial homology modulo 2* [18].

Definition 2.6.1. A **1-chain** in the graph is a union of some edges of the graph $\mathcal{C} = \cup e_i$. The **boundary** of a 1-chain \mathcal{C} , which is denoted as $\partial_1 \mathcal{C}$, is a union of vertices that are included in \mathcal{C} an odd number of times.

Similarly, we define a **2-chain** as a union of some faces of the graph $\mathcal{A} = \cup f_i$. The **boundary** $\partial_2 \mathcal{A}$ of a 2-chain \mathcal{A} is a union of edges that are included in \mathcal{A} an odd number of times ⁶.

⁶For this 2d complex, any edge can belong to no more then two faces. This is not going to be

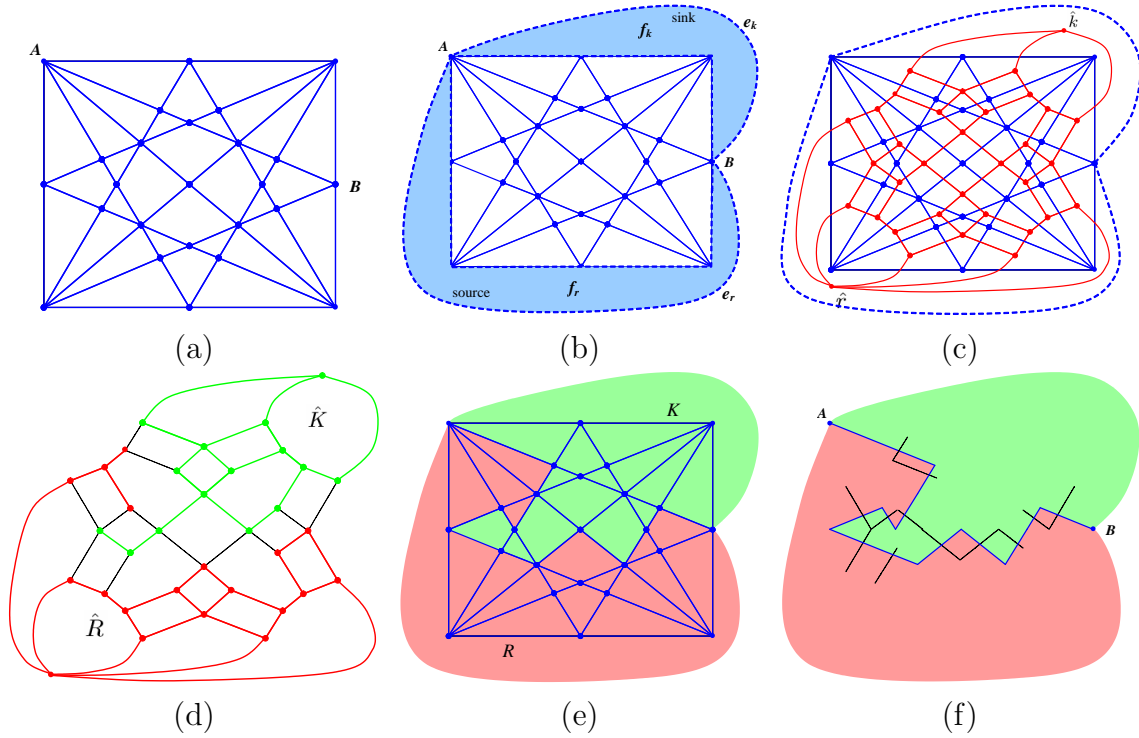


Figure 2.3: Duality for the minimum valid curve problem. (a) A primal graph with boundary conditions. (b) Source and sink faces and edges are added to the primal graph. (c) Dual graph is shown in red. (d) The minimal cut of the dual graph. The edges across the cut are shown in black. (e) A cut of the dual graph correspond to a partition of the primal graph. (f) The boundary of the (R, K) partition is a minimum valid curve on the primal graph. Every edge of the curve corresponds to an edge across the dual cut.

For our problem, we are interested only in 1-chains that satisfy boundary conditions $\Gamma = \{A, B\}$.

Definition 2.6.2. Let us define a **valid curve** on the graph described above to be any 1-chain \mathcal{C} with $e_r \notin \mathcal{C}$ and $e_k \notin \mathcal{C}$ such that $\partial_1 \mathcal{C} = \Gamma$. The cost of a valid curve is a sum of the weights of its edges.

The *minimal valid curve problem* is to find a valid curve with the minimal cost.

As it is proven in section 2.11, this is easily accomplished using a dual graph D (Figure 2.3, c). That is, we associate with each face f_i a dual vertex \hat{v}_i . We also associate a single source vertex \hat{r} with a source face f_r and a single sink vertex \hat{k} with a sink face f_r . With each edge e_i bounding two faces f_j and f_k , we associate a dual

true for higher-dimensional problems.

edge \hat{e}_i that connects \hat{v}_j and \hat{v}_k . We set the *capacity* of each dual edge the cost of the associated primal edge $c(\hat{e}_i) = c(e_i)$.

Definition 2.6.3. A **cut** is a partition of the vertices \hat{v} into two sets \hat{R} and \hat{K} with $\hat{r} \in \hat{R}$ and $\hat{k} \in \hat{K}$. The **cost** of a cut $c(\hat{R}, \hat{K})$ is the sum of the capacities of the edges between \hat{R} and \hat{K} .

Because there is a one-to-one correspondence between the vertices of the dual graph and the faces of the primal graph, we can also denote the cut as a partition of the faces of the primal graph into two 2-chains R and K with $f_r \in R$, $f_k \in K$.

Theorem 2.6.4. There is a one-to-one correspondence between the valid curves on the primal graph and cuts on the dual graph. The minimal valid curve problem over the primal undirected graph is equivalent to a minimal cut problem for the dual graph.

The proof of the theorem is given in the appendix 2.11.

The minimal cut is a well-known problem that can be solved in polynomial time.

Additional constraints

In the previous section we considered a general problem of finding a minimal valid curve in G . Here we describe how to solve its restricted version where we want to find a minimal curve with some special properties. Our final purpose is to ensure that the chosen curve is a *function*.

Theorem (2.6.4) states that every valid curve \mathcal{C} corresponds to a cut that is denoted as $(R_{\mathcal{C}}, K_{\mathcal{C}})$ in the primal graph and $(\hat{R}_{\mathcal{C}}, \hat{K}_{\mathcal{C}})$ in the dual graph. If the curve passes through an edge e , one of its adjacent faces belongs to the source 2-chain $R_{\mathcal{C}}$ and the other one belongs to the sink 2-chain $K_{\mathcal{C}}$. We specify an additional constraint to be satisfied when the curve passes through this edge.

Constraint 2.6.5. Let e to be an edge of the primal graph. We take one of its adjacent faces and denote it as f_e^R , the other one is denoted as f_e^K . We also denote corresponding vertices in the dual graph as \hat{v}_e^R and \hat{v}_e^K . If $e \in \mathcal{C}$, we require that $f_e^R \in R_{\mathcal{C}}$ and $f_e^K \in K_{\mathcal{C}}$ (similarly, $\hat{v}_e^R \in \hat{R}_{\mathcal{C}}$, $\hat{v}_e^K \in \hat{K}_{\mathcal{C}}$).

This condition can be specified independently for every edge (and it is quite possible that some face f belongs to a source 2-chain for one edge and belongs to the sink 2-chain for another edge). Note, that the constraints do not depend on the particular curve \mathcal{C} and simply narrow the set of possible valid curves on the primal graph.

In order to satisfy constraints (2.6.5), instead of undirected dual graph, we create the *directed* dual graph \hat{G} . If the constraint (2.6.5) is specified for the primal edge e , then the corresponding edge \hat{e} of the undirected dual graph is split into two directed edges \hat{e}^R (from \hat{v}_e^R to \hat{v}_e^K) and \hat{e}^K (from \hat{v}_e^K to \hat{v}_e^R). The capacities of the dual edges are defined as $c(\hat{e}^R) = c(e)$ and $c(\hat{e}^K) = \infty$. If the constraint is not specified for e , then the capacities are set as $c(\hat{e}^R) = c(\hat{e}^K) = c(e)$.

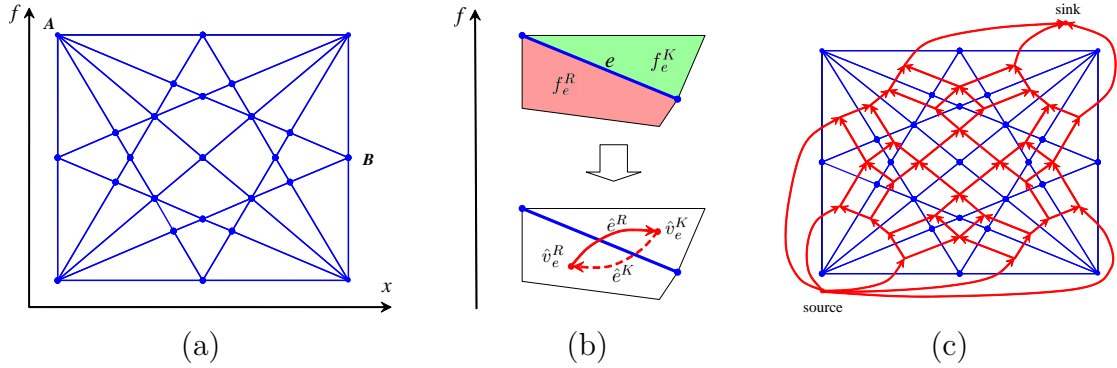


Figure 2.4: Construction of the dual directed graph. (a) We want the minimal valid curve to be a function $f(x)$. (b) Constraints. If an edge e (top) belongs to the solution, the adjacent upper face belongs to a sink part of the cut and the adjacent lower face belongs to a source part of the cut. We enforce this condition by introducing two directed edges in the dual graph (bottom). The edge with the infinite weight is dashed. (c) The resulting directed dual graph (edges with infinite weights are not shown). Compare it with fig. 2.3c.

Definition 2.6.6. The **cost** of a cut $c(\hat{R}, \hat{K})$ in \hat{G} is the sum of the capacities of the edges from \hat{R} to \hat{K} .⁷

Theorem 2.6.7. Let \mathcal{C} be a valid curve on the primal graph with constraint (2.6.5) defined for every edge, (\hat{R}, \hat{K}) be a corresponding cut of the dual directed graph. The curve \mathcal{C} satisfies the constraints if and only if the cost of the cut is finite, $c(\hat{R}, \hat{K}) < \infty$. Therefore, min-cut of the dual directed graph corresponds to the curve that has the minimal weight and satisfies all of the constraints (2.6.5).

Proof. There is a one-to-one correspondence between valid curves and cuts.

If \mathcal{C} satisfies the constraints (2.6.5), the weight of the corresponding cut is finite because all the edges from \hat{R} to \hat{K} are finite by construction.

If the cut has a finite weight, every edge \hat{e} from \hat{R} to \hat{K} is finite. Therefore, every edge e of the corresponding curve satisfies the condition, otherwise $c(\hat{e})$ would be infinite. \square

Duality for minimum functions

Thus far, the particular embedding of the planar primal graph has not been important. In our application, the primal graph represents a grid which has been used to discretize the univariate variational problem (section 2.5.2). In particular, this

⁷Compare with the definition (2.6.3) of the undirected graph cut cost.

primal graph is embedded in the rectangular region $[0, 1] \times [0, 1]$ and every vertex graph has two coordinates $v_i = (x_i, y_i)$. At least one of the boundary vertices has its first coordinate equal to 0, and at least one vertex has its first coordinate equal to 1. The edges of the primal graph (except for $e_{\hat{r}}$ and $e_{\hat{k}}$) are straight line segments.

Only some valid curves on this graph will correspond to (piece-wise linear) functions $f : [0, 1] \mapsto [0, 1]$. We wish to add our additional constraints in such a way as to restrict our solution to this subset.

One can think of every piece-wise linear function f as dividing $[0, 1] \times [0, 1]$ into two regions, one of them is "below" the function and the other, above. In our cut solution, we want the region below the function to be the source-connected 2-chain R_f .

Therefore, for every edge e of the primal graph we introduce an additional condition. If the edge belongs to the output function f , we require the face below the edge to belong to R_f and the edge above f to belong to K_f . This is exactly a constraint (2.6.5) described in the previous section.

Formally, the additional constraints are defined as follows. Every primal edge e_i is a segment of a line $y = a_i x + b_i$ ⁸. The adjacent face in the half-plane $y < a_i x + b_i$ is denoted as $f_{e_i}^R$, the face in the half-plane $y > a_i x + b_i$ is denoted as $f_{e_i}^K$. The constraint for e_i is defined as (2.6.5). The construction of the directed dual graph for these constraints is described in (2.6.1). Example of the dual graph can be seen on the figure (2.4).

Theorem 2.6.8. *The curve \mathcal{C} is a function if and only if for every edge it satisfies the constraint defined above.*

Proof. If the curve is a function, it satisfies the constraints by definition. Indeed, all the faces below the function are in the source 2-chain, all the faces above the function are in the sink 2-chain.

Let us now prove that if a curve satisfies the constraints above, it is a function, i.e. the curve \mathcal{C} defines a unique mapping $f : \Omega \mapsto [0, 1]$. Let's prove it by contradiction. If the mapping is not unique in some point $x \in \Omega$, there exist two edges $e_1, e_2 \in \mathcal{C}$ and two points on these edges with the same first coordinate $(x, y_1) \in e_1, (x, y_2) \in e_2$. Without loss of generality we can assume that $y_1 < y_2$ and there are no other edge $e_3 \in \mathcal{C}$ and $(x, y_3) \in e_3$, such that $y_1 < y_3 < y_2$.

By definition the face $f_{e_1}^K$ is located above the edge e_1 and belongs to a sink 2-chain $K_{\mathcal{C}}$. Similarly, the face $f_{e_2}^R$ is located below the edge e_2 and belongs to a sink 2-chain $R_{\mathcal{C}}$. Let us consider the set F of all faces intersected by an open segment $((x, y_1), (x, y_2))$. On the one hand, this segment does not intersect any edges of \mathcal{C} and therefore either $F \subset R_{\mathcal{C}}$ or $F \subset K_{\mathcal{C}}$. On the other hand, $f_{e_1}^K \in F$ and $f_{e_2}^R \in F$, which is a contradiction. Therefore, the mapping is unique. \square

⁸We can avoid the vertical edges in two ways. The first one is to "shake" all vertices of the primal graph by some small random value to make sure that there are no vertical edges in the graph. The second one is to assign infinite weight with both of the dual edges.

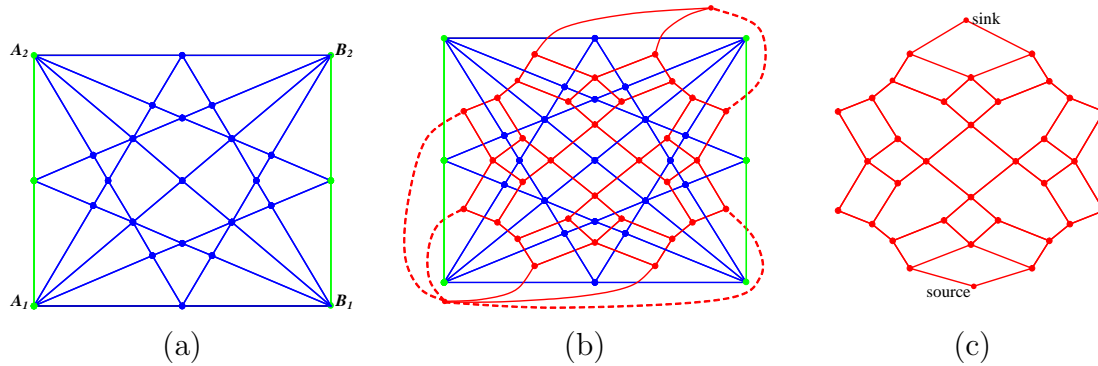


Figure 2.5: Enforcing relaxed boundary conditions. (a) Exterior edges that are within the region allowed by the relaxed boundary condition are shown in green. (b) In the dual graph, the edges that intersect the green edges are removed (removed edges are dashed). (c) Resulting dual graph.

Corollary 2.6.9. *The minimal function problem on the grid can be reduced to the minimal cut problem for the directed dual graph defined above.*

Relaxed boundary conditions

So far we considered the strict boundary conditions (2.4.2); in the univariate case it can be stated as $f(0) = A$, $f(1) = B$. In fact, by a very small modification we can allow the relaxed boundary conditions (2.4.3); in the univariate case it can be stated as $A_1 \leq f(0) \leq A_2$, $B_1 \leq f(1) \leq B_2$.

To enforce these constraints (see figure 2.5), we remove from the dual graph all edges (shown in dotted red) that are dual to exterior edges (shown in green) that are within the region allowed by the relaxed boundary condition.

The removal of these dual edges is equivalent to assigning a zero cost to their associated primal edges. This allows the boundary of the function to "slide" up and down in the allowed region with no added cost. It can be easily seen that after this modification there is a one-to-one correspondence between functions that satisfy the relaxed boundary conditions and cuts in the dual graphs.

2.6.2 Minimal cost valid surface

The generalization of the shortest path problem to higher dimensions is straightforward⁹. Here we show its application in 3D. In the discrete spatial minimal cost

⁹Unlike the minimal curve problem, which can be solved efficiently even for non-planar graphs, the minimal cost surface problem can only be solved efficiently for a complex that is embedded in three dimensional space. Without this restriction, the minimal cost two dimensional function problem becomes NP-Complete by reduction to min-cost planar triangulation [43].

surface problem a three-dimensional spatial complex G is homeomorphic to B^3 and consists of vertices v_i , edges e_i , faces f_i , and cells c_i . Associated with each face f_i is a cost w_i . Also input is a closed boundary 1-chain Γ , which is on the exterior of the complex and which is homeomorphic to a circle S^1 .

The boundary Γ partition the exterior faces of the complex into two sets which we call upper U and lower L . These sets are both disks, due to the Jordan Curve Theorem. Following the recipe of section (2.6.1), we attach a source and a sink cell c_r and c_k to the complex. The source cell c_r is bounded by the faces of L and one additional outer face f_r . The sink cell c_k is bounded by the faces of U and one additional outer face f_k . This new complex is still homeomorphic to the ball B^3 .

For our problem, we are interested only in 2-chains that satisfy the boundary condition Γ .

Definition 2.6.10. *A valid surface in the spatial complex is any 2-chain \mathcal{A} with $f_r \notin \mathcal{A}$ and $f_k \notin \mathcal{A}$ such that $\partial_2 \mathcal{A} = \Gamma$.*

Note that under this definition, our “surface” can include handles, have disconnected components, or even be non-manifold.

The *minimal valid surface problem* is to find a valid surface with minimal cost.

Omitting the intermediate steps that were described in details for the univariate case, we construct the directed dual graph as follows. That is, we associate with each cell c_i a dual vertex \hat{v}_i . A source cell c_r corresponds to a source dual vertex \hat{r} , a sink cell c_k corresponds to a sink dual vertex \hat{k} . Each non-exterior face f_i is a segment of a plane $z = a_i x + b_i y + d_i$ and has two adjacent cells. One of them, $c_{f_i}^r$, is below the plane, the other one, $c_{f_i}^k$ is above the plane. In the dual directed graph, with each face f_i we associate two dual edges. One of them, \hat{e}_i^r , goes from $\hat{v}_{f_i}^r$ to $\hat{v}_{f_i}^k$, the other one, \hat{e}_i^k , goes the opposite direction. The capacities of the directed edges are set as $c(\hat{e}_i^r) = w(f)$, $c(\hat{e}_i^k) = \infty$.

Theorem 2.6.11. *The minimal function problem on the grid can be reduced to the minimal cut problem for the directed dual graph defined above.*

The proof of this theorem is identical to that of the univariate case, except that is based on 3-chains and their properties in a complex homeomorphic to B^3 .

For the relaxed boundary case, we have an upper and lower boundary curve, Γ_1 and Γ_2 . In the graph construction we remove from the dual graph all edges that are dual to exterior primal *faces* that are within the region allowed by the relaxed boundary condition.

2.7 Implementation and results

We implemented our algorithm for the case of minimal cost surfaces. Again, the corresponding spatial complex is 3-dimensional.

The spatial complex is represented using a BSP-tree data structure. In this tree, the root node is associated with the volume of the entire bounding 3D cube. Random planes entered into the tree one by one. As each plane is entered, each node whose associated volume is intersected by the plane is split into two "child" nodes representing the associated halves of the parent node's volume. At the end of this construction, each leaf of the tree represents a cell in the spatial complex. The nice property of this representation is that the leaves that are intersected by a hyperplane can be located hierarchically, because the hyperplane intersects the child cell in the binary tree only if it intersects the parent cell. Some interesting properties of the dual graph are investigated in the section 2.12.

Integrals over each of the faces (2.5.2) are computed analytically or evaluated numerically by Monte-Carlo methods. We also used the elegant implementation of the min-cut algorithm written by Kolmogorov [5].

For problems with a fixed boundary condition Γ we force a small number of the planes (usually, less than 10% of the total number of the planes) to go through the piece-wise linear segments of the boundary constraints. In other words, each segment of the boundary becomes a pencil of planes; the orientations of the planes are chosen randomly.

As we have proven above, as the number of the planes goes to infinity, the discrete surface will approach the continuous minimal surface. Unfortunately, our computing constraints currently limit us to the use of only few hundreds of planes. In order to deal practically with these limitations we have explored the use of a number of heuristic-extensions.

Relaxation. One obvious extension which we call *relaxation* is to use the output of the our combinatorial optimization as the starting guess for a numerical iterative method. Because the numerical method only has to discretize the 2D domain, it can be discretized to a much higher resolution. The goal of this step is to refine the answer to a high-resolution, highly accurate local minima that is nearby to the combinatorial global optimum. For our numerical method, we employed a multi-resolution gradient descent method. The multi-resolution hierarchy was made up of three grids with sizes 20x20, 40x40, and 80x80. A finite differencing approach was used to derive the discrete optimization problem.

Plane selection. Because we use random grids, a natural way to use the method is to run the discrete algorithm several times and to pick up the solution with the smallest energy. We can use this idea to develop the following extension, which we call *plane selection*. With this extension we run our discrete algorithm several times for a single problem. After each iteration, we keep the planes that have been included in the current minimum, drop the planes that are not included in the minimum, and replace these with new random planes. At each iteration, the

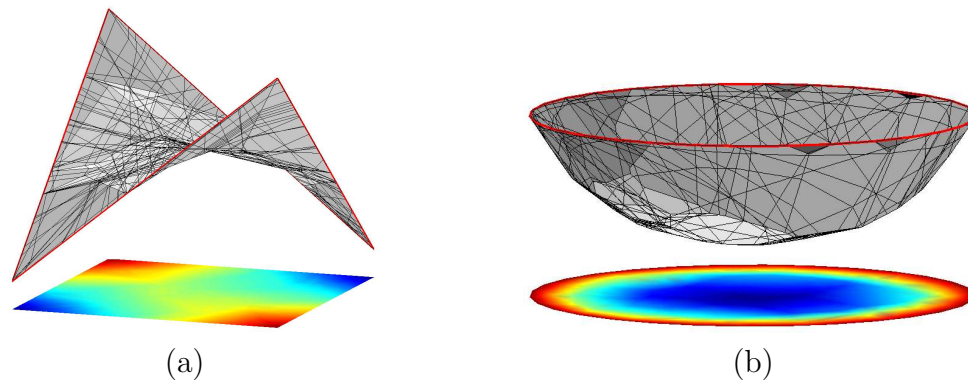


Figure 2.6: Results of the plane selection algorithm. Boundary conditions are shown in red. (a) Minimal area surface for the saddle boundary conditions, 150 random planes. (b) Membrane in the gravity field, 200 random planes.

energy must necessarily not increase. Of course, plane selection can be coupled with relaxation.

Adaptive Refinement The last extension, to adaptively refine the grid by only subdividing cells that lie near to the current solution. More specifically, we run the algorithm a number of times and compute an average solution as well as deviation from this average. From these data we construct an upper and lower boundary (“envelope”) of the desired answer. In the succeeding iterations we add more random planes, but all the cells above (resp. below) the envelope are merged with the source (resp. sink).

2.7.1 Simple examples

We first show how our algorithm behaves on some simple classical variational problems. Next we show how our algorithm works on a problem that has numerous local minima.

Our first example is the famous problem of finding the function of the minimal area that “spans” a given boundary curve. Such a function $f(x, y)$ minimizes the integral

$$F_1(f) = \iint_{\Omega} \sqrt{1 + f'_x(x, y)^2 + f'_y(x, y)^2} dx dy. \quad (2.7.1)$$

Figure 2.6a shows the result when using a saddle boundary configuration.

A standard elasticity problem is the deformation of a membrane in a gravity field.

Such a membrane function $f(x, y)$ minimizes the integral

$$F_2(f) = F_1(f) - \iint_{\Omega} k(x, y)f(x, y)g \, dx \, dy \quad (2.7.2)$$

In Figure 2.6b, we show a solution with a constant density $k(x, y) = c$, though nothing changes in our method if we use any other distribution of the density or gravity field.

The space and time costs of our algorithm can be seen in the table (2.2). In this experiment a cubic cell Φ was subdivided by N_{planes} random planes. We measured time of the construction of the cell tree, time of the min-cut computation, number of the total cells in the cell tree, and total memory usage. The experiments were performed at Athlon 1.5GHz machine with 2GB of memory. From the table, we observe that the major bottlenecks of the algorithm are memory usage and min-cut computation time. In theory the worst case for n planes has n^3 memory complexity and $n^6 \log n$ time complexity. From experiments with various sized problems, we observed that in practice the algorithm had $n^{2.7}$ memory complexity and $n^{4.6}$ time complexity.

N_{planes}	$T_{cell \, tree}$, sec.	$T_{min-cut}$, sec.	N_{cells} , thousands	Memory used, Mb
40	0.5	0.5	11	3
60	2	3	40	13
80	4	14	107	35
100	8	34	205	68
120	14	95	375	123
140	22	210	590	195
160	34	380	903	300
180	48	830	1300	428
200	68	1350	1810	596
220	90	2100	2410	790
240	117	3400	3150	1040

Table 2.2: Experimental time and memory usage by the minimal surface algorithm. N_{planes} is a number of the planes used for the unit cube subdivision. $T_{cell \, tree}$ and $T_{min-cut}$ are the times used for the construction of the cell tree and computation of the min-cut. N_{cells} is the number of the cells in the cell tree.

2.7.2 Examples with many local minima

Here we experimented with a class of problems that has a huge number of local minimums and can hardly be solved in reasonable time by existing methods.

Our problems were created by starting with a 3D unit cube that has some given 3D density distribution $D(x, y, z)$ in it. Our problem was then to find the surface that

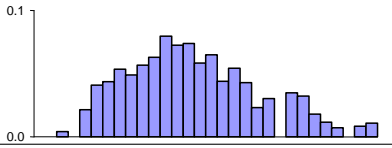
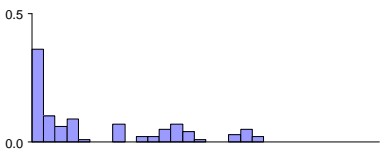

	time, <i>sec</i>	memory, <i>Mb</i>	energy histogram
a) local optimization, starting from random guess	13.5	3	
b) combinatorial optimization, 240 planes, + relaxation	3590	1040	
c) plane selection, 150 planes, 5 iterations, + relaxation	1365	200	

Table 2.3: $2 \cdot 10^6$ possible local minimum surfaces. (a) Local optimization with random starting guess. (b) Global combinatorial optimization with 240 planes. (c) Global optimization enhanced with a plane selection heuristic.

breaks the cube into two pieces using minimal effort. In this case we must minimize the functional

$$\iint_{\Omega} D(x, y, f(x, y)) \sqrt{1 + f'_x(x, y)^2 + f'_y(x, y)^2} dx dy \quad (2.7.3)$$

We only enforced the relaxed condition that the boundaries map to $[0..1]$.

The distributions were created by summing up a number of 3D Gaussian blobs. We chose m locations over Ω , above each location we placed k Gaussian blobs at random heights. This results in $(k + 1)^m$ locally optimal functions. The density distribution can be very complex; a 2D slice of such a distribution is shown in Figure (2.7,a).

In our experiments we compared our algorithm against a standard numerical optimization method. For the numerical method, we used a multi-resolution gradient descent method with random starting conditions. We terminated the iterations when the improvements were below 10^{-5} .

For our first experiment we constructed a density field with $m = 9$ and $k = 4$, there are about $2 \cdot 10^6$ possible local minimums for this example. The results of the experiment can be found in the table 2.3. In the rightmost column, we show a histogram of the minimal energies found by each method. We have shifted and the energies so that the leftmost column only counts solutions that are within epsilon of the absolutely smallest energy reported by any of the algorithms. (Since we do not have a closed form solution for this problem, we, of course, have no proof that this in-fact represents the global minimum.)

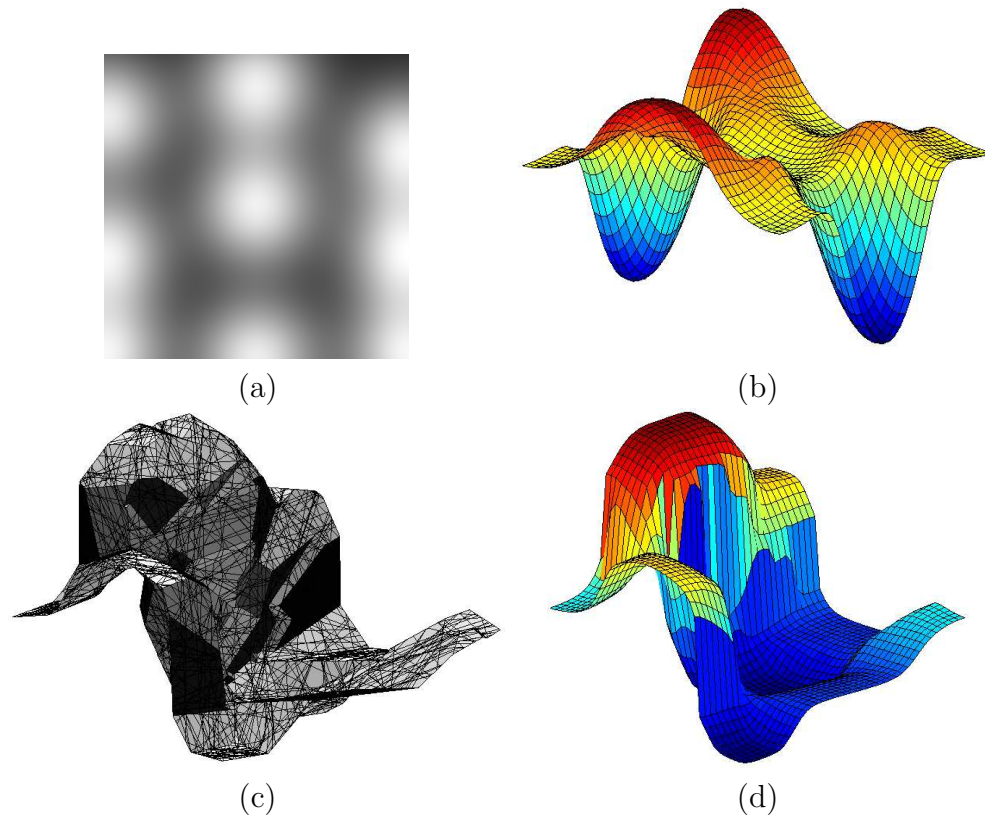


Figure 2.7: (a) A two-dimensional slice of the density function. Intensity of the color is proportional to the density. (b) Numerical optimization methods converge to the local minimum. (c) The minimal discrete surface using a grid of random planes. (d) Our method followed by numerical relaxation.

As we can see here, the local optimizer finds solutions with a range of energies, although it never finds the best one. The 240-plane combinatorial method, most often finds the best solution, but occasionally finds non-optimal ones. The use of plane selection with only 150 planes always finds the same solution at a fraction of the cost in both time and memory.

All of the combinatorial solutions were followed by relaxation. The typical effect of this relaxation is to improve the quality of the result by about 1%. More specifically the quantity “(relaxed - unrelaxed)/relaxed” is on the order of 10^{-2} .

Next we experimented with a density field that had two levels of detail. In particular, we started with the functional from the previous experiment and randomly added the same number of Gaussian blobs, but these blobs were 5 times smaller than the original ones.

For the adaptive refinement extension, we ran three steps of refinement using an

	time, <i>sec</i>	memory, <i>Mb</i>	energy histogram
a) local optimization, starting from random guess	13.5	3	
b) combinatorial optimization, 240 planes, + relaxation	3590	1040	
c) plane selection, 150 planes, 20 iterations, + relaxation	5450	200	
d) adaptive refinement, 150 planes, 3 iterations (5 plane selection iterations each), + relaxation	4120	290	

Table 2.4: Comparison of the different algorithms for the functional with two level of details.

envelope offset of 1, 0.2, and 0.05 respectively. At every step we use 5 iterations of the plane selection extension to construct the best possible solution and the next envelope. This method always succeeded in finding the global minimum. These results are summarized in table 2.4.

2.8 Discussion

We are currently investigating various applications where our algorithms may be applicable. These include segmentation of volume data, and constructing iso-surfaces from contours. We hope that, combinatorial methods, such as the ones described here may find more applications and impact in the scientific computing community.

It would be nice to be able to solve variational problems of higher co-dimensions, such as minimal 2-surfaces in R^4 , but it seems unlikely that such a generalization exists. Similarly, it seems difficult to generalize these methods for functionals that depend on the second derivative of the function.

In this thesis we limited ourselves by constructing the grids as intersection of the hyper-planes. This is one of the reasons that time and space complexity of our method is so high. Every hyperplane that was added to the list intersected a lot of cells, most of which are located far from the solution. One of the possible

directions for future research is to construct grids that can be refined locally in order to increase precision of the approximation without overcomplicating the grid structure. For example, in the planar case we investigated subdivision rules for triangular grids. Unfortunately, the simple subdivision rules (such as 1-to-4 split) do not ensure the necessary density properties of the resulting grids and more complicated rules (such as pinwheel grids [47]) are hard to implement and generalize to higher dimensions.

Using spatial octrees (for 3D hyper-planes) is not as elegant and easy to implement as binary trees, but it is likely to increase the performance of the algorithm and better balance of the tree.

The dual graphs that arise from our algorithm are very sparse and have a special structure. For example, it can be easily shown (section 2.12) that the dual graph of any hyperplane grid is bi-chromatic. Exploiting these properties in MIN-CUT algorithm seems like a very interesting direction of future research.

2.9 Proof of the density theorem (2.5.7)

Before we begin to prove the theorem, we formulate technical lemma that we will use in the proof.

Lemma 2.9.1. *Let $f(x) : [a, b] \mapsto \mathbb{R}$ be a twice differentiable function, $f(a) = A$, $f(b) = B$. Then there exists such a point $\bar{x} \in [a, b]$ that*

$$\frac{df}{dx}(\bar{x}) = \frac{B - A}{b - a}$$

Corollary 2.9.2. *If $f(a) = A + \varepsilon_1$, $f(b) = B + \varepsilon_2$, then there exists such a point $\bar{x} \in [a, b]$ that*

$$\left| \frac{df}{dx}(\bar{x}) - \frac{B - A}{b - a} \right| \leq \frac{|\varepsilon_1| + |\varepsilon_2|}{b - a}$$

Theorem 2.9.3. *Let us consider a continuous twice differentiable function $f(x) : [0, 1] \mapsto [0, 1]$ with bounded first and second derivative, $|f'(x)| \leq C_1$, $|f''(x)| \leq C_2$. For any $\varepsilon > 0$ there exists $g \in \mathcal{L}$ such that $\|f - g\| < \varepsilon$.*

Proof. Every piecewise linear function generated by the grid C_n can be coded by the second coordinate of its vertices

$$(dk_0, dk_1, \dots, dk_n) \mid k_i \in \{0, 1, \dots, n^2\},$$

where $d = 1/n^2$. In this notation the closest piece-wise linear approximation $g(x)$ of the function $f(x)$ will look like

$$\left(d \left[\frac{f(0)}{d} \right], d \left[\frac{f(h)}{d} \right], d \left[\frac{f(2h)}{d} \right], \dots, d \left[\frac{f(nh)}{d} \right] \right)$$

where $[\cdot]$ denotes the integer part of a number and $h = 1/n$. By choosing $g(x)$ this way we make sure that $|f(hi) - g(hi)| \leq d/2$ for any $i = 0, 1, \dots, n$.

Let us now assess the difference between f and g from above, using the fact that f has limited first and second derivatives.

$$\begin{aligned} \|f - g\| &= \int_0^1 |f(x) - g(x)| + |f'(x) - g'(x)| \, dx \\ &\leq \max_{x \in [0,1]} (|f(x) - g(x)|) + \max_{x \in [0,1]} (|f'(x) - g'(x)|) \end{aligned} \quad (2.9.1)$$

The difference $|f(x) - g(x)|$ in the points hi , $i = 0, 1, \dots, n$ is less than $d/2$ and therefore

$$|f(x) - g(x)| \leq d/2 + C_1 h \quad (2.9.2)$$

for all $x \in [0, 1]$.

Let us take a look on the interval $[hi, h(i+1)]$ for some $0 \leq i \leq n-1$. The derivative of the piecewise linear function $g(x)$ is constant on this interval and equal to $(g(h(i+1)) - g(hi))/h$. We also know that at the ends of the interval the distance between $f(x)$ and $g(x)$ is less than $d/2$. Applying the corollary (2.9.2), we can say that there exists a point $\bar{x} \in [hi, h(i+1)]$ such that

$$|f'(\bar{x}) - g'(\bar{x})| \leq \frac{d}{h}.$$

Using the fact that the second derivative is limited, we can conclude that

$$|f'(x) - g'(x)| \leq \frac{d}{h} + C_2 h. \quad (2.9.3)$$

is true for all $x \in [hi, h(i+1)]$ and therefore for all $x \in [0, 1]$.

Combining (2.9.1), (2.9.2) and (2.9.3), and taking into account that $h = 1/n$, $d = 1/n^2$, we have

$$\begin{aligned} \|f - g\| &\leq d/2 + C_1 h + d/h + C_2 h \\ &= \frac{1}{2n^2} + \frac{1}{n} (C_1 + C_2 + 1) \xrightarrow{n \rightarrow \infty} 0. \end{aligned}$$

Thus, for any $\varepsilon > 0$ we can choose such an integer n and such a function $g \in \mathcal{M}(C_n) \subset \mathcal{L}$ that $\|f - g\| < \varepsilon$. □

We can easily generalize this theorem for piece-wise differentiable functions.

Corollary 2.9.4. *Let us consider a continuous **piece-wise** twice differentiable function $f(x) : [0, 1] \mapsto [0, 1]$ that has a finite number of pieces. The first and second derivative are defined at every internal point of these pieces and bounded, $|f'(x)| \leq C_1$, $|f''(x)| \leq C_2$. For any $\varepsilon > 0$ there still exists $g \in \mathcal{L}$ such that $\|f - g\| < \varepsilon$.*

Proof. We create the piece-wise linear function g the same way we did it in the previous theorem. The only difference from the previous proof is that now the function f has a finite number of points where the first and second derivatives are discontinuous. Let us consider one of these points \hat{x} that is covered by k -th segment of the function g . Because the first derivative is bounded, we have

$$\int_{kh}^{(k+1)h} |f(x) - g(x)| + |f'(x) - g'(x)| dx \leq (2 + C_1)h.$$

Therefore, the total difference introduced by these points of discontinuity goes to zero when h goes to zero. \square

The proof of the theorem (2.5.7) now becomes trivial. We proved that for every function $f \in \mathcal{P}$ and any $\varepsilon > 0$ there still exists $g \in \mathcal{L}$ such that $\|f - g\| < \varepsilon$. Therefore, \mathcal{L} is dense in \mathcal{P} .

2.10 Proof of the density theorem (2.5.8)

Our investigation of the random grids will be based on the obvious fact that can be stated as follows.

Lemma 2.10.1. *Let $f(x)$ be a piece-wise linear function with n linear segments, R_m be a random grid with $m \geq n$, and $\varepsilon > 0$. Then there is a non-zero probability $p(f, m) > 0$ that there exists such a function $g(x) \in \mathcal{M}(R_m)$ that $\|f - g\| < \varepsilon$.*

All the lines are generated independently, therefore we can consider the grid R_{2m} to be the intersection of two independent grids of the size m . Using the basics of the probability theory, we can conclude that for the conditions given in the lemma

$$p(f, 2m) \geq 2p(f, m) - p(f, m)^2.$$

We know that the sequence $p_{n+1} = p_n + p_n(1 - p_n)$ with the starting point $0 < p_1 < 1$ converges to 1. That is why

$$p(f, m) \xrightarrow{m \rightarrow \infty} 1. \quad (2.10.1)$$

In other words, we can state the following corollary.

Corollary 2.10.2. *Let $f(x)$ be a piece-wise linear function, and $\{R_m, m = 1, 2, \dots\}$ be a family of the random grids, and $\varepsilon > 0$. Then the probability that there exists such a function $g(x) \in \mathcal{L}_r$ that $\|f - g\| < \varepsilon$ is equal to 1.*

Corollary 2.10.3. *Let $\{R_m, m = 1, 2, \dots\}$ be a family of the random grids. Then \mathcal{L}_r is dense in \mathcal{L} and therefore \mathcal{L}_r is dense in \mathcal{P} .*

2.11 Proof of the duality theorem (2.6.4)

In section 2.6.1 we already defined 1- and 2-chains and their boundaries.

Definition 2.11.1. *The chain is called closed if its boundary is the empty set.*

The following lemmata reduce the minimum valid curve problem to one of planar partitioning.

Lemma 2.11.2. *On our complex, there is a one-to-one correspondence between valid curves and closed 1-chains which include the edge e_r and do not include e_k .*

Proof. The correspondence between a 1-chain \mathcal{C} satisfying the properties of the lemma and a valid curve \mathcal{S} is $\mathcal{C} = \mathcal{S} \cup e_r$. \square

Lemma 2.11.3. *On any complex homeomorphic to B^2 , there is a one-to-one correspondence between 2-chains and closed 1-chains. The relationship between the corresponding 2-chain \mathcal{A} and 1-chain \mathcal{C} can be expressed as $\partial_2 \mathcal{A} = \mathcal{C}$.*

Proof. The first homology group H_1 is defined as the quotient group: $H_1 = \frac{\ker(\partial_1)}{\text{Im}(\partial_2)}$. In any complex homeomorphic to B^2 , H_1 is the trivial group (see [18]), therefore $\ker(\partial_1) = \text{Im}(\partial_2)$. \square

Combining these two lemmas together, we establish the following correspondence.

Theorem 2.11.4. *On our complex, there is a one-to-one correspondence between valid curves and 2-chains that include the face f_r and do not include the face f_k .*

Because there is a one-to-one correspondence between the faces of the primal graph and the vertices of the dual graph, there is also a one-to one correspondence between 2-chains that includes the face f_r and does not include the face f_k and cuts in the dual graph.

This proves theorem (2.6.4).

2.12 Dual graph is bi-chromatic

The dual graph of the complex obtained by the plane intersection has special properties. Some of them might be interesting to the future researchers. Note, that the complex and dual graph considered here do not include source and sink vertices.

Definition 2.12.1. *Let the d -dimensional space be divided by n non-equivalent hyperplanes $A_i x + B_i = 0$, $i = 1, 2, \dots, n$, A_i is a d -dimensional vector. The **subdivision code** of a point \bar{x} is an n -dimensional vector $b(\bar{x})_i = \text{sign}(A_i \bar{x} + B_i)$.*

Lemma 2.12.2. *Two points are located in the same cell if and only if they have the same subdivision code.*

We do not prove this lemma because of its triviality.

Lemma 2.12.3. *Two points are located in the neighboring cells if and only if their subdivision codes are the same except for one element which is 1 for one of the points and -1 for another.*

Proof. Two neighboring cells have one hyperplane that separate them. That is one exactly one element in their subdivision codes is different. \square

Lemma 2.12.4. *The dual graph of any grid obtained by the hyperplanes intersection is two-colorable (except for the source and sink vertices).*

Proof. For every cell k we can introduce a parameter P_k which is equal to the number of all positive elements in the cell subdivision code. Let us color with the white color all the cells where P_k is even and with the black color all the cells where P_k is odd.

By the lemma 2.12.4 all cells have the neighbors of the opposite color only.

The dual graph has a vertex for every cell and an edge between every two neighbors. Therefore, the dual graph is also two-colorable. \square

Minimum Weight Triangulations

3

Finding the minimum weight triangulation of the set of points on the plane is a classical problem of computational geometry. Though it might seem to be very far from the problems considered in the previous chapter, these problems exhibit a tremendous similarity in the context of spatial complexes. In this chapter we show that the spatial complexes arising in the minimum weight triangulation are not embeddable and that embeddability is crucial in finding the solution of the problem in polynomial time.

Minimal discrete surface on the embeddable complex can be found in polynomial time using the duality graph introduced in chapter (2.6). While the duality trick does not work for non-embeddable complexes, we explore linear programming techniques to construct an approximate solution for the minimal weight triangulation (MWT) problem.

In particular, we reduce the minimum weight triangulation problem of n points to the linear program with $O(n^3)$ variables and (in)equalities. This approach has a number of interesting properties of both theoretical and practical value.

- Integer solution of our linear program is equal to MWT.
- In general case, the straightforward implementation of our approach successfully computes MWT of up to two hundreds random points.
- Our approach can be merged with existing methods to provide fast implementations of MWT algorithm.

Though in some special cases we can prove that the solution of our LP is always integer, in general there exist a counterexamples where the solution of the linear problem is not integer. Still, we believe that the linear programming approach can be of a practical use for the MWT problem.

3.1 Edge-based and triangle-based definitions of the problem

Let S be a set of n points in Euclidean plane. A *triangulation* of this set is a maximum set of non-crossing edges between the points of S . The *weight* of the triangulation is equal to the sum of the lengths of its edges. To avoid degeneracies, in this thesis we suppose that no three points are located on a line, and no two triangulations of S have the same weights. Clearly, these conditions can always be satisfied by infinitely small perturbation of the positions of the points. The problem of finding the triangulation with the minimum total Euclidean length of its edges is historically called the *minimum weight triangulation* (MWT) problem. This problem is the major problem addressed in this chapter. For the clarity sake, in this text we

will sometimes call this problem *Euclidean MWT*. The complexity of the Euclidean MWT problem is currently unknown.

We can also consider the generalization of Euclidean MWT problem when the weights of the edges are arbitrary positive numbers. Let us call this problem *the general edge-based MWT* problem. This problem is known to be NP-hard.

In this thesis, we redefine triangulation in terms of triangles, which seems more natural for our approach.

Definition 3.1.1. A *set T_e of empty triangles* consists of all triangles with the endpoints in S that do not have any other points of S in their interiors.

Obviously, any triangulation T consists of the triangles from the set T_e . Note, that because the set of points S is non-degenerate, no points can appear exactly on the edge of any triangle. For our purpose, we introduce the set \mathcal{Q} of all points of the plane inside the convex hull of S , and the set \mathcal{E} of all points of the plane covered by the edges with the endpoints in S .

Definition 3.1.2. A set of the triangles $T = \{t_i\} \subset T_e$ is a **triangulation** of the point set S if $\cup t_i = \mathcal{Q}$, and triangles overlap only by their edges, i.e. $t_i \cap t_j \setminus \mathcal{E} = \emptyset$.

To make sure that the weight of the triangulation corresponds to the weight from previous definitions, we have to remember that every edge in the triangulation belongs to two triangles, except for the edges on the convex hull of S .

Definition 3.1.3. The **weight** w of the triangle with the edges e_1, e_2, e_3 is computed as

$$w = \alpha_1 |e_1| + \alpha_2 |e_2| + \alpha_3 |e_3| \quad (3.1.1)$$

where $\alpha_i = 1$ for the edges on the convex hull and $\alpha_i = 0.5$ for all other edges.

After having defined the weight of a triangulation as a sum of the weights of its triangles, it is possible to talk about MWT. Clearly, for the non-degenerate sets new definition of the MWT problem corresponds to the old one.

When the weights of the triangles are arbitrary positive numbers, the problem of finding the triangulation of minimum weights becomes *the general triangle-based MWT* problem. Obviously, it is more general than Euclidean and edge-based MWT and therefore it is also NP-hard.

3.2 Previous work on MWT.

In 1977 Lloyd [16] proved that the following problem is equivalent to the 3-satisfiability and therefore NP-complete. *If not all edges are allowed, find any triangulation that consists of the allowed edges or show that no such triangulation exists.* This problem has an immediate application for our case. By assigning zero weight

to the allowed edges and non-zero positive weight to the not allowed edges, Lloyd's problem becomes the special case of general edge-based MWT, and thus the latter one is NP-hard.

On the contrary, the problem of finding Euclidean MWT is open for more than thirty years [50, 17], but its complexity still remains unknown. No polynomial time algorithms are constructed so far, nor is it proven that the problem is NP-hard. Detailed review of the previous work can be seen in the extensive survey by Aurenhammer and Xu [2]. Here we overview the main directions of recent research.

Apart from the direct studying of the complexity of the problem, several approaches have been explored. It is possible to approximate MWT by triangulations that are based on greedy and Delauney triangulations [1]. Though these methods are not guaranteed to find the edges of the MWT, in some cases they give a good approximation of the MWT.

For the purpose of this thesis, we are more interested in the methods that can locate the actual edges of the MWT. When S is a vertex set of a convex polygon, dynamic programming leads to a simple n^3 algorithm [35], that nowadays can be found in many textbooks. In fact, this algorithm does not exploit convexity and can be easily generalized for any interior face of a planar straight line graph [2] and any positive triangle weights, i.e. it solves the general triangle-based MWT for this restricted setup.

Local methods such as β -skeleton [11] and exclusion regions [15] can give us edges that are guaranteed to be in the MWT or to be out of it. Global methods, such as intersection of all locally minimum triangulations [14], usually produce a connected subgraph of MWT.

Recent experimental papers [4] combine these approaches to generate MWT for up to several thousands randomly distributed points. Still, there exist examples where all the methods fail and the construction of MWT in reasonable time is not possible.

3.2.1 Previous edge-based linear programming approach.

Though linear programming (LP) can be used for solving or approximating many of the discrete problems, we found only one significant attempt to apply it to edge-based MWT problem [2]. This linear program is edge-based. Each of the $N = \binom{n}{2}$ edges e_i is assigned a constant weight $w_i = |e_i|$ and a real-valued variable x_i . The objective is to minimize

$$\sum_{i=1}^N x_i w_i \tag{3.2.1}$$

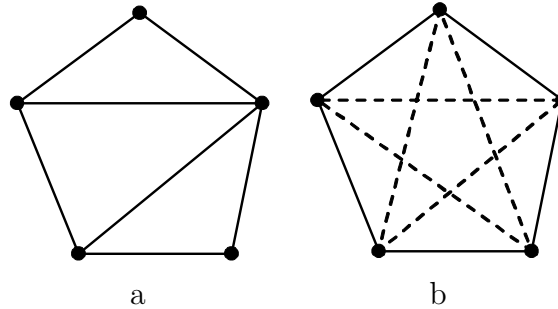


Figure 3.1: $n = 5$. Solution of the edge-based linear program, the edges with non-zero variables x_i are shown. (a) Integer LP gives minimum weight triangulation; $x_i = 1$ for all solid edges. (b) Continuous LP gives non-integer solution which is smaller than (a); $x_i = 1$ for the solid edges, $x_i = 1/3$ for the dashed edges.

subject to constraints

$$0 \leq x_i \leq 1, \quad (3.2.2)$$

$$x_i + x_j \leq 1 \text{ for } e_i \cap e_j \neq \emptyset, \quad (3.2.3)$$

$$x_i + \sum_{e_i \cap e_j \neq \emptyset} x_j \geq 1. \quad (3.2.4)$$

Condition (3.2.2) provides the integer boundaries for the real-valued variables. Condition (3.2.3) is established for every crossing pair of edges, it ensures that edges of the triangulation do not intersect each other. Condition (3.2.4) is established for every edge. If the edge is not in the triangulation, it should be intersected by at least one of the triangulation edges.

Integer solution of the problem (3.2.1 - 3.2.4) corresponds to a minimum weight triangulation of S [2]. Each edge in MWT is indicated by $x_i = 1$, each edge outside of MWT is indicated by $x_i = 0$. Unfortunately, integer LP is NP-complete, and the optimal solution of the continuous LP is not integer in most cases (Figure 3.1).

3.3 Minimum weight triangulation problem as a spatial complex

Let us consider the general triangle-based MWT problem¹ and construct a spatial complex for it.

Triangulation is a planar problem, all vertices have two coordinates. Let us "lift" these vertices at the arbitrary height from this base plane, in other words let us introduce the third coordinate for each vertex and give it some random finite value.

¹Again, this problem is known to be NP-hard.

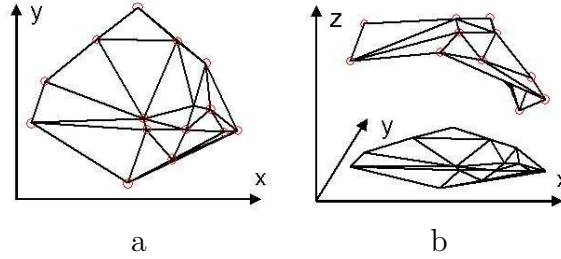


Figure 3.2: A subset of the spatial complex that corresponds to a single triangulation. (a) Triangulation of the set of points on the plane. (b) Random third coordinate is introduced for each vertex.

These coordinates are linearly interpolated for the edges and triangles. For notation purpose all lifted objects have the upper index l .

It is hard to visualize the resulting spatial complex, because the lifted triangles intersect each other. On the figure 3.2 we show only a subset of the spatial complex that corresponds to a single triangulation.

The union of all lifted vertices, edges, and triangles establish a 3-dimensional spatial complex G^l similar to the complexes described in the previous chapter. There are several simple facts about the spatial complex that are stated here without a formal proof.

Lemma 3.3.1. *The boundary ∂Q of the convex hull Q of the vertices in the base plane corresponds to a closed² 1-chain Γ^l in G^l .*

Lemma 3.3.2. *There is a one-to-one correspondence between the triangulations T in the base plane and 2-chain T^l in G^l with the following properties:*

- $\partial T^l = \Gamma^l$
- T^l is a continuous piece-wise linear function over Q

The problem of finding the triangulation with the minimum weight now becomes the problem of finding the minimal 2-chain that satisfies additional constraints. It might seem that this is exactly the problem that is solved in the chapter (2.6) in polynomial time. What is the catch?

There is one crucial difference between the lifted complex G^l and the complexes considered in the chapter (2.6). Defining G^l , we introduced vertices, edges and faces, but we did not introduce the cells. Indeed, the faces of the complex G^l intersect not only on the edges but in some arbitrary places. That is why the cells are not defined and the duality algorithm cannot be applied to the complex.

²the definition of closed 1-chain is given in the section (2.6.4).

Note, that if we could construct the embedding of G^l in 3D such that faces intersect on the edges only, then all the conditions of the chapter (2.6) would be fulfilled and we could find the minimum triangulation in polynomial time. This might be possible, for example, when not all triangles are allowed in triangulation.

3.4 Triangle-based linear program

Linear programming provides another interesting approach to the MWT problem. The rest of the chapter is devoted to the thorough investigation of this method. Note, that similar linear program can be set up for the spatial complex in the previous chapter, but duality method is more computationally efficient.

Triangle-based linear program is defined as follows. Every triangle $t_i \in T_e$ is assigned a real-valued variable x_i . The objective is to minimize the weight of the triangulation

$$F(\mathbf{x}) = \sum_{T_e} x_i w_i \quad (3.4.1)$$

Each variable has to be nonnegative.³

$$x_i \geq 0 \quad (3.4.2)$$

Every edge of the convex hull e_k is in MWT. Therefore, one of the triangles adjacent to this edge is also represented in MWT. This condition can be expressed by the following equality.

$$\sum_{t_i \ni e_k} x_i = 1 \quad (3.4.3)$$

Finally, there is an equality for every edge e_k that is not on the convex hull. The line, that this edge belongs to, divides the plane onto two half-planes. The set of the triangles that are adjacent to e_k can be split accordingly onto two sets, T_k^1 and T_k^2 . If e_k belongs to MWT, then exactly one triangle from T_k^1 and one from T_k^2 belong to MWT. If e_k is not in MWT, then non of the adjacent triangles is in the MWT. This idea is expressed in the following equality.

$$\sum_{T_k^1} x_i = \sum_{T_k^2} x_i \quad (3.4.4)$$

It is interesting to note, that (3.4.1 - 3.4.4) is a direct generalization of the well-known LP approach to the shortest path problem.⁴

³Unlike (3.2.2), we do not define the upper boundary, because it will be satisfied automatically.

⁴In some sense, we are trying to construct the "shortest surface" with the boundary conditions (3.4.3) and connectivity condition (3.4.4). The crucial difference between these two problems is that shortest path LP has integer solutions for *any* positive weights w_i . In our case, the problem of finding the minimum triangulation with arbitrary weights is NP-complete and the success of the method is provided by the weight definition (3.1.1).

3.4.1 Analysis of the triangle-based integer LP

Every triangulation T is represented by the integer-valued vector \mathbf{x} , where $x_i = 1$ for all triangles in T and $x_i = 0$ for all triangles not in T . By our definition of triangulation (3.1), this representation corresponds to the following constraint. First we choose any point p that is inside the convex hull of S and not on any edge in S , $p \in \mathcal{Q} \setminus \mathcal{E}$. The set of triangles that cover p is denoted as $C(p) \subset T_e$.

$$\sum_{C(p)} x_i = 1 \quad (3.4.5)$$

In fact, this constraint is automatically satisfied by the initial constraints of the problem for both integer-valued and real-valued instances of LP.

Theorem 3.4.1. *A real-valued vector \mathbf{x} satisfies the condition (3.4.5) if and only if it satisfies the conditions (3.4.3 - 3.4.4).*

Proof. (3.4.3 - 3.4.4) \Rightarrow (3.4.5). First, let us take any edge e_k of the convex hull of S and the set of the triangles $T_k = \{t_i\}$ adjacent to this edge. The condition (3.4.5) is directly ensured by (3.4.3) for every point p that is covered only by the triangles of T_k , i.e. for all

$$p \in Q_k = \bigcap_{T_k} t_i \setminus \bigcup_{T_e \setminus T_k} t_i \setminus e_k.$$

Because S is a non-degenerate set, the set Q_k is not empty and includes some area adjacent to e_k .

Second, let us take any point p and connect it with a point on the boundary of the convex hull by a straight segment γ that does not pass through any point of S . In the previous paragraph we have shown that the condition (3.4.5) is satisfied at least for some part of γ . If it is not satisfied in p , then there exists a point $\bar{p} \in \gamma$ such that the condition is satisfied on the one side of this point and not satisfied on the other one. Clearly, \bar{p} belongs to some internal edge e , because (3.4.5) can change its value only on the edge. But in this case the condition (3.4.4) is not satisfied for the edge e .

(3.4.5) \Rightarrow (3.4.3 - 3.4.4). This part of proof is constructed in a similar manner and omitted here for the sake of space. \square

Corollary 3.4.2. *Integer solution of the linear program (3.4.1 - 3.4.4) corresponds to MWT.*

Proof. Integer solution x of the linear program satisfies (3.4.2), therefore x_i can be only 0 or 1. Constraint (3.4.5) ensures that x corresponds to a triangulation defined in (3.1). Because any triangulation satisfies all the constraints of LP, x corresponds to the triangulation with the minimum weight. \square

3.4.2 Solving integer LP using continuous LP

Unfortunately, integer LP is NP-complete and there is no known polynomial-time algorithm for this problem. Instead, we can solve the problem (3.4.1 - 3.4.4) for the optimal real-valued vector \mathbf{x} and hope that the solution is integer.

Definition 3.4.3. *The lifted complex is **embedded** if its faces intersect only on the edges. The complex is **embeddable** if such an embedding exists.*⁵

Theorem 3.4.4. *For the embeddable lifted complex continuous LP always has an integer solution.*

The proof of the theorem is given in the section 3.6.1.

The theorem proves that linear programming can be used to solve the problems arising in the previous chapter. Unfortunately, the lifted complex is not embeddable. Though for some special case (chapter 3.6) we can prove that the solution of the continuous LP always has integer values, in general this is not true.

Note, that the linear problem described above can be applied to any minimum triangulation problem, including general triangle-based one, which is known to be NP-hard. So we know⁶, at least for triangle-based MWT, that continuous linear program should give non-integer solutions in some cases. These examples have a very special structure and, as predicted, exploit the fact that the spatial complex is not embeddable.

3.4.3 Double-coverings and non-integer solutions of LP

Any triangulation T is a subset of T_e that covers points in the triangulation area $\mathcal{Q} \setminus \mathcal{E}$ exactly one time. It is important to consider another family of subsets of T_e that cover this area exactly two times. We call such a subset $D \subset T_e$ *double covering*. The simplest example of double covering is as a union of two triangulations $D = T_1 \cup T_2$, but in general case this representation is not possible. If for every triangle in the double covering we assign $x_i = 0.5$, then the constraint (3.4.5) is automatically satisfied. Therefore, double covering satisfies all the conditions of our LP and theoretically can be a solution of the continuous LP. Of course, we can easily construct triple-coverings e.t.c.

In fact, there exist double coverings that cannot be represented as a union of two triangulations. The simple six-point example is given on the figure (3.3). In general, a double-covering can be a solution of the triangle-based LP, if the weights of the triangles of the double covering are small and the weights of all other triangles are large.

⁵The embeddability of the spatial complexes can be considered as a generalization of the planarity to the next dimension.

⁶unless P=NP

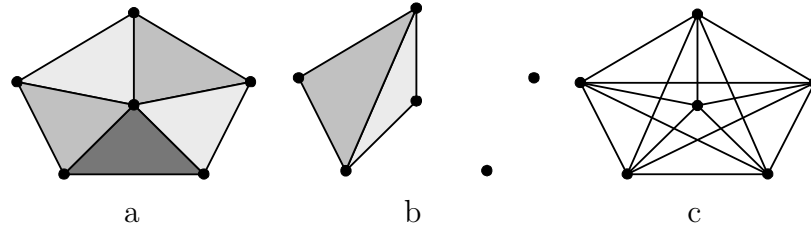


Figure 3.3: Single- and double- coverings. (a) Triangulation (single-covering) of 6-point set consists of five triangles. (b) Double-covering of the same set consists of ten triangles. For simplicity, only two triangles are shown, the rest can be obtained by the rotation of this pair around the center point by $4\pi k/5$, where $k = 1, 2, 3, 4$. (c) The same double-covering. All ten triangles are shown (transparent). Note, that this double-covering cannot be split onto two triangulations.

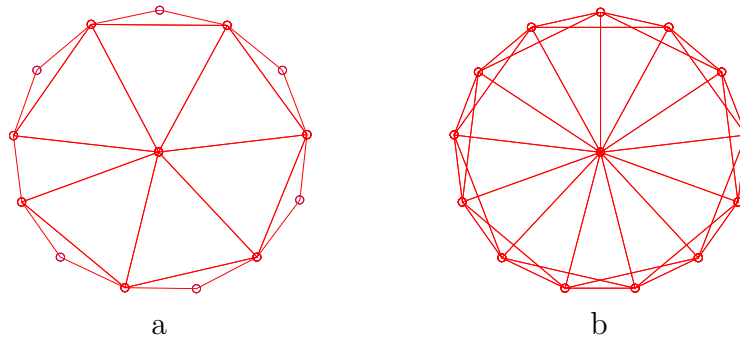


Figure 3.4: 13-point counterexample for Euclidean MWT. (a) Minimum weight triangulation weights 18.798. (b) Solution of the continuous LP corresponds to a double covering with a weight 18.764. Triangles are shown transparent.

Intuitively, this situation is unlikely to happen when the weights of triangles are defined as Euclidean length (3.1.1). For example, on the figure (3.3c) we had to use relatively long and "heavy" edges to construct a double covering. For a long time we could not construct the counter example for the Euclidean MWT, nor could we prove that the solution of the LP is always integer. Numerous experiments showed that LP gives integer solution when points are randomly distributed on the plane (the experiments are described in the following chapters).

The triskaidekaphobic counterexample that we found is surprisingly similar to the counterexample for LMT method [14]. One vertex is located in the middle, and exactly 13 vertices are uniformly distributed on the unit circle around it (figure 3.4). The weight of the minimal triangulation is 18.798, the weight of the minimal double covering is 18.764.

3.4.4 Random experiments

We believe that the proposed linear program might be of a practical use when the vertices are uniformly distributed on the plane. The experiments are essential to check the robustness of the algorithm. The number of the empty triangles is $O(n^3)$. Indeed, in the worst case when S is a vertex set of a convex polygon, every triangle is possible. The same is the number of inequalities (3.4.2). The number of equalities (3.4.3 - 3.4.4) is equal to the number of the edges $\binom{n}{2}$ and therefore quadratic. Given that the best known LP algorithm is cubic, the time complexity of the straightforward implementation of the algorithm is $O(n^9)$.

We used CPLEX to solve the resulting linear program, points were distributed uniformly in the unit square. For such a setup, our straightforward implementation of the algorithm allows to compute MWT of up to two hundreds randomly distributed points. For 200 points, the linear program consists of about 70,000 variables and inequalities and 20,000 equalities; it takes CPLEX about twenty minutes to converge.

We tested random point sets from 10, 20, 60 and 200 points. The total number of sets tested is about 100,000. In all tested cases the solution of the LP problem was integer and therefore the exact MWT was constructed.

Using previous methods to minimize the number of variables

The algorithm proposed above is not practical, since for the randomly distributed sets of points it is possible to construct a large subgraph of the MWT for up to hundred thousands points in reasonable time [4]. When the resulting subgraph is connected, each of the remaining faces can be triangulated in $O(n^3)$ time. Some modification of our algorithm should be used when the resulting subgraph is not connected.

Indeed, we can efficiently use the information about edges that are guaranteed to be in MWT and that are guaranteed to be out of MWT. If the curtain edge is guaranteed to be out of MWT, then all the triangles adjacent to this edge are no longer valid and can be ignored in linear program constructing. If the edge is guaranteed to be in the MWT, then all triangles that intersect this edge can be also eliminated.

In our experiments, we used the diamond test [15] to locate the edges that are not in MWT. This simple but powerful test significantly reduced the number of the valid triangles and allowed us to compute MWT of up to two thousands points.

3.5 Discussion

Our method and LMT [14]. Interesting enough, the results of our method are very similar to the results shown by LMT, though methods are very different by their nature. There are two major similarities:

- Both methods seem to work fine for the sets of randomly distributed points on

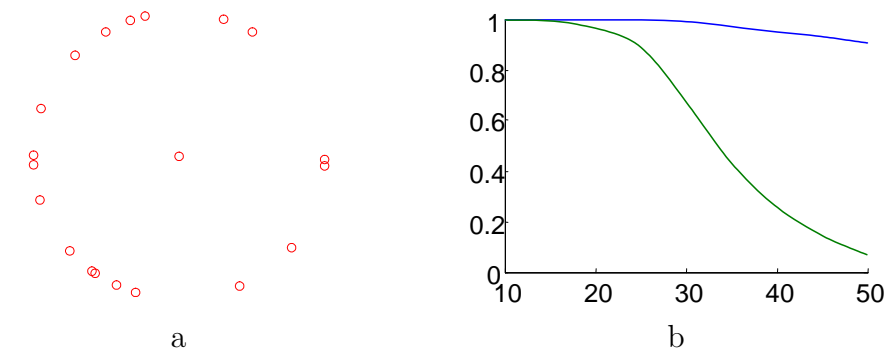


Figure 3.5: Comparison of the linear programming and LMT approach for a random wheel example. (a) Typical random setup for $n = 20$. (b) Experimental probability of success by LMT (green line) and our LP method (blue line) depends on the number of points in the example.

the plane. LMT finds the connected subgraph of the MWT, and our method converges to the integer solution.

- Counterexamples for both methods have very similar wheel-shapes (figure 3.4).

It is hard to come up with a fair experiment to compare both methods. Still, there is some evidence that our linear programming approach is a little bit more general than LMT. To show this, we did the following experiment. N points are randomly placed on the circle around one central point. The probability of tested method to succeed is shown on the figure (3.5).

Different metrics. Another good thing about LP approach is that it is very flexible. We can apply it not only for Euclidean MWT problem, but to any other problem where triangles are assigned the positive weights. If the solution of LP is integer, the answer corresponds to minimal weight triangulation due to the corollary (3.4.2). If the weights of the triangles are arbitrary, the integer solution of the LP is highly unlikely, but if the weights of the triangles come from some metric, the probability of success is much higher.

In particular, we experimented with two different metrics, $L^1 = |\Delta x| + |\Delta y|$ and $L^\infty = \max(|\Delta x|, |\Delta y|)$. For random point distribution, our algorithm works for all the example considered. It seems, that these metrics are "strict" enough to avoid double-covering. What properties of the metrics provide integer solution for triangle-based LP, is an open question.

Higher order problems. We tried to generalize our approach to the higher order problems such as finding the minimum weight tetrahedralization of the set of

points in 3D. When the weight of the tetrahedralization is the sum of the areas ⁷ of its faces, the tetrahedra-based generalization of the linear problem is straightforward.

Unfortunately, it turned out that the double coverings happen all the time in the LP solutions of the tetrahedralization problem. We believe that it happens because of the one significant difference from the triangulation problem. Every triangulation of some point set have exactly the same number of triangles and edges. For tetrahedralization, this is not true; two tetrahedralizations of the same point set can have different amounts of tetrahedra. The double coverings "exploit" this fact to provide non-integer solutions of the linear program.

3.6 Special cases

In this section we show that in some special cases the solution of the LP is guaranteed to be integer and therefore we can find MWT in polynomial time. One of the instances of MWT problem when polynomial time solution is known is the case when S is a vertex set of a convex polygon. Note, that edge-based linear program does not work for this problem (Figure 3.1).

Lemma 3.6.1. *When S is a vertex set of a convex polygon, triangle-based LP always has an integer solution.*

Proof. Essentially, our proof is based on the fact that for this setup every double-covering can be represented as a union of two triangulations. We also use the fact that if the vectors $\mathbf{x} = \{x_i\}$ and $\mathbf{y} = \{y_i\}$ satisfy the constraints (3.4.2 - 3.4.4) of the linear program, than for every $c \in [0, 1]$ the vector $\mathbf{z} = (1 - c)\mathbf{x} + c\mathbf{y}$ also satisfied them.

Let us now suppose that \mathbf{x} is a non-integer solution of a triangle-based LP, $F(\mathbf{x}) = C$. We can prove that \mathbf{x} is not optimal. To do that, we extract a triangulation from \mathbf{x} (figure 3.6) as following. Fix any triangle with a positive variable x_i , one of its edges, and one of the vertices on the edge. Constraints (3.4.4) guarantee that there exist another positive triangle adjacent to the chosen edge. In this manner we can add triangles counter- or clockwise around the chosen vertex until we hit the boundary of the convex hull. Now we can fix all these triangles, fix any outer edge and vertex on this edge and repeat the whole process until we complete the triangulation T . We can define $c_1 = \min(x_i | t_i \subset T)$. If \mathbf{y} is an integer representation of the triangulation T , then \mathbf{x} can be represented as a mix of two solutions \mathbf{x}_1 and \mathbf{y} :

$$\mathbf{x} = (1 - c_1)\mathbf{x}_1 + c_1\mathbf{y}$$

⁷In fact, the generalization of the LP works if the weight of a face is arbitrary positive number. On the contrary, it is not obvious how to generalize LP if we define the wight of the tetrahedralization as the sum of the lengths of its edges. The reason is that each internal face belongs to exactly two tetrahedra and we can apply a formula similar to (3.1.1) in order to compute the weights of the tetrahedra. The number o the tetrahedra that contain the curtain edge is unknown in advance and it is not clear how to split the weight of the edge among them.

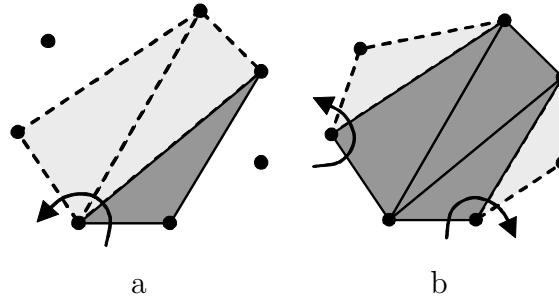


Figure 3.6: For the convex set S , any non-integer solution \mathbf{x} consists of several triangulations. Here is how we can extract one of them. (a) Fix any triangle with a positive x_i (dark grey) and one of its vertices. Constraint (3.4.4) guarantees that going around this vertex in a clockwise or counter-clockwise direction we can always find a triangle with a positive variable x_j . These triangles are shown in light gray. (b) Fixing the triangles found on the previous step, we can repeat procedure until the whole triangulation is complete.

It is easy to check that \mathbf{x}_1 and \mathbf{y} satisfy the constraint (3.4.5). Repeating the procedure for \mathbf{x}_1 , we can represent \mathbf{x} as

$$\mathbf{x} = (1 - c_1)(1 - c_2)\mathbf{x}_2 + c_1\mathbf{y} + (1 - c_1)c_2\mathbf{z},$$

where \mathbf{z} is a triangulation different from \mathbf{y} . We can construct two new vectors that satisfy all the constraints of the problem.

$$\begin{aligned}\mathbf{x}_3 &= (1 - c)\mathbf{x}_2 + c\mathbf{y} \\ \mathbf{x}_4 &= (1 - c)\mathbf{x}_2 + c\mathbf{z}\end{aligned}$$

where $c = c_1 + c_2 - c_1c_2$. Because the set S is not degenerate, then either $F(\mathbf{y}) < F(\mathbf{z})$ or $F(\mathbf{z}) < F(\mathbf{y})$. Therefore by linearity either $F(\mathbf{x}_3) < F(\mathbf{x})$ or $F(\mathbf{x}_4) < F(\mathbf{x})$. \square

In the proof of the lemma (3.6.1) we never used the fact that the set S is convex. In fact, this lemma can be generalized for a larger class of problems.

Corollary 3.6.2. *When S is a vertex set of an interior face of a planar straight line graph, triangle-based LP always has an integer solution.*

3.6.1 Proof of the theorem 3.4.4

Proof. The proof by contradiction is based on the very same idea as the proof given in the previous section. For the embeddable complex every non-integer solution \mathbf{x} of the continuous linear program can be split into two different solutions, and therefore

it cannot be optimal. Because of the lack of space, here we provide only the general idea of the proof.

Because the complex G^l is lifted, every triangle t_i^l of the complex can be also considered as a linear mapping L_i of the triangle t_i from the base plane \mathcal{Q} , i.e. $L_i: t_i \subset \mathcal{Q} \mapsto \mathfrak{R}$.

Definition 3.6.3. *The lifted triangle t_i^l is **lower** than the triangle t_j^l , if there exists at least one point in the base plane $q \in t_i \cap t_j$ such that $L_i(q) < L_j(q)$.*

For the embeddable complexes the notion of "lower" is well defined, because the triangles intersect on the edges only. For the non-embeddable complexes this definition does not make a lot of sense and the rest of the proof does not work.

Let us consider the set L^l that consists of the lifted triangles t_i^l with the following properties:

- the LP variable x_i of this triangle is non-zero, $x_i > 0$
- there is no triangle t_j^l lower than t_i^l such that $x_j > 0$

We will show now that the 2-chain L^l corresponds to a valid triangulation by the lemma 3.3.2. Indeed, L^l corresponds to a continuous piece-wise linear function over the domain \mathcal{Q} .

- L^l does not cover any area of the domain twice by construction.
- L^l is continuous by the LP constraint (3.4.4).
- L^l fits the boundary condition by the constraint (3.4.3).

Therefore, the triangulation represented by L^l can be an integer solution \mathbf{y} of the linear program. The initial solution \mathbf{x} can be split into \mathbf{y} and some other solution \mathbf{x}_1 similar to the previous section,

$$\mathbf{x} = (1 - c_1)\mathbf{x}_1 + c_1\mathbf{y},$$

where $c_1 = \min(x_i | t_i \subset L^l)$.

The proof that in this case the initial solution cannot be optimal is given in the previous section. \square

Geodesics

4

In the previous chapters we considered the problems that could be reduced to the 2-dimensional minimal discrete surfaces on the 3-dimensional spatial complexes. In the rest of the thesis we will be interested in 1-dimensional minimal curves on the piece-wise linear triangulated surfaces (meshes).

In this chapter we present practical methods for computing both exact and approximate geodesic (shortest) paths on triangular meshes. These paths typically cut across faces in the mesh and so are not directly found by graph algorithms (such as Dijkstra’s shortest path algorithm). The exact version of the algorithm is based on the interval propagation idea introduced by Mitchell, Mount, and Papadimitriou ¹, and has the same $O(n^2 \log n)$ worst case time complexity. The fastest approximate version works in roughly $O(n \log n)$ time and still guarantees computing exact geodesics on any subdivision of a plane. The desired tradeoff between the time complexity and the error of the approximation can be achieved by setting the interval simplification threshold. The algorithms were evaluated on meshes with up to 100,000 vertices.

Many mesh-processing algorithms require a subroutine that efficiently computes geodesic paths between vertices on a triangular mesh. When parameterizing a complicated mesh, one often breaks it up into a set of charts; (See, for example, [37] for a manual chartification system, and [49] for an automatic one). In these cases, the parameterization can be done more efficiently (with less distortion and better packing efficiency) when the charts are bounded by geodesic paths. Geodesic paths also are needed when segmenting a mesh into subparts, such as done in [33]. In mesh editing systems, such as [36], one desires straight boundaries to define the bounding extents of editing operations.

Many mesh processing algorithms require geodesic distances between vertices in a triangular mesh. For example, radial-basis interpolation over a mesh requires point to point distances. This type of interpolation is used in numerical applications, such as skinning [51], and in mesh watermarking [45]. Shape analysis algorithms such as [24] use Morse analysis of a geodesic distance field. Mesh parameterization methods [54] based on isomap type algorithms [53] are also driven by point to point geodesic distances.

In this chapter, we first give a simplified interpretation of, and describe a simple way to implement, the “MMP exact geodesic algorithm” [40]. The MMP algorithm has worst case running time of $O(n^2 \log n)$ but runs much faster on most meshes. The original description of the MMP algorithm is quite complicated, and has never been implemented as far as we can tell. Our simplified interpretation makes an implementation very straightforward. In particular it involves no special handling of saddle vertices ², and guarantees no gaps in the distance field due to numerical errors.

The MMP algorithm splits up each edge on the mesh into a set of smaller intervals over which exact distance computation can be dealt with atomically. In the second

¹Further in the text their algorithm is referred as MMP.

²A saddle vertex is a vertex for which the sum of the angles of the adjacent mesh faces is more than 2π .

part of this chapter, we explain how approximate geodesics can be computed by running a version of MMP that skips these edge-splits. This approximate approach is (essentially) of comparable complexity to Dijkstra. Our approximate algorithm also has the property that it computes the exact geodesics if given a flat (planar) mesh.

4.0.2 Related Work

An exact geodesic algorithm with worst case time complexity of $O(n^2)$ was described by Chen and Han [10]. This algorithm was partially implemented by Kaneva and O’Rourke [31]. We show in our results section that our exact implementation runs many times faster than that method.

An exact geodesic algorithm with worst case time complexity of $O(n \log^2 n)$ was described by Kapoor [32]. This is a very complicated algorithm which calls as “sub-routines” many other complicated computational geometry algorithms. It is not clear if this algorithm will ever be implemented.

Approximate geodesics (with guaranteed error bounds) can be obtained by adding extra edges into the mesh and running Dijkstra on the one-skeleton of this augmented mesh [30], [38]. These algorithm requires the addition of numerous extra edges to obtain accurate geodesics. The algorithms described in [30] relies on the selective refinement, and therefore significantly depends on the first approximation path found. If this approximation is far from the actual solution, the subdivision method might converge to the local minimal path (instead of the global geodesic one), or it might take a very large number of iterations until the refinement area moves to the vicinity of the actual geodesic and the process converge.

Approximate geodesics are computed by Kimmel and Sethian [34] using an algorithm that runs in $O(n \log n)$ time. The approximate geodesics found by this method can be quite inaccurate, even for planar meshes. We show in our results section that our approximate geodesic computation is much more accurate than that method.

Polthier and Schmies [44] describe a different definition of a geodesic path on meshes using a notion of “straightest” instead of “shortest”. This notion may be inappropriate for some applications of geodesics.

4.1 Exact Algorithm

Our implementation of the exact algorithm is based on a very simple idea. Let us fix the source vertex on the mesh. The minimal distance from the source is a scalar function that is defined for every point on the surface of the mesh. In particular, for every edge e_i , that is parameterized by the parameter x , the *minimal distance function* $D_{e_i}(x)$ is well defined.

Let us consider two edges e_1 and e_2 that belong to the same face. If the shortest path from some point $x_2 \in e_2$ goes through the point $x_1 \in e_1$, then the distance func-

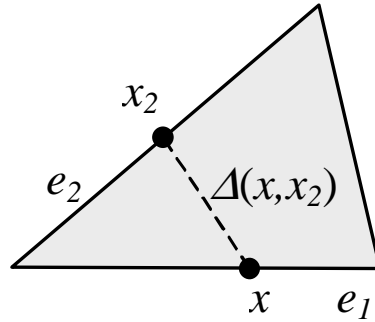


Figure 4.1: If the shortest path from the point x_2 of the edge e_2 crosses the edge e_1 , the distance function in this point can be computed by theorem (4.1.1).

tion in the point x_2 can be simply computed as $D_{e_2}(x_2) = D_{e_1}(x_1) + \Delta(x_1, x_2)$, where the $\Delta(x_1, x_2)$ is Euclidean distance between these points across the face. Because this path is the shortest possible one, we can generalize this observation.

Theorem 4.1.1. *If the shortest path from the point $x_2 \in e_2$ goes through the edge e_1 that belongs to the same face (fig. 4.1), then*

$$D_{e_2}(x_2) = \min_{x \in e_1} (D_{e_1}(x) + \Delta(x, x_2)) \quad (4.1.1)$$

Based on this theorem, we can write a simple algorithm (table 4.1) that is guaranteed to find all the distance functions on the edges correctly. This algorithm works for any convex or non-convex triangulated mesh with or without boundary. If the queue is sorted by the minimal value of the edge distance function, we have a Dijkstra-style propagation of the distance function across the surface.

The algorithm described above is a close relative of many algorithms developed for this problem before. On the one hand, it can be considered to be a continuous version of the discrete subdivision algorithm described in [38]. On the other hand, this interval-based algorithm is also in the heart of [40].

4.1.1 Back-tracing the shortest path

When all the distance functions on the edges are computed, tracing the shortest path is very simple. Any geodesic path is piece-wise linear curve with nodes on the edges of the initial mesh. Geodesics can be traced backward step-by-step by finding the next appropriate nodes. Let us take some initial node q that belongs to face f with edges e_1 , e_2 and e_3 . If q does not belong to any of the edges, the next node of

algorithm	FindMinimalDistanceFunctions
<ol style="list-style-type: none"> 1. Compute distance functions for all edges that belong to the same faces as source. Put these edges in the queue. Define distance functions to be infinite for the rest of the edges. 2. until the queue is empty 3. Remove the next edge e from the queue. 4. for all edges e_i that belong to same faces as e. 5. Compute $\hat{D}_{e_i}(x)$ according to (4.1.1) for all $x \in e_i$ 6. Update distance function at e_i as $D_{e_i}(x) = \min(D_{e_i}(x), \hat{D}_{e_i}(x))$ 7. If the function $D_{e_i}(x)$ is changed, put the edge e_i on the queue. 8. end for 9. end until 	

Table 4.1: Simple algorithm for computing minimal distance functions on the edges.

the shortest path is point \hat{x} that minimizes the distance function

$$\min_{x \in \{e_1, e_2, e_3\}} (D(x) + \Delta(x, q)) \quad (4.1.2)$$

Otherwise, if $q \in e_1$, the next node of the shortest path \hat{x} minimizes

$$\min_{x \in \{e_2, e_3, e_4, e_5\}} (D(x) + \Delta(x, q)), \quad (4.1.3)$$

where e_4, e_5 are the edges of the other face adjacent to e_1 . We can keep tracing the shortest path until we hit the origin.

4.1.2 Intervals of optimality

To derive the general representation of the distance functions $D_{e_i}(x)$, we briefly review the interval theory developed in [40].

First, let us consider convex polygons without boundary. The shortest path from some point on the edge to the source crosses some set of faces $F = \{f_i\}$. It can be shown that on a convex polygon with no boundary the shortest path never

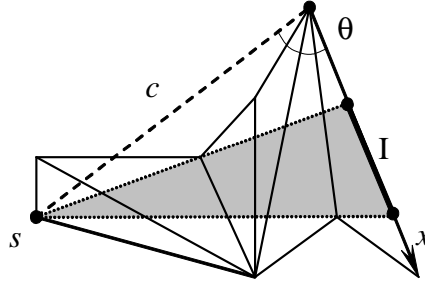


Figure 4.2: When the faces are unfolded on the plane, the shortest path from the the source (or pseudo-source) s to any point on the interval of optimality p is represented as a straight line. c is the distance from the (pseudo-)source to the edge origin, x is a parameter along the edge.

goes through the vertices of the polygon, except for the source vertex and, possibly, geodesic endpoint. We can *unfold* this sequence of faces to lay them flat on a plane. In this unfolding, the shortest path must be a straight line. In fact, the sets of points on an edge whose shortest pathes go through the same sets of faces, form the continuous *intervals of optimality* on the edge (fig. 4.2).

Using the cosine theorem, the distance function on such an interval p can be represented as

$$D_{p \subset e_i}(x) = \sqrt{x^2 - 2xc \cos \theta + c^2}, \quad (4.1.4)$$

where x is a distance from the origin of the edge, c is a distance from the source to the edge origin and θ is an angle is the angle between the edge and vector \vec{c} .

In the presence of a boundary, or when the polygon is non-convex, shortest passes can go through the vertices or follow edges of the polygon. In this case, other vertices can act as *pseudo-sources* for the intervals of optimality and one more parameter is needed to represent the distance function on the interval.

$$D_{p \subset e_i}(x) = d + \sqrt{x^2 - 2xc \cos \theta + c^2}, \quad (4.1.5)$$

where d is a distance from the source to the pseudo-source, c is now a distance from the pseudo-source to the origin of the edge. The distance function on the interval can therefore be uniquely defined by three parameters (d, c, θ) .

In general, the distance function on the edge is a continuous piece-wise smooth function that consists of the interval functions of the type (4.1.5).

4.1.3 Interval propagation

Given the distance function D_{e_1} on the edge e_1 , the propagated distance function on the adjacent edge e_2 is defined by formula (4.1.1). It is possible to solve this prob-

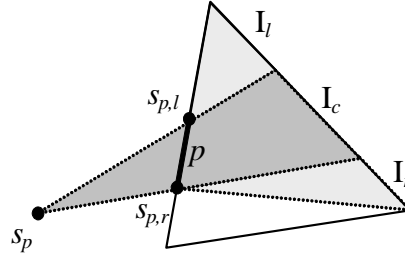


Figure 4.3: When interval p propagates onto the adjacent edge, it can create up to three new intervals. Central intervals I_c has its (pseudo-)source located in the same point s_p ; left and right intervals I_l and I_r have their (pseudo-)sources located in the points s_l and s_r respectively.

lem algebraically, using the distance function representation (4.1.5), but it is much simpler to employ geometrical reasoning. We can propagate intervals independently and denote as $D_{e_2|_{p \subset e_1}}(x_2)$ the distance function that is created on the edge e_2 by the interval $p \subset e_1$,

$$D_{e_2|_{p \subset e_1}}(x_2) = \min_{x \in p \subset e_1} (D_{e_1}(x) + \Delta(x, x_2)). \quad (4.1.6)$$

The equation (4.1.1) can be restated as minimum over the interval propagated functions $D_{e_2,p}$,

$$D_{e_2}(x_2) = \min_{p \subset e_1} D_{e_2|_{p \subset e_1}}(x_2). \quad (4.1.7)$$

The interval propagation function $D_{e_2|_{p \subset e_1}}(x_2)$ can be computed from the geometrical reasons as follows. Part of the edge e_2 can "see" the pseudo-source s_p of the interval p directly through the interval p . Obviously, for the points in this region, the shortest path to the pseudo-source is a straight line. Therefore, this whole region can be represented as an interval with the pseudo-source s_p (fig. 4.3).

The shortest path of the points to the "left" side of this region have to pass through the "left" end of the interval p . That is why these points can be represented as an interval with new pseudo-source $s_{p,l}$. Similarly, the points on the "right" side of the edge can be represented as an interval with new pseudo-source $s_{p,r}$.

Summarizing, the interval propagation function $D_{e_2|_{p \subset e_1}}(x_2)$ consist of up to three intervals with different pseudo-sources. In theory, the shortest path cannot bend on the edge (it can bend only at the saddle or boundary vertex) and construction of the "left" and "right" intervals might seem unnecessary in many cases. However, by introducing these additional intervals, we achieve two goals simultaneously. First, we do not have to come up with a special treatment of the saddle and boundary vertices.

Second, and more important, this trick provides numerical stability of the algorithm and covers small gaps in the "wavefront".

4.1.4 Interval intersection

In order to compute the minimum of two interval functions in formula (4.1.7) and algorithm (4.1), we need to find the points where two interval functions are equal to each other. Unlike the interval propagation, this operation is easier to do algebraically. Let us say that two different functions $D_1(x)$ and $D_2(x)$ are defined on the edge e . We want to find such points \hat{x} that $D_1(\hat{x}) = D_2(\hat{x})$. When the pseudo-sources are the same distance from the initial source, $d_1 = d_2$, there is one intersection at most.

$$\hat{x} = \frac{c_2^2 - c_1^2}{2(c_1 \cos \theta_1 - c_2 \cos \theta_2)} \quad (4.1.8)$$

In general, when $d_1 \neq d_2$, we have to solve a quadratic equation

$$A\hat{x}^2 + B\hat{x} + C = 0, \quad (4.1.9)$$

where

$$\begin{aligned} A &= P^2 - D^2, & B &= PQ - 2c_1 \cos \theta_1 D^2, \\ C &= Q^2/4 - c_1^2 D^2, & P &= c_1 \cos \theta_1 - c_2 \cos \theta_2, \\ & & Q &= D^2 + c_1^2 - c_2^2, & D &= d_1 - d_2. \end{aligned}$$

4.1.5 Interval-based algorithm

Putting it together, we can now derive final interval-based algorithm. In this algorithm, each edge distance function $D_{e_i}(x)$ is represented as a list of intervals L_{e_i} . Therefore, when we propagate an interval onto another edge, we create a new list of intervals that consists of up to three intervals (section 4.1.3). When we find the minimum of two distance function, we intersect two lists of intervals and take the smallest ones (section 4.1.4).

Obviously, all the edges that belong to the same faces as the source, can be seen from the source directly and therefore have only one interval each.

The interval propagation algorithm can be seen in the table (4.2). We maintain the interval priority queue sorted by the minimum value of the distance function on the interval. This queue is described in [40] in order to maintain a Dijkstra-style properties of the interval propagation. Computing the minimum value of the distance function on the interval is straightforward and omitted here for the sake of space.

In our implementation of the algorithm, each interval also has a bit that shows the direction of the interval propagation. Instead of propagating the interval into two directions, we propagate it only into one direction. In practice it makes the code faster, because the interval propagation procedure is computationally expensive.

algorithm	IntervalPropagation
<ol style="list-style-type: none"> 1. Represent each edge that belongs to the same face as source vertex by one interval. Put these intervals in the priority queue. 2. until the queue is empty 3. Remove the next interval p from the queue. It belongs to an edge e. 4. for all edges e_i that belong to same faces as e. 5. Compute propagated interval list $\hat{L}_{e_i,p}$. 6. Update the interval list of the edge at e_i as $L_{e_i} = \min(L_{e_i}, \hat{L}_{e_i})$. 7. Remove deleted intervals from the priority queue. 8. Add created intervals to the priority queue. 9. end for 10. end until 	

Table 4.2: Interval-based algorithm for computing minimal distance functions on the edges. When the interval list is updated, some intervals might be deleted and/or created.

4.1.6 Properties of the interval-based algorithm

Our interval-based algorithm is based on the algorithm developed in [40], but has several crucial differences.

We do not have to worry about saddle vertices and boundary vertices, which otherwise require additional treatment. In the original algorithm, every saddle vertex became a pseudo-source of intervals. Similarly, it was reported by [31] about Chen&Han algorithm, that such vertices required significant additional processing time in their implementation of the shortest path algorithm.

Even more important property is robustness. When the intervals are propagating, some of them become very small. The processing of the small intervals can require a lot of time and memory, and become a source of numerical instability. However, in MMP algorithm, it was not possible to clip small intervals, because it could result in large gaps when the intervals are propagated. By propagating side lobes of the intervals, we guarantee that all the gaps are covered in the very beginning.

This robustness property gives us an opportunity to construct a wide class of the

approximate algorithms. We can simplify the intervals when they are small or when two neighboring intervals are similar. By construction of our algorithm it will never lead to gaps when the intervals are propagated.

Also, in contrast to [40], we do not have to consider edge-face pairs and can treat each edge independently of where the signal comes from. This simplifies algorithm implementation.

4.2 Merge simplifications

Unfortunately, the exact algorithm is in the worst case quadratic in memory and $n^2 \log n$ in time [40]. When the mesh has hundreds of thousands of vertices, this non-linear growth can become a bottleneck. The reason of such a growth is that far from the source intervals become smaller and smaller. In many cases, a set of small intervals can be approximated by a single interval with high precision.

In both MMP and Chen&Han algorithms, even the slightest modification of any interval could lead to a potential distance function gap somewhere on the mesh. On the contrary, our algorithm remains stable after interval approximation, which is demonstrated in the following lemma.

Lemma 4.2.1. *Let us take the results of the exact algorithm, pick one of the edges, e_k , and one interval p_m on this edge. Let us say that the distance function on this interval, $D_{p_m \subset e_k}(x)$ is defined by the constants d, c, θ . Let us also define a new distance function $\bar{D}_{p_m \subset e_k}(x)$ by changing these parameters arbitrarily and re-propagate the interval through the mesh. Then the resulting distance field in any point of the mesh will not change more than ε , where*

$$\varepsilon = \max_x |\bar{D}_{p_m \subset e_k}(x) - D_{p_m \subset e_k}(x)|.$$

To make a bridge to our flat-exact algorithm developed in the next chapter, here we describe a possible simplification algorithm that is based on interval merging. Because of the perfect results shown by the flat-exact algorithm, we did not implemented the merging algorithm described here.

The easiest merging strategy is to check the neighbors of the interval before its propagation, and if their distance function are approximated by its distance function well enough, merge them together.

There are still two issues remain. First, after a chain of simplifications the algorithm can come to the loop, i.e. some interval become modified by its own "children" intervals. Second, simplifications can create local minimums in the distance field of the mesh and the tracing-back algorithm that computes geodesics can stuck in such a minimum.

It is possible that both issues can be handled by applying some restrictions on the simplification rules. We did not find the set of restrictions that is elegant and fast to compute; that is why we resolve both issues by the following method.

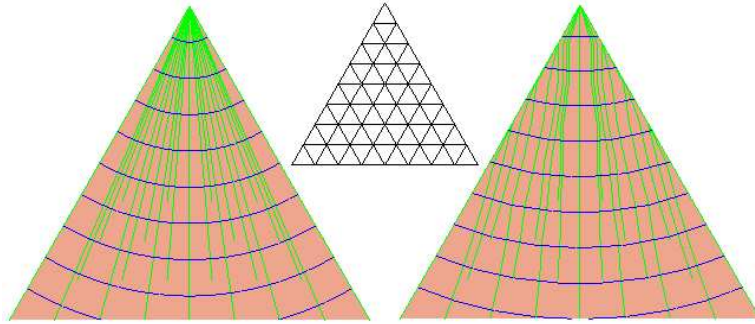


Figure 4.4: Comparison of the approximate geodesics algorithms for the planar mesh, which is subdivision of a triangle (top). The source is located in the upper corner of the mesh. Kimmel-Sethian algorithm is on the right, our flat-exact one is on the left. Geodesics are shown in green, iso-distant curves are shown in blue.

4.2.1 Dependence DAG

In our algorithm, every interval is a result of the propagation of another interval (for mathematical accuracy, we have to mention that the intervals adjacent to the source vertex are the results of the propagation of this vertex). We can consider a dependence graph. A vertex of this graph corresponds to an interval of the mesh, and the edges of the graph are directed from the parent intervals to their children. At every stage of our exact algorithm, this directed graph is acyclic (DAG). The idea is to maintain such a dependence DAG for approximation algorithm.

In the exact algorithm, an interval has only one parent by construction. The only difference of the approximation algorithm is that one interval can have many parents. Whenever the intervals are merged, the resulting interval inherits all their children and parents. Similarly, whenever we create new interval that covers the area of the edge that belonged to some other interval, it has only one parent but inherits all children of this previous interval.

We merge or propagate intervals only if the dependence graph remains acyclic. This rule solves the cycling problem automatically. By back-tracing the geodesics only from children to parents, we make sure that the tracing algorithm always converges to the origin.

Checking whether the resulting graph remains acyclic in theory requires breadth-searching the whole graph, and therefore ruins the complexity of our algorithm. However, in practice this check is extremely fast, because most of the time we do not have to consider the entire graph. The updated intervals are located on the frontier of the distance propagation wave, i.e. they are located very close to the leaves of the DAG. Therefore, if the graph remains acyclic, the search will stop after processing the few descendants of the updated interval.

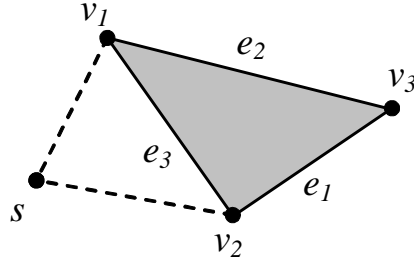


Figure 4.5: Edge e_3 with source s is used to compute the propagated distance function in the vertex v_3 .

4.3 Flat-exact algorithm

Though interval simplification improves both time and memory complexity of the algorithm, it is even more complicated to implement this algorithm from scratch than the original exact algorithm. That is why we focused on the approximation algorithm that is also very simple to implement. Though it can be considered as an extreme case of the interval simplification described in the previous chapter, this algorithm can also be related to algorithm described in [34].

The following section describes an approximation algorithm where each edge is represented by a single interval. Now terms "edge" and "interval" are equivalent and we will call interval parameters to be edge parameters.

In the extreme case when the entire mesh is located in one plane, the resulting distance function will coincide with the exact one. Because of this reason, we call the simplification to be flat-exact. This property is extremely important in practice. For example, figure (4.4) shows the planar mesh that was created as a regular subdivision of a triangle with the source in the upper corner of the mesh.

The main structure of the interval-based algorithm remains unchanged. Initially, one interval is created for all edges that belong to the same faces as the source. Then the intervals are propagated by a very simple rules across the mesh.

4.3.1 Simplified interval structure and propagation rules

Another simplification we make is that the distance field on the whole mesh can be represented by the distances values in the vertices of the mesh. In particular, the distance function on the edge is defined by its values in the vertices of the edge, $D(v_1)$ and $D(v_2)$. The distance function on the edge is still of the form (4.1.4), that has three parameters (d, c, θ) . In order to compute them from the two vertex distance values, from now on we consider $d = 0$, i.e. we ignore the pseudo-sources. After this assumption, the one-to-one correspondence between the (c, θ) and $(D(v_1), D(v_2))$ is

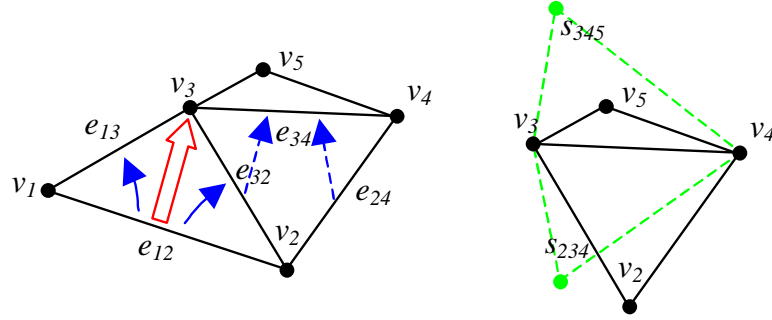


Figure 4.6: Construction of the dependance DAG. The vertex v_3 is updated from the edge e_{12} (red arrow), the blue arrows show the desirable parent-children links. On the right, two possible pseudo-sources for the edge e_{34} are located symmetrically to this edge.

trivial

$$D(v_1) = c, \quad (4.3.1)$$

$$D(v_2)^2 = c^2 + L^2 - 2cL \cos \theta, \quad (4.3.2)$$

where L is the length of the edge. Our new construction also implies that the distance function is continuous on the 1-skeleton of the mesh.

The propagation rules change as follows. Given the edge e_3 to be propagated, we first define the distance in the opposite vertex of the triangle (fig. 4.5). If the quadrilateral (s, v_1, v_3, v_2) is convex, i.e. the opposite vertex can "see" the source s of the interval, the propagated distance $\hat{D}(v_3)$ is computed as Euclidean distance between two points, $\hat{D}(v_3) = d(s_e, v_3)$. Otherwise, the distance is computed along the edge, $\hat{D}(v_3) = \min(D(v_1) + |e_2|, D(v_1) + |e_2|)$.

If new distance in the vertex v_3 is smaller than the old one, we update all the edges adjacent to v_3 and put them in a priority queue. The simplified algorithm is represented in the table 4.3.

4.3.2 Dependance DAG for the simplified algorithm

In order to avoid cycling, we use the same idea of the dependance DAG as before. Each edges of the is represented by a vertex in the DAG. The dependence information is now maintained as follows. Let us say that the vertex v_3 has its distance updated from the edge e_{12} that belong to the face f_{123} (fig. 4.6). The distance function over all the edges adjacent to v_3 have now been affected by e_{12} . So to avoid cycling in the distance propagation, we add a pointer from e_{12} to all of the edges adjacent to v_3 .

During shortest path back-tracing, we will only trace paths that go towards ancestors edges, as recorded in the DAG. If an edge has no ancestor edges that it shares

a face with, then we only trace the path back to one of its vertices. So, when v_3 is updated to allow as much edge to edge back-tracing when possible we would also like to other DAG edges.

We do this by "estimating" the direction to the source and adding extra dependencies into the DAG as follows. Consider edge e_{34} , and assume that the weights of both v_2 and v_5 are finite. To assess direction of the source we compute the parameters (c_{35}, θ_{35}) and find the positions of two possible pseudo-sources, s_{234} and s_{345} . Then we also assess how well these pseudo-sources approximate the distance function values in the vertices v_2 and v_5 .

$$\begin{aligned} Q_{234} &= |D(v_4) - \|s_{234} - v_4\|| + |D(v_3) - \|s_{234} - v_3\||, \\ Q_{345} &= |D(v_4) - \|s_{345} - v_4\|| + |D(v_3) - \|s_{345} - v_3\||. \end{aligned}$$

If $Q_{234} < Q_{345}$, we set e_{32} and e_{24} as parents of e_{34} otherwise we set e_{35} and e_{54} to be parents of e_{34} .

If both of the vertices v_2 and v_5 have infinite weights, then no extra DAG information is added, and back-tracing will have to go through edges in the graph.

Again, the initial vertex v_3 is updated only in case that after this update the resulting dependency graph remains acyclic.

Finally, we want to mention that the situations when the dependency graph may become cyclic are very rare and have only been seen in hand-crafted examples.

4.4 Results

In order to test the algorithms, we compared them with several existing ones. The fastest (and least precise) algorithm for approximate geodesic path computation is Dijkstra shortest path that runs on the edges of the mesh. We also implemented an acute version of the approximation algorithm proposed by Kimmel and Sethian [34] and used publicly available implementation of Chen and Han's algorithm by Kaneva and O'Rourke [31] (due to unknown implementation reasons, we were unable to test this algorithm to all the meshes we have). The algorithms were tested on an Athlon 1.5GHz machine with 2Gb of memory.

To assess the relative error, we used the following test. Given the mesh and a randomly selected source vertex, we computed the shortest paths to all vertices of the mesh. The relative error was computed as a sum over all vertices

$$E = \frac{1}{N_p} \sum_i \left| \frac{|\bar{p}_i| - |p_i|}{|p_i|} \right|, \quad (4.4.1)$$

where $|p_i|$ is the length of the shortest path obtained by the exact algorithm, $|\bar{p}_i|$ is the length of the shortest path obtained by an approximation algorithm, and N_p is the total number of the computed paths.

algorithm	SimplifiedIntervalPropagation
<ol style="list-style-type: none"> 1. Compute the distance for all vertices adjacent to the source. Put all edges that belong to the same faces as source in the priority queue. 2. until the queue is empty 3. Remove the next edge e from the queue. 4. for both faces f_i adjacent to e. 5. Consider the vertex v opposite to e. 6. Compute updated distance function $\hat{D}(v)$. 7. Update the dependence graph. 8. if $\hat{D}(v) < D(v)$ and the dependence graph is acyclic 9. $D(v) = \hat{D}(v)$ 10. Put all edges adjacent to v in a queue. 11. end if 12. end for 13. end until 	

Table 4.3: Simplified algorithm for computing minimal distance function.

The results of both exact and flat-exact algorithms are shown in figures 4.8, 4.4.

One of the interesting effects that we observed with our exact algorithms, is that it usually works much faster for the models with a lot saddle vertices and gives the slowest performance on the saddle objects (compare *sphere2* vs. *parasaur* models). This happens because on the convex models the intervals do not eliminate each and become smaller and smaller while the "wavefront" propagates further from the source.

The flat-exact algorithm runs a little bit slower than Kimmel-Sethian algorithm because of the necessity to maintain the DAG structure, but gives significantly better results.

The only limitation of the flat-exact algorithm that we could find can be observed when the two distance "wavefronts" meet on the opposite sides of some obstacle. In this case our one-interval simplification rule oversimplifies the distance field and the

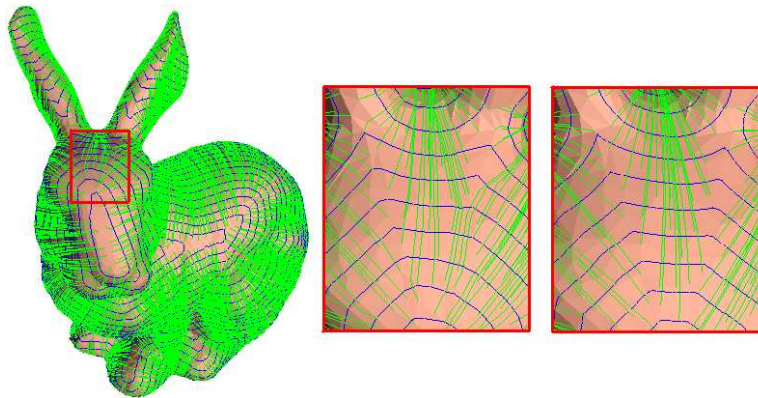


Figure 4.7: Limitations of our flat-exact algorithms. In the areas where several wavefronts meet together, the one-interval approximation of the edge distance function leads to geodesic distortion. Enlarged area of the bunny head is shown in the middle (flat-exact). Result of the exact algorithm is shown to the right. Geodesics are shown in green, iso-distance curves are shown in blue.

geodesics are visibly distorted (fig. 4.7).

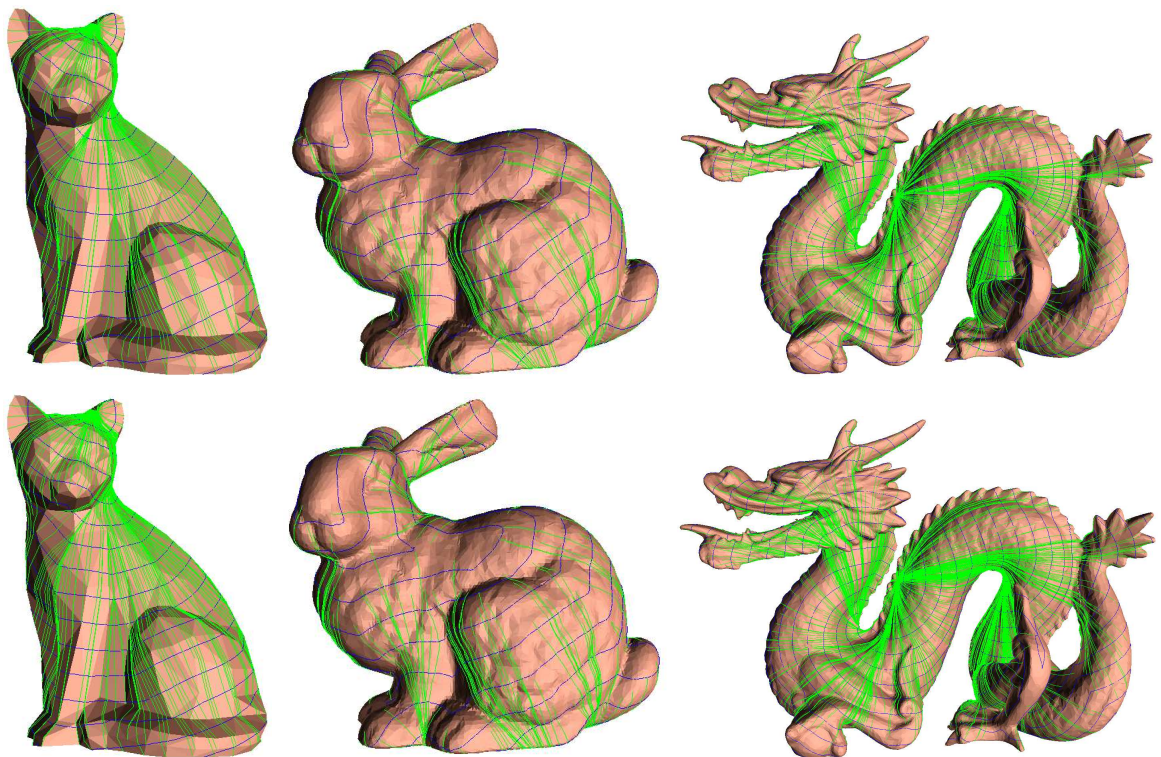


Figure 4.8: The results of the our exact (top row) and flat-exact (bottom row) algorithms are shown on the cat, bunny, and dragon models (366, 10000, and 50000 vertices respectively). Geodesics are shown in green, iso-distant curves are shown in blue. In the bunny model, the source is on the back side.

model (# of vertices)	Dijkstra	Kimmel-Sethian	Chen-Han	Our Exact	Our Flat-exact
cat (366)	0/6.1	0.011/2.0	5.0/0	0.12/0	0.015/0.27
plane (561)	0/7.3	0.015/0.4	5.2/0	0.031/0	0.016/0
sphere1 (1562)	0.01/7.2	0.047/0.89	11.0/0	2.56/0	0.063/0.02
sphere2 (6242)	0.015/7.2	0.203/0.82	-	40.1/0	0.21/0.01
handmale (7609)	0.023/5.7	0.26/2.6	-	11.5/0	0.34/0.24
bunny (10000)	0.025/5.3	0.37/1.8	44160.0/0	12.5/0	0.41/0.15
buddhaf (14990)	0.06/4.8	0.62/3.7	101500.0/0	10.9/0	0.85/0.66
parasaur (21935)	0.082/6.3	0.78/1.1	-	45.5/0	0.86/0.1
rockerarm (40177)	0.16/5.7	1.6/0.34	-	235.0/0	1.7/0.02
horse (48500)	0.21/4.9	1.9/0.4	-	343.0/0	2.2/0.05
dragon (50000)	0.23/6.4	2.3/2.0	-	64.0/0	2.7/0.19

Table 4.4: Time/error statistics for the geodesics algorithms. Time is given in seconds, the relative error is computed by 4.4.1 and shown in percents.

Silhouettes

5

In this chapter we investigate one more problem that can be stated as a problem of finding the globally minimum curve on the surface. It turned out that the duality introduced for the globally minimum surfaces can be also used in this case.

Complex meshes tend to have intricate, detailed silhouettes. In this chapter we propose two algorithms for extracting a simpler, approximate silhouette from a high-resolution model. Our methods preserve the important features of the silhouette by using the silhouette of a coarser, simplified mesh as a guide. Our simple silhouettes have significantly fewer edges than the original silhouette, while still preserving its appearance.

Silhouettes play an important role in several areas of computer graphics, such as interactive visualization, non-photorealistic rendering and shadow computation. There are several systems that draw silhouettes in various ways to help depict geometric models ([20, 21, 23, 39], and many others). Unfortunately, many high-resolution geometric meshes often have overly complicated silhouettes. For example, see second column of the figure 5.5, which shows the silhouette of the models as viewed from both the eye position as well as an auxiliary side view; these silhouettes contain a large number of loops, many intertwined loops, and loops that are quite wiggly. This silhouette complexity is usually unnecessary and sometimes costly, both in terms of speed and quality of rendering algorithms applied on silhouette edges. In this thesis, we propose methods for computing "approximate silhouettes" that attempt to represent the most salient features of the silhouette, while suppressing the extra detail and complexity. Our basic approach is to first compute the actual silhouette of a smoothed coarse geometric mesh that approximates the original model (see figure 5.5, column 1). This "coarse silhouette" typically has a simpler structure. Our goal is to create a silhouette for the fine mesh that is similar in structure to the coarse silhouette. This results in a high resolution, but "simple" silhouette (see figure 5.5, cols. 3,4). We have explored a few approaches, and in this thesis we describe two different methods that present various tradeoffs.

5.1 Previous Work

There have been a number of interesting approaches to efficiently compute silhouettes on models. Sander et al. [48] build a spatial hierarchy that allows one to quickly dismiss subsets of mesh edges as being non-silhouette. Barequet et al. [3] and Hertzmann and Zorin [23] describe methods that also build a hierarchy, but do their computation in a dual space. Barequet et al. [3] also describe how their method can be used incrementally to more quickly recompute silhouettes under small viewpoint changes. Markosian et al. [39] propose a method to find silhouettes by tracking them from frame to frame as well as randomly testing for the creation of new silhouette loops. This method tends to find the larger loops more reliably. Northrup et al. [42] address the simplification of the silhouettes in the image space by linking the vis-

ible segments into the chains, thus resulting in a simpler, approximate silhouette. Hertzmann and Zorin [23] describe an elegant way of finding simple approximate silhouettes on meshes. Silhouettes found by this method are quite smooth and are always comprised as a set of non-intersecting loops. In their method, the silhouette edges all span across mesh triangles and are not original mesh edges. In our methods the silhouette is actually composed of actual mesh edges. Furthermore, redundant short loops are usually removed from our computed simple silhouettes.

5.2 Approach

The input to our system is a high-resolution fine mesh. From this mesh, we create a progressive mesh [25]. We then extract a simpler, coarser mesh, and apply smoothing on it in order to prevent artificially generated silhouettes. During runtime, for a given viewpoint, the objective is to create a silhouette for the fine mesh that is similar in structure to that of the coarse mesh. In this section we will describe two approaches to achieve this objective. In particular we will describe one method in which the set of the edges in the coarse silhouette are constrained to be some subset of the actual silhouette edges. In this method, loops from the actual silhouette are iteratively included into the simple silhouette as long as they sufficiently match the coarse silhouette. The second method constrains the simple silhouette to be structurally similar to the coarse silhouette. For each coarse-silhouette loop, a corresponding loop is found on the original mesh. With this second approach, it may be necessary to sometimes include non-silhouette edges in the simple silhouette. Next we will describe how we decompose the silhouette into loops - a sub-step that is required for both methods, - and then we present the two silhouette extraction methods.

5.2.1 Loop decomposition

On a smooth surface, the silhouette is comprised of a set of disjoint closed loops. On a triangle mesh, these loops can intersect (for example, a silhouette vertex can be incident to four silhouette edges) creating a slightly more complicated structure. Our first step will be to take a complicated silhouette and describe it as the union of (possibly intersecting) loops. Given a viewpoint, a silhouette edge is an edge that is adjacent to one front-facing triangle and one back-facing triangle. Silhouette edges are directed edges, where the left adjacent face is front-facing. A silhouette vertex is a vertex adjacent to a silhouette edge. In particular, on a triangle mesh, the faces around a silhouette vertex can be partitioned into an alternating set of front-facing and back-facing triangles, thus the number of silhouette edges adjacent to a silhouette vertex must be *even*. Since each silhouette vertex has an even "silhouette edge valence," we can decompose the silhouette into a set of loops. We desire a decomposition that results in loops without self-intersections. We achieve this by

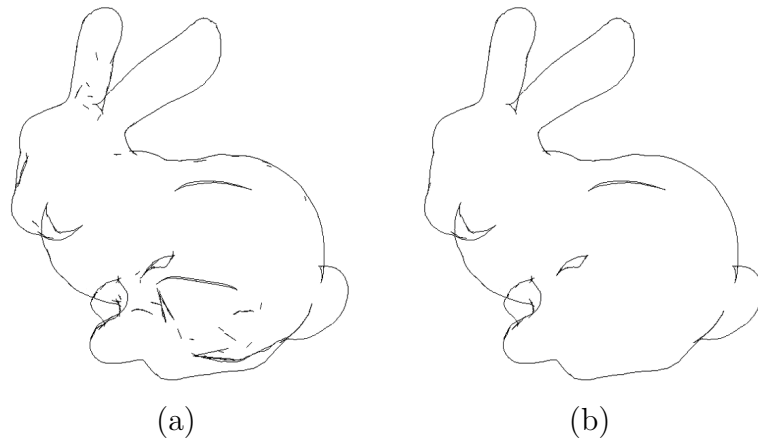


Figure 5.1: Fine silhouette (left) and simple silhouette (right) computed using the loop picking algorithm. The simple silhouette consists of only four loops.

starting with an arbitrary vertex and walking along the directed silhouette edges until we get back to the original vertex, thus closing the loop. We repeat this process until all silhouette edges have been assigned to loops. In order to accelerate the walking algorithm, we use a hash table of directed silhouette edges indexed by the source vertices [48].

During this walk, we may get to intersections where we have pick between two or more edges to follow. When that happens, we just arbitrarily pick any one of the possible edges. If at any point during a walk we encounter a sub-loop, we "detach" it and store it as a separate loop. This method does not produce unique loop decomposition, as it is sensitive to the location where the walk starts and the decisions made at silhouette intersections. However, this greedy approach is extremely fast, and by snapping off sub-loops, it achieves our objective of constructing loops without self-intersections.

5.2.2 Loop picking

In this approach, we first extract the silhouette of both the coarse and fine meshes using the algorithm from [48]. We then perform loop decomposition on the fine silhouette, yielding a set of silhouette loops. Our goal is to pick a subset of the fine silhouette loops that minimizes an error metric based on distance to the coarse mesh silhouette. More specifically, the metric is the sum of the squared distances between each coarse silhouette vertex, and its closest point on the fine silhouette. The loops are picked one by one in greedy fashion, minimizing this metric. We terminate when the error gets below a specified threshold that is proportional to the amount of detail that is desired. In order to do this efficiently, we first pre-compute the distances from

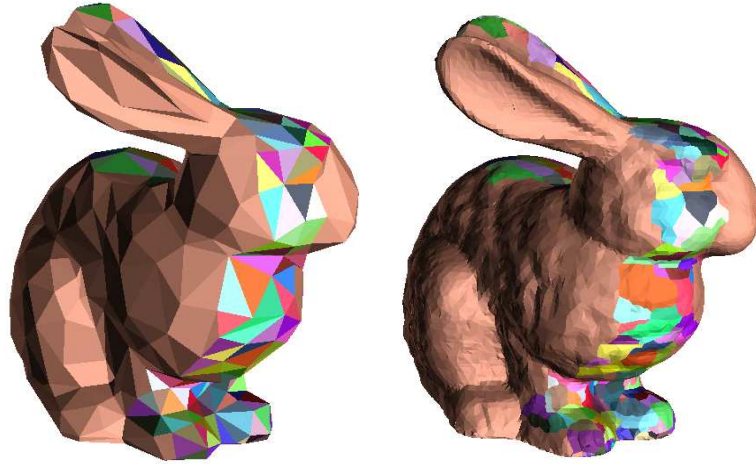


Figure 5.2: Coarse and fine bands are shown with colored triangles on the bunny modes. In order to show the mapping between models, each triangle on the coarse mesh (left) has the same color as corresponding fine triangles on the fine mesh (right).

every coarse silhouette vertex to its closest vertex on each of the fine silhouette loops, and store these distances in a 2D array. When evaluating a candidate loop, we just do look-ups to this array to compute the error. Applying the above algorithm results in a set of long silhouette loops that resemble the silhouette of the coarse mesh. Short loops and loops that are not geometrically close to a coarse mesh silhouette edge are not picked because they do not significantly decrease the error. Very short loops are pruned and discarded a priori in order to make the search more efficient. Figures 5.1 and 5.5(col. 3) show examples of simple silhouettes extracted using the above algorithm. For performance results, refer to the Section 5.3 .

5.2.3 Method B: Loop mapping

In the previous method, we started with the fine silhouette, and selected loops based on their proximity to the coarse silhouette. In this method, we instead start with the coarse silhouette, and try to create loops over the fine mesh that resemble the coarse loops. For every loop of the coarse silhouette we find a corresponding loop on the fine mesh. The goal of this method is to retain all major features of the silhouette and ensure continuity when the model is rotated. By meeting this goal we do not guarantee that all edges of the fine loop belong to the fine silhouette.

The first step of this algorithm is to decompose the coarse silhouette onto non-self-intersecting loops using the method described in Section 5.2.1. We also need a mapping between coarse and fine models, i.e. for each fine triangle we need to know the corresponding coarse triangle (figure 5.2). To create such a mapping, we

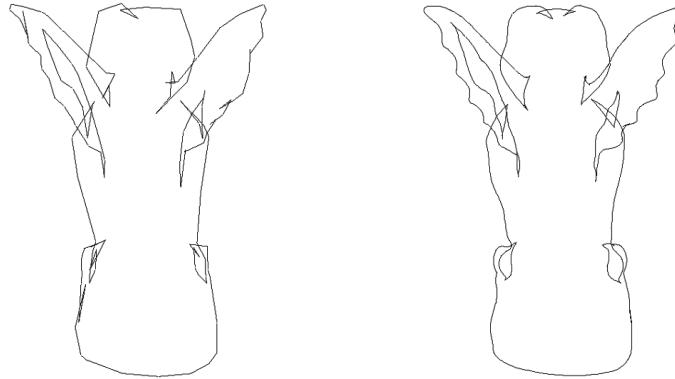


Figure 5.3: One loop of the coarse silhouette of the gargoyle (left) and its mapping on the fine mesh (right).

simplify the mesh using half-edge collapses. Each coarse mesh edge corresponds to the shortest path of fine mesh edges between the two analogous vertices on the fine mesh. Therefore, each face can then be trivially mapped to a set of faces on the fine mesh. We define the *coarse band* of a coarse loop as all triangles that have at least one common vertex with this loop. The coarse loop splits the coarse band into two parts, internal and external. The outer boundary of the internal part of the band is called *internal boundary*, the outer boundary of the external part of the band is called *external boundary*. The internal and external boundaries of the coarse band could consist of more than one loop, but this does not affect our algorithm. We can map the coarse band from the coarse mesh onto the fine band on the fine mesh (figure 5.2). Note, that internal and external boundaries of the coarse band are mapped to the internal and external boundaries of the fine band. Our goal is to approximate the coarse loop with the best possible loop in the fine band. To do this, we find a minimal-cost loop of edges over the fine mesh that separates the internal and external boundaries. In order to measure the cost of an edge, the simplest approach would be to set its weight to be equal to its length. In this case, we will be looking for the shortest loop possible. To ensure that the fine loop is close to the fine silhouette, we multiply the weights of the fine edges that are on the fine silhouette by the small parameter α . Increasing the value of this parameter makes the loop smoother and decreasing forces it to include more edges from the actual silhouette. In our experiments we set $\alpha = 0.1$. Figure 5.3 shows a loop from the coarse silhouette and its corresponding loop on the fine mesh. Now we have to find a minimal path on the weighted graph that partitions it into two parts. This is a standard problem that can be easily solved with the minimal cut approach. First, we construct the dual graph ¹ (figure 5.4). That is, we associate

¹compare to section 2.6.1

with each face of the band f_i a dual vertex F_i . We also create a single source vertex R and a sink vertex K . With each non-boundary edge e_i bounding two faces f_j and f_k , we associate a dual edge E_i that connects F_j and F_k . For each edge e_i belonging to the external boundary we associate a dual edge E_i which connects the sink K with the single face that e_i bounds. For each edge e_i belonging to the internal boundary we associate a dual edge E_i which connects the source R with the single face that e_i bounds. We set the capacity of each dual edge the weight of the associated primal edge $c(E_i) = w_i$. We define a cut of the dual graph as a partition of the vertices F_i into two sets F and Q with $R \in F$ and $K \in Q$. The cost of a cut is the sum of the capacities of the edges between these sets. Similar to section 2.6.1, it can be shown that the edges across the minimal cut in the dual graph correspond to dual to the edges of the minimal separating path in the primal graph. The only difference is that in this setup we do not have a boundary and the curve is closed.

In summary, the entire algorithm can be described as follows:

- Preprocess:
 - Simplify original mesh and establish mapping between the simplified coarse and the original fine mesh.
- Runtime:
 - Decompose coarse silhouette into loops.
 - For every coarse loop:
 - * Calculate the coarse band.
 - * Find the internal and external boundaries of the band.
 - * Find the corresponding fine band and its boundaries.
 - * Compute the weights of the edges and create a dual graph.
 - * Compute the min-cut of the dual graph.
 - * Find the corresponding fine loop.

5.3 Results

We implemented both algorithms, and applied them to several models. Table 5.1 shows the mesh resolutions and timings of our algorithm on the bunny and gargoyle models. Figure 5.5 shows silhouette renderings for both methods. We also show side views of the silhouette, in order to demonstrate how intricate the original fine mesh silhouette is. For silhouette extraction, we used an optimized version of the algorithm from Sander et al. [48]. It extracts the silhouette of a 20,000-face bunny mesh in half a millisecond. The loop decomposition step from Section 5.2.1 also takes approximately half a millisecond for that model. The total silhouette extraction time

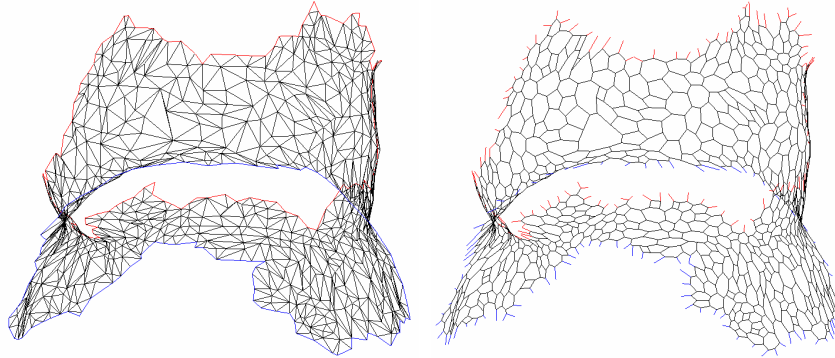


Figure 5.4: Primal (left) and dual (right) graphs. Internal boundary (red) on the primal graph corresponds to the source edges on the dual graph. External boundary (blue) on the primal graph corresponds to the sink edges on the dual graph.

	bunny	gargoyle
The number of coarse mesh faces	1,000	1,000
The number of fine mesh faces	20,000	30,000
coarse mesh sil. extraction time (ms)	0.038	0.042
+ loop decomposition	0.098	0.102
fine mesh sil. extraction time (ms)	0.553	0.917
+ loop decomposition	1.051	1.602
Loop picking total time (ms)	5.172	9.582
Loop mapping total time (ms)	190.000	521.000

Table 5.1: Quantitative results (Pentium 4, 2.0Mhz).

for the loop picking algorithm is 5 and 10 milliseconds for the bunny and gargoyle models, respectively. So, we can compute the simple silhouette of a 20,000-face bunny model at a rate of 200 frames/sec. Note that this does not include model or silhouette rendering time. The loop mapping algorithm is significantly slower, running at a rate of 2 to 5 frames/sec. However, it results in a rendering with far fewer silhouette edges, while still resembling the fine mesh silhouette, as shown in Figure 5.5.

5.4 Discussion

In this chapter, we described two algorithms to extract a simple silhouette from a high-resolution mesh by using the silhouette of a coarse, simplified mesh as a guide. The silhouette produced by both of these methods are composed by actual edges

of the original model. The loop picking algorithm is very efficient and extracts a subset of the actual silhouette edges. The loop mapping algorithm is slow, but it is able to capture all the important features of the silhouette with far fewer loops, thus further eliminating the redundancy present on the silhouette of high resolution meshes. Another advantage of this method is that only small amount of the edges of the fine mesh has to be processed. The major limitation of both proposed approaches is that we cannot guarantee the temporal coherence of the silhouette approximation for the moving models.













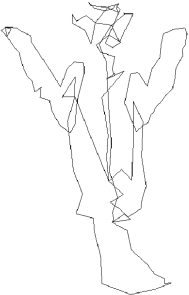

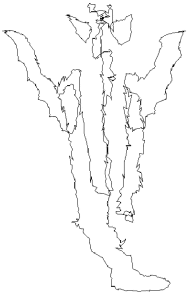
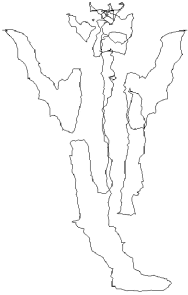
coarse mesh silh.	fine mesh silh.	loop picking	loop mapping
			
			
154 edges	1668 edges	1014 edges	720 edges
			
			
155 edges	1845 edges	1081 edges	898 edges

Figure 5.5: Silhouette rendering comparisons of the bunny and gargoyle meshes. Model parameters and benchmarks can be seen in the table 5.1. Front (row 1) and side (row 2) views of bunny mesh silhouette. Front (row 3) and side (row 4) views of gargoyle mesh silhouette.

Conclusion

6

In this thesis we discussed several methods that can be considered to be the generalizations of the Dijkstra shortest path method. In chapters 2 and 3 we go to higher dimensions and explore the problem of "shortest" discrete surface. The duality algorithm described there can be also successfully applied in the 1-dimensional case (chapter 5). In chapter 4 we allow the shortest path to cut across the mesh face and investigate the "continuous" version of Dijkstra.

The main property of all the algorithms described is their ability to find the *globally* minimal curves and surfaces in polynomial time. The natural price for this property is the higher-than-linear time and memory complexity ¹, which prevents us from running the algorithms on the data structures of a very high resolution.

The standard application of these algorithms could be generating the initial guess for the fast *local* methods that can be used on a structures of very high resolution but converge to the local minimums only. The balance between global and local methods still remains unclear. There are a lot of open questions in this relatively unexplored topic.

Another issue considered in this thesis is the close relationship between continuous and discrete methods. It is shown in chapter 2 that the continuous variational problem can be reduced to a purely discrete combinatorial optimization problem. On the other hand, in chapter 3 we addressed a discrete minimum weight triangulation problem by a continuous linear programming approach. The geodesic problem on the mesh can be successfully approached by the discrete algorithm (chapter 3) or by the numerical solution of the differential equation [34].

Unfortunately, it seems that very often the discrete and continuous communities do not interact very closely. Applied physics and engineering major students often the advanced algorithms classes; computer science major students ignore the numerical modelling classes. We hope that our thesis will increase the attention to the problems that are located on the boundary of these two worlds.

¹ $n \log n$ Dijkstra shortest path is a nice exception.

Bibliography

- [1] AICHHOLZER, O., AND AURENHAMMER, F. Constant level greedy triangulations approximate the mwt well. *Journal of Combinatorial Optimization 2* (1999), 361–369.
- [2] AURENHAMMER, F., AND XU, Y.-F. Optimal triangulations. *SFB Report F003-099, Tu Graz, Austria* (1999).
- [3] BAREQUET, G., DUNCAN, C., GOODRICH, T., KUMAR, S., AND POP, M. Efficient perspective-accurate silhouette computation. In *Symposium on Computational Geometry* (1999), pp. 60 – 68.
- [4] BEIROUTI, R., AND SNOEYINK, J. Implementations of the LMT heuristics for minimum weight triangulations. In *Proc. 14th ACM Symp. on Comp. Geom.* (1998), pp. 96–105.
- [5] BOYKOV, Y., AND KOLMOGOROV, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in computer vision. In *EMM-CVPR* (2001).
- [6] BOYKOV, Y., AND KOLMOGOROV, V. Computing geodesics and minimal surfaces via graph cuts. In *Proc. ICCV* (2003).
- [7] BOYKOV, Y., VEKSLER, O., AND ZABIH, R. Fast approximate energy minimization via graph cuts. In *Proc. ICCV* (1999).
- [8] BRECHTKEN-MANDERSHEID, U. *Introduction to the Calculus of Variations*. Chapman and Hall, 1991.
- [9] BUEHLER, C., GORTLER, S. J., COHEN, M. F., AND MCMILLAN, L. Minimal surfaces for stereo. *ECCV* (2002), 885–899.
- [10] CHEN, J., AND HAN, Y. Shortest paths on a polyhedron; part i: computing shortest paths. *Int. J. Comput. Geom. & Appl.* 6, 2 (1996), 127–144.
- [11] CHENG, S.-W., AND XU, Y.-F. Approaching the largest β -skeleton within a minimum weight triangulation. In *Proc. 12th ACM Symp. on Comp. Geom.* (1996), pp. 196–203.

-
- [12] CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms*. MIT Press, 2001.
- [13] DACOROGNA, B. *Direct Methods in the Calculus of Variations*. Springer-Verlag, 1989.
- [14] DICKERSON, M., AND MONTAGUE, M. A (usually?) connected subgraph of the minimum weight triangulation. In *Proc. 12th ACM Symp. on Comp. Geom.* (1996), pp. 204–213.
- [15] DRYSDALE, R., MCELFRISH, S., AND SNOEYINK, J. On exclusion regions for optimal triangulations. *Discrete Applied Mathematics* 109 (2001), 49–65.
- [16] E.L.LLOYD. On triangulation of a set of points in the plane. In *Proc. 18th IEEE Symp. on Foundations of Computer Science* (1977), pp. 228–240.
- [17] GAREY, M., AND JOHNSON, D. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [18] GIBLIN, P. J. *Graphs, Surfaces and Homology*. Chapman and Hill, 1981.
- [19] GLOWINSKI, R. *Numerical Methods for Nonlinear Variational Problems*. Springer-Verlag, 1984.
- [20] GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH* (1998), pp. 447 – 452.
- [21] GOOCH, B., SLOAN, P., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. Interactive technical illustration. In *ACM Symposium on Interactive 3D graphics* (1999), pp. 31 – 38.
- [22] HATCHER, A. *Algebraic Topology*. Cambridge University Press, 2001.
- [23] HERTZMANN, A., AND ZORIN, D. Illustrating smooth surfaces. In *SIGGRAPH* (2000), pp. 517 – 526.
- [24] HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of ACM SIGGRAPH 2001* (2001), Computer Graphics Proceedings, Annual Conference Series, pp. 203–212.
- [25] HOPPE, H. Progressive meshes. In *SIGGRAPH* (1996), pp. 99 – 108.
- [26] HU, T. C. *Integer Programming and Network Flows*. Addison-Wesley, 1969.

- [27] HU, T. C., KAHNG, A. B., AND ROBINS, G. Optimal minimum-surface computations using network flow. *To appear in Mathematical Programming*.
- [28] HU, T. C., KAHNG, A. B., AND ROBINS, G. Solution of the discrete plateau problem. In *Proc. of the National Academy of Sciences* (1992), vol. 89, pp. 9235–9236.
- [29] JOHNSON, D. B. Parallel algorithms for minimum cuts and maximum flow in planar networks. *J. of the ACM* 34, 4 (1987), 950–967.
- [30] KANAI, T., AND SUZUKI, H. Approximate shortest path on a polyhedral surface and its applications. *Computer-Aided Design* 33, 11 (2001), 801–811.
- [31] KANEVA, B., AND O’ROURKE, J. An implementation of chen & han’s shortest paths algorithm. In *Proc. of the 12th Canadian Conf. on Comput. Geom.* (2000), pp. 139–146.
- [32] KAPOOR, S. Efficient computation of geodesic shortest paths. In *Proc. 32nd Annu. ACM Sympos. Theory Comput.* (1999), pp. 770–779.
- [33] KATZ, S., AND TAL, A. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics* 22, 3 (2003), 954–961.
- [34] KIMMEL, R., AND SETHIAN, J. A. Computing geodesic paths on manifolds. *Proc. National. Academy of Sciences* 95, 15 (1998), 8431–8435.
- [35] KLINCSEK, G. Minimal triangulations of polygonal domains. *Annals of discrete mathematics* 9 (1980), 127–128.
- [36] KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of SIGGRAPH 98* (1998), Computer Graphics Proceedings, Annual Conference Series, pp. 105–114.
- [37] KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of SIGGRAPH 96* (1996), Computer Graphics Proceedings, Annual Conference Series, pp. 313–324.
- [38] LANTHIER, M., MAHESHWARI, A., AND SACK, J.-R. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.* (1997), pp. 274–283.
- [39] MARKOSIAN, L., KOWALSKI, M., TRYCHIN, S., BOURDEV, L., GOLDSTEIN, D., AND HUGUES, J. Real time non photorealistic rendering. In *SIGGRAPH* (1997), pp. 415 – 420.

-
- [40] MITCHELL, J., MOUNT, D. M., AND PAPADIMITRIOU, C. H. The discrete geodesic problem. *SIAM J. Comput.* 16 (1987), 647–668.
- [41] MORGAN, F. *Geometric Measure Theory: a Beginner's Guide*. Academic Press, 1988.
- [42] NORTHRUP, J., AND MARKOSIAN, L. Artistic silhouettes: A hybrid approach. In *NPAR* (2000), pp. 31 – 38.
- [43] O’ROURKE, J. *Computational Geometry in C*. Cambridge University Press, 1994.
- [44] POLTHIER, K., AND SCHMIES, M. Straightest geodesics on polyhedral surfaces. *Mathematical Visualization, Ed: H.C. Hege, K. Polthier, Springer Verlag* (1998), 391.
- [45] PRAUN, E., HOPPE, H., AND FINKELSTEIN, A. Robust mesh watermarking. In *Proceedings of SIGGRAPH 99* (1999), Computer Graphics Proceedings, Annual Conference Series, pp. 49–56.
- [46] PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.
- [47] RADIN, C., AND SADUN, L. The isoperimetric problem for pinwheel tilings. *Comm. Math. Phys* 177 (1996), 255–263.
- [48] SANDER, P., GU, X., GORTLER, S., HOPPE, H., AND SNYDER, J. Silhouette clipping. In *SIGGRAPH* (2000), pp. 327 – 334.
- [49] SANDER, P., WOOD, Z., GORTLER, S., SNYDER, J., AND HOPPE, H. Multi-chart geometry images. *ACM Symposium on Geometry Processing 2003*.
- [50] SHAMOS, M., AND HOEY, D. Closest point problems. In *Proc. 16th IEEE Symp. on Computer Science* (1975), pp. 151–162.
- [51] SLOAN, P.-P. J., III, C. F. R., AND COHEN, M. F. Shape by example. In *2001 ACM Symposium on Interactive 3D Graphics* (2001), pp. 135–144.
- [52] SULLIVAN, J. *A Crystalline Approximation Theorem for Hypersurfaces*. PhD thesis, Princeton University, 1990.
- [53] TENENBAUM, J. B., DE SILVA, V., AND LANGFORD, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science* 290, 5500 (2000), 2319–2323.

- [54] ZIGELMAN, G., KIMMEL, R., AND KIRYATI, N. Texture mapping using surface flattening via multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics* 8, 2 (2002), 198–207.