Some Basis Function Methods for Surface Approximation

A thesis submitted in partial fulfilment of the
requirements for the Degree of
Doctor of Philosophy
in Mathematics

by Wen Eng Ong

Department of Mathematics and Statistics
University of Canterbury

2012

Abstract

This thesis considers issues in surface reconstruction such as identifying approximation methods that work well for certain applications and developing efficient methods to compute and manipulate these approximations.

The first part of the thesis illustrates a new fast evaluation scheme to efficiently calculate thin-plate splines in two dimensions. In the fast multipole method scheme, exponential expansions/approximations are used as an intermediate step in converting far field series to local polynomial approximations. The contributions here are extending the scheme to the thin-plate spline and a new error analysis. The error analysis covers the practically important case where truncated series are used throughout, and through off line computation of error constants gives sharp error bounds.

In the second part of this thesis, we investigates fitting a surface to an object using blobby models as a coarse level approximation. The aim is to achieve a given quality of approximation with relatively few parameters. This process involves an optimization procedure where a number of blobs (ellipses or ellipsoids) are separately fitted to a cloud of points. Then the optimized blobs are combined to yield an implicit surface approximating the cloud of points. The results for our test cases in 2 and 3 dimensions are very encouraging. For many applications, the coarse level blobby model itself will be sufficient. For example adding texture on top of the blobby surface can give a surprisingly realistic image.

The last part of the thesis describes a method to reconstruct surfaces with known discontinuities. We fit a surface to the data points by performing a scattered data interpolation using compactly supported RBFs with respect to a geodesic distance. Techniques from computational geometry such as the visibility graph are used to compute the shortest Euclidean distance between two points, avoiding any obstacles. Results have shown that discontinuities on the surface were clearly reconstructed, and the “across fault” influence is suitably small.

Acknowledgements

I would like to express my deepest gratitude and appreciation to my supervisors Rick Beatson and Chris Price. They have supported me throughout my thesis with their knowledge and patience. Their excellent guidance and encouragement have enabled me to complete this thesis.

I am grateful to the Ministry of Higher Education Malaysia and Universiti Sains Malaysia, for providing me the opportunity and funding my postgraduate studies.

Special thanks to all staff and colleagues in the Department of Mathematics and Statistics, University of Canterbury, for their valuable assistance and cooperation, making my stay in the department a very pleasant one. I am also thankful to all my peers and friends in Christchurch and in Malaysia, for their friendship, encouragement and support.

Finally I am indebted to my parents and siblings. Without their love and support, I would not have made it this far.

Contents

Chapter 1

Overview

1.1 Background of Surface Reconstruction

Surface reconstruction is one of the significant applications of scattered data interpolation and approximation (Lewis et al., 2010). This area of research is motivated by its many applications to different fields such as applied mathematics, geology, engineering, biology and etc. In practice, it is very much related to the subject of computational geometry and object modeling where curve, surface, solid or object representation is to be sought from a given finite set of data. The obtained mathematical representation is often studied and the reconstructed object is placed in computer graphics environment for viewing, printing, or further manipulation. Such work in seeking good representation and construction techniques is classified as a major branch of CAGD - Computer Aided Geometric Design. The term CAGD was first used by R. E. Barnhill and R. F. Riesenfeld as the title of a symposium organized at University of Utah in 1974. Surfaces reconstruction in CAGD has since then become a popular topic for research. Earlier applications of reconstructed surfaces were mainly in manufacturing aircraft, ships and cars. S. A. Coons and P. Bézier has pioneered methods of merging classical drawing or drafting a surface with computational techniques. More on history and early development in this area of CAGD can be found in the surveys by Barnhill (1985) and Farin (2002a).

Reconstructing a surface from a finite set of point data is sometimes known as surface fitting. Various surface reconstruction or surface fitting techniques exist, (Söderkvist, 1999; Chivate and Jablokow, 1995; Turk and O'Brien, 2002; Weiss et al., 2002; Carr et al., 2003), the choice of technique heavily depends on the application and the format of input data. A major part of work in this thesis focused on seeking approximation based surface reconstruction methods for certain applications of scattered data fitting. By scattered data we mean irregularly spaced and randomly distributed unorganized data points in contrast to points on a grid. Let us briefly review some of the well known techniques for this problem.

1.2 Surface Representations

Scattered data sets that are obtained from 3D scanners have become common nowadays. In fitting surfaces to such data, a continuous surface representation is desirable. The fitted surface is required to approximate each data point within some tolerance. It is also essential to be predictable, meaning that it should curve when necessary, but nowhere else. Other than that, the fitted surface is also expected to reasonably describe regions where no data is available. Criteria that determine the choice of surfaces representation include accuracy, conciseness, local support, affine invariance and guaranteed levels of parametric on geometric continuity, to name a few. Generally, surface representations for scattered data sets can be divided into two categories, explicit or implicit. Explicit surface representation includes parametric surfaces, subdivision surfaces and triangulated surfaces (Cohen et al., 1980; Pratt, 1985; Hoppe et al., 1994; Hagen et al., 1996).

Generally, parametric surfaces can be represented in terms of two parameters u and v. Boundary representation of such a surface consists of three parametric functions, $x = x(u, v)$, $y = y(u, v)$ and $z = z(u, v)$. Examples of parametric surfaces are Bézier surfaces (Lodha and Warren (1990) and Farin (2002b, Chap. 14 - 17)), B-splines surfaces (Milroy et al., 1995; Eck and Hoppe, 1996; Shi et al., 2004) and nonuniform rational B-spline surfaces (NURBS) (Piegl, 1991; Ma and Kruth, 1998; Leal et al., 2010; Brujic et al., 2010). Parametric surfaces have the advantage of locating points correctly on a surface. It is also possible to describe complex shape by partitioning a surface into piecewise parametric surface patches. Figure 1.2 shows an example of a parametric patch.

Bézier surfaces were independently developed and applied in car manufacturing industry in the late 1950s by P. Bézier and J. de Casteljau (Barnhill, 1985). Such a surface can be defined in a generic form of

$$P(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j} B_{i,m}(u) B_{j,n}(v) \tag{1.1}$$

where $P_{i,j}$ are an $(m+1) \times (n+1)$ array of control points. $B_{i,m}(u)$ and $B_{j,n}(v)$ represent the Bernstein polynomials of degrees m and n in the variables u and v, respectively. It is clear that each term is obtained from a control point and the product of two univariate

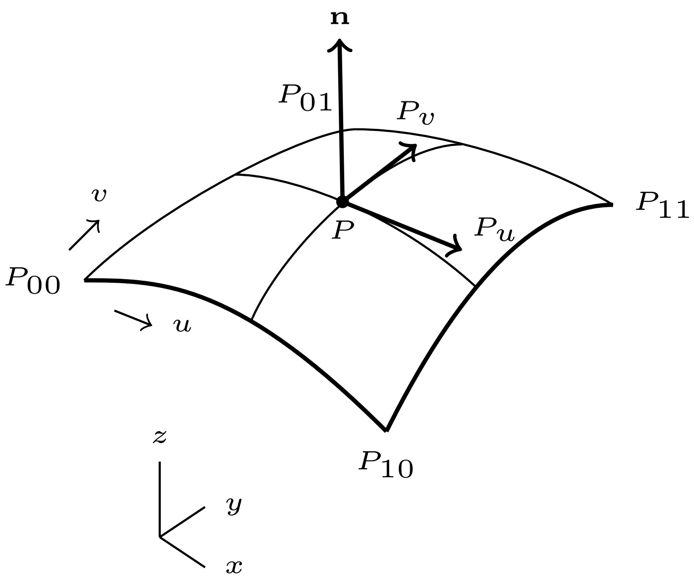


Figure 1.1: Parametric surface patch.

Bernstein polynomials. Therefore each product forms a basis function of the surface. A Bézier surface does not pass through the control points in general except for the boundary or end control points. It remains within the convex hull of the control points.

Instead of Bernstein polynomials, B-splines surfaces uses piecewise polynomials as their basis functions defined over a knot vector. Generally, a B-spline surface P of order k in the u direction and order l in the v direction is a bivariate vector-valued piecewise function of the form

$$P(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} N_{i,k}(u) N_{j,l}(v) \tag{1.2}$$

where the $P_{i,j}$ form the control net and $N_{i,k}(u)$ and $N_{j,l}(v)$ are the normalised B-spline functions defined on the knot vectors:

$$U = \{u_1, u_2, \ldots u_{m+k}\} \quad \text{and} \quad V = \{v_1, v_2, \ldots v_{n+l}\}.$$

In the case of uniform B-splines, the knots are equidistant. If the knots are not equidistant, non-uniform splines are used. NURBS has become a popular choice of surface form in the industry nowadays (Farin, 2002a). Using rational polynomials as their basis functions, a NURBS surface patch can be defined as:

$$P(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} w_{i,j} N_{i,k}(u) N_{j,l}(v)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{i,j} N_{i,k}(u) N_{j,l}(v)} \tag{1.3}$$

where the $P_{i,j}$, $N_{i,k}(u)$ and $N_{j,l}(v)$ are as given previously and $w_{i,j}$ are the weights. More theoretical details on NURBS can be found in Farin (2002b, Chap. 17).

There exist a number of algorithms to reconstruct a parametric surface. See Chivate and Jablokow (1995), Vida et al. (1994), Lim and Haron (2012) and references therein. Most of these involve techniques such as parametrization, optimization and least-squares approach (Liu and Wang, 2008; Weiss et al., 2002). Efficiency of these algorithms are guaranteed when the data to be fitted is gridded (Piegl, 1991). If the data is scattered, sometimes a suitable fitting algorithm can be applied after converting scattered data to gridded data. The major drawback with parametric surfaces is extrapolation. It is sometimes desirable to be able to extend the surfaces outside the boundary of the data points. Another important issue is the smoothness (Du and Schmitt, 1990; Ma and Peng, 1995) of the surface. The term related to the smoothness of a surface is called geometric continuity (Farin, 2002b, Chap 11.). A surface can be defined as having G^n continuity where n is the order of smoothness. For example G^1 continuity means the tangent plane varies continuously with position on the surface. In particular, the tangent planes from two adjacent patches must be coplanar along their join. Assembling the patches together with certain level of continuity is a difficult task (Vida et al., 1994). Liu and Hoschek (1989) reported sufficient conditions for G^1 continuity in blending different types of parametric patches together such as rectangular and triangular patches. In some cases, even high order smoothness (parametric as well as geometric continuity) is required, see Hartmann (2001).

As we are interested in scattered data sets, it is reasonable to use implicit rather than explicit representation for the reconstructed surface. An implicit description of a surface is an equation satisfied by all points in the surface. In other words, an implicit surface can be defined by a continuous scalar-valued function over the domain $\mathbb{R}^3$. It is an infinite surface and not bounded by the bounding box of the data points. The advantage of this is no additional work is required for computing intersections. Implicit representations are well known to have topological flexibility. It is also very efficient in checking if a point is on, inside or outside an implicit surface.

Substantial techniques (Bloomenthal et al., 1997; Zhao et al., 2000; Tobor et al., 2004b; Ye et al., 2010) have been proposed to reconstruct an implicit surface from scattered data (these techniques are sometimes called meshless methods). In the last decade, methods based on Radial Basis Functions (RBFs) for fitting a surface to unorganized points are relatively popular in CAGD (Savchenko et al., 1995; Turk and O'Brien, 2002; Carr et al., 2003; Morse et al., 2005). A surface that is reconstructed using RBFs is globally smooth. Much theory has been developed regarding RBF approximation. A particularly important property is the poisedness of interpolation problems. That is the associated system of linear equations is invertible under very weak conditions as detailed in Micchelli (1986) and Cheney and Light (2000).

Let us now define the surface reconstruction problem as a scattered data interpolation problem. Given a set of distinct data points $x = \{x_1, x_2, \cdots x_N\} \in \mathbb{R}^d$ and a set of function values $f = (f_1, f_2, \cdots f_N)$, we intend to find an interpolant function s such that

$$s(x_i) = f_i, \quad 1 \leq i \leq N. \tag{1.4}$$

Using appropriate RBFs, we can write the interpolation function in this form:

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \, \phi\big(|x - x_i|\big) \tag{1.5}$$

where p is a polynomial of low degree and the basic function ϕ is a real valued function on $[0, \infty)$. The λ_i's and the x_i's are the weights and centers of RBFs. Micchelli (1986) has shown that the above interpolation system is invertible whenever $\phi\big(|x - x_i|\big)$ is strictly conditionally positive definite of some order k and $\{x_i\}$ is unisolvent with respect to the polynomials of total degree at most $k-1$. Below is a treatment based on Beatson et al. (2001b), showing that the interpolation system has a unique solution.

We consider the multivariate interpolation problem as: Given m points in $\mathbb{R}^d$ and a function $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, find a function of the form

$$s(x) = p(x) + \sum_{i=1}^{m} \lambda_i \Phi(x, x_i) \tag{1.6}$$

where $p(x) \in \pi_{k-1}^d$, the space of polynomials of total degree $k-1$ in d variables, taking specified values at the points x_i. In $\mathbb{R}^d$ we usually take $\Phi(x, y) = \phi(|x - y|)$. Let us now give the definition of a strictly conditionally positive definite function.

Definition 1.2.1. *A symmetric function $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is strictly conditionally positive definite of order k on $\mathbb{R}^d$ if for all sets $X = \{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^d$ of distinct* *points, and all vectors* $\lambda \in \mathbb{R}^m$ *satisfying the orthogonality conditions*

$$\sum_{i=1}^{m} \lambda_i q(x_i) = 0 \tag{1.7}$$

for all $q \in \pi_{k-1}^d$*, the quadratic form*

$$\lambda^T \mathsf{A} \lambda = \sum_{i,j=1}^{m} \lambda_i \lambda_j \Phi(x_i, x_j),$$

is positive whenever $\lambda \neq 0$*.*

A set of distinct points $X = \{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^d$ is said to be unisolvent for polynomials of degree $k-1$ (order k) if the only polynomial in π_{k-1}^d that has the value zero at all the points is the zero polynomial.

Lemma 1.2.2. *Let* Φ *be a strictly conditionally positive definite function of order* k *and* $X = \{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^d$ *be unisolvent for polynomials of degree* $k-1$*. Then given any* m *values* $(f_1, f_2, \ldots, f_m)$ *there is a unique function of the form* (1.6) *satisfying the orthogonality conditions* (1.7) *and the interpolation conditions*

$$s(x_i) = f_i \qquad 1 \leq i \leq m. \tag{1.8}$$

Proof. The statement of the lemma is equivalent to saying the system

$$\begin{bmatrix} \mathsf{A} & P \\ P^T & \mathsf{0} \end{bmatrix} \begin{bmatrix} \lambda \\ c \end{bmatrix} = \begin{bmatrix} f \\ \mathsf{0} \end{bmatrix} \tag{1.9}$$

has a unique solution, where

$$\mathsf{A}_{i,j} = \Phi(x_i, x_j), \qquad i, j = 1, 2, \ldots, m$$

and

$$P_{i,j} = p_j(x_i), \qquad i = 1, 2, \ldots, m, j = 1, 2, \ldots, \ell,$$

where $\{p_j\}_{j=1}^{\ell}$ is some basis for π_{k-1}^d and c are scalars $c_1, c_2, \ldots c_\ell$. Now let Q be an $m \times (m - \ell)$ matrix whose columns span the orthogonal complement of the column space of P. This implies in particular that Q has full rank. Then any $\lambda \in \mathbb{R}^m$ satisfying $P^T \lambda = 0$ can be uniquely expressed in the form $\lambda = Q\gamma$ where $\gamma \in \mathbb{R}^{m-\ell}$. Thus when we write $\lambda = Q\gamma$ the constraints on λ are automatically satisfied and we only need to solve the unconstrained system

$$\mathsf{A}(Q\gamma) + Pc = f.$$

Premultiplying by Q^T gives

$$(Q^T \mathsf{A} Q)\gamma = Q^T f \tag{1.10}$$

(note we have used $Q^T P = 0$). Observe that for any $u \in \mathbb{R}^{m-\ell}$ with $u \neq 0$,

$$u^T Q^T \mathsf{A} Q u = (Qu)^T \mathsf{A}(Qu) > 0$$

since Q has full rank and Φ is strictly conditionally positive definite of order k. Hence $Q^T \mathsf{A} Q$ is strictly positive definite symmetric matrix. Therefore Equation (1.10) uniquely specifies γ and $\lambda = Q\gamma$. Note also that Equation (1.10) can be rewritten as

$$0 = Q^T(f - \mathsf{A}Q\gamma) = Q^T(f - \mathsf{A}\lambda).$$

Hence $f - \mathsf{A}\lambda$ is in the column space of P and there is a unique c such that

$$Pc = f - \mathsf{A}\lambda.$$

Thus there is a unique interpolant with parameters $\lambda = Q\gamma$ and c as was to be proven. □

Despite all the favourable characteristics of RBFs, one drawback is that they have been computationally expensive to evaluate and fit, and therefore can appear infeasible for large scale (more than a few tens of thousands of data points) computation. To remedy this problem, the use of fast multipole methods and other hierarchical schemes allow fast evaluation and fitting of RBFs (see Beatson and Greengard (1997)). Mathematical analysis and techniques to increase computational efficiency for fast multipole related methods were reported in Beatson and Newsam (1992); Beatson et al. (1999); Cherrie et al. (2002); Beatson et al. (2001a).

1.3 Themes of Thesis

This thesis emphasises on two issues in surface reconstruction. The first issue that we look into is to develop efficient methods to compute and manipulate these approximations. In particular, we analyze and present a faster evaluation scheme for thin-plate splines in two dimensions. The new feature of the method is that exponential approximations are used as an intermediate step in obtaining local polynomial approximations.

Second, we intend to identify approximation methods that work well for certain applications. In regards to this, a suitable method for reconstructing smooth objects is proposed. The two stage process involves fitting blobby primitives to a set of points sampled from a surface and blending them together to obtain an implicit representation describing the surface of the object. Besides, we also investigated a method to reconstruct surfaces with known faults or barriers. This idea is motivated by geophysical phenomena such as fault lines on earth surface. We incorporated techniques from computational geometry such as the visibility graph, and methods for solving the shortest distance problem in our method. Both methods that we propose have been applied successfully to test cases.

The outline of this thesis is as follows. The first chapter in the thesis illustrates a new fast evaluation scheme to efficiently calculate thin-plate splines in two dimensions. We have adopted the idea by Hrycak and Rokhlin (1998) in using exponential approximation as an intermediate representation. In the fast multipole method scheme, exponential expansions/approximations are used as an intermediate step in converting far field series to local polynomial approximations. The contributions here are extending the scheme to the thin-plate spline and a new error analysis. The error analysis covers the practically important case where truncated series are used throughout, and through off line computation of error constants gives sharp error bounds.

In Chapter $3-5$, we present a method to reconstruct a surface of an object using blobby models as a coarse level approximation. The method is rather simple and easy to apply. It involves optimization and a blending process to give the final implicit surface. Both reconstruction of a blobby curve in two dimensions and a blobby surface in three dimensions are reported. We also give the corresponding reconstruction algorithms in two schemes, one being user-initiated and the other being fully automated.

Chapter $6-8$ describe a method in fitting a surface to data points with known discontinuities. We performed a scattered data interpolation using compactly supported RBFs with respect to a geodesic distance. A visibility graph based technique is used to compute the shortest Euclidean distance between two points, avoiding any obstacles. Results have shown that discontinuities on the surface were clearly reconstructed, and the "across fault" influence is suitably small.

Chapter 2

Faster Fast Evaluation

In this chapter, we present a new method for fast evaluation of thin-plate splines in two dimensions.

2.1 Introduction

Radial basis functions (RBFs) on $\mathbb{R}^d$ are functions of the form

$$s(\cdot) = p(\cdot) + \sum_{i=1}^{N} \lambda_i \Phi(\cdot - x_i). \tag{2.1}$$

Here, Φ, the basic function, is a fixed function mapping $\mathbb{R}^d \to \mathbb{R}$, and p is a low degree polynomial. The points $\{x_i\}$ are called the centers. Initially these functions were studied for their beautiful theory. More recently they have become a standard tool in many applications. Examples are surface reconstruction, uses in medicine such as in the custom manufacture of artificial limbs and of cranial protheses, use in modelling acquifers, and use in mineral exploration software such as the Leapfrog package. For all these applications there is one immediate drawback in the form given in equation (2.1). Namely, algorithms not taking advantage of the special properties of a particular Φ will require $\mathcal{O}(N)$ arithmetic operations for evaluation of s at a single extra point x. Further, non–specialised methods for fitting such an RBF s to N pieces of data, for example by interpolation, will require $\mathcal{O}(N^3)$ arithmetic operations. Fortunately, the use of fast algorithms, see for example Beatson and Newsam (1992), or of compactly supported kernels, see Wendland (2005), reduces the incremental cost of a single extra evaluation dramatically, for example to $\mathcal{O}(1)$ operations, ignoring the cost of finding the panel containing the evaluation point. Also, numerical experiments show that iterative fitting methods employing these fast algorithms to compute matrix–vector products, typically solve interpolation problems with N nodes in $\mathcal{O}(N(\log N)^2)$ operations, rather than $\mathcal{O}(N^3)$. In this work, we present an improved fast evaluation method for thin-plate splines in two dimensions. The new feature of the method is that exponential approximations are used as an intermediate step in obtaining local polynomial approximations. This intermediate exponential approximation idea was first used by Hrycak and Rokhlin (1998) in their method for fast evaluation of two dimensional potentials.

In this chapter, intermediate exponential approximations will be used in a fast multipole method for the thin-plate spline. The error analysis given is new and differs from that in Hrycak and Rokhlin (1998).

2.2 Hrycak and Rokhlin's improved fast multipole method

The fast multipole method (FMM) is listed as one of the top ten algorithms of the century (Dongarra and Sullivan, 2000). It is developed for achieving fast products of large-scale matrices with vectors. Hrycak and Rokhlin (1998) developed a fast multipole method for potentials $\dfrac{1}{z-t}$ employing, as an intermediate representation, exponential approximations of the form

$$\frac{1}{z-t} \approx \sum_{\ell=1}^{L} w_\ell \, e^{-x_\ell (z-t)}, \qquad z, t \in \mathbb{C}, \tag{2.2}$$

where w_ℓ and x_ℓ are specified coefficients, $t \in Q$, $z \in R_0$, and Q and R_0 are sets specified as follows. Figure 2.2 shows the partitioning used when the panel size h equals 1. In the diagram, Q is the region $[\frac{-1}{2}, \frac{1}{2}] \times [\frac{-1}{2}, \frac{1}{2}]$, and R_0 is the region consisting of the rectangle $[\frac{3}{2}, \frac{7}{2}] \times [-\frac{7}{2}, \frac{7}{2}]$ minus the squares $[\frac{3}{2}, \frac{5}{2}] \times [\frac{5}{2}, \frac{7}{2}]$ and $[\frac{3}{2}, \frac{5}{2}] \times [-\frac{7}{2}, -\frac{5}{2}]$. Region R_r is obtained from region R_0 by rotating anti-clockwise by $r\pi/2$ radians.

In this case, most translation operators are diagonal. This is different from previous implementation of FMM using Laurent (multipole) and Taylor expansions for the potential field (Greengard and Rokhlin, 1987). The exponential approximations give impressively high accuracy for the number of terms L. The numerical tool for finding the exponential approximation formulas is using finite quadratures for integrals of the form

$$\frac{1}{z-t} = \int_0^\infty e^{-x(z-t)} dx,$$

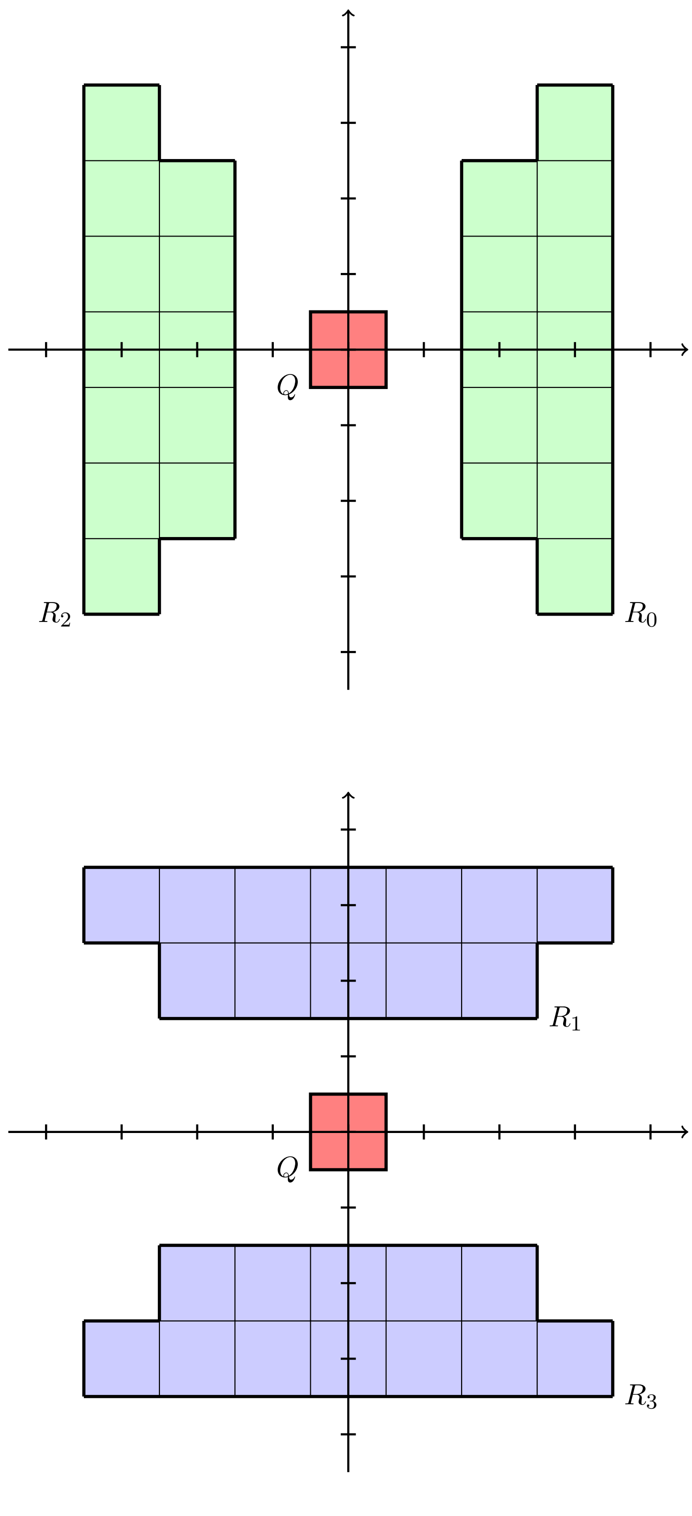


Figure 2.1: The domains Q, R_0, R_1, R_2 and R_3

developed by Yarvin and Rokhlin (1998). The integrals were approximated by expression of the form

$$\int_0^\infty e^{-x(z-t)}dx \approx \sum_{\ell=1}^{L} w_\ell\, e^{-x_\ell(z-t)},$$

where w_ℓ and x_ℓ are chosen to give low approximation error. In our work, the potential $\dfrac{1}{z-t}$ is considered as an initial step to obtain approximations for the thin plate spline because the exponential approximation to $\dfrac{1}{z-t}$ leads to the exponential approximations to $\mathrm{Log}(z-t)$. Further details will be given in Section 2.4.

In algorithm 1 below, we briefly outline a generic fast multipole method which consists of two stages: a setup stage and an evaluation stage. The reader who is not familiar with these algorithms will find more detailed descriptions in Beatson and Newsam (1992); Beatson and Greengard (1997).

Algorithm 1 A fast multipole procedure for evaluation of $s(x) = \sum_{i=1}^{N} \lambda_i \Phi(x - x_i)$

SETUP: Given the centers and weights $\{(x_i, \lambda_i)\}$

- Step 1: Construct a tree which at each new level creates child panels by splitting parent panels, assigning the centers x_i belonging to the parent to the appropriate children. In $\mathbb{R}^2$ the tree is often chosen to be a quadtree (see Figure 2.2).
- Step 2: Starting at the leaf nodes work up the tree constructing for each panel, $\mathcal{T}$, a truncated far field expansion which approximates the influence of the panel, at points far from the panel. In many cases it is efficient to calculate the truncated far field expansion of a parent panel indirectly by merging the expansions of its children, rather than directly from the centers an weights $\{x_i, \lambda_i)\}$.
- Step 3: Work down the tree level by level (see Figure 2.3). At level ℓ and working on panel $\mathcal{T}$, first recenter the polynomial approximation associated with the parent of $\mathcal{T}$, and then add to this polynomials obtained by approximating far field approximations impacting on panel $\mathcal{T}$. These far field approximations arise from the up to 27 well separated panels on the interaction list of $\mathcal{T}$. See Figure 2.4.

EVALUATION: Given a point x at which to evaluate $s(x)$.

- Step 1: Find the panel $\mathcal{T}$ containing x.
- Step 2: Evaluate $s(x)$ by using the local polynomial approximation summarising the influence of all the panels far away from $\mathcal{T}$ to approximate the far field, and evaluating the near field directly. See Figure 2.4 for illustration of far field and near field panels.

In the case of the potential $\sum_{i=1}^{N} \lambda_i/(z - z_i)$, Hrycak and Rokhlin (1998) use ex-

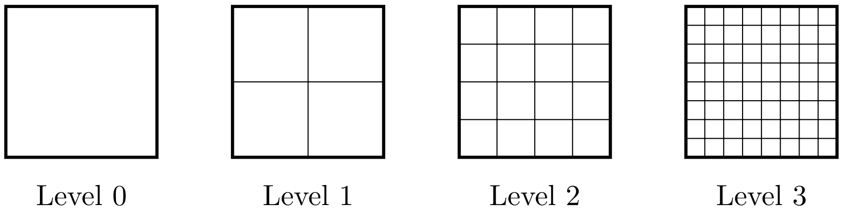


Figure 2.2: Example of a quadtree in two dimensions. Taking a square as an initial parent, repeatedly subdivide each parent panel into four equally sized child panels.

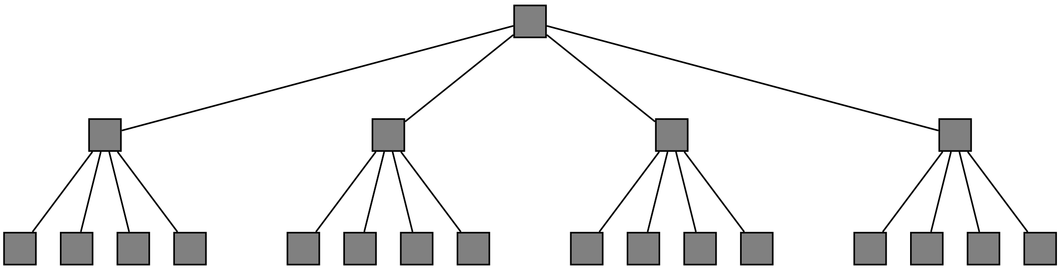


Figure 2.3: Tree of panels: each node at level ℓ refers to a square panel of area $4^{-\ell}$.

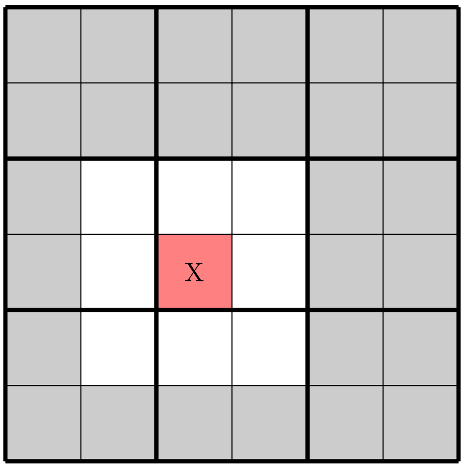


Figure 2.4: The interaction list of panel $\mathcal{T}$. The interaction list of $\mathcal{T}$ is all panels at the same level, whose influence has not been accounted for at a coarser level. Let $\mathcal{T}$ (red) be the panel containing x. Boxes in grey are considered far away (or well separated) from $\mathcal{T}$ and boxes in white are near neighbors of $\mathcal{T}$.

ponential approximations to improve step 3 of the setup stage of Algorithm 1. They do this by first converting far field approximations to exponential approximations, and then approximating these exponential approximations by local Taylor polynomials. The procedure can be anticipated to be an improvement for at least three reasons. First, the exponential approximations have relatively few terms for the given accuracy. Second, recentering exponentials $e^{-x_\ell z}$ to the center, t, of a target panel is a 'diagonal' operation, since $e^{-x_\ell z} = e^{-x_\ell t} e^{-x_\ell (z-t)}$. Third, Taylor series approximations about t to the function of exponential type $\sum_{\ell=1}^{L} a_\ell \, e^{-x_\ell (z-t)}$ will converge very much faster than Taylor series approximations about t to the Laurent expansions $\sum_{k=1}^{K} b_k \, z^{-k}$. The singularity in the Laurent expansion implies the radius of convergence of the Taylor series is $|t|$. The convergence or divergence of the Taylor series at various radii leads to the following conclusions. The coefficients $\{c_n\}$ of the Taylor series of the Laurent expansion will decay as $\mathcal{O}\big((|t| - \epsilon)^{-n}\big)$ but not as $\mathcal{O}\big((|t| + \epsilon)^{-n}\big)$ for all small positive ϵ. More precisely,

$$\limsup_{n\to\infty} \frac{1}{\sqrt[n]{|c_n|}} = |t|.$$

On the other hand, the coefficients $\{d_n\}$ of the Taylor series of the entire function will decay as $\mathcal{O}(R^{-n})$ for every $R > 0$. More precisely, Boas (Boas, 1954, Theorem 2.2.2) shows that the coefficients $\{d_n\}$ of the Taylor series of any non trivial combination of exponentials will decay so that

$$\limsup_{n\to\infty} \frac{n \log n}{-\log(|d_n|)} = 1.$$

2.3 Laurent-like expansion of a thin-plate spline

In this section we discuss the mathematics underlying the formulation of the far field series for a panel, as needed in step 2 of Algorithm 1.

We define the multipole coefficients as

$$a_k = \frac{1}{k!} \sum_{j=1}^{J} \lambda_j z_j^k \quad \text{and} \quad b_k = \frac{1}{k!} \sum_{j=1}^{J} \lambda_j \bar{z}_j z_j^k \quad \text{where} \quad k \geq 0. \tag{2.3}$$

Write A_k, B_k, α, β and γ for the multipole coefficients defined in the multipole expansion of the biharmonic given by Beatson and Newsam (1992). Then the relationships between the coefficients there and those here are (assuming all the weights λ_j are real),

$$a_0 = \alpha, \quad a_1 = \beta, \quad b_0 = \overline{\beta} = \overline{a_1}, \quad b_1 = \gamma = B_0, \tag{2.4}$$

$$a_{k+1} = \begin{cases} -A_0, & k = 0, \\ \dfrac{1}{(k-1)!} A_k, & k > 0, \end{cases} \tag{2.5}$$

and

$$b_{k+1} = \begin{cases} B_0, & k = 0, \\ -\dfrac{1}{(k-1)!} B_k, & k > 0. \end{cases} \tag{2.6}$$

The change in notation here will simplify certain formulas in the exponential expansion sections 2.4 and 2.5. The approximation of a thin-plate spline by a far field / Laurent series is described by

Lemma 2.3.1. *(Beatson and Newsam, 1992, Lemma 2) Suppose J centers $z_j, j = 1, \cdots, J$, with $|z_j| \leq r$ and J corresponding real coefficients λ_j are given. Then, whenever $|z| > r$ the thin-plate spline associated with this cluster of centers,*

$$s(z) = \sum_{j=1}^{J} \lambda_j \, \phi_j(z) = \sum_{j=1}^{J} \lambda_j |z - z_j|^2 \log|z - z_j|, \tag{2.7}$$

is alternatively given by the expansion

$$s(z) = \Big[a_0|z|^2 - 2\,\Re e\big\{\bar{a}_1 z\big\} + b_1\Big] \log|z| + \Re e\left\{\sum_{k=0}^{\infty} \eta(k)(a_{k+1}\,\bar{z} - b_{k+1}) z^{-k}\right\}, \tag{2.8}$$

where

$$a_k = \frac{1}{k!}\sum_{j=1}^{J} \lambda_j \, z_j^k\,, \qquad b_k = \frac{1}{k!}\sum_{j=1}^{J} \lambda_j \, \bar{z}_j \, z_j^k\,, \quad k \geq 0, \tag{2.9}$$

and

$$\eta(k) = \begin{cases} -1, & k = 0, \\ (k-1)\,!\,, & k \geq 1. \end{cases} \tag{2.10}$$

Furthermore, defining

$$M = \sum_{j=1}^{J} |\lambda_j| \quad \textit{and} \quad c = \left|\frac{z}{r}\right|, \tag{2.11}$$

then for any $K \geq 1$ and $|z| > r$,

$$\left| s(z) - \left(\Big[a_0|z|^2 - 2\,\Re e\big\{\bar{a}_1 z\big\} + b_1\Big] \log|z| + \Re e\left\{\sum_{k=0}^{K} \eta(k)(a_{k+1}\,\bar{z} - b_{k+1}) z^{-k}\right\}\right)\right|$$
$$\leq \frac{Mr^2}{(K+1)(K+2)} \left(\frac{c+1}{c-1}\right)\left(\frac{1}{c}\right)^{K}. \tag{2.12}$$

The following lemma enables the shifting of the center of a Laurent-like series approximation to a thin-plate spline. The lemma enables calculation of the multipole coefficents corresponding to a panel indirectly from those of its children, rather than directly from its centers and weights. This can lead to a considerable saving of compute time when short truncated series summarise the influence of panels containing many thousands of centers.

Lemma 2.3.2. *(Beatson and Newsam, 1992, Lemma 3) Let*

$$\begin{aligned} s(z) &= \Big[a_0|z-t|^2 - 2\,\Re e\big\{\bar{a}_1(z-t)\big\} + b_1\Big]\log|z-t| \\ &\quad + \Re e\left\{\sum_{k=0}^{\infty}\eta(k)\Big[a_{k+1}\,(\bar{z}-\bar{t}) - b_{k+1}\Big](z-t)^{-k}\right\}, \end{aligned} \tag{2.13}$$

be the Laurent-like expansion about t of the thin-plate spline, s, associated with J centers z_j and corresponding real coefficients λ_j. Suppose that all the centers are located in a disk radius R about t. Then for all $|z| > R + |t|$,

$$s(z) = \Big[\widehat{a}_0|z|^2 - 2\,\Re e\big\{\overline{\widehat{a}}_1 z\big\} + \widehat{b}_1\Big]\log|z| + \Re e\left\{\sum_{k=0}^{\infty}\eta(k)(\widehat{a}_{k+1}\,\bar{z} - \widehat{b}_{k+1})z^{-k}\right\}, \tag{2.14}$$

where

$$\widehat{a}_k = \begin{cases} a_0\,, & k=0,\\ a_0\,t + a_1\,, & k=1,\\ \dfrac{\widehat{a}_1\,t^{k-1}}{(k-1)!} - \dfrac{a_0\,t^k}{(k-2)!k} + \displaystyle\sum_{j=1}^{k-1}\dfrac{a_{j+1}\,t^{k-j-1}}{(k-j-1)!}, & k \geq 2. \end{cases} \tag{2.15}$$

and

$$\widehat{b}_k = \begin{cases} a_0\bar{t} + b_0\,, & k=0,\\ a_0|t|^2 + 2\,\Re e\big\{b_0\,t\big\} + b_1, & k=1,\\ -\dfrac{\widehat{b}_0\,t^k}{(k-2)!k} + \dfrac{\widehat{b}_1\,t^{k-1}}{(k-1)!} + \displaystyle\sum_{j=1}^{k-1}\dfrac{\big(\bar{t}\,a_{j+1} + b_{j+1}\big)\,t^{k-j-1}}{(k-j-1)!}, & k \geq 2. \end{cases} \tag{2.16}$$

Furthermore, the coefficients β, γ, $\{\widehat{a}_k\}$ and $\{\widehat{b}_k\}$ of the shifted expansion are the same as those obtained if the thin-plate spline is directly expanded about 0 *using lemma* (2.3.1).

Hence, the error bound

$$\left| s(z) - \left(\Big[a_0 |z|^2 - 2\,\mathfrak{Re}\big\{\bar{\beta} z\big\} + \gamma \Big] \log|z| + \mathfrak{Re}\left\{ \sum_{k=0}^{K} \eta(k) (\widehat{a}_{k+1}\,\bar{z} - \widehat{b}_{k+1}) z^{-k} \right\} \right) \right|$$
$$\leq \frac{Mu^2}{(K+1)(K+2)} \left(\frac{c+1}{c-1}\right) \left(\frac{1}{c}\right)^K . \qquad (2.17)$$

holds where

$$u = \max_{1 \leq i \leq J} |z_i| \leq R + |t| \qquad \textit{and} \qquad c = \left|\frac{z}{u}\right| \qquad (2.18)$$

Proof. As in the statement of the lemma any coefficient with a $\widehat{\ }$ on top, is a coefficient of the shifted expansion. This lemma follows from translating the formulas given in Beatson and Newsam (1992, Lemma 3), to be in terms of the new multipole coefficients introduced at the beginning of this section.

$$\begin{aligned}
\widehat{\alpha} = \alpha \quad &\Longrightarrow \quad \widehat{a}_0 = a_0 \\
\widehat{\beta} = \alpha t + \beta \quad &\Longrightarrow \quad \widehat{a}_1 = a_0 t + a_1 \\
\widehat{\overline{\beta}} = \alpha \bar{t} + \overline{\beta} \quad &\Longrightarrow \quad \widehat{b}_0 = a_0 \bar{t} + b_0 \\
\widehat{A}_0 = -\alpha t + A_0 \quad &\Longrightarrow \quad \widehat{a}_1 = a_0 t + a_1 \\
\widehat{\gamma} = \gamma + 2\mathfrak{Re}\{\overline{\beta} t\} + \alpha|t|^2 \quad &\Longrightarrow \quad \widehat{b}_1 = b_1 + 2\mathfrak{Re}\{b_0\, t\} + a_0 |t|^2 .
\end{aligned}$$

Also

$$\widehat{A}_k = \frac{\widehat{\beta} t^k}{k} - \frac{\alpha t^{k+1}}{k+1} + \sum_{j=1}^{k} A_j \binom{k-1}{j-1} t^{k-j}, \quad k \geq 1,$$

implies

$$\begin{aligned}
(k-1)!\,\widehat{a}_{k+1} \quad &= \quad \frac{\widehat{a}_1 t^k}{k} - \frac{a_0 t^{k+1}}{k+1} + \sum_{j=1}^{k} (j-1)!\, a_{j+1} \binom{k-1}{j-1} t^{k-j} \\
\Longrightarrow\ \widehat{a}_{k+1} \quad &= \quad \frac{\widehat{a}_1 t^k}{k!} - \frac{a_0 t^{k+1}}{(k-1)!\,(k+1)} + \sum_{j=1}^{k} \frac{a_{j+1} t^{k-j}}{(k-j)!}, \quad k \geq 1, \\
\Longrightarrow\ \widehat{a}_k \quad &= \quad \frac{\widehat{a}_1 t^{k-1}}{(k-1)!} - \frac{a_0 t^k}{(k-2)!\,k} + \sum_{j=1}^{k-1} \frac{a_{j+1} t^{k-j-1}}{(k-j-1)!}, \quad k \geq 2.
\end{aligned}$$

Finally,

$$\widehat{B}_k = \frac{\widehat{\overline{\beta}} t^{k+1}}{k+1} - \frac{\widehat{\gamma} t^k}{k} + \sum_{j=1}^{k} \left(B_j - A_j \bar{t}\right) \binom{k-1}{j-1} t^{k-j}, \quad k \geq 1,$$

implies

$$
\begin{aligned}
-(k-1)!\,\widehat{b}_{k+1} &= \frac{\widehat{b}_0 t^{k+1}}{k+1} - \frac{\widehat{b}_1 t^k}{k} + \sum_{j=1}^{k} (j-1)!\,(b_{j+1} + a_{j+1}\overline{t}) \binom{k-1}{j-1} t^{k-j} \\
\Longrightarrow \widehat{b}_{k+1} &= \frac{\widehat{b}_1 t^k}{k!} - \frac{\widehat{b}_0 t^{k+1}}{(k-1)!\,(k+1)} + \sum_{j=1}^{k} \frac{\left(b_{j+1} + a_{j+1}\overline{t}\right)\, t^{k-j}}{(k-j)!}, \quad k \geq 1, \\
\Longrightarrow \widehat{b}_k &= \frac{\widehat{b}_1 t^{k-1}}{(k-1)!} - \frac{\widehat{b}_0 t^k}{(k-2)!\,k} + \sum_{j=1}^{k-1} \frac{\left(b_{j+1} + a_{j+1}\,\overline{t}\right)\, t^{k-j-1}}{(k-j-1)!}, \quad k \geq 2.
\end{aligned}
$$

□

2.4 Exponential approximation of thin-plate splines

This section develops an exponential approximation of thin-plate splines $\sum_{j=1}^{J} \lambda_j |z - z_j|^2 \log|z - z_j|$ based upon the 'quadrature formulas' for approximation of potentials the form $\displaystyle\sum_{j=1}^{J} \frac{\lambda_j}{z - z_j}$, given by Hrycak and Rokhlin (1998).

Hrycak and Rokhlin (1998) give approximation formulas of the form

$$
\frac{1}{z-t} \approx \sum_{\ell=1}^{L} w_\ell\, e^{x_\ell (z-t)}, \quad t \in Q, \quad z \in R_0, \tag{2.19}
$$

which have relatively high accuracy for the number of nodes L. The quadrature nodes and weights are listed in Hrycak and Rokhlin (1998) and are available at the URL `http://www.netlib.org/pdes/multipole/pwts4.f`. The following table summarizes the performance of these quadratures such that

$$
\left| \frac{1}{z-t} - \sum_{\ell=1}^{L} w_\ell\, e^{x_\ell (z-t)} \right| < \epsilon \tag{2.20}
$$

for all $t \in Q,\ z \in R_0$.

L	8	10	16	33
ϵ	10^{-3}	10^{-4}	10^{-7}	10^{-15}

Table 2.1: Performance of different quadrature formulas.

2.4.1 An exponential approximation of Log(z-t)

The exponential approximations to the function $1/(z-t)$ developed by Hrycak and Rokhlin (1998) leads to exponential approximations to $\mathrm{Log}(z-t)$. Let U be a simply connected region within $\mathbb{C}\backslash(-\infty,0]$. Then $\mathrm{Log}(u) = \int_1^u \frac{1}{v}\,dv$, where Log is the principal branch of the complex logarithm, and the path integral is over a path not touching the branch cut. Suppose that the quadrature formula (2.19) has absolute error bounded above by ϵ when $t \in Q$ and $z \in R_0$. Then

$$\mathrm{Log}(u) = \int_1^u \frac{1}{v}\,dv \approx \int_1^u \sum_{\ell=1}^{L} w_\ell\, e^{-x_\ell v}\,dv = \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell\, u} \tag{2.21}$$

and

$$\left|\mathrm{Log}(u) - \left\{\sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell\, u}\right\}\right| = \left|\int_1^u \frac{1}{v} - \sum_{\ell=1}^{L} w_\ell\, e^{-x_\ell v}\,dv\right| \le \epsilon\mathcal{L}, \tag{2.22}$$

where $\mathcal{L}$ is the maximum length of minimal path in $R_0 - Q$ from 1 to u.

2.4.2 An exponential approximation of (y-s)Log(y-s)

Integration by parts gives us the formula

$$v\,\mathrm{Log}(v) = \int_1^v \mathrm{Log}(u)\,du + v - 1 \tag{2.23}$$

for v in the right half plane. Therefore

$$
\begin{aligned}
(y-s)\,\mathrm{Log}(y-s) &= \int_1^{y-s} \mathrm{Log}(u)\,du + (y-s) - 1 \\
&\approx \int_1^{y-s} \left(\sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell u} \right) du + (y-s) - 1 \\
&= \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell}\, u\Big|_{u=1}^{y-s} - \sum_{\ell=1}^{L} \frac{w_\ell}{-x_\ell^2}\, e^{-x_\ell u}\Big|_{u=1}^{y-s} + (y-s) - 1 \\
&= \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell}(y-s-1) + (y-s) - 1 + \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2} \left(e^{-x_\ell(y-s)} - e^{-x_\ell} \right) \\
&= \alpha\, y - \alpha\, s - \alpha + \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{x_\ell s}\, e^{-x_\ell y} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{-x_\ell} \\
&= \alpha\, y - \alpha\, s - (\alpha+\beta) + \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{x_\ell s}\, e^{-x_\ell y}, \qquad (2.24)
\end{aligned}
$$

$$
\text{where} \quad \alpha = \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell} + 1 \qquad \text{and} \qquad \beta = \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{-x_\ell}.
$$

2.4.3 Exponential approximation of biharmonic RBFs

Using the approximation for $(y-s)\,\mathrm{Log}(y-s)$ derived above, an approximation of a single thin-plate source at $t \in Q$ with weight 1, and $y \in R_0$, is

$$
\begin{aligned}
\Phi(y-t) &= |y-t|^2 \log|y-t| \\
&= \mathfrak{Re}\Big\{ (\bar{y}-\bar{t})(y-t)\,\mathrm{Log}(y-t) \Big\} \\
&\approx \mathfrak{Re}\Big\{ (\bar{y}-\bar{t}) \Big[\alpha\, y - \alpha\, t - (\alpha+\beta) + \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{x_\ell t}\, e^{-x_\ell y} \Big] \Big\} \\
&= \alpha \Big[|y|^2 + |t|^2 - 2\,\mathfrak{Re}(\bar{t}y) \Big] - (\alpha+\beta)\,\mathfrak{Re}\Big\{ \bar{y} - \bar{t} \Big\} \\
&\quad + \quad \mathfrak{Re}\Big\{ \Big[\bar{y} \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{x_\ell t} + \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell^2}\, e^{x_\ell t}(-\bar{t}) \Big] e^{-x_\ell y} \Big\} \\
&= \alpha \Big[|y|^2 + |t|^2 - 2\,\mathfrak{Re}(\bar{t}y) \Big] - (\alpha+\beta)\,\mathfrak{Re}\Big\{ \bar{y} - \bar{t} \Big\} \\
&\quad + \quad \mathfrak{Re}\Big\{ \sum_{\ell=1}^{L} \Big[\bar{y} c'_\ell - d'_\ell \Big] e^{-x_\ell y} \Big\}, \qquad (2.25)
\end{aligned}
$$

where

$$c'_\ell = \frac{w_\ell}{x_\ell^2}\, e^{x_\ell t} \quad \text{and} \quad d'_\ell = \frac{w_\ell}{x_\ell^2}\, \bar{t}\, e^{x_\ell t}\,.$$

Using (2.25), an exponential approximation of the thin-plate spline, corresponding to a cluster of sources and weights $\Big\{(z_j, \lambda_j) : 1 \leqslant j \leqslant J\Big\} \subset Q \times \mathbb{R}$, and evaluation points $z \in R_0$, is

$$\begin{aligned}
s(z) &= \sum_{j=1}^{J} \lambda_j \left|z - z_j\right|^2 \log\left|z - z_j\right| \\
&= \mathfrak{Re}\Bigg\{\sum_{j=1}^{J} \lambda_j (\bar{z} - \bar{z}_j)(z - z_j)\,\mathrm{Log}(z - z_j)\Bigg\} \\
&\approx \alpha \left[\sum_{j=1}^{J} \lambda_j |z|^2 + \sum_{j=1}^{J} \lambda_j |z_j|^2 - 2\,\mathfrak{Re}\Bigg\{\sum_{j=1}^{J} \lambda_j \bar{z}_j z\Bigg\}\right] \\
&\qquad - (\alpha+\beta)\,\mathfrak{Re}\Bigg\{\sum_{j=1}^{J} \lambda_j z - \sum_{j=1}^{J} \lambda_j z_j\Bigg\} + \mathfrak{Re}\Bigg\{\sum_{\ell=1}^{L} \Big[\bar{z} c_\ell - d_\ell\Big] e^{-x_\ell z}\Bigg\} \\
&= \mathfrak{Re}\Bigg\{\gamma_2 |z|^2 + \gamma_1 z + \gamma_0\Bigg\} + \mathfrak{Re}\Bigg\{\sum_{\ell=1}^{L} \Big[\bar{z} c_{\ell,\infty} - d_{\ell,\infty}\Big] e^{-x_\ell z}\Bigg\}, \qquad (2.26)
\end{aligned}$$

where

$$\gamma_2 = \alpha\, a_0\,, \quad \gamma_1 = -2\alpha\, b_0 - (\alpha+\beta)\, a_0 \quad \gamma_0 = \alpha\, b_1 + (\alpha+\beta)\, a_1\,, \qquad (2.27)$$

$$c_{\ell,\infty} = \frac{w_\ell}{x_\ell^2} \sum_{j=1}^{J} \lambda_j\, e^{x_\ell z_j} \qquad \text{and} \qquad d_{\ell,\infty} = \frac{w_\ell}{x_\ell^2} \sum_{j=1}^{J} \lambda_j\, \bar{z}_j\, e^{x_\ell z_j}\,. \qquad (2.28)$$

Note that

$$\sum_{j=1}^{J} \lambda_j\, e^{x_\ell\, z_j} = \sum_{j=1}^{J} \lambda_j \sum_{k=0}^{\infty} \frac{(x_\ell z_j)^k}{k!} = \sum_{k=0}^{\infty} x_\ell^k \frac{1}{k!} \sum_{j=1}^{J} \lambda_j\, z_j^k = \sum_{k=0}^{\infty} x_\ell^k\, a_k\,, \qquad (2.29a)$$

$$\sum_{j=1}^{J} \lambda_j\, \bar{z}_j\, e^{x_\ell z_j} = \sum_{j=1}^{J} \lambda_j\, \bar{z}_j \sum_{k=0}^{\infty} \frac{(x_\ell z_j)^k}{k!} = \sum_{k=0}^{\infty} x_\ell^k \frac{1}{k!} \sum_{j=1}^{J} \lambda_j\, \bar{z}_j\, z_j^k = \sum_{k=0}^{\infty} x_\ell^k\, b_k\,, \qquad (2.29b)$$

where a_k and b_k are the multipole coefficients as defined in (2.3). The expressions on the right of (2.29) are not directly applicable in a multipole code as they require the calculation of infinitely many multipole coefficents for each source panel. Therefore, we consider calculating approximations $c_{\ell,K}$ and $d_{\ell,K}$ to $c_{\ell,\infty}$ and $d_{\ell,\infty}$ by truncating the series in (2.29). We define

$$c_{\ell,K} = \frac{w_\ell}{x_\ell^2} \sum_{k=0}^{K} x_\ell^k \, a_k \approx \frac{w_\ell}{x_\ell^2} \sum_{k=0}^{\infty} x_\ell^k \, a_k = c_{\ell,\infty} \text{ and } d_{\ell,K} = \frac{w_\ell}{x_\ell^2} \sum_{k=0}^{K} x_\ell^k \, b_k \approx d_{\ell\,\infty}, \tag{2.30}$$

which leads to an exponential approximation to $s(z)$ of the form

$$\Psi_{L,K}(z) := \Re e\left\{\gamma_2|z|^2 + \gamma_1 z + \gamma_0\right\} + \Re e\left\{\sum_{\ell=1}^{L}\left[\bar{z}c_{\ell,K} - d_{\ell,K}\right]e^{-x_\ell z}\right\}. \tag{2.31}$$

2.5 Exponential approximations of thin-plate spline in scaled and rotated settings

The purpose of this section is to develop the expansions and error estimates underlying step 3 of Algorithm 1, when exponential expansions are used as an intermediate step in converting far field series to local approximations.

In step 3, at level ℓ in the tree, all the panels are squares of size $h \times h$, and the panels in the level above are all $2h \times 2h$. The first part of step 3 at level ℓ is to sweep over all the panels initializing the local polynomial approximation by shifting the polynomial approximation of its parent, for example by using the complete Horner scheme. Next, in a sweep regarding every level ℓ panel in turn as a source, and shifting the origin so that it becomes the panel hQ, exponential approximations are calculated for the regions hR_0, hR_1, hR_2 and hR_3. Finally, the exponential approximation holding in hR_r is converted to a Taylor polynomial approximation in each $h \times h$ target subpanel, $\mathcal{T}$, within hR_r. It is clear from this discussion that exponential expansions and error bounds, relevant when the source panel is hQ and the target regions are hR_0, hR_1, hR_2 and hR_3, are needed.

Fundamental to the error estimates to be given below is the maximum error in approximating over R_0, a single thin-plate source with centre $t \in Q$ and weight unity. That is

$$\epsilon_{L,K} := \max_{t\in Q} \max_{z\in R_0} \left| |z-t|^2 \log|z-t| - \Psi_{L,K,t}(z) \right|, \tag{2.32}$$

where $\Psi_{L,K,t}$ is the approximation $\Psi_{L,K}$ given in (2.31), when the cluster of sources reduces to a single source at t, with weight unity. $\epsilon_{L,\infty}$ can be bounded by analyzing the steps made in deriving the approximation in section 2.4. However, tighter estimates of $\epsilon_{L,\infty}$ can be found numerically, for example by maximising the error in approximating a single source, with t and z restricted to very fine grids.

In an implementation of this fast evaluation method considerable computational savings can be achieved by deriving the exponential expansions for a panel from the coefficients of a short truncated Laurent expansion, rather than directly from the (possibly millions of) centers and weights, $\{(z_j, \lambda_j)\}$, associated with the panel. Surprisingly, error analysis for these truncated exponential expansions is not discussed in Hrycak and Rokhlin (1998). It is important to note that in such an implementation the relevant error constants are the $\epsilon_{L,K}$, with K finite, rather than the $\epsilon_{L,\infty}$. Fortunately, the *offline computation strategy* discussed for computation of $\epsilon_{L,\infty}$, can also be applied to the practically important case of computing $\epsilon_{L,K}$ with K finite. Such computations were carried out giving the results listed in Table 2.2. The error constants in the table were calculated offline using a mesh size of 1/512. The computations were performed using the Nvidia CUDA parallel computing environment. The error constants for $L = 33$ have

L	$\epsilon_{L,L-2}$	$\epsilon_{L,L}$	$\epsilon_{L,\infty}$
8	3.5948(−4)	3.5438(−4)	3.5389(−4)
10	3.9052(−5)	3.9048(−5)	3.9048(−5)
16	1.3322(−7)	2.8925(−8)	2.8925(−8)
33	8.3176(−14)	1.5761(−14)	3.8698(−17)

Table 2.2: Error constants, $\epsilon_{L,K}$, for exponential approximation of thin-plate splines with various values of L and K.

been computed in quadrule precision. If the $L = 33$ error computations are carried out in double precision then we see numbers like 3.13(−13) due to roundoff. Error estimates for the exponential approximation of thin-plate splines in the case of scaled and rotated domains are given in the Theorem below. Note the level dependent factor h^2 in the error estimates.

Theorem 2.5.1. *Consider the thin plate spline,*

$$s(z) = \sum_{j=1}^{J} \lambda_j \left| z - z_j \right|^2 \log \left| z - z_j \right|, \tag{2.33}$$

associated with a cluster of centers $z_1, \ldots, z_J$. Suppose that $\epsilon_{L,K}$ is defined as in (2.32)*. Then for all $\{(z_j, \lambda_j) : 1 \le j \le J\} \subset hQ \times \mathbb{R}$ and $z \in hR_r$, $0 \le r \le 3$,*

$$\left| s(z) - \Psi_{L,K,h,r}(\omega) \right| \le h^2 \, \epsilon_{L,K} \sum_{j=1}^{J} |\lambda_j| \tag{2.34}$$

where $\omega = z/h$ *and the truncated series expansion of* (2.33),

$$\Psi_{L,K,h,r}(\omega) = \mathfrak{Re}\left\{\gamma_{2,h,r}|\omega|^2 + \gamma_{1,h,r}\,\omega + \gamma_{0,h,r}\right\} + \mathfrak{Re}\left\{\sum_{\ell=1}^{L}\left[\bar{\omega}\,c_{\ell,K,h,r} - d_{\ell,K,h,r}\right]e^{-x_\ell(-\mathfrak{i})^r\omega}\right\}, \tag{2.35}$$

$$c_{\ell,K,h,r} = \frac{h^2\,w_\ell}{x_\ell^2}\sum_{k=0}^{K} x_\ell^k\,(-\mathfrak{i})^{r(k-1)}\,\frac{a_k}{h^k}, \qquad d_{\ell,K,h,r} = \frac{h^2\,w_\ell}{x_\ell^2}\sum_{k=0}^{K} x_\ell^k\,(-\mathfrak{i})^{r(k-1)}\,\frac{b_k}{h^{k+1}}, \tag{2.36}$$

$$\gamma_{2,h,r} = (\alpha + \log h)\,h^2\,a_0, \qquad \gamma_{1,h,r} = -\,(\alpha+\beta)\,h^2\,(-\mathfrak{i})^r\,a_0 - 2\,(\alpha+\log h)\,h\,b_0,$$
$$\textit{and} \qquad \gamma_{0,h,r} = (\alpha+\log h)\,b_1 + (\alpha+\beta)\,h\,(-\mathfrak{i})^r\,a_1. \tag{2.37}$$

Proof. Let $\{(z_j,\lambda_j) : 1 \le j \le J\} \subset hQ \times \mathbb{R}$ and $z \in hR_r$, $0 \le r \le 3$. Define $\omega_j = z_j/h$ and $\omega = z/h$. Then, from the definition of $\epsilon_{L,K}$, and considering sources $\{((-\mathfrak{i})^r\omega_j\,,\lambda_j)\} \subset Q \times \mathbb{R}$ and an evaluation point $(-\mathfrak{i})^r\omega \in R_0$,

$$\left|\sum_{j=1}^{J}\lambda_j|\omega-\omega_j|^2\log|\omega-\omega_j| - \Psi_{L,K,r}((-\mathfrak{i})^r\omega)\right| \le \epsilon_{L,K}\sum_{j=1}^{J}|\lambda_j|, \tag{2.38}$$

where $\Psi_{L,K,r}$ is $\Psi_{L,K}$ calculated using the $(-\mathfrak{i})^r\omega_j$'s rather than the z_j's.

Note that

$$\sum_{j=1}^{J}\lambda_j|z-z_j|^2\log|z-z_j| = h^2\,\log(h)\sum_{j=1}^{J}\lambda_j|\omega-\omega_j|^2 + h^2\sum_{j=1}^{J}\lambda_j|\omega-\omega_j|^2\log|(-\mathfrak{i})^r(\omega-\omega_j)|\,. \tag{2.39}$$

Therefore, the original thin-plate spline becomes after scaling and rotating z and the z_j's, a quadratic polynomial plus h^2 times a new thin-plate spline. This new thin-plate spline has sources and evaluation point in the standard (Q, R_0), geometry. Therefore, define

$$\Psi_{L,K,h,r}(\omega) = h^2\,\log(h)\sum_{j=1}^{J}\lambda_j|\omega-\omega_j|^2 + h^2\Psi_{L,K,r}((-\mathfrak{i})^r\omega), \tag{2.40}$$

which amounts to approximating the quadratic polynomial part above exactly and the TPS in standard geometry with $\Psi_{L,K,r}$. Then from (2.38) the error estimate (2.34) holds.

It remains to show that the approximation (2.40) can be rewritten in the form given in the statement of the Theorem.

Explicitly, from (2.30), (2.27) and using a hat "$\widehat{\ }$" to indicate an expression calculated from the $(-\mathrm{i})^r\omega_j$'s, rather than the z_j's

$$\begin{aligned}\Psi_{L,K,r}((-\mathrm{i})^r\omega) &= \mathfrak{Re}\left\{\widehat{\gamma}_2|\omega|^2 + \widehat{\gamma}_1(-\mathrm{i})^r\omega + \widehat{\gamma}_0\right\} \\ &\quad + \mathfrak{Re}\left\{\sum_{\ell=1}^{L}\left[\mathrm{i}^r\,\overline{\omega}\,\widehat{c}_{\ell,K} - \widehat{d}_{\ell,K}\right]e^{-x_\ell(-\mathrm{i})^r\omega}\right\},\end{aligned} \tag{2.41}$$

where

$$\widehat{c}_{\ell,K} = \frac{w_\ell}{x_\ell^2}\sum_{k=0}^{K}x_\ell^k\,\widehat{a}_k, \qquad \widehat{d}_{\ell,K} = \frac{w_\ell}{x_\ell^2}\sum_{k=0}^{K}x_\ell^k\,\widehat{b}_k,$$

$$\begin{aligned}\widehat{a}_k &= \frac{(-\mathrm{i})^{rk}}{k!}\sum_{j=1}^{J}\lambda_j\,\omega_j^k = (-\mathrm{i})^{rk}h^{-k}a_k\,, \\ \widehat{b}_k &= \frac{(-\mathrm{i})^{r(k-1)}}{k!}\sum_{j=1}^{J}\lambda_j\bar{\omega}_j\,\omega_j^k = (-\mathrm{i})^{r(k-1)}\,h^{-(k+1)}b_k\,,\end{aligned}$$

and the a_k, b_k are multipole coefficients for the unscaled and unrotated sources $\{(z_j,\lambda_j)\}_{j=1}^{J}$. Also,

$$\begin{aligned}\widehat{\gamma}_2 &= \alpha\,\widehat{a}_0 = \alpha a_0 \\ \widehat{\gamma}_1 &= -2\,\alpha\,\widehat{b}_0 - (\alpha+\beta)\,\widehat{a}_0 = -2\,\alpha\frac{\mathrm{i}^r\,b_0}{h} - (\alpha+\beta)a_0 \\ \widehat{\gamma}_0 &= \alpha\,\widehat{b}_1 + (\alpha+\beta)\,\widehat{a}_1 = \alpha\frac{b_1}{h^2} + (\alpha+\beta)\frac{(-\mathrm{i})^r\,a_1}{h}.\end{aligned}$$

Therefore from (2.40) and (2.41)

$$\begin{aligned}\Psi_{L,K,h,r}(\omega) &= h^2\log(h)\sum_{j=1}^{J}\lambda_j\,\mathfrak{Re}\left\{|\omega|^2 - 2\omega\overline{\omega}_j + |\omega_j|^2\right\} + h^2\mathfrak{Re}\left\{\widehat{\gamma}_2|\omega|^2 + \widehat{\gamma}_1(-\mathrm{i})^r\omega + \widehat{\gamma}_0\right\} \\ &\quad + h^2\mathfrak{Re}\left\{\sum_{\ell=1}^{L}\left[\mathrm{i}^r\,\overline{\omega}\,\widehat{c}_{\ell,K} - \widehat{d}_{\ell,K}\right]e^{-x_\ell(-\mathrm{i})^r\omega}\right\} \\ &= \log(h)\,\mathfrak{Re}\left\{h^2\,a_0|\omega|^2 - 2\,hb_0\,\omega + b_1\right\} + h^2\,\mathfrak{Re}\left\{\widehat{\gamma}_2|\omega|^2 + \widehat{\gamma}_1(-\mathrm{i})^r\omega + \widehat{\gamma}_0\right\} \\ &\quad + h^2\,\mathfrak{Re}\left\{\sum_{\ell=1}^{L}\left[\overline{\omega}\;\mathrm{i}^r\widehat{c}_{\ell,K} - \widehat{d}_{\ell,K}\right]e^{-x_\ell(-\mathrm{i})^r\omega}\right\} \\ &= \mathfrak{Re}\left\{\gamma_{2,h,r}|\omega|^2 + \gamma_{1,h,r}\,\omega + \gamma_{0,h,r}\right\} + \mathfrak{Re}\left\{\sum_{\ell=1}^{L}\left[\bar{\omega}\,c_{\ell,K,h,r} - d_{\ell,K,h,r}\right]e^{-x_\ell(-\mathrm{i})^r\omega}\right\},\end{aligned}$$

as required. □

2.6 Converting exponential expansions into polynomial approximations

This subsection concerns converting exponential approximations into Taylor polynomial approximations.

The original exponential approximation approximates the influence of sources in hQ throughout one of the target regions hR_r. In terms of the scaled variable, $\omega = z/h$, it approximates the influence of sources in Q throughout one of the regions R_r. A Taylor polynomial approximation will be formed for each of the 1×1 subsquares of R_r, see Figure 2.2. The piecewise polynomial approximation obtained will (in a single target square) be of the form

$$\Upsilon_{L,K,q,r,h}(u) = \Re e \left\{ h_{q+1} u^{q+1} + \sum_{j=0}^{q} (\bar{u}\, g_j + h_j)\, u^j \right\} \approx \Psi_{L,K,h,r}(\omega) \approx s(z), \tag{2.42}$$

where $u = \omega - t = (z - ht)/h$, and t is the center of one of the subsquares of R_r. This will give $\mathcal{O}\left(|u|^{q+2}\right)$ order approximation to $\Psi_{L,K,h,r}(u+t)$, and can therefore be identified as the Taylor polynomial in two real variables.

Explicitly, within the target square with center t,

$$\begin{aligned} \Psi_{L,K,h,r}(\omega) &= \Psi_{L,K,h,r}(u+t) \\ &= \Re e \left\{ \gamma_{2,h,r} |u+t|^2 + \gamma_{1,h,r}(u+t) + \gamma_{0,h,r} \right\} \qquad (2.43) \\ &\quad + \Re e \left\{ \sum_{\ell=1}^{L} \left[(\bar{u} + \bar{t})\, c_{\ell,K,h,r} - d_{\ell,K,h,r} \right] e^{-x_\ell (-\mathrm{i})^r t}\, e^{-x_\ell (-\mathrm{i})^r u} \right\}. \qquad (2.44) \end{aligned}$$

Expanding all the terms involving u on the right of (2.44) and neglecting those which are $\mathcal{O}(|u|^{q+2})$ as $|u| \to 0$, we obtain the required approximation $\Upsilon_{L,K,q,h,r}(u)$ of the form

(2.42), with coefficients

$$g_j = \begin{cases} \sum_{\ell=1}^{L} c_{\ell,K,h,r}\, e^{-x_\ell\,(-\mathrm{i})^r t}\,, & j=0, \\ \gamma_{2,h,r} + \sum_{\ell=1}^{L} c_{\ell,K,h,r}\,(-x_\ell)(-\mathrm{i})^r\, e^{-x_\ell\,(-\mathrm{i})^r t}, & j=1, \\ \sum_{\ell=1}^{L} c_{\ell,K,h,r}\, \frac{(-x_\ell)^j(-\mathrm{i})^{rj}}{j\,!}\, e^{-x_\ell\,(-\mathrm{i})^r t}, & j \geq 2, \end{cases} \tag{2.45}$$

and

$$h_j = \begin{cases} \gamma_{2,h,r}|t|^2 + \gamma_{1,h,r}t + \gamma_{0,h,r} + \sum_{\ell=1}^{L}\Big[\bar{t}\, c_{\ell,K,h,r} - d_{\ell,K,h,r}\Big]\, e^{-x_\ell\,(-\mathrm{i})^r t}\,, & j=0, \\ 2\,\gamma_{2,h,r}\bar{t} + \gamma_{1,h,r} + \sum_{\ell=1}^{L}\Big[\bar{t}\, c_{\ell,K,h,r} - d_{\ell,K,h,r}\Big]\,(-x_\ell)(-\mathrm{i})^r\, e^{-x_\ell\,(-\mathrm{i})^r t}, & j=1, \\ \sum_{\ell=1}^{L}\Big[\bar{t}\, c_{\ell,K,h,r} - d_{\ell,K,h,r}\Big]\, \frac{(-x_\ell)^j(-\mathrm{i})^{rj} e^{-x_\ell\,(-\mathrm{i})^r t}}{j\,!}\,, & j \geq 2. \end{cases} \tag{2.46}$$

Fundamental to the error estimates to be given below is the maximum error in approximating over R_0, a single thinplate source with centre $v \in Q$, and weight unity. That is

$$\epsilon_{L,K,q} := \max_{v\in Q}\max_{z\in R_0} \left| |z-v|^2 \log|z-v| - \Upsilon_{L,K,q,v}(z) \right|, \tag{2.47}$$

where $\Upsilon_{L,K,q,v}$ is the piecewise polynomial approximation in R_0 to this single source generated by the procedure described in the previous paragraph with $r=0$, $h=1$ and series length parameters L, K, and q. The approximation is piecewise polynomial as the local polynomial changes as the 1×1 target subsquare with center t in region R_0 changes. We estimate the error constants $\epsilon_{L,K,q}$ numerically, by maximizing the error over very fine grids with spacing 1/512.

Theorem 2.6.1. *Suppose that $\epsilon_{L,K,q}$ is defined as in* (2.47)*. Then for all $\{(z_j,\lambda_j) : 1 \leq j \leq J\} \subset hQ \times \mathbb{R}$ and $z \in hR_r$, $0 \leq r \leq 3$,*

$$\left| s(z) - \Upsilon_{L,K,q,h,r}(\omega) \right| \leq h^2\, \epsilon_{L,K,q} \sum_{j=1}^{J} |\lambda_j|, \tag{2.48}$$

where $\omega = z/h$, and $\Upsilon_{L,K,q,r,h}$ is a piecewise polynomial which has the form given in (2.42) *in each $h\times h$ subsquare in hR_r. Equations* (2.45) *and* (2.46) *specify the coefficients $\{g_j\}$ and $\{h_j\}$ of each polynomial piece in terms of the approximation $\Phi_{L,K,h,r}$ of Theorem 2.5.1.*

Proof. The proof of this Theorem is along similar lines to that of Theorem 2.5.1.

Let $\{(z_j, \lambda_j) : 1 \le j \le J\} \subset hQ \times \mathbb{R}$ and $z \in hR_r$, $0 \le r \le 3$. Define $\omega_j = z_j/h$ and $\omega = z/h$. Then, from the definition of $\epsilon_{L,K,q}$, and considering sources $\{((-\mathfrak{i})^r\omega_j\,, \lambda_j)\} \subset Q \times \mathbb{R}$ and an evaluation point $(-\mathfrak{i})^r\omega \in R_0$,

$$\left|\sum_{j=1}^{J} \lambda_j|\omega - \omega_j|^2 \log|\omega - \omega_j| - \Upsilon_{L,K,r}((-\mathfrak{i})^r\omega)\right| \le \epsilon_{L,K,q} \sum_{j=1}^{J} |\lambda_j|, \tag{2.49}$$

where $\Upsilon_{L,K,q,r}$ is $\Upsilon_{L,K,q}$ calculated using the $(-\mathfrak{i})^r\omega_j$'s rather than the z_j's.

Recall, equation (2.39) showing how the original thinplate spline becomes after scaling and rotating z and the z_j's, a quadratic polynomial plus h^2 times a new thinplate spline. This new thinplate spline has sources and evaluation point in the standard (Q, R_0), geometry. Therefore, define

$$\Upsilon_{L,K,q,r,h}(\omega) = h^2 \ \log(h) \sum_{j=1}^{J} \lambda_j|\omega - \omega_j|^2 + h^2\Upsilon_{L,K,q,r}((-\mathfrak{i})^r\omega), \tag{2.50}$$

which amounts to approximating the quadratic polynomial part on the right of (2.39) exactly, and the TPS in standard geometry with $\Upsilon_{L,K,q,r}$. Then from (2.49) the error estimate (2.48) holds.

It remains only to show that the approximation $\Upsilon_{L,K,q,r,h}$ can be rewritten in the form given in (2.42). The details of these calculations are omitted.

□

Values of the error constants $\epsilon_{L,K,q}$ for various choices of L, K and q are given in Table 2.3 below. In the corresponding computer code an acceptable maximum error, η, for the influence of a single source panel, $\mathcal{S}$, on a single target panel, $\mathcal{T}$, at the same level, is determined. Then, whenever the influence of a source panel $\mathcal{S}$ is to be approximated by a polynomial in Step 3 of Algorithm 1, L,K, and q are determined as the first choice from the table such that

$$h^2\epsilon_{L,K,q} \sum_{i:z_i\in\mathcal{S}} |\lambda_i| < \eta.$$

Besides calculating the error constant, we also recorded some timings and compare our method (EEFM) with the NFFT method by Potts et al. (2004), which is an algorithm based on fast Fourier transform at nonequispaced knots. The timings and the relative errors achieved were tabulated in Table 2.4, 2.5 and 2.6. These numerical experiments were conducted via evaluating the algorithm with the number of sources (N), and the

L	K	q	$\epsilon_{L,K,q}$	L	K	q	$\epsilon_{L,K,q}$
2	2	2	$9.495(-1)$	17	17	17	$9.946(-9)$
3	3	2	$2.832(-1)$	18	18	18	$4.141(-9)$
4	4	2	$5.056(-2)$	19	19	19	$1.771(-9)$
5	5	3	$2.051(-2)$	20	20	20	$7.615(-10)$
6	6	4	$6.755(-3)$	21	21	21	$3.155(-10)$
7	7	5	$1.909(-3)$	22	22	22	$1.356(-10)$
8	8	6	$3.564(-4)$	23	23	23	$6.012(-11)$
9	9	7	$1.193(-4)$	24	24	24	$2.611(-11)$
10	10	8	$3.907(-5)$	25	25	25	$1.114(-11)$
11	11	10	$8.844(-6)$	26	26	26	$4.801(-12)$
12	12	10	$4.028(-6)$	27	27	27	$2.179(-12)$
13	13	12	$6.680(-7)$	28	28	28	$9.304(-13)$
14	14	14	$1.965(-7)$	29	29	29	$4.246(-13)$
15	15	14	$1.131(-7)$	30	30	30	$3.091(-13)$
16	16	16	$2.899(-8)$				

Table 2.3: Error constants, $\epsilon_{L,K,q}$, for exponential approximation of thin plate splines with various values of L, K and q.

number of evaluation points equal to $\left(1+\sqrt{\lfloor N \rfloor}\right)^2$, a number usually slightly larger than N. The weights λ_i were all set equal to 1. The times taken by the algorithm are consistent with an approximate order N overall execution time. The order constant depending on the required relative precision.

All the timings shown are the smallest seen in 5 runs on an unloaded machine. The code is single threaded and ran on an otherwise unloaded machine with a 3.4GHz Intel I7-2600K CPU. In the tables the notation $r.stu(-v)$ indicates the number $r.stu \times 10^{-v}$. We expect to be able to improve our implementation further for a moderate further increase in speed.

We show the performance of both algorithms by calculating the speed up factor (i.e. direct time/algorithm time) for various different N's and different target relative precisions. These are presented in Table 2.7. Figure 2.5 shows the comparison of performance for both algorithm at different precisions. Note that our method generally outperformed the NFFT method. Also, for large N, our method can be dramatically faster than direct evaluation. Over 200 times faster for $N = 64,000$ and almost 1000 times faster for $N = 256,000$.

Number of sources	Number of evaluation points	Direct (sec)	NFFT (sec)	EEFM (sec)	relative error NFFT	relative error EEFM
2000	2025	3.712(−2)	6.999(−3)	4.888(−3)	3.807(−5)	7.658(−5)
4000	4096	1.505(−1)	1.400(−2)	1.030(−2)	1.139(−5)	7.953(−5)
8000	8100	5.942(−1)	3.200(−2)	1.818(−2)	6.729(−5)	7.816(−5)
16000	16129	2.375(00)	5.299(−2)	3.661(−2)	4.089(−5)	8.014(−5)
32000	32041	9.433(00)	9.498(−2)	7.007(−2)	3.800(−5)	6.071(−5)
64000	64009	3.769(+1)	1.830(−1)	1.455(−1)	4.717(−5)	8.076(−5)
128000	128164	1.505(+2)	3.619(−1)	2.835(−1)	5.275(−5)	6.382(−5)
256000	256036	6.077(+2)	7.329(−1)	6.106(−1)	5.683(−5)	5.074(−5)

Table 2.4: Timings and relative errors for various computations at approximate relative precision of 1.0(−4).

Number of sources	Number of evaluation points	Direct (sec)	NFFT (sec)	EEFM (sec)	relative error NFFT	relative error EEFM
2000	2025	3.712(−2)	5.499(−2)	9.482(−3)	5.476(−9)	2.739(−9)
4000	4096	1.505(−1)	7.099(−2)	1.871(−2)	1.131(−9)	6.128(−9)
8000	8100	5.942(−1)	1.160(−1)	4.060(−2)	1.150(−9)	3.164(−9)
16000	16129	2.375(00)	1.650(−1)	7.145(−2)	1.043(−9)	9.433(−9)
32000	32041	9.433(00)	2.660(−1)	1.463(−1)	1.027(−9)	7.318(−9)
64000	64009	3.769(+1)	5.269(−1)	2.883(−1)	1.100(−9)	5.581(−9)
128000	128164	1.505(+2)	1.309(00)	5.565(−1)	1.123(−9)	6.726(−9)
256000	256036	6.077(+2)	3.847(00)	1.094(00)	1.133(−9)	4.439(−9)

Table 2.5: Timings and relative errors for various computations at approximate relative precision of 1.0(−8).

Number of sources	Number of evaluation points	Direct (sec)	NFFT (sec)	EEFM (sec)	relative error NFFT	relative error EEFM
2000	2025	3.712(−2)	5.489(−1)	1.508(−2)	5.517(−13)	2.998(−13)
4000	4096	1.505(−1)	5.669(−1)	3.199(−2)	2.683(−13)	2.469(−13)
8000	8100	5.942(−1)	6.199(−1)	6.829(−2)	9.022(−13)	3.646(−13)
16000	16129	2.375(00)	5.109(−1)	1.280(−1)	9.019(−13)	3.382(−13)
32000	32041	9.433(00)	8.699(−1)	2.580(−1)	3.538(−13)	8.773(−13)
64000	64009	3.769(+1)	2.123(00)	4.941(−1)	4.383(−13)	5.940(−13)
128000	128164	1.505(+2)	6.698(00)	1.011(00)	4.138(−13)	4.891(−13)
256000	256036	6.077(+2)	2.405(+1)	1.925(00)	3.560(−13)	6.331(−13)

Table 2.6: Timings and relative errors for various computations at approximate relative precision of 1.0(−12).

Number of sources	Number of evaluation points	Approximate relative precision					
		1.0(−4)		1.0(−8)		1.0(−12)	
		EEFM	NFFT	EEFM	NFFT	EEFM	NFFT
2000	2025	7.59	5.30	3.91	0.68	2.46	0.068
4000	4096	14.62	10.75	8.05	2.12	4.71	0.27
8000	8100	32.68	18.59	14.64	5.12	8.70	0.96
16000	16129	64.85	44.82	33.23	14.39	18.55	4.65
32000	32041	134.62	99.32	64.50	35.46	36.56	10.84
64000	64009	258.94	205.96	130.73	71.53	76.27	17.75
128000	128164	531.04	415.86	270.51	114.97	148.93	22.47
256000	256036	995.16	829.17	555.26	157.97	315.63	25.27

Table 2.7: Speed up factor for EEFM and NFFT computations at various approximate precisions.

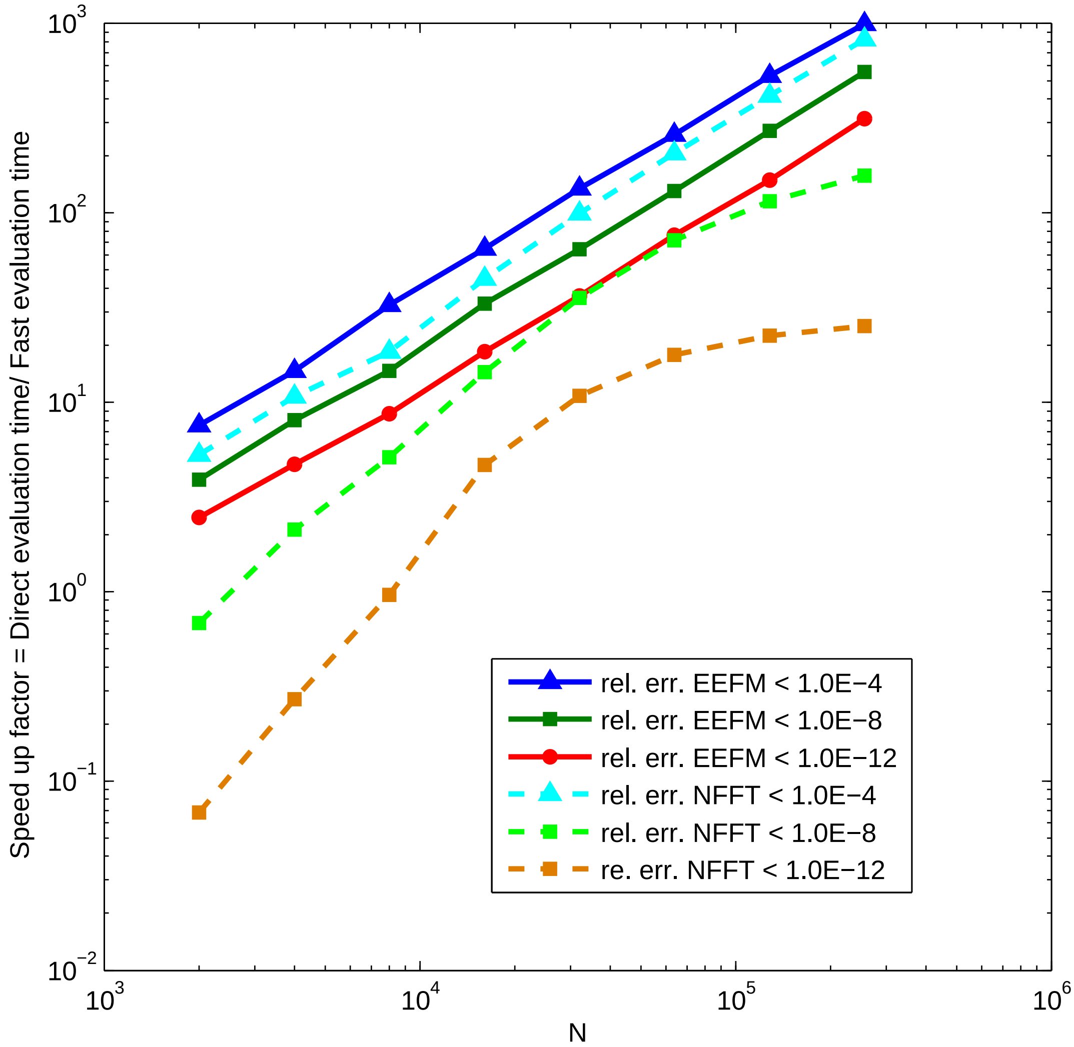


Figure 2.5: Comparison of the speed up factor for EEFM and NFFT computations at various approximate precisions.

2.7 Exploiting rotational symmetry

2.7.1 Exploiting symmetry in forming exponential series

In this section, we will show how the coefficients $c_{\ell,K,h,r}$ and $d_{\ell,K,h,r}$ of the four different exponential approximations corresponding to a single source panel can be calculated efficiently. Saving operations in this way is discussed in Hrycak and Rokhlin (1998).

Recall that $c_{\ell,K,h,r} = \dfrac{h^2\, w_\ell}{x_\ell^2} \displaystyle\sum_{k=0}^{K} \left(\frac{x_\ell}{h}\right)^k (-\mathfrak{i})^{r(k-1)}\, a_k$, as defined in equation (2.36) of Theorem 2.5.1. Let us define the notation $\displaystyle\sum_{k=j,\,4}^{K} f(k)$ to mean the sum of $f(k)$ over indices k such that $j \le k \le K$ and $k = j \pmod 4$. Since $(-\mathfrak{i})^k$ is four periodic, it follows that

$$
\begin{aligned}
c_{\ell,K,h,r} &= \frac{h^2\, w_\ell}{x_\ell^2}\, \mathfrak{i}^r \sum_{k=0}^{K} \left(\frac{x_\ell}{h}\right)^k \left((-\mathfrak{i})^k\right)^r a_k \\
&= \frac{h^2\, w_\ell}{x_\ell^2} \Bigg[\mathfrak{i}^r \sum_{k=0,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k a_k + \sum_{k=1,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k a_k \\
&\qquad + (-\mathfrak{i})^r \sum_{k=2,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k a_k + (-1)^r \sum_{k=3,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k a_k \Bigg] \\
&= \frac{h^2\, w_\ell}{x_\ell^2} \Bigg[\mathfrak{i}^r f_{0,\ell} + f_{1,\ell} + (-\mathfrak{i})^r f_{2,\ell} + (-1)^r f_{3,\ell} \Bigg],
\end{aligned}
$$

where $f_{j,\ell} = \displaystyle\sum_{k=j,4}^{K} \left(\frac{x_\ell}{h}\right)^k a_k$. This allows efficient computation of the coefficients $c_{\ell,K,h,r}$, for $0 \le r \le 3$, simultaneously. This way, one can first compute $f_{j,\ell}$ for $0 \le j \le 3$. Then coefficients $c_{\ell,K,h,r}$ for $0 \le r \le 3$ can be computed by fixed code without computing $\mathfrak{i}^r$, $(\mathfrak{i})^r$ and $(-1)^r$. Similarly, $d_{\ell,K,h,r}$ can also be computed by the same technique,

$$
\begin{aligned}
d_{\ell,K,h,r} &= \frac{h\, w_\ell}{x_\ell^2}\, \mathfrak{i}^r \sum_{k=0}^{K} \left(\frac{x_\ell}{h}\right)^k \left((-\mathfrak{i})^k\right)^r b_k \\
&= \frac{h\, w_\ell}{x_\ell^2} \Bigg[\mathfrak{i}^r \sum_{k=0,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k b_k + \sum_{k=1,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k b_k \\
&\qquad + (-\mathfrak{i})^r \sum_{k=2,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k b_k + (-1)^r \sum_{k=3,\,4}^{K} \left(\frac{x_\ell}{h}\right)^k b_k \Bigg].
\end{aligned}
$$

2.7.2 Exploiting symmetry in recentering polynomials

In this subsection, we discuss how to recenter the polynomial approximation corresponding to a parent panel efficiently to the centers of the children. The approach taken is via the complete Horner scheme and differs from that discussed in Hrycak and Rokhlin (1998).

The complete Horner scheme for expressing a polynomial $p(z) = \sum_{k=0}^{K} a_k z^k$ expanded about 0, as a polynomial $\sum_{k=0}^{K} b_k (z-t)^k$ expanded about t, is given in the algorithm below.

Algorithm 2 The complete Horner scheme

for $j = 0$ **to** $K-1$ **do**
  for $k = K-j-1$ **to** $K-1$ **do**
    $a_k := a_k + t a_{k+1}$
  end for
end for

In this pseudo code, the final a_k's are the desired coefficients b_k. As written, this scheme takes $\frac{(K+1)K}{2}$ complex multiplications and $\frac{(K+1)K}{2}$ complex additions. However if t is one of $1, \mathfrak{i}, -1$ or $-\mathfrak{i}$ then, all the multiplications can be avoided and the arithmetic work reduces to $\frac{(K+1)K}{2}$ complex additions.

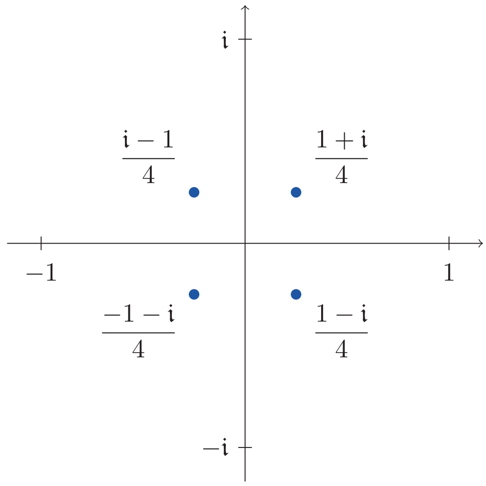
For example in the particular case $t = \mathfrak{i}$, Algorithm 2 can be recoded as

for $j = 0$ **to** $K-1$ **do**
  for $k = K-j-1$ **to** $K-1$ **do**
    $\mathrm{Re}(a_k) := \mathrm{Re}(a_k) - \mathrm{Im}(a_{k+1})$
    $\mathrm{Im}(a_k) := \mathrm{Im}(a_k) + \mathrm{Re}(a_{k+1})$
  end for
end for

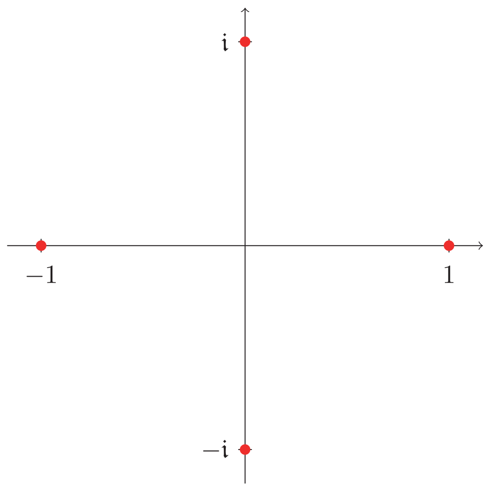
This observation can be exploited to enable efficient recentering of the polynomial of a parent panel to the centers of the four children.

As in Figure 2.6, we first make the change of variable $w = \frac{4}{1+\mathfrak{i}} z = (2-2\mathfrak{i})\, z$ obtaining from $p(z)$ a polynomial

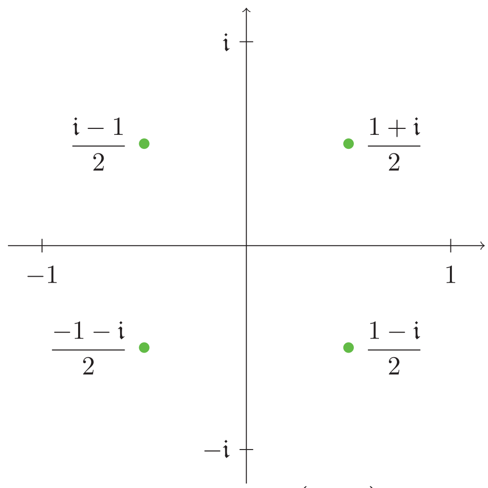
$$q(w) = \sum_{k=0}^{K} b_k w^k.$$



(a) $z-$ plane.



(b) $w-$ plane, $w = (2 - 2\mathrm{i})z$.



(c) $v-$ plane, $v = \left(\dfrac{1+i}{2}\right) w$.

Figure 2.6: The $z-$plane represents the parent domain level ℓ, and $v-$plane represents the child domain level $\ell + 1$.

This change of variable is accomplished in $\mathcal{O}(K)$ floating point operations. Then in the $w-$plane setting, we recenter q to be expressed about $1\,,\mathfrak{i}\,,-1$ and $-\,\mathfrak{i}$. This recentering is accomplished with the efficient, no multiplication, forms of complete Horner previously discussed.

Finally we make another change of variable $v = \left(\frac{1+\mathfrak{i}}{2}\right) w$ in the "child" polynomials. This rotates and scales, see Figure 2.6, so that the "child" polynomials are scaled for 1×1 panels centered at $\frac{1+\mathfrak{i}}{2}\,, \frac{\mathfrak{i}-1}{2}\,, \frac{-1-\mathfrak{i}}{2}$ and $\frac{1-\mathfrak{i}}{2}$.

This strategy achieves a considerable lessening of the overall work, replacing $2K(K+1)$ complex multiplications occuring in four "ordinary" complete Horner's, by $\mathcal{O}(5K)$ complex multiplications occuring in the scaling and rotating.

2.8 An alternative Exponential Expansions for TPS

In this section, we present an alternative exponential expansions to approximate $|z-t|^2 \log|z-t|$. Instead of using the exponential approximation to $(z-t)\log|z-t|$, we use the one approximating $\log|z-t|$, and simply multiply it by $|z-t|^2$. The expansions obtained can also be written in separated form where term being expressed as a product of stuff depending on a source and stuff depending on the evaluation point. Such series can be summed over a cluster of sources retaining the same basic form. The series obtained appear to be less advantageous for numerical computations than the exponential series of equation (2.31) as they contain more terms. We include them for the sake of completeness.

Recall from Section 2.4, Equation (2.21), we have

$$\mathrm{Log}(z-t) \approx \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\, e^{-x_\ell\,(z-t)} \tag{2.51}$$

with a suitable error bound. Therefore, an approximation of a single thin-plate source at $t \in Q$ with weight 1 and $z \in R_0$ is

$$
\begin{aligned}
|z-t|^2 \log|z-t| &= \mathfrak{Re}\Big\{|z-t|^2 \,\mathrm{Log}\,|z-t|\Big\} \\
&\approx \mathfrak{Re}\left\{(\bar z-\bar t)(z-t)\left[\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{-x_\ell}-\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{-x_\ell\,(z-t)}\right]\right\} \\
&= \Big[|z|^2-2\,\mathfrak{Re}(\bar t z)+|t|^2\Big]\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{-x_\ell} \\
&\qquad -\Big(|z|^2+|t|^2\Big)\,\mathfrak{Re}\left\{\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{x_\ell t}\,e^{-x_\ell z}\right\} \\
&\qquad +\mathfrak{Re}\left\{\Big(\bar t z+\bar z t\Big)\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{x_\ell t}\,e^{-x_\ell z}\right\}.
\end{aligned} \tag{2.52}
$$

For a cluster of sources $\{z_j\}_{j=1}^{J}$ where $z_j \in Q$ is associated with weight $\lambda_j \in \mathbb{R}$, the corresponding approximation is

$$
\begin{aligned}
s(z) &= \sum_{j=1}^{J}\lambda_j|z-z_j|^2\log|z-z_j| \\
&= \left[\sum_{j=1}^{J}\lambda_j|z|^2-2\,\mathfrak{Re}\left(\sum_{j=1}^{J}\lambda_j\bar z_j z\right)+\sum_{j=1}^{J}\lambda_j|z_j|^2\right]\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{-x_\ell} \\
&\quad+\ \mathfrak{Re}\left\{-|z|^2\ \sum_{j=1}^{J}\lambda_j\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{x_\ell z_j}\,e^{-x_\ell z}-\sum_{j=1}^{J}\lambda_j|z_j|^2\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,e^{x_\ell z_j}\,e^{-x_\ell z}\right. \\
&\qquad\left.+\,\bar z\ \sum_{j=1}^{J}\lambda_j\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,z_j\,e^{x_\ell z_j}\,e^{-x_\ell z}+z\ \sum_{j=1}^{J}\lambda_j\sum_{\ell=1}^{L}\frac{w_\ell}{x_\ell}\,\bar z_j\,e^{x_\ell z_j}\,e^{-x_\ell z}\right\}.
\end{aligned} \tag{2.53}
$$

Note that $e^{x_\ell\, z_j} = \displaystyle\sum_{k=0}^{\infty}\frac{(x_\ell z_j)^k}{k!}$ and truncating the series on the right above at

$k = K$, the approximation becomes

$$
\begin{aligned}
s(z) \approx{}& \left[\sum_{j=1}^{J} \lambda_j |z|^2 - 2\,\Re e\left(\sum_{j=1}^{J} \lambda_j \bar{z}_j z\right) + \sum_{j=1}^{J} \lambda_j |z_j|^2\right] \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\; e^{-x_\ell} \\
&+\; \Re e\Bigg\{ -|z|^2 \;\sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\; \left(\sum_{j=1}^{J}\sum_{k=0}^{K} \frac{\lambda_j z_j^k}{k!} x_\ell^k\right)\; e^{-x_\ell z} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\; \left(\sum_{j=1}^{J}\sum_{k=0}^{K} \frac{\lambda_j z_j^{k+1}}{k!} \bar{z}_j x_\ell^k\right)\; e^{-x_\ell z} \\
&\quad + \bar{z}\; \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\; \left(\sum_{j=1}^{J}\sum_{k=0}^{K} \frac{\lambda_j z_j^{k+1}}{k!} x_\ell^k\right)\; e^{-x_\ell z} + z\; \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\; \left(\sum_{j=1}^{J}\sum_{k=0}^{K} \frac{\lambda_j z_j^k}{k!} \bar{z}_j x_\ell^k\right)\; e^{-x_\ell z} \Bigg\} \\
={}& \Big[a_0 |z|^2 - 2\,\Re e(b_0 z) + b_1\Big] A \\
&+\; \Re e\Bigg\{ -|z|^2 \;\sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell} \left(\sum_{k=0}^{K} a_k\, x_\ell^k\right) e^{-x_\ell z} - \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell} \left(\sum_{k=0}^{K} (k+1)\, b_{k+1}\, x_\ell^k\right) e^{-x_\ell z} \\
&\quad + \bar{z}\; \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell} \left(\sum_{k=0}^{K} (k+1)\, a_{k+1}\, x_\ell^k\right) e^{-x_\ell z} + z\; \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell} \left(\sum_{k=0}^{K} b_k\, x_\ell^k\right) e^{-x_\ell z} \Bigg\} \\
={}& \Big[a_0 |z|^2 - 2\,\Re e(b_0 z) + b_1\Big] A \\
&+\; \Re e\left\{ \sum_{\ell=1}^{L} \Big(|z|^2\, c_{\ell,K} + \bar{z}\, d_{\ell,K} + z\, e_{\ell,K} + f_{\ell,K} \Big) e^{-x_\ell z} \right\}
\end{aligned}
\tag{2.54}
$$

where

$$
A = \sum_{\ell=1}^{L} \frac{w_\ell}{x_\ell}\; e^{-x_\ell}, \qquad a_k = \frac{1}{k!} \sum_{j=1}^{J} \lambda_j z_j^k, \qquad b_k = \frac{1}{k!} \sum_{j=1}^{J} \lambda_j \bar{z}_j z_j^k\,,
$$

$$
\begin{aligned}
c_{\ell,K} &= -\frac{w_\ell}{x_\ell}\; \sum_{k=0}^{K} a_k\, x_\ell^k\,, & d_{\ell,K} &= \frac{w_\ell}{x_\ell}\; \sum_{k=0}^{K} (k+1)\, a_{k+1}\, x_\ell^k\,, \\
e_{\ell,K} &= \frac{w_\ell}{x_\ell}\; \sum_{k=0}^{K} b_k\, x_\ell^k\,, \qquad \text{and} & f_{\ell,K} &= -\frac{w_\ell}{x_\ell}\; \sum_{k=0}^{K} (k+1)\, b_{k+1}\, x_\ell^k\,.
\end{aligned}
\tag{2.55}
$$

Chapter 3

Reconstruction with Blobby Shapes

The wide spread adoption of 3D scanning devices has enabled the acquisition of many large data sets or cloud of points. A cloud of points refers to a collection of points assume to be on or very near to a surface. These can represent a variety of objects. For example, points obtain from the laser scan of sculptures, body parts, and machine parts, through to the interiors of mines. For various applications, it is required to convert the point clouds into a "water tight" surface, meaning that no holes should be allowed in the surface. This process is classified into the research area called *surface reconstruction*. Much attention has been paid by researchers in this area as a main part of reverse engineering of shapes, which aims to create physical models of existing objects for further utilization.

Various reconstruction strategies and methods were reported in the literature (Hoppe et al., 1992; Amenta et al., 1998; Zhao et al., 2000; Duan et al., 2004; Ye et al., 2010). Most of them were designed for a specific type of problem, based on the important aspects listed below:

- **Prior knowledge of the physical model.** If the topology or geometric information of the point clouds is known, a more specific and efficient method to reconstruct the shape may be used. This is particularly important for methods that involve segmentation and fitting primitives to the point clouds (Attene et al., 2006; Liu et al., 2006). Prior knowledge is indeed useful for achieving better accuracy in reconstructing shapes from point clouds. Unfortunately, this is not common in practical situations such as recovery of biological shapes or when multiple view point range data is used. Calibration, accuracy, and accessibility in acquiring useable data also add to the difficulty in making prior assumptions of the physical model (Varady et al., 1997).

- **Fitting algorithms.** Interpolation and approximation are the two main techniques in fitting a surface to point clouds. One would use an interpolation method if there is no appreciable noise in the point cloud, that means the reconstructed surface passes exactly through each point in the cloud. Related work can be found in (Franke and Nielson, 1991; Alfeld, 1989; Ohtake et al., 2005). However, if the point cloud is noisy (contains measurement errors) or incomplete (lack of samples in certain regions), approximating the surface will be more appropriate. There exist many fitting solutions to this problem which include fitting of subdivision surfaces (Hoppe et al., 1994; Cheng et al., 2007), parametric surfaces (Floater and Reimers, 2001; Hormann, 2003), Splines Functions (B-splines and NURBS) (Forsey and Wong, 1998; Farin, 2002b) and Radial Basis Functions (RBFs) (Carr et al., 2003; Tobor et al., 2004a).

- **Performance: accuracy, speed and user interactivity.** The trade-off between accuracy (how well the surface is fitted) and speed (computational costs: time and memory) has always been an issue of research. With increasing availability of high speed computers and improved algorithms (Zhou et al., 2008; Mullen et al., 2010; **?**), one can wisely choose a robust method to achieve desired quality with low cost. This decision is usually motivated by the application of the reconstructed surface. Another interesting issue is whether the process can be controlled by the user, or is fully automatic (Hoppe et al., 1992; Attene and Spagnuolo, 2000). An automatic algorithm is used in cases where a complete (very dense with no noise) point cloud is given and much information of the physical model is known in advance. On the other hand, algorithms that enable user input to make intelligent decisions along the process are often needed. A user interactive environment is especially useful in a segmenting shape or surface and extracting part of shape to be reconstructed or discarded (Sharf et al., 2007; Gregory et al., 1999).

- **Application of the reconstructed surface.** Medical diagnosis (You et al., 2008; Liu et al., 2008; Carr et al., 1997; Lorensen and Cline, 1987), and object and image morphing techniques (Gregory et al., 1999; Jin et al., 2005) in computer visualization and animation (Liu et al., 2007) are among the many applications nowadays. By knowing the end use of the reconstructed surface, one can then choose a suitable method or algorithm, which enhance the efficiency of the surface reconstruction process.

The motivation of this work is to identify the basic structure of an object by blobby model representation. Blobby representation is ideal for modeling smooth objects with large curved surfaces that exist in most organic objects. Here we present a method to approximate the shape of a given cloud of points with a low parameter model

- using ellipses in two dimensional space (2D)
- using ellipsoids in three dimensional space (3D).

Our aim is to find a very efficient coarse level approximation for an hierarchical implicit curve (2D), or surface (3D), model. Here by efficient we mean a coarse level approximation that reproduces the overall shape of the underlying object while being very fast to evaluate. The speed of evaluation of the coarse level approximation is critical as it will be used for every function evaluation occurring in rendering the surface. Extensive computational experience with implicit radial basis function models (see e.g. Carr et al. (2001)) and hierarchical radial basis function models (Langton, 2009, Chap. 7), indicates to us that the current method easily outperforms radial basis function models for this application.

Suitable objective functions and heuristics have been developed which work well on our test cases, particularly in fitting the outline of a hand in 2D and 3D. The 2-stage process involves initialization and optimization of blobs (ellipses or ellipsoids) approximating parts of a point cloud. After that an implicit blobby model is fitted by combining all the optimized blobs through least squares fitting to all the data. We present both user-initialized and automatic algorithms for blob identification in the first stage. The user-initialized algorithm enables a user to visualize and identify regions for blobs to be optimized. This is very straight forward and efficiency is guaranteed. In the automatic algorithm, the parts approximated by different blobs are not identified explicitly and will usually overlap. We will demonstrate our algorithm in detail later. We will also provide solutions to some drawbacks that arise in our method.

3.1 Previous work

A good general overview on various methods used in surface reconstruction prior to 1999 can be found in Söderkvist (1999). Since we are focusing on blobby model shape reconstruction, we only review related research; particularly those reported recently. By the word related, we mean all these works share a common core idea with our work: the process of reconstructing the surface. They all constructed an implicit surface with steps using blobs in fitting primitives or decomposition, employed a field function, and extract the isosurface from the implicit function to represent the shape. However, detailed techniques and approach used are quite different.

Muraki (1991) is the first to automatically generate a “Blobby Model” for a given set of range data, using spherical primitives. Without segmenting range data, the procedure of fitting the primitives is done by “selection and division” of the primitives sequence and minimizing the energy function. The proposed selection method that examines all primitives at each step is claimed to consume much computational time.

Bischoff and Kobbelt (2002a) presented a technique in fitting ellipsoids to the volume bounded in a given mesh. They first decompose the original mesh into a set of ellipsoids until they completely fill up the interior of the mesh. This results in an initial mesh which they later refine using a robust transmission technique (Bischoff and Kobbelt, 2002b) by inserting sample points from the original mesh. The robust transmission of geometric objects in the ellipsoidal decomposition guarantees a good approximation even if some parts of the data get lost during transmission.

Simari and Singh (2005) restructured and re-meshed polygon meshes using ellipsoidal based segmentation. Their algorithm allows fitting the ellipsoids to a given mesh by surface or volume enclosed orientation. They also take "negative ellipsoids" as primitives to capture bowl-shaped concavities. The shape of the model is then obtained by smoothing segmentation boundaries.

Jin et al. (2005) proposed an automatic algorithm for approximating polygonal meshes with blobby objects based on the application of 3D liquid morphing. They constructed a medial axis sphere-tree of the given mesh and used the spheres as initial blobs to approximate the object's shape. Once the sphere representation is achieved, an isosurface of the object can be generated employing field functions in the blobby model. An implicit representation of the object is constructed through minimizing the distance between the isosurface of blobs and the given mesh. This work is then extended by Liu et al. (2007) to reconstruct a blobby model using ellipsoidal blobs. They employed a modified scheme of Bischoff and Kobbelt (2002a) in the ellipsoidal decomposition process. Their main contributions are applying the ellipsoidal blobby model in geometry data reduction and giving two geometric-based schemes for cloud animation.

Chapter 4

Fitting Ellipses in 2D

Curve reconstruction in 2D can be defined as the problem of connecting a given set of points sampled from a smooth curve, in the order they appear on that curve. While much attention has been focused on developing algorithms for surface reconstruction in 3D, there is still substantial interest for similar work in 2D (Wang et al., 2010; Kumar et al., 2004; Lee, 2000; Dey and Kumar, 1999). This problem is assumed to be simpler as it is often be regarded as a special case of the surface reconstruction problem. However, the importance of understanding and solving the problem in 2D should not be underestimated. The problem is worth studying as it is fundamental to ensuring the feasibility of extending the algorithm to higher dimensions.

Here, we introduce a method to reconstruct the curve of an ordered point cloud using blobby model representation. We would like to describe the shape or outline formed by some scattered data points in a plane. In 2D ellipses are a good choice of primitive for capturing the curvature properties. More formally, given a set of scattered data points in the plane, (x_i, y_i) where $i = 1, 2, \cdots, N$, we aim to approximate part of the shape of the scattered points by an ellipse.

Let us now look at some basic properties of an ellipse and how we exploit them in our method. Generally, an arbitrarily oriented ellipse with major and minor semi axes, a and b can be defined as

$$(\boldsymbol{x} - \boldsymbol{c})^T \mathbf{A} (\boldsymbol{x} - \boldsymbol{c}) = 1 \tag{4.1}$$

where $\boldsymbol{x} = (x, y)$ are points on the ellipse, $\boldsymbol{c} = (x_c, y_c)$ is the center of the ellipse and $\mathbf{A}$ is a symmetric positive definite matrix that specifies the size and orientation of the ellipse. $\mathbf{A}$ can be decomposed as $\mathbf{A} = QDQ^T$ where the eigenvalues of $\mathbf{A}$ are given by

$$D = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \tag{4.2}$$

and the columns of Q are the corresponding eigenvectors. These eigenvalues are the inverse square of the length of the major and minor semi axes, a and b, i.e.

$$a = \frac{1}{\sqrt{\sigma_1}} \quad \text{and} \quad b = \frac{1}{\sqrt{\sigma_2}}. \tag{4.3}$$

Q represents the rotation matrix,

$$Q = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{4.4}$$

where θ is the angle of rotation of the ellipse counter clockwise from the x-axis, see Figure 4.1.

In the first stage of our method, ellipses are fitted inside the point clouds independently of each other. Therefore, we set up the fitting problem as a constrained optimization problem of the form

$$\text{minimize } f(\boldsymbol{z}) \text{ such that } l_b \leq \boldsymbol{z} \leq u_b, \tag{4.5}$$

subject to additional nonlinear constraints preventing points from lying inside the fitted ellipse. The problem's decision variables are $\boldsymbol{z} = \begin{bmatrix} x_c & y_c & a & b & \theta \end{bmatrix}^T$ and the objective function $f(\boldsymbol{z})$, and constraint functions will now be specified.

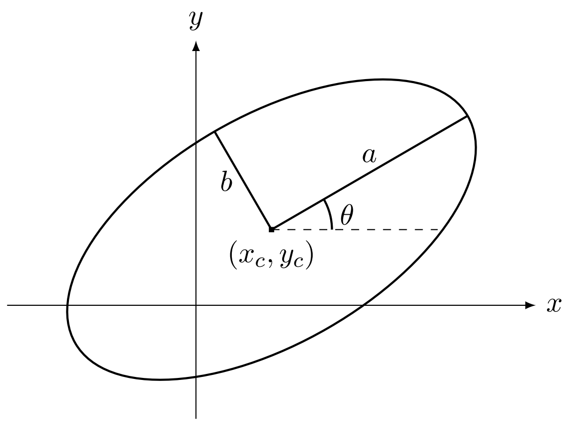


Figure 4.1: Parameters of an ellipse used in the constrained optimization step.

Here, we set x_c, y_c, a, b, and θ as the 5 parameters to be optimized, see Figure 4.1.

Since we are fitting an ellipse inside the point cloud, we need an objective function so that the growing ellipse is bounded by the point cloud.

Let us define the offset of a point $\boldsymbol{x}_i$ from the ellipse as

$$\rho_i = (\boldsymbol{x}_i - \boldsymbol{c})^T \mathbf{A} (\boldsymbol{x}_i - \boldsymbol{c}) - 1. \tag{4.6}$$

Then,

$$\rho_i = \begin{cases} > 0, & \text{if } \boldsymbol{x}_i \text{ lies outside the ellipse;} \\ = 0, & \text{if } \boldsymbol{x}_i \text{ is on the ellipse;} \\ < 0, & \text{if } \boldsymbol{x}_i \text{ lies inside the ellipse.} \end{cases} \tag{4.7}$$

Therefore, to encourage points outside to be on the ellipse, we set our objective function as

$$f(\boldsymbol{z}) = \sum_{i=1}^{N} \frac{\big[\rho_i\big]_+^2}{K + \big[\rho_i\big]_+^2} \tag{4.8}$$

for some $K > 0$ and $\big[\rho_i\big]_+ = \max\big(\rho_i, 0\big)$. The constraints consist of simple bounds on the decision variables

$$\begin{gathered} x_{min} \leq x_c \leq x_{max}\,, \\ y_{min} \leq y_c \leq y_{max}\,, \\ \ell_{min} \leq a, b \leq \ell_{max}\,, \\ 0 \leq \theta \leq \pi\,, \end{gathered}$$

together with the nonlinear constraints $c_i = -\rho_i \leq 0,\ \forall\ i = 1, 2, \cdots, N$, which are designed to prevent any data point from lying inside the optimized ellipse.

Several ellipses are fitted using the optimization procedure described in the last three paragraphs. Once this forming ellipses phase is finished, we need a method to blend these ellipses to describe the shape of the scattered data points. Each optimized ellipse now takes the form of equation (4.1) for some symmetric matrix $\mathbf{A}$ and center $\boldsymbol{c}$. The left hand side of equation (4.1) is a positive definite quadratic form. The square root of a quadratic form is suitable for representing a single ellipse, but not for combining several ellipses. The difficulty with the root quadratic form is that it becomes larger as one moves further away from the ellipse's center. In a linear combination of such functions the distant ellipses' root quadratic forms can swamp those of closer ellipses. To avoid this effect we need a function which is negligible (and preferably zero) far from an ellipse.

To this end we introduce the sigmoidal function $\phi(s)$

$$\phi(s) = \begin{cases} 2, & \text{if } t \le 0; \\ 2 - \dfrac{t^3}{3h^3}, & \text{if } 0 < t \le h; \\ 2 - \left(\dfrac{1}{3} + \dfrac{t-h}{h} + \dfrac{(t-h)^2}{h^2} - \dfrac{2(t-h)^3}{3h^3}\right), & \text{if } h < t \le 2h; \\ \dfrac{(3h-t)^3}{3h^3}, & \text{if } 2h < t \le 3h; \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

where

$$s = \sqrt{(\boldsymbol{x}-\boldsymbol{c})^T\mathbf{A}(\boldsymbol{x}-\boldsymbol{c})} \quad \text{and} \quad t = s - \left(1 - \frac{3}{2}h\right).$$

We have chosen this particular sigmoidal function $\phi(s)$ because it is identically zero when

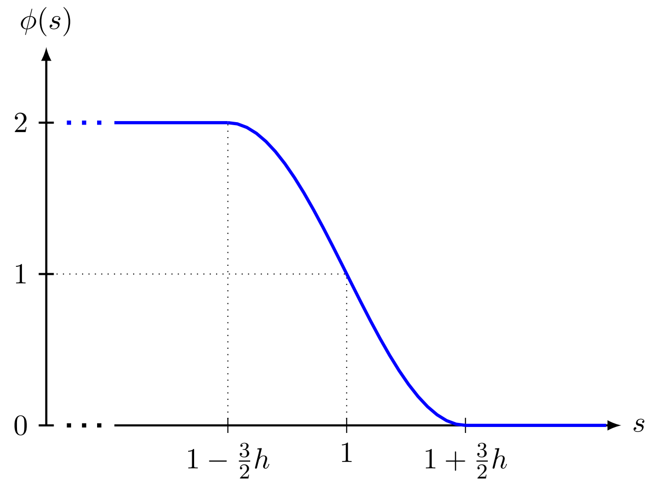


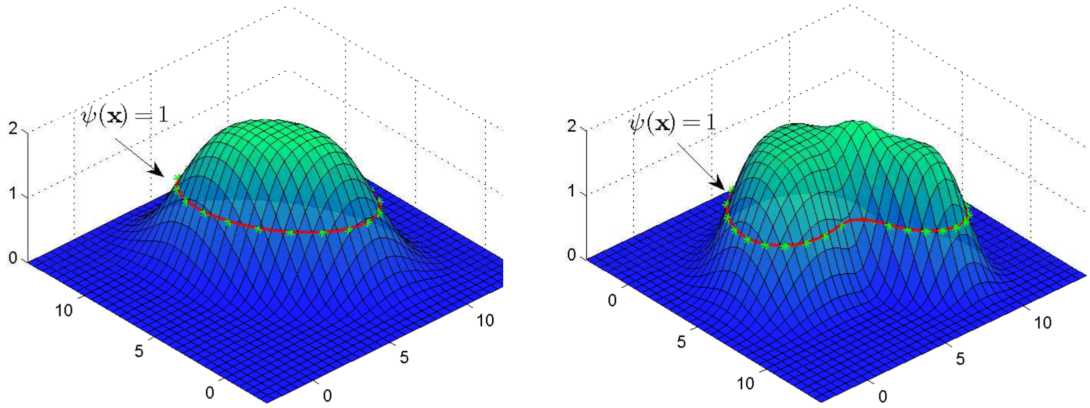
Figure 4.2: The sigmoidal function $\phi(s)$

the distance function s is large, meaning that this particular ϕ makes no contribution far outside its ellipse. Being a combination of cubic B-splines, $\phi(s)$ is relatively easy to calculate. Note that, $\phi(s)$ equals 2 for small s and decreases monotonically as s increases through the interval $\left[1-\frac{3}{2}h, 1+\frac{3}{2}h\right]$, as is shown in Figure 4.2. A single ellipse can be recovered from $\phi(s)$ via the equation $\phi(s) = 1$. Different ellipses give rise to different distance functions s_j. Applying the sigmoidal function we define

$$\psi(\mathbf{x}) = \sum_j \lambda_j \phi(s_j(\mathbf{x})), \tag{4.10}$$

where

$$s_j(\mathbf{x}) = \sqrt{(\mathbf{x}-\mathbf{c}_j)^T\mathbf{A}_j(\mathbf{x}-\mathbf{c}_j)}. \tag{4.11}$$



(a) *Reconstructed curve for a single ellipse* (b) *Reconstructed curve for two intersected ellipses*

Figure 4.3: Plot of $\psi(\mathbf{x})$ for one and two ellipses. The contour $\psi(\mathbf{x}) = 1$ approximating the ellipse(s) is also shown.

The curve corresponding to the combined ellipses is given by $\psi(\mathbf{x}) = 1$ (see Figure 4.3). The λ_j's are chosen to optimize the fit of $\psi(\mathbf{x}) = 1$ to the data points. This is done by performing a least squares solution of the system

$$\psi(\mathbf{x}_i) = \sum_j \lambda_j \phi(s_j(\mathbf{x}_i)) \approx 1, \qquad \text{for all } i, \tag{4.12}$$

where $\mathbf{x}_i$ is the i^{th} data point. This system is usually overdetermined as the number of ellipses is small compared to the number of data points.

4.1 Algorithm

As described earlier, our proposed method begins with the optimization which requires an initial estimate of the parameters. In this section, we present both user-initialized and automatic initialization algorithms for the optimization step. Prototype implementations of our algorithms were written in Matlab.

For user-initialized algorithm, a graphical interface tool in Matlab, `ginput.m` is used. It enables the user to interactively specify some initial parameters, in this case centers of the ellipses. Then the optimization procedure optimizes the ellipses, followed by the process of blending the ellipses together, see Algorithm 3.

The algorithm is direct and very efficient with user input information. However,

Algorithm 3 User-initialized algorithm for coarse level approximation in 2D.

Input: List of data points and user input initial parameters.

Output: Reconstructed curve.

1: User approximates centers of ellipses via a graphical interface tool.
2: Solve constrained optimization problems to obtain ellipses for gross features.
3: Construct for each ellipse a sigmoidal function $\phi(s)$ where s is the square root of a quadratic which has the function value of 1 on the ellipse.
4: Fit by least squares $\psi(\mathbf{x}) = \sum_j \lambda_j \phi(s_j) \approx 1$ to all the data points.
5: Extract the contour implicitly defined by $\psi(\mathbf{x}) = 1$.

there are cases where user interaction might not be applicable, for example when a larger number of data points is involved. Therefore we investigate and set up an automatic algorithm to identify potential centers of ellipses to be optimized one by one. We first aim to optimize the largest inscribed ellipse. This can be done by generating the midpoints of some randomly chosen pairs of data points. Then, choose the midpoint that has the largest distance from its nearest data point to be optimized as the largest inscribed ellipse. A *kd-tree* (De Berg et al., 2008) structure is constructed for finding the nearest point of any point in the given data set.

Once we have identified the first ellipse, subsequent ellipses are added to be optimized by updating the implicit function $\psi(\mathbf{x})$. We make use of equation (4.12) to pick out potential centers of ellipses. Note that any data point on the implicit curve will return a value 1. Therefore we can locate data points that are far away from the first ellipse, and we denote these data points as an uncovered region. To insert an ellipse for optimization in this region, we identify one far away data point $\boldsymbol{x}_{far}$ and its neighboring data points. We fit a best line to this cluster of points and return the inward normal $\mathbf{n}_{in}$ of the line. $\mathbf{n}_{in}$ indicates a good direction to estimate the center of an ellipse in this region. Hence we define

$$\boldsymbol{x}^* = \boldsymbol{x}_{far} + k\, d\, \mathbf{n}_{in}, \tag{4.13}$$

where $\boldsymbol{x}^*$ is the estimated center from $\boldsymbol{x}_{far}$, d is the distance from $\boldsymbol{x}_{far}$ to its closest point and k is some parameter with $1 < k < 2$.

After each optimization step, we calculate the residual sum of squares (RSS),

$$RSS = \sum_{i=1}^{N} \left(\psi(\mathbf{x}_i) - 1\right)^2, \tag{4.14}$$

for all data points. The process of updating the implicit function $\psi(\mathbf{x}) = 1$ and inserting an ellipse is repeated until the change in RSS due to adding a new ellipse becomes insignificant, in this case less than 1.

Algorithm 4 Automatic algorithm for coarse level approximation in 2D.

Input: List of data points and their corresponding inward normals.

Output: Reconstructed curve.

1: Generate 20 midpoints from randomly paired data points.
2: Find closest data point for every midpoint using *kd-tree*.
3: Take midpoint with largest distance to its closest point be the initial center.
4: Solve constrained optimization problem to obtain the first ellipse.
5: **Repeat**
6: Identify $\boldsymbol{x}_{far}$ in the uncovered region.
7: Estimate center of ellipse with $\boldsymbol{x}^* = \boldsymbol{x}_{far} + k\, d\, \mathbf{n}_{in}$.
8: Solve constrained optimization problem to obtain ellipse in the uncovered region.
9: Construct for each ellipse a sigmoidal function $\phi(s)$ where s is square root of a quadratic which has the function value of 1 on the ellipse.
10: Determine the λ_j's by least squares fitting $\psi(\boldsymbol{x}) : \sum_j \lambda_j \phi(s_j(\boldsymbol{x})) \approx 1$ to all the data points.
11: Calculate RSS.
12: **IF** RSS significant, include current ellipse.
13: Until RSS insignificant.
14: Extract the contour implicitly defined by $\psi(\mathbf{x}) = 1$.

4.2 2D Results and Discussion

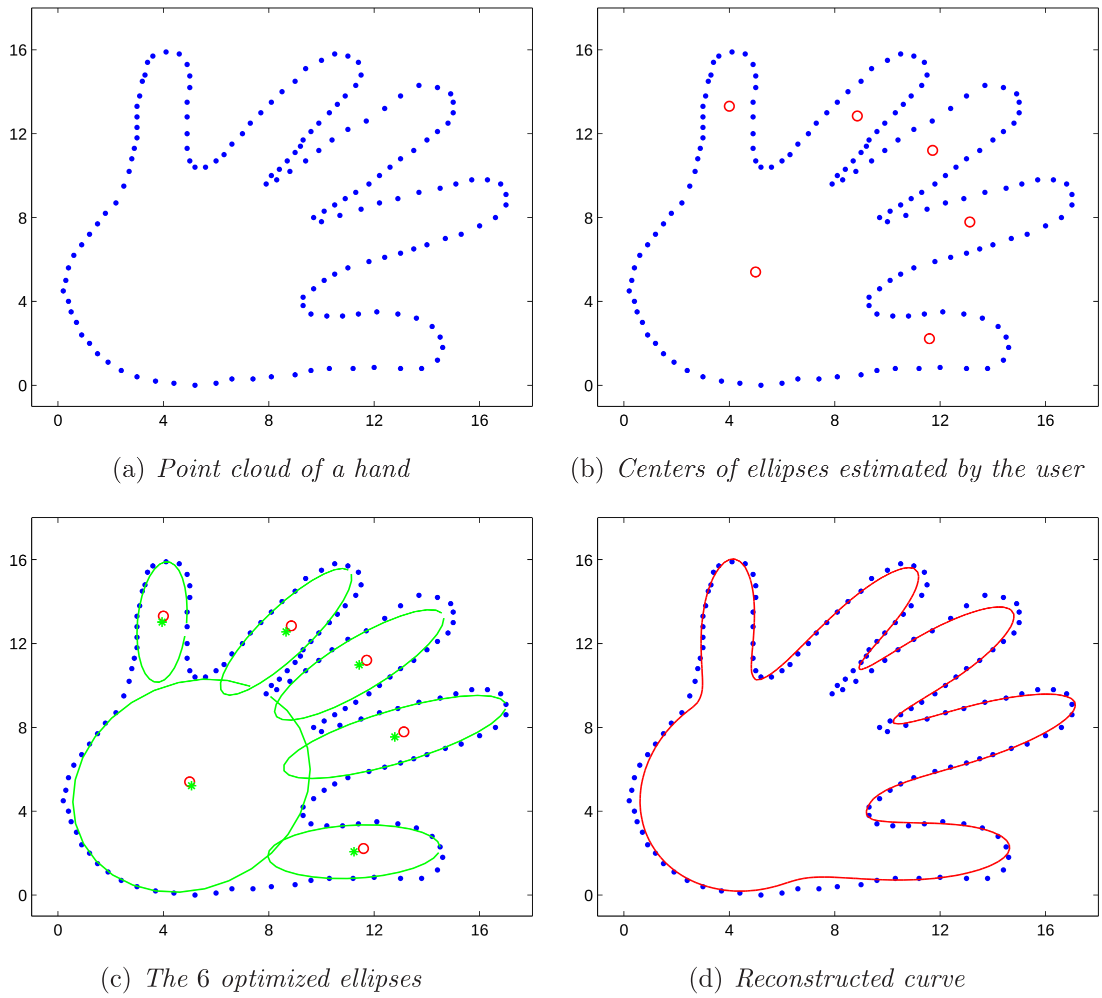
Figure 4.4 shows the process of coarsely reconstructing the curve of a hand using ellipses, with user initialization. The point cloud of a hand in 2D contains 151 vertices as shown in Figure 4.4(a). The user can visualize the point cloud and interactively specifies initial values for ellipse centers. The estimated centers are shown in Figure 4.4(b). At the same time, the number of ellipses to be optimized is determined by the user. Then the optimization procedure optimizes the ellipses. The six optimized ellipses are shown in Figure 4.4(c) together with the estimated centers and original point cloud. In the process of blending the ellipses together, the parameter we used in equation (4.9) is $h = 0.6$. Figure 4.4(d) shows the extracted contour of the implicit function, $\psi(\mathbf{x})$ at 1. The curve of a hand describing the point cloud is clearly reconstructed.

In the optimization steps, the method used was `fmincon.m` active set method which is a sequential quadrature programming method employing a quasi-Newton (BFGS) approximation to the Hessian of the Lagrangian. It has global convergence to a local minimizer or stationary point of the problem, because an exact penalty function is employed (Han, 1977; Powell, 1978a,b).

Each optimization performed halted when the relevant stopping conditions were satisfied. The method halted when the constraint violations and the changes in the predicted or actual function values between successive iterations were less than 1.0×10^{-6}.

The algorithm stopped if either the predicted or actual changes in the objective function fell below 1.0×10^{-6}. but with different output messages on termination. If `fmincon.m` stopped because the predicted change in the objective function is less than 1.0×10^{-6} and constraints are satisfied to within 1.0×10^{-6}, then `fmincon.m` returns the message "Local minimum possible. Constraints satisfied." If optimization completed because the objective function is non-decreasing in feasible directions, to within 1.0×10^{-6}, and constraints are satisfied to within 1.0×10^{-6}, then `fmincon.m` returns the message "Local minimum found that satisfies the constraints."

In general, the optimization problems have multiple local minima. As evidence of this, the various ellipse optimizations for the hand gave 1 ellipse per finger, and one for the hand's palm. Hence, the method will converge to a local minimizer or stationary point with different minimizers or stationary points being located from different initial points.



(a) *Point cloud of a hand*

(b) *Centers of ellipses estimated by the user*

(c) *The 6 optimized ellipses*

(d) *Reconstructed curve*

Figure 4.4: User-initialized curve reconstruction of a hand using ellipses.

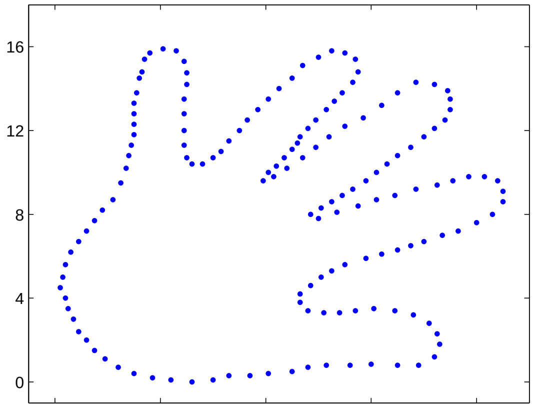
The automatic curve reconstruction process is shown in Figure 4.5. For the same point cloud in 2D, our proposed method managed to estimate the center of the largest inscribed ellipse (marked as red star) and the optimized largest inscribed ellipse is shown in Figure 4.5(b). Figures 4.5(c) to 4.5(h) show the progressive steps in adding ellipses for optimization. In each step, the implicit curve $\psi(\mathbf{x}) = 1$ is updated (marked as red curve) and $\boldsymbol{x}_{far}$ is identified (marked as the red circle). The estimated center, $\boldsymbol{x}^*$ can then be calculated using equation (4.13) with parameter $k = 1.7$. The corresponding optimized ellipse is marked as green curve. Note that the estimated center (marked as red star) can be quite far from the optimized center (marked as green star) for an ellipse. This implies that accurate initial estimation to optimized an ellipse is not needed. A total of seven ellipses were optimized, then the algorithm terminated (see Figure 4.5(i)). Note that the algorithm terminates when the last added ellipse no longer improves the reconstructed curve in describing the point cloud. Therefore we can regard the last added ellipse as a redundant ellipse. The redundant ellipse is removed when blending all the ellipses together where the parameter we used in equation (4.9) is $h = 0.6$. Figure 4.5(j) shows the extracted contour of the implicit function, $\psi(\mathbf{x})$ at 1. The curve of the hand is clearly reconstructed.

We note that different initial estimates can produce slightly different final approximated curves. However the algorithm is reasonably robust as most of the results capture the geometry of the data points. In terms of accuracy of the approximation, we can quantify the error in the final approximation by using the residual sum of squares, that is $RSS = \sum_{i=1}^{N} (\psi(\mathbf{x}_i) - 1)^2$. The idea is similar with detecting a far away point (a point in the uncovered region) from the approximated curve described in the automatic algorithm. If a point x_i is far away from the approximated curve, the value of $\psi(x_i)$ will not be 1 or close to 1. Therefore, we can use RSS as a phantom for the error in the approximation to measure the discrepancy between the points and the approximation. For example, Table 4.1 shows the RSS at each step in the automatic algorithm shown in Figure 4.5. In that case the iteration was stopped when adding a new ellipse changed the RSS by less than 1.

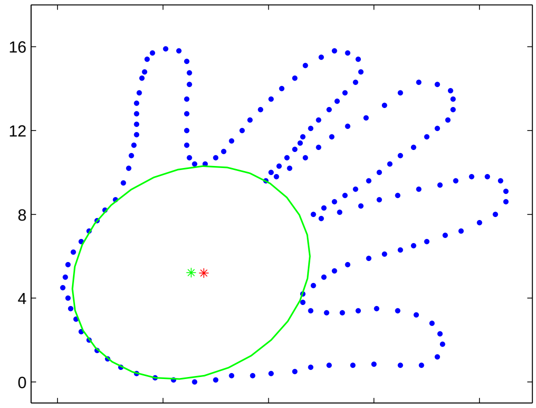
Number of inserted ellipses	1	2	3	4	5	6	7
RSS	87.78	69.02	51.85	37.80	25.44	7.87	7.37

Table 4.1: RSS in the automatic algorithm shown in Figure 4.5.

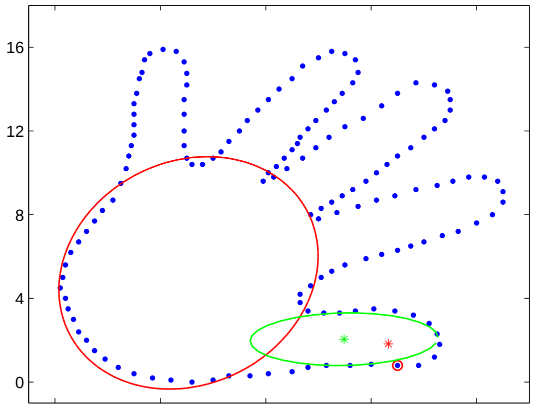
In the test cases we have considered, our proposed algorithm produces a qualitatively correct blobby curve approximation to the given input data set.



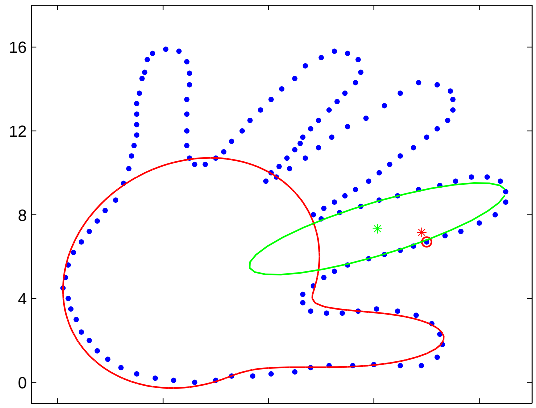
(a) Point cloud of a hand



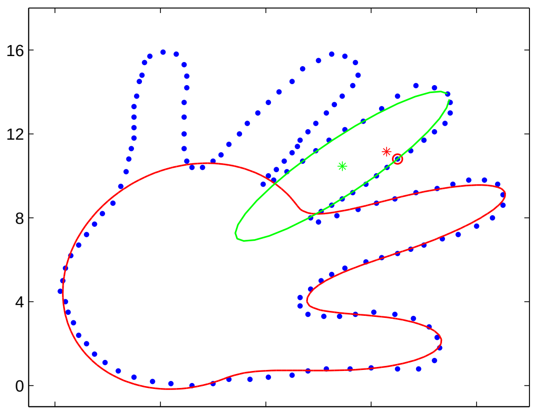
(b) First optimized ellipse



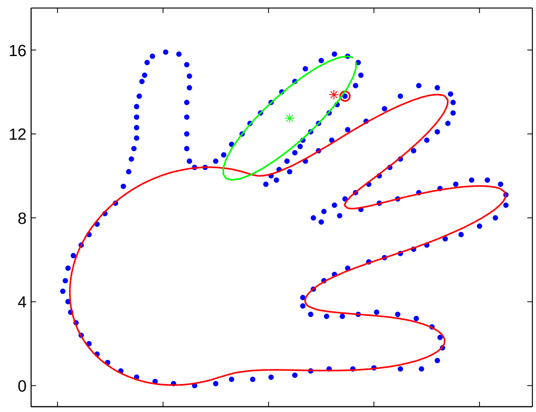
(c)



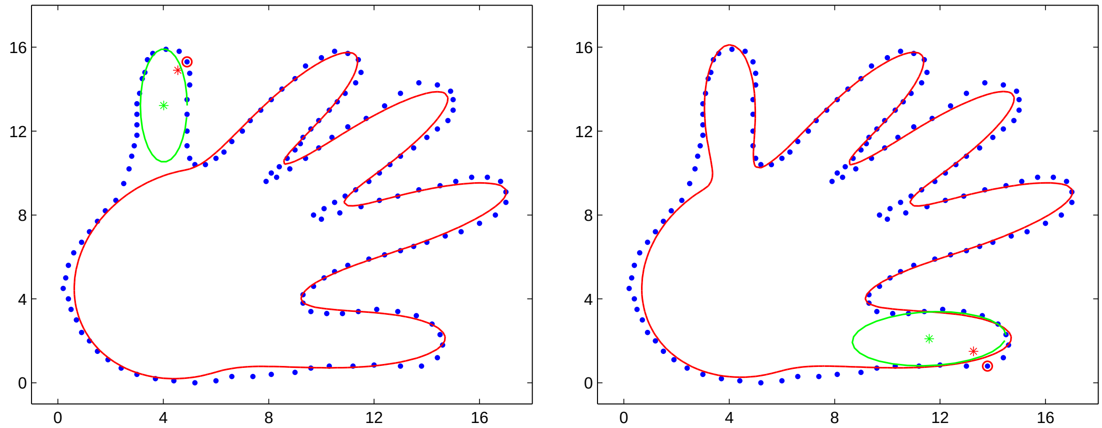
(d)



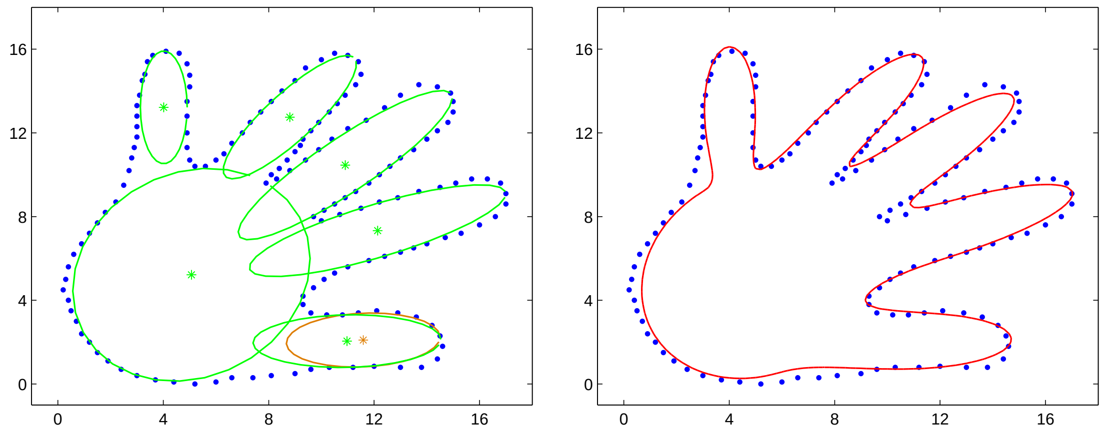
(e)



(f)



(g) (h)



(i) 7 *optimized ellipses when the automatic algorithm terminated*

(j) *Reconstructed curve*

Figure 4.5: Automatic curve reconstruction of a hand using ellipses.

Chapter 5

Fitting Ellipsoids in 3D

In this section, we extend our proposed algorithm in 2D to obtain a coarse shape approximation of a point cloud in three dimensional (3D) space.

Given a set of scattered data points in 3D, (x_i, y_i, z_i) where $i = 1, 2, \cdots, N$, we aim to fit the shape of the scattered points by some ellipsoids. Therefore, let the equation of an ellipsoid be

$$(\boldsymbol{x} - \boldsymbol{c})^T \mathbf{A} (\boldsymbol{x} - \boldsymbol{c}) = 1$$

where $\boldsymbol{x} = (x, y, z)$ is a point on the ellipsoid, $\boldsymbol{c} = (x_c, y_c, z_c)$ is the center of the ellipsoid and $\mathbf{A}$ is a 3 by 3 symmetric positive definite matrix. $\mathbf{A}$ has spectral decomposition $\mathbf{A} = QDQ^T$ where

$$D = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}$$

are the eigenvalues of $\mathbf{A}$ and Q is a 3×3 rotation matrix. These eigenvalues give information about the radii of the ellipsoid, where

$$a = \frac{1}{\sqrt{\sigma_1}}, \quad b = \frac{1}{\sqrt{\sigma_2}}, \quad c = \frac{1}{\sqrt{\sigma_3}} \tag{5.1}$$

as shown in Figure 5.1. Note that a, b and c are all positive real numbers that determine the shape of the ellipsoid.

To describe the orientation of an ellipsoid, we need a suitable representation for the rotation matrix, Q. In 2D, only a single parameter is required to determine the rotation of an ellipse, that is the angle of rotation, θ. It becomes more complicated in 3D as 3D rotations are not commutative, and more parameters are involved. There are many ways to represent a 3×3 rotation matrix. In our work we use Rodrigues' formula

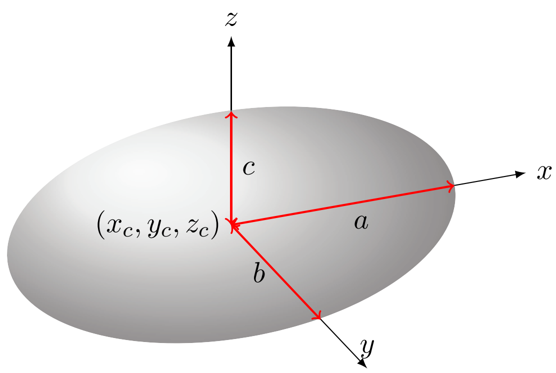


Figure 5.1: Some of the parameters of an ellipsoid used in the constrained optimization step.

for the rotation matrix in exponential representation (see Ma et al. (2004, chap. 2)). It is simpler and more intuitive.

Other choices such as a special orthogonal matrix, Euler angles, or quaternion representation have disadvantages in the optimization procedure. For example, a 3×3 special orthogonal rotation matrix R contains nine entries which will significantly increase the number of parameters to be optimized. Furthermore, it must also satisfy the constraint $R^T R = I$ and the determinant of R must be 1. In using Euler angle representations for a rotation matrix, there are several different conventions can be used. This representation includes bounds that must be dealt with such as defining the reference frame and line of nodes, conditioning the signs and ranges of angles. Quaternions is another rotation representation that is relatively easier to work with than matrices. It is a generalization of the complex numbers. Nevertheless, it consists of four parameters, which is one parameter extra compared to the exponential representation that we used.

Let us now give some details on the expression of a rotation matrix in exponential form.

Lemma 5.0.1. *Given $w \in \mathbb{R}^3$, let $\widehat{w}$ be the skew symmetric matrix*

$$\widehat{w} = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}. \tag{5.2}$$

Then $Q = e^{\widehat{w}}$ is a rotation about axis w through $\|w\| = \sqrt{w_1^2 + w_2^2 + w_3^2}$ radians.

Proof. To show that Q is a rotation matrix, we recall that any rotation matrix can be characterized as an orthogonal matrix with determinant 1. Since $\widehat{w}$ is a skew symmetric matrix, $\widehat{w}^T = -\widehat{w}$. Therefore,

$$\begin{aligned}
QQ^T &= e^{\widehat{w}}(e^{\widehat{w}})^T \\
&= \left(I + \widehat{w} + \frac{\widehat{w}^2}{2!} + \dots\right)\left(I + \widehat{w}^T + \frac{(\widehat{w}^T)^2}{2!} + \dots\right) \\
&= \left(I + \widehat{w} + \frac{\widehat{w}^2}{2!} + \dots\right)\left(I - \widehat{w} + \frac{(-\widehat{w})^2}{2!} + \dots\right) \\
&= e^{\widehat{w}}e^{-\widehat{w}} \\
&= I.
\end{aligned}$$

Hence, matrix Q is orthogonal and the determinant is either $+1$ or -1. Now consider $Q(t) = e^{t\widehat{w}}$ where $0 \le t \le 1$. Then $Q(0) = e^0 = I$ and $\det(Q(0)) = 1$. Since $\det(Q(t)) = \pm 1$ for all t from above, and $\det(Q(t))$ is a continuous function of t, it follows that $\det(Q(t)) = 1$ for all t. That is $Q(1) = e^{\widehat{w}}$ is a rotation.

Also, since $\widehat{w}w = \mathbf{0}$, it follows that

$$Qw = e^{\widehat{w}}w = \left(I + \widehat{w} + \frac{\widehat{w}^2}{2!} + \cdots\right)w = w.$$

Hence the axis of the rotation Q is w.

Note that the angle of rotation, θ, for any rotation matrix R in $\mathbb{R}^3$ can be expressed in terms of the trace of R as

$$\theta = \cos^{-1}\left(\frac{\operatorname{tr}(R) - 1}{2}\right).$$

Also, the trace of a square matrix is the sum of the eigenvalues, or more generally

$$\operatorname{tr}(A^k) = \sum_{i=1}^{n} \lambda_i^k$$

where A is a $n \times n$ matrix and $\lambda_1, \lambda_2, \dots \lambda_n$ are the eigenvalues of A.

Then the eigenvalues of $\widehat{w}$ can be found as follows,

$$\begin{aligned}
|\widehat{w} - \lambda I| &= \begin{vmatrix} -\lambda & -w_3 & w_2 \\ w_3 & -\lambda & -w_1 \\ -w_2 & w_1 & -\lambda \end{vmatrix} \\
&= -\lambda\left(\lambda^2 + w_1^2\right) - w_3\left(\lambda w_3 - w_1 w_2\right) - w_2\left(w_1 w_3 + \lambda w_2\right) \\
&= -\lambda\left(\lambda^2 + w_1^2 + w_2^2 + w_3^2\right) \\
&= -\lambda\left(\lambda^2 + \|w\|^2\right).
\end{aligned}$$

Therefore, the eigenvalues of $\widehat{w}$ are

$$\lambda_1 = 0, \qquad \lambda_2 = \|w\|\mathfrak{i} \quad \text{and} \quad \lambda_3 = -\|w\|\mathfrak{i}.$$

Hence, $\operatorname{tr}\left(\widehat{w}^{\,2k}\right) = (-1)^k\, 2\, \|w\|^{2k}$ and

$$\begin{aligned}
\operatorname{tr}(Q) &= \operatorname{tr}\left(I + \widehat{w} + \frac{\widehat{w}^2}{2!} + \frac{\widehat{w}^3}{3!} + \cdots\right) \\
&= \operatorname{tr}(I) + \operatorname{tr}(\widehat{w}) + \frac{\operatorname{tr}(\widehat{w}^2)}{2!} + \frac{\operatorname{tr}(\widehat{w}^3)}{3!} + \cdots \\
&= 3 + 0 + \frac{-2\|w\|^2}{2!} + 0 + \frac{2\|w\|^4}{4!} + 0 + \frac{-2\|w\|^6}{6!} + \ldots \\
&= 1 + 2\left(1 - \frac{\|w\|^2}{2!} + \frac{\|w\|^4}{4!} - \frac{\|w\|^6}{6!} + \cdots\right) \\
&= 1 + 2\cos(\|w\|)
\end{aligned}$$

It is clear that the angle of rotation for Q is $\|w\|$. □

The matrix exponential of $\widehat{w}$,

$$e^{\widehat{w}} = I + \widehat{w} + \frac{\widehat{w}^2}{2!} + \frac{\widehat{w}^3}{3!} + \cdots \tag{5.3}$$

is alternatively given by the Rodrigues formula

$$e^{\widehat{w}} = I + \frac{\widehat{w}}{\|w\|}\sin(\|w\|) + \frac{\widehat{w}^2}{\|w\|^2}\left(1 - \cos(\|w\|)\right). \tag{5.4}$$

To avoid loss of significance when $\|w\|$ is small, we rewrite equation (5.4) by applying a half angle formula obtaining

$$e^{\widehat{w}} = I + \frac{\widehat{w}}{\|w\|}\sin(\|w\|) + 2\sin^2\left(\frac{\|w\|}{2}\right)\frac{\widehat{w}^2}{\|w\|^2}. \tag{5.5}$$

It is clear from equation (5.5) that

$$\lim_{\|w\|\to 0} e^{\widehat{w}} = I.$$

We therefore define the rotation matrix as,

$$Q = e^{\widehat{w}} = \begin{cases} I + \dfrac{\widehat{w}}{\|w\|}\sin(\|w\|) + 2\sin^2\left(\dfrac{\|w\|}{2}\right)\dfrac{\widehat{w}^2}{\|w\|^2}, & w^T w > 0 \\ I, & \text{otherwise.} \end{cases} \tag{5.6}$$

Using this definition, the optimizer can adjust w without constraint. This is a stable way of evaluating the rotation matrix.

In fitting an ellipsoid, the same optimization procedure in 2D is used with minor modifications. We now have 9 parameters to be optimized, 3 to indicate the ellipsoids center, 3 to represent the lengths of the semi axes, and 3 to represent its orientation, i.e. $\boldsymbol{z} = \begin{bmatrix} x_c & y_c & z_c & a & b & c & w_1 & w_2 & w_3 \end{bmatrix}^T$. The objective function in the constrained optimization remains unchanged, i.e.

$$\min f\big(\boldsymbol{z}\big) = \sum_{i=1}^{N} \frac{\big[\rho_i\big]_+^2}{K + \big[\rho_i\big]_+^2} \tag{5.7}$$

for some $K > 0$ and $\big[\rho_i\big]_+ = \max\big(\rho_i, 0\big)$ where

$$\rho_i = (\boldsymbol{x}_i - \boldsymbol{c})^T \mathbf{A}(\boldsymbol{x}_i - \boldsymbol{c}) - 1. \tag{5.8}$$

The constraints consist of simple bounds on the decision variables

$$\begin{gathered} x_{min} \leq x_c \leq x_{max}\,, \\ y_{min} \leq y_c \leq y_{max}\,, \\ z_{min} \leq z_c \leq z_{max}\,, \\ \ell_{min} \leq a, b, c \leq \ell_{max}\,, \\ -\infty \leq w_1, w_2, w_3 \leq +\infty\,, \end{gathered}$$

together with the nonlinear constraints $c_i = -\rho_i \leq 0$, which prevent any data point from lying inside the optimized ellipsoid.

5.1 Algorithm

For the 3D case, we also developed user-initialized and automatic methods in identifying the initial ellipsoids for coarse level approximation. In this section, both algorithms are presented. The former algorithm is designed such that the user can select the region or cluster of points to be fitted by a single ellipsoid. This can be done by using a software tool called *rbb3select* (Kotliarov, 2005). We have developed a graphical user interface incorporating this tool. It allows user to interactively rotate the point cloud and select some cluster of points. Each cluster of points is then recorded for optimization in isolation. Based on the selected clusters of points, the number of ellipsoids, and the ellipsoids' initial center are determined. Here, we take the center of each cluster of data points to be the initial centers. The optimized ellipsoids are then blended together to form a shape that approximates the point cloud. The detailed algorithm is given in Algorithm 5.

Algorithm 5 User-initialized algorithm for coarse level approximation in 3D.

Input: List of data points and user selected clusters of points.
Output: Reconstructed shape.
1: User selects cluster of points using *rbb3select*.
2: Initial centers are determined based on cluster of points.
3: Solve constrained optimization problems to obtain ellipsoids for gross features.
4: Construct for each ellipsoid a sigmoidal function $\phi(s)$ where s is square root of a quadratic which has the function value of 1 on the ellipsoid.
5: Fit by least squares $\psi(\boldsymbol{x}) = \sum_j \lambda_j \phi(s_j) \approx 1$ to all the data points.
6: Extract the isosurface implicitly defined by $\psi(\mathbf{x}) = 1$.

For automatic initialization in 3D, a similar procedure proposed for the 2D case is used. To begin the fitting process, a small number of ellipsoids will be optimized randomly. This can be done by randomly choosing a few data points. For each such random point, $\boldsymbol{x}_{rand}$, we form an initial center of an ellipsoid by using a modified version of equation (4.13),

$$\boldsymbol{x}^* = \boldsymbol{x}_{rand} + k\, d\, \mathbf{n}_{in}, \tag{5.9}$$

based on these random data points. Note that $\boldsymbol{x}^*$ is the estimated center from $\boldsymbol{x}_{rand}$, $\mathbf{n}_{in}$ is the inward point normal at $\boldsymbol{x}_{rand}$ and d is the distance from $\boldsymbol{x}_{rand}$ to its closest data point and k is some parameter with $1 < k < 2$.

With these initial centers, ellipsoids are optimized. The first ellipsoid is then selected by choosing the largest ellipsoid among them. After that, the process of updating the implicit function ψ, identifying a poorly fitted region and inserting subsequent ellipsoids for optimization in that region is applied. The detailed algorithm is given in

Algorithm 6.

Algorithm 6 Automatic algorithm for coarse level approximation in 3D.

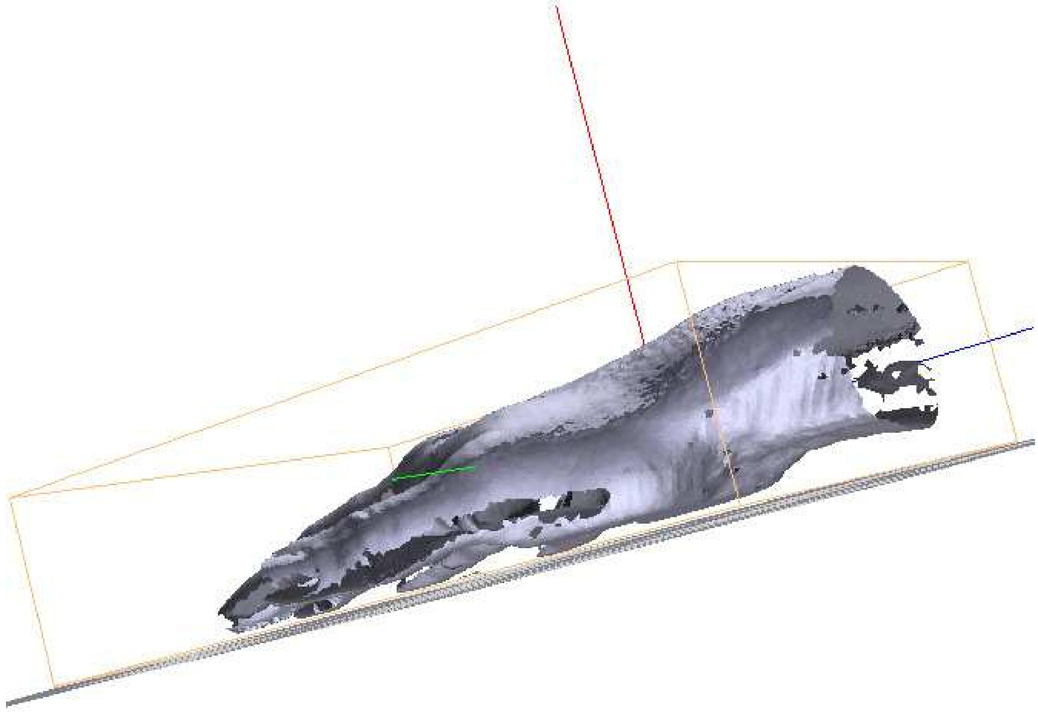
Input: List of data points and their corresponding point normals.
Output: Reconstructed shape.

1: Randomly identify five data points, $\boldsymbol{x}_{rand}$.
2: Estimate center of ellipsoid with $\boldsymbol{x}^* = \boldsymbol{x}_{rand} + k\,d\,\mathbf{n}_{in}$.
3: Solve constrained optimization problem to obtain random ellipsoids.
4: Choose the largest ellipsoid to be the first ellipsoid.
5: **Repeat**
6: Identify $\boldsymbol{x}_{far}$ in the uncovered region.
7: Estimate center of ellipsoid with $\boldsymbol{x}^* = \boldsymbol{x}_{far} + k\,d\,\mathbf{n}_{in}$.
8: Solve constrained optimization problem to obtain ellipsoid in the uncovered region.
9: Construct for each ellipsoid a sigmoidal function $\phi(s)$ where s is square root of a quadratic which has the function value of 1 on the ellipsoid.
10: Determine the λ_j's by least squares fitting $\psi(\boldsymbol{x}) : \sum_j \lambda_j \phi(s_j(\boldsymbol{x})) \approx 1$ to all the data points.
11: Calculate RSS.
12: **IF** RSS significant, include current ellipsoid.
13: Until RSS insignificant.
14: Extract the isosurface implicitly defined by $\psi(\mathbf{x}) = 1$.

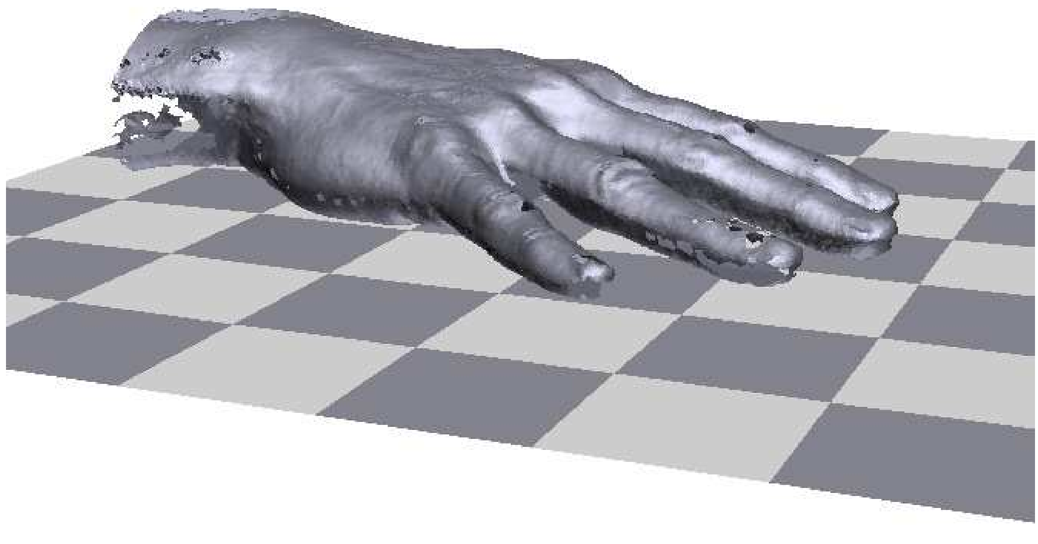
5.2 3D Results and Discussion

The 3D hand model used in our test case contains 1905 data points shown in Figure 5.3. It is a heavily downsampled version of the 57,126 point laser scan supplied to us by Brent Price of Applied Research Associates New Zealand Limited. Even the original data set contains significant holes (see Figure 5.2), particularly on sides of the fingers and the thumb, and at the wrist. The downsampled data set also has holes at the fingertips. Since our optimization algorithm is designed such that any ellipsoid will grow within the data points, ellipsoids may grow through the holes. This problem usually does not occur in the user-initialized algorithm where ellipsoid is optimized within a cluster of data points selected by the user. Therefore, the simple bound constraints are adjusted accordingly to prevent the center of ellipsoid from moving too far from the initial center. Note that the size of ellipsoid is also bounded by the range of the selected cluster of data points. However in the automatic algorithm, the constraints are bounded by the range of the data set allowing the flexibility of growing an ellipsoid. The trade off is some ellipsoids may grow through the holes or even outside the point cloud, for example in between fingers.

Figure 5.4 shows the process of a user-initialized shape reconstruction. In this case, the user selected 8 clusters of points to be optimized as ellipsoids. These clusters are

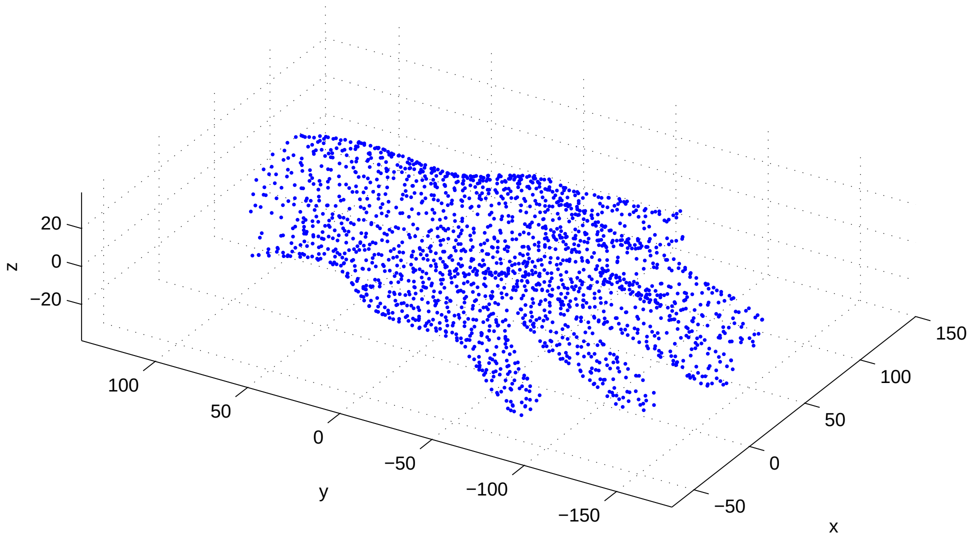


(a)

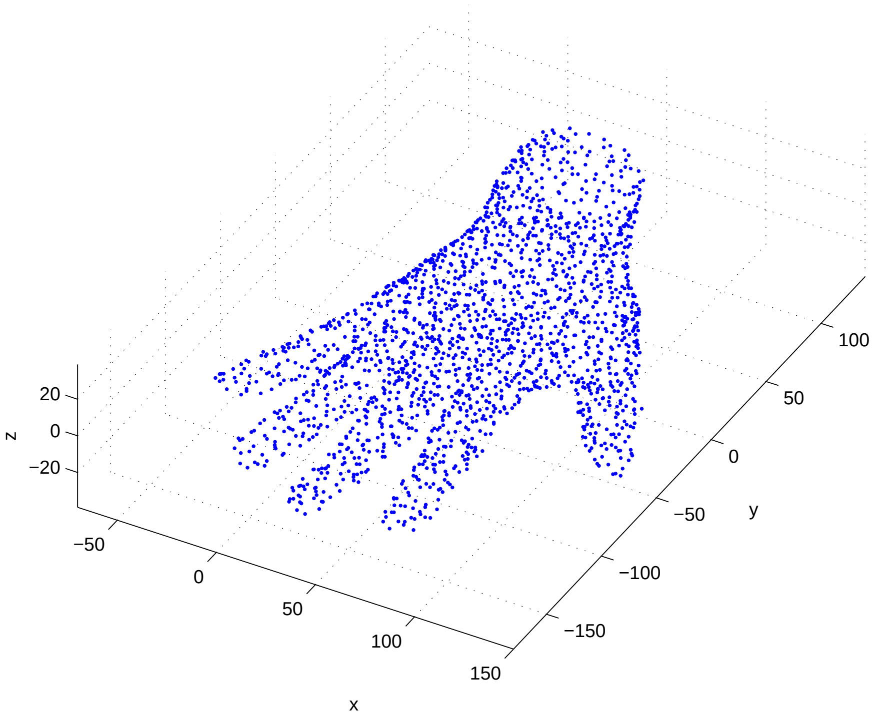


(b)

Figure 5.2: Original data set of a hand with 57, 126 laser scan points.

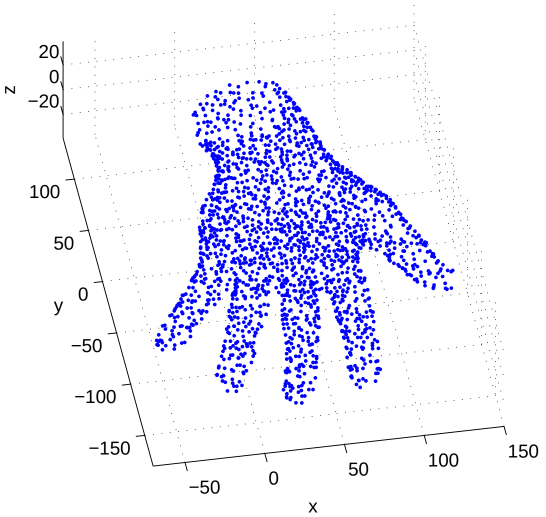


(a)

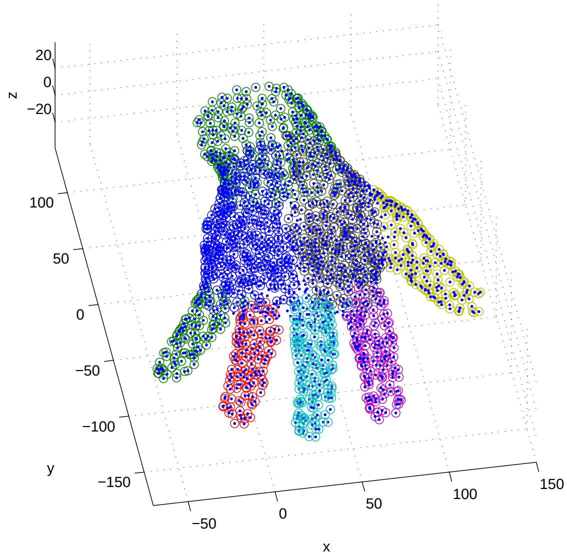


(b)

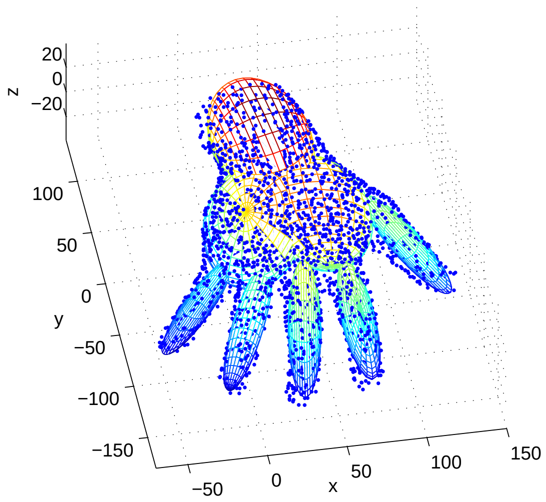
Figure 5.3: Downsampled data set of a hand (1905 data points).



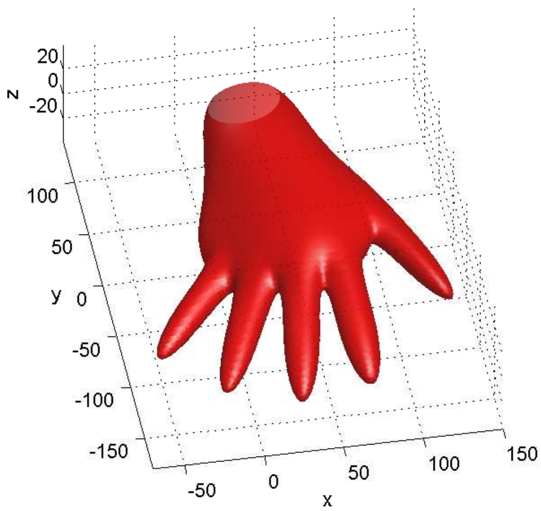
(a) *Point cloud of a hand*



(b) *Clusters of data points selected by the user*



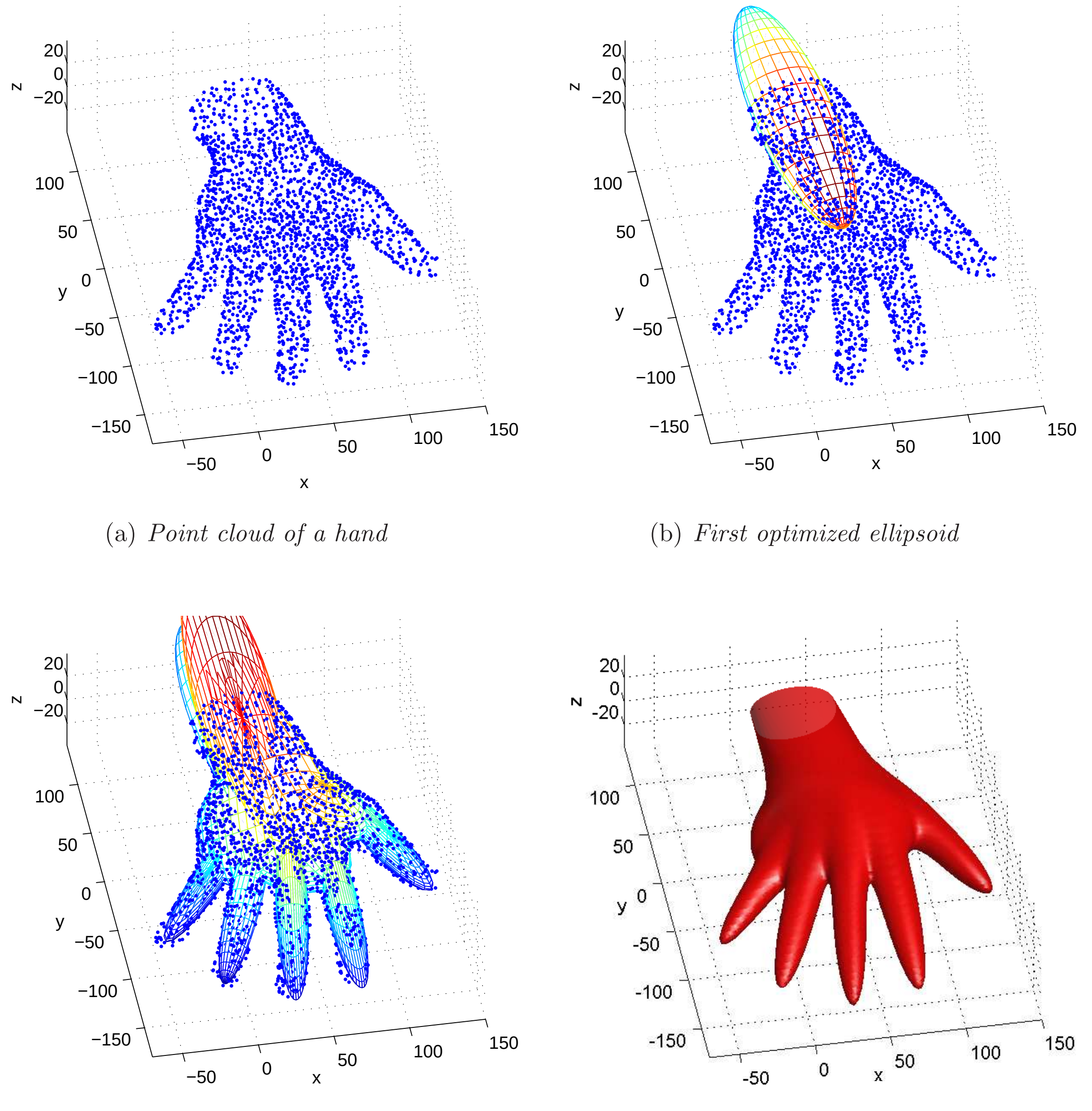
(c) 8 *optimized ellipsoids*



(d) *Reconstructed shape*

Figure 5.4: User-initialized shape reconstruction of a hand using ellipsoids.

highlighted with different colours as shown in Figure 5.4(b). Figure 5.4(c) shows all the optimized ellipsoids with the data points. The final isosurface generated by blending all ellipsoids with $h = 0.7$ is shown in Figure 5.4(d).



(a) *Point cloud of a hand*

(b) *First optimized ellipsoid*

(c) 15 *optimized ellipsoids*

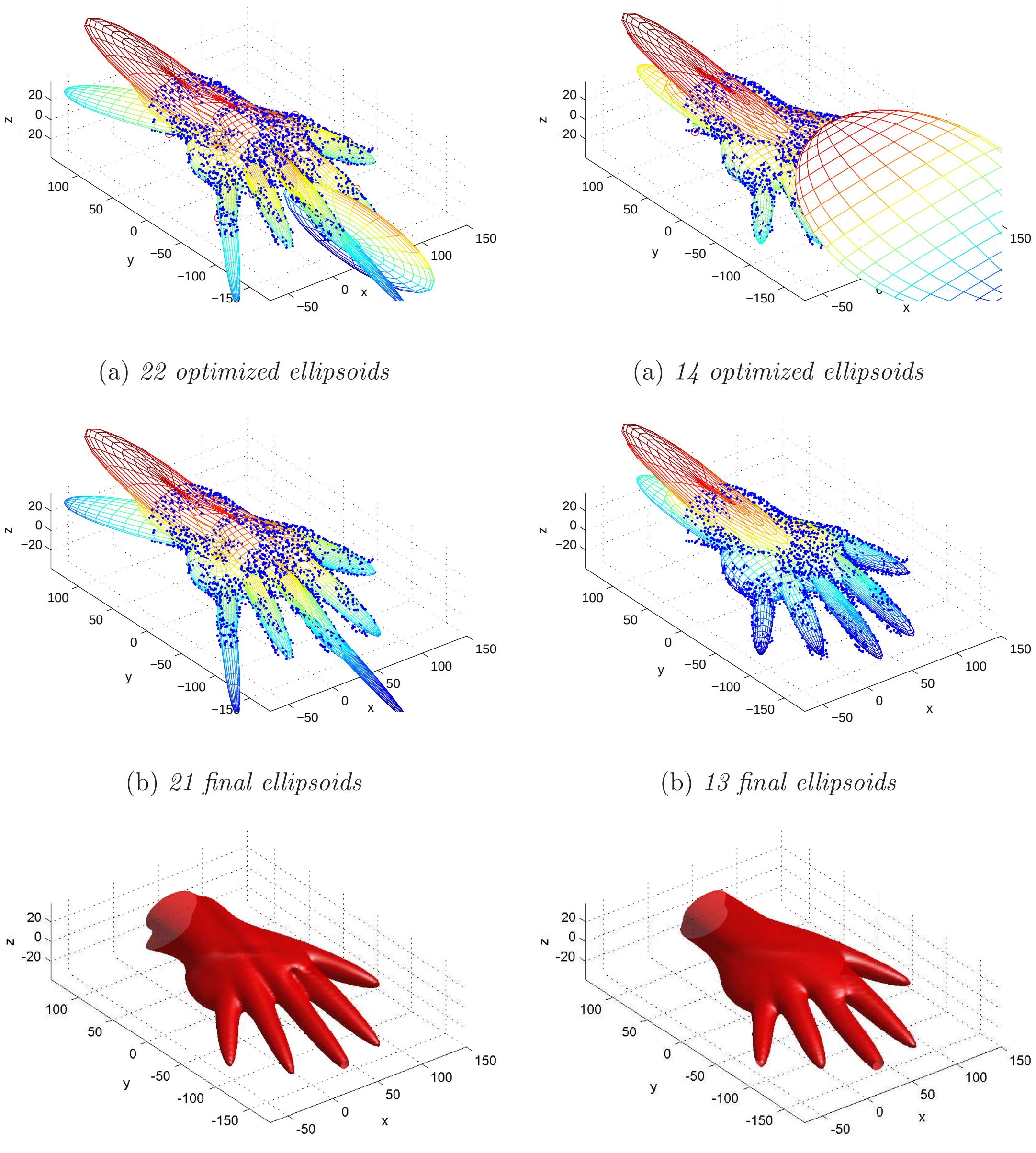
(d) *Reconstructed shape*

Figure 5.5: Automatic shape reconstruction of a hand using ellipsoids.

Figure 5.5 shows an example of reconstructing the shape of a hand automatically. The first optimized ellipsoid shown in Figure 5.5(b) has some part of it growing outside the data range. The back of the wrist has no data points, hence this ellipsoid is allowed to grow through the hole. The final 15 optimized ellipsoids are shown in Figure 5.5(c) using the parameter value $k = 1.3$ in equation (4.13). However, we are only interested in recovering the coarse surface of the approximation within the data range. In regards to that, our final implicit surface is not affected by ellipsoids that partly grow outside the data range. Figure 5.5(d) shows the isosurface of ψ corresponding to $\psi(\mathbf{x}) = 1$ with $h = 0.6$.

To show the consistency of our automatic algorithm, we will show a few examples using the same point cloud. In our experience ellipsoids that grow totally outside the data do not improve the model. These were therefore detected and removed, increasing the simplicity of the model. Detecting such ellipsoids is easily done by comparing normals to the ellipsoids and data normals at data points lying on the ellipsoid. As shown in all the examples, the coarse shape of a hand can be successfully reconstructed.

Example 1 Example 2



(a) *22 optimized ellipsoids* (a) *14 optimized ellipsoids*

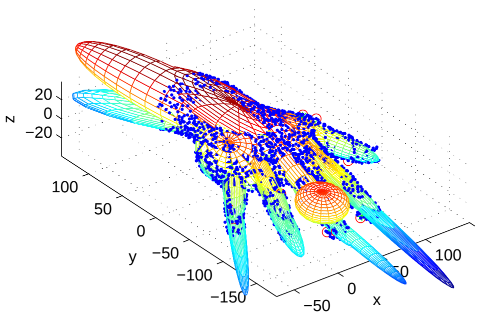
(b) *21 final ellipsoids* (b) *13 final ellipsoids*

(c) *Reconstructed shape* (c) *Reconstructed shape*

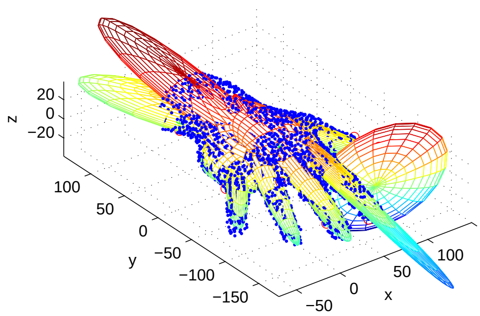
Figure 5.6: 3D hand automatic reconstruction, example 1 and 2.

Example 3

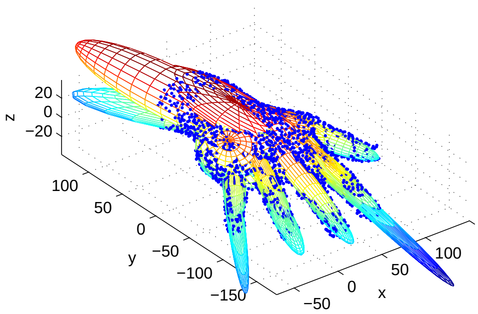
Example 4



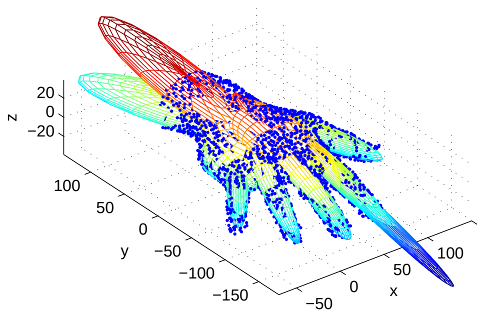
(a) *22 optimized ellipsoids*



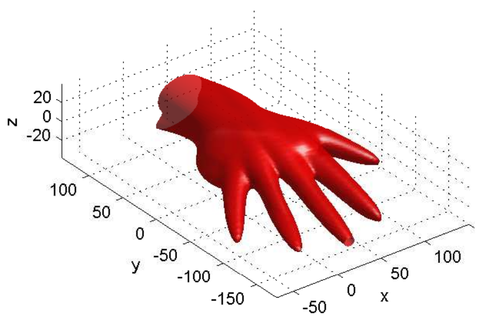
(a) *17 optimized ellipsoids*



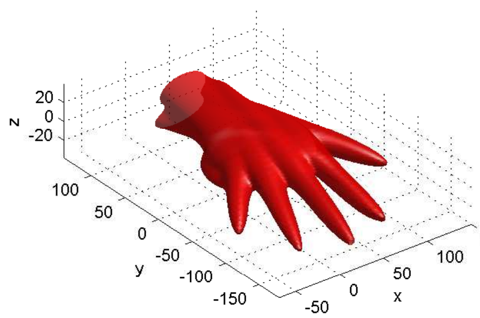
(b) *21 final ellipsoids*



(b) *16 final ellipsoids*



(c) *Reconstructed shape*



(c) *Reconstructed shape*

Figure 5.7: 3D hand automatic reconstruction, example 3 and 4.

5.3 Conclusions and Future Directions

We have demonstrated a method to approximate a given point cloud with an implicit blobby representation. Our work focused on coarse level approximation with relatively few parameters. The results (Ong et al., 2011) show that the reconstructed curves and shapes can describe the original point clouds successfully in both 2D and 3D.

At this stage, we do not report any timing as the approximation is at a coarse level. In the future, we hope to combine the current method and other methods such as implicit radial basis function models or hierarchical radial basis function models for fine level approximation in 3D. Only then, comparison of timing between methods or algorithms for large scale data sets such as the Stanford bunny would be meaningful.

For simple blobby object reconstruction, our proposed method will be sufficient. However, we need to look into cases where the given data points are incomplete or sparse. We would like to be able to detect sparse regions or missing data points (holes in the surface) and fill the holes with smooth surface.

In addition, we would also like to add 'negative ellipses or ellipsoids' as fitting primitives in our method to capture wavy curvature such as a bowl or crescent moon shape.

Chapter 6

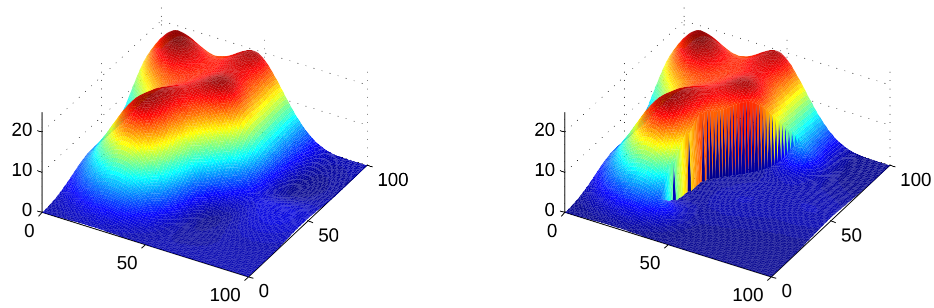
Surface Fitting with Discontinuities

6.1 Introduction

Surface fitting or surface reconstruction is a fast growing research area motivated by its many applications in different fields (Farin, 2002b; Carr et al., 1997). In mathematical description, the problem of surface fitting is to find a function that represents the surface based on some given information about the surface (data points and data values). Given a set of data $x = \{x_1, x_2, \cdots x_N\}$ and the data values $f_i = f(x_i), \quad 1 \leq i \leq N$, we need to find a function s that satisfies $s(x_i) = f_i, \quad 1 \leq i \leq N$. This is equivalent to the problem of scattered data interpolation, where a continuous real valued function of two or more independent variables that interpolates the given data values is sought.

An early survey of method for scattered data interpolation is due to Schumaker (1982). In a recent book, Wendland (2005) provides a thorough description of the concept of scattered data interpolation and approximation. It also covers relevant topics such as multivariate approximation theory and radial basis function theory. Numerous methods with many variants have been investigated and reported for this problem (Weiss et al., 2002; Amidror, 2002; Franke and Nielson, 1991; Alfeld, 1989). In particular, we are interested in methods that uses radial basis functions (Buhmann, 2000, 2003) as a tool to interpolate the scattered data. We shall give more details on radial basis function and discuss some related concepts such as positive definite and compactly supported functions later in this chapter.

Now let us get back to the motivation of this work. We are interested in fitting a surface in the presence of obstacles or barriers. Suppose we are given a set of scattered data with some obstacles or barriers whose location are known a priori. We intend to fit this set of data with a surface showing clearly the discontinuity across the barriers. Note that the data from two sides of the barrier are from two different parts of the surfaces. See Figure 6.1(b) for an example of a smooth surface with discontinuity.



(a) (b)

Figure 6.1: Example of a smooth surface (a) and a surface with discontinuity (b).

The method we proposed here is very suitable for fitting geophysical data in the presence of faults. Geophysical data analysis is an essential part of earth sciences and geophysics related research including environmental modeling. Raw geophysical data often need substantial treatment before they can be used for further analysis. This is due to noise during data collection and the data recorded are generally irregularly sampled. El Abbass et al. (1990) has made a comparison of various algorithms in surface fitting for geophysical data. They have chosen five commonly used methods to investigate and concluded with the suitability and limitations of each method according to the distribution of data and quality of interpolation required. Interpolation and smooth surface fitting of geophysical data using continuous global surfaces can be found in Billings et al. (2002a,b). They have shown a method that is computationally efficient especially on large scale data sets (Beatson et al., 2001b).

Surprisingly, there is not much literature available specifically on surface fitting with known obstacles. Flöry (2009) has demonstrated a method that fits a surface to an unordered point cloud in the presence of obstacles (known geometric primitives such as sphere, cylinder, and cube). They first solved the fitting problem iteratively as an optimization problem. They then derived constraints to the optimization from the obstacles and reconstruct a B-spline surface avoiding these obstacles.

Some other relevant work is fitting a set of scattered data while detecting or preserving discontinuities. This approach is more like a technique to identify and distinguish between noise and discontinuity in the data. Eric et al. (1985) suggested a method that is able to correctly detect discontinuities in the data set provided the existing noise in the data set has relatively small amplitude compared to the discontinuity. The method of detecting discontinuities is by using a local statistical analysis on the residuals (difference between the actual and approximated value) of the data. Sinha and Schunck (1992) proposed a two-stage algorithm for reconstructing a surface from scattered data while preserving discontinuities. The first stage uses a robust regression local approximation to remove outliers and grid the data from the scattered data. The second stage follows by using the gridded data that contains discontinuities to construct a piecewise smooth surface using a weighted bicubic splines approximation.

6.1.1 Problem Statement

In our work, we investigate a method to interpolate a surface from scattered data that includes faults or barriers. Note that physical attributes such as height (or depth) will be discontinuous across the faults.

We approximate a real valued function f by a function s, given the set of values $f = (f_1, f_2, \cdots f_N)$ at the distinct data points $x = \{x_1, x_2, \cdots x_N\} \in \mathbb{R}^d$. One simple way is to use radial basis functions to interpolate a function with N data points by using N radial basis functions centered at these data points. Radial basis functions (Buhmann, 2000, 2003) can be viewed as circularly-symmetric functions centered at a given point. Thus, the interpolant function becomes

$$s(x) = \sum_{i=1}^{N} \lambda_i \, \phi\big(\|x - x_i\|\big), \tag{6.1}$$

where each x_i is a known distinct data point (center) and λ_i is the weight of the radial basis function located at that point. Here, $\|x - x_i\|$ is a metric and $\phi : [0, \infty) \to \mathbb{R}$. The weights (coefficients) λ_i can be found from the interpolation conditions

$$s(x_i) = f_i, \quad 1 \leq i \leq N, \tag{6.2}$$

which lead to the requirement that the interpolation matrix $\mathbf{A}$ be positive definite, where

$$\mathbf{A} = \begin{pmatrix} \phi(\|x_1 - x_1\|) & \phi(\|x_1 - x_2\|) & \cdots & \phi(\|x_1 - x_N\|) \\ \phi(\|x_2 - x_1\|) & \phi(\|x_2 - x_2\|) & \cdots & \phi(\|x_2 - x_N\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|x_N - x_1\|) & \phi(\|x_N - x_2\|) & \cdots & \phi(\|x_N - x_N\|) \end{pmatrix}. \tag{6.3}$$

For settings such as $\mathbb{R}^d$ with the usual Euclidean distance, there is some beautiful theory identifying classes of strictly and conditionally positive definite functions ϕ.

Interpolation problems corresponding to such ϕ's are guaranteed to be poised, that is have a unique solution (Micchelli, 1986; Buhmann, 2003; Wendland, 2005). Unfortunately, in our setting, conditions implying ϕ is strictly positive definite have not been identified. Nevertheless, in practice we have not had problems with singularity of matrices. Moreover, we can always replace interpolation by least squares approximation if necessary.

In the cases when the radial basis function used is only conditionally positive definite, for example the thin-plate splines, it is necessary to include a low degree polynomial $p(x)$ to ensure the poisedness of the interpolation. Hence, the interpolant functions becomes:

$$s(x) = \sum_{i=1}^{N} \lambda_i \, \phi\big(\|x - x_i\|\big) + p(x). \tag{6.4}$$

The weights (coefficients) λ_i, must satisfy Equation (6.2). Further, the side conditions

$$\sum_{i=1}^{N} \lambda_i \, \hat{p}(x_i) = 0, \tag{6.5}$$

for all polynomials $\hat{p}$ of degree at most $k-1$ are imposed on the coefficients. Let $\{p_1, p_2, \cdots p_\ell\}$ be a basis for polynomials of degree at most $k-1$ and let $\boldsymbol{c} = \{c_1, c_2, \cdots c_\ell\}$ be the coefficients that give $\hat{p}$ in terms of this basis. Then Equation (6.2) and (6.5) may be written in matrix form as

$$\begin{pmatrix} \mathbf{A} & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{c} \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{6.6}$$

where $\mathbf{A}$ is as defined in Equation (6.3), $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_N)^T$ and $P = p_{ij} = p_j(x_i)$, for all $1 \le i \le N$, and $1 \le j \le \ell$. Hence, solving the linear system (6.6) determines $\boldsymbol{c}$ and $\boldsymbol{\lambda}$.

Similarly, in the case of no polynomial part, the parameters $\boldsymbol{\lambda}$ of the interpolant are given by solving

$$\mathbf{A}\boldsymbol{\lambda} = f. \tag{6.7}$$

In the context of our problem, we replace the Euclidean norm $\|\cdot\|$ in Equation (6.1) by the 'actual' Euclidean distance, that is the Euclidean distance between two points avoiding the faults. Let us now define the interpolant functions,

$$s(x) = \sum_{i=1}^{N} \lambda_i \, \phi\big(d(x, x_i)\big), \quad x \in \mathbb{R}^d, \tag{6.8}$$

where λ_i is a real valued weight and $d(x, x_i)$ denotes the geodesic distance (shortest distance) from x to x_i not crossing the faults. Using this distance is intuitively reasonable and gives the desired result in test cases shown later in Chapter 8.

Definition 6.1.1. *Shortest distance metric*
The shortest distance, $r = d(a, b)$ is a metric such that for all $a, b, c \in \mathbb{R}^2$,

1. $d(a, b) \geq 0$ *and* $d(a, b) = 0$ *if and only if* $a = b$
2. $d(a, b) = d(b, a)$
3. $d(a, b) \leq d(a, c) + d(c, b)$

6.1.2 Using Compactly Supported Radial Basis Functions

The usefulness of radial basis functions for interpolation or approximation is now well established, see for example Powell (1987), Carr et al. (2001), Wang and Liu (2002) and Buhmann (2003). Some of the well known and commonly used types of radial basis functions are given in Table 6.1.

Thin plane splines	$\phi(r) = r^2 \log(r)$
Gaussians	$\phi(r) = e^{-\alpha r^2}$
Multiquadrics	$\phi(r) = \sqrt{\alpha^2 + r^2}$
Inverse Multiquadrics	$\phi(r) = \dfrac{1}{\sqrt{\alpha^2 + r^2}}$
Inverse Quadrics	$\phi(r) = \dfrac{1}{1 + r^2}$
Polyharmonic splines	$\phi(r) = r^k$, with $k = 1, 3, 5, \ldots$, when d is odd $\phi(r) = r^k \log(r)$, with $k = 2, 4, 6, \ldots$, when d is even

Table 6.1: Types of radial basis functions where $r = \|x - x_i\|$ and α is a positive parameter.

These radial basis functions are globally supported and conditionally positive definite. As mentioned earlier in this chapter, a positive definite radial basis function is desired to ensure the well-posedness of the problem (there exists a unique solution to the problem if and only if the interpolation matrix $\mathbf{A}$ is non-singular, see Buhmann (2003)). Therefore it is natural to restrict the choices of ϕ in Equation (6.8) to locally supported positive definite radial basis functions.

One family of locally (compactly) supported radial basis functions which are strictly positive definite was developed by Wendland (1995, 1999). He has solved the minimum degree polynomial solution for compactly supported functions which has the general form of

$$\phi(r) = \begin{cases} (1-r)^p p(r) & 0 \le r < 1, \\ 0 & \text{otherwise,} \end{cases} \tag{6.9}$$

where p represents a univariate polynomial. A function of the form of Equation (6.9) is a positive definite function on $\mathbb{R}^d$ (See Wendland (2005, Theorem 6.2)). He also derived various compactly supported functions based on the smoothness (continuity, C^k) and space dimension (d) where the function is positive definite (See Table 6.2). More compactly supported functions were proposed (Wu, 1995; Wong et al., 2002; De Boer et al., 2007) based on Wendland functions. A comparison of their behaviors can be found in Menandro et al. (2010).

Positive Definite	**Function**	**Smoothness** (C^k)
$\mathbb{R}$	$\phi(r) = (1-r)$	C^0
$\mathbb{R}$	$\phi(r) = (1-r)^3(3r+1)$	C^2
$\mathbb{R}$	$\phi(r) = (1-r)^5(8r^2+5r+1)$	C^4
$\mathbb{R}^3$	$\phi(r) = (1-r)^2$	C^0
$\mathbb{R}^3$	$\phi(r) = (1-r)^4(4r+1)$	C^2
$\mathbb{R}^3$	$\phi(r) = (1-r)^6(35r^2+18r+3)$	C^4
$\mathbb{R}^3$	$\phi(r) = (1-r)^8(32r^3+25r^2+8r+1)$	C^6
$\mathbb{R}^5$	$\phi(r) = (1-r)^3$	C^0
$\mathbb{R}^5$	$\phi(r) = (1-r)^5(5r+1)$	C^2
$\mathbb{R}^5$	$\phi(r) = (1-r)^7(16r^2+7r+1)$	C^4

Table 6.2: Wendland functions (Wendland, 2005).

The functions given in Table 6.2 have radius of support $0 \le r \le 1$. If a different radius of support is needed, scaling the support of ϕ can be easily done by $\phi(r/\alpha)$ where α is the desired radius of support. Schaback (1995), Floater and Iske (1996), and Morse et al. (2005) are works that apply compactly supported radial basis functions to interpolate surfaces from scattered data. Morse et al. (2005) emphasized that the local nature of compactly supported radial basis functions has increased computational efficiency on large data set due to the sparse interpolation matrix generated by these functions.

In the next chapter, we will describe the method we used to compute $d(a, b)$, the geodesic distance (shortest distance) between two given points a and b without crossing some given obstacles. This lead us to the shortest path problem, a common topic in motion planning (Latombe, 1999). Here, we consider faults or barriers as obstacles in $\mathbb{R}^2$ that are simply line segments that can be disjoint or connected. The technique of using the visibility graph to solve the shortest path problem will also be described in detail in the next chapter.

Chapter 7

Shortest Path Problems

This chapter considers shortest path problems. It reviews the literature and particularly the visibility graph approach of Overmars and Welzl (1988) and Alt and Welzl (1988). Some extensions are made to make the method applicable to polygonal boundaries with nodes not in a general position.

Computing the shortest path between two nodes is a fundamental problem in computational geometry. It is also a well-known and natural problem in network optimization (route or path planning) that has many applications including operations research, real road networks in transportation (Zhan and Noon, 1998; Jacob et al., 1999; Gonzalez et al., 2007), geographic information systems (GIS), routing in data networks, industrial automation, and motion planning in robotics (Latombe, 1999; Choset et al., 2005; Siciliano et al., 2009).

The basic form of the problem is: Given a collection of obstacles in a specified domain or environment, find a shortest path avoiding obstacles between two given nodes. This path is also known as the optimal collision-free path or geodesic path. More complicated problems can be classified according to some parameters, such as

- Dimension: two dimensions (plane), three dimensions (space), or higher dimension?
- Input geometry: complete geometry of the problem (information of the obstacles and environment) is known or unknown?
- Path Constraints: Is there any restriction on the path between the nodes? Are there nodes that must be visited along the path?
- Static or dynamic environment: Are the obstacles movable?

The type of shortest path problem usually determines the design of the algorithm and the technique used to solve the problem efficiently. There exists a large body of literature on shortest path algorithms (Guibas et al., 1987; Gallo and Pallottino, 1988; Mitchell, 1991; Ahuja et al., 1990; Kapoor et al., 1997; Hershberger and Suri, 1999). Time and storage complexity of these algorithms haven often been analyzed and reported. An optimal algorithm is often desired for practical implementation on real-life problems. Though this area has been well and widely studied, there still exist several open problems and implementation issues like robustness, see the survey by Mitchell (2000).

At this stage, we will give details of our shortest path problem, define some notation and basic terminologies, and review some papers that are relevant to our work. We are interested in computing the shortest Euclidean distance between two given nodes in a plane, avoiding segment type obstacles or barriers. Generally, there are two approaches to solve a shortest path problem with multiple obstacles in a plane: the shortest path map method and the visibility graph method. The first approach builds a subdivision of the plane for a given source point. This produces a map with multiple regions where all the points in a specific region will have the same path sequence in their shortest path to the source point. Mitchell (1993) uses some advanced range searching data structures to compute the shortest path map and published an $O(n^{5/3+\varepsilon})$ time and space algorithm. He later improved the running time to $O(n^{3/2+\varepsilon})$ for any fixed $\varepsilon > 0$ (Mitchell, 1996). Hershberger and Suri (1999) developed an optimal algorithm which runs in the worst-case time $O(n \log n)$ and $O(n \log n)$ space. Their algorithm, based on the implementation of wavefront propagation among polygonal obstacles, has significantly improved the time complexity of previously published algorithms. However, the algorithm appears to be rather complicated and it is difficult for us to implement in our application.

In our work, we use the second approach where we first construct a visibility graph based on the set of obstacles, the source node and the goal node. Then the shortest path between the nodes can be obtained by Dijkstra's algorithm on this graph (Dijkstra, 1959; Cormen et al., 2001).

Before we go into the details on the visibility graph method, let us now define the notation and terminology used throughout the rest of the thesis.

- Nodes (V): A set of points or vertices of obstacles (vertices of polygonal obstacles or endpoints of line segment obstacles), source node, and goal node.

- Edges (E): A set of line segments or arcs connecting two mutually visible nodes, which can be either directed or not. Weighted edges refer to edges with values or weights attached to them. The weight can be duration, costs, or length, depending on the objective of the problem.

- Visibility Graph (VG): A graph VG consists of a set of nodes V and a set of edges E. It can be written as $VG = (V, E)$. It is also known as a roadmap or network.
- A path in VG is a sequence of edges, such that consecutive edges share a common node, and each node belongs to at most two edges on the path.
- The total cost of a path in VG is the sum of the costs of the edges on the path.
- A shortest path between two nodes P and Q, denoted by $SP(P, Q)$, is a path connecting P and Q that has the minimum cost, assuming such a path exists.
- The shortest distance between P and Q, denoted by $d(P, Q)$, is the length of $SP(P, Q)$, or infinity if no path exists.

Here we give the basic idea of solving the shortest path problem using the visibility graph method (De Berg et al., 2008, Chap. 13 and 15). Given a set of obstacles $\mathcal{S}$, a source node P and a goal node Q (see example in Figure 7.1), we can compute the shortest collision-free path from P to Q that does not intersect any of the obstacle edges as follows.

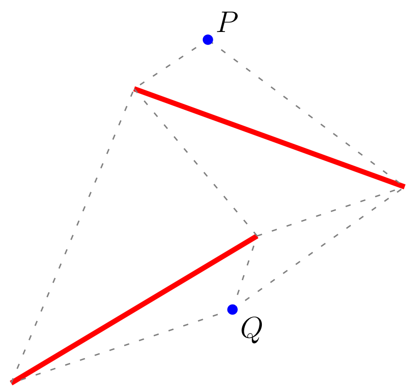
Step 1 Construct the visibility graph VG given $\mathcal{S}$, P, and Q where every edge $\overline{v_1 v_2}$, has the Euclidean length of the segment $d(v_1, v_2)$ as the weight.

Step 2 Use Dijkstra's algorithm to compute shortest path between P and Q.

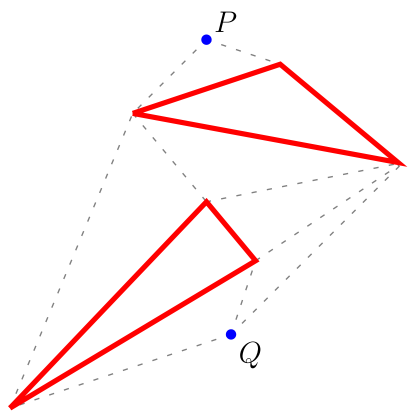
Figures 7.1(a) and 7.1(b) show examples of visibility graphs for segments and polygonal obstacles. Figures 7.1(c) and 7.1(d) show the corresponding shortest paths from P to Q avoiding the obstacles. The shortest path consists of the source and goal nodes linked by a set of edges. The following lemma gives the description of a shortest path for polygonal obstacles.

Lemma 7.0.2. *(De Berg et al., 2008, Lemma 15.1) A shortest path between two nodes that avoids a set $\mathcal{S}$ of polygonal obstacles is a polygonal path whose connecting nodes are nodes of the obstacles $\mathcal{S}$.*

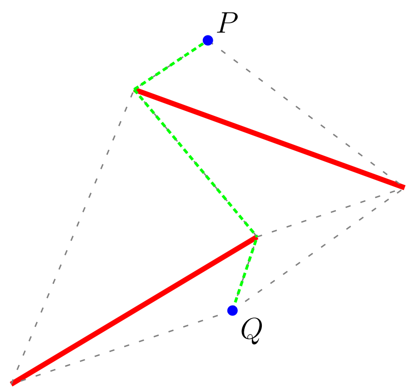
From Lemma 7.0.2, a shortest path is linked by a sequence of edges connecting the source and goal nodes. Each of these edges does not intersect the interior of any obstacles and the endpoints of the edges must be mutually visible. Note that the endpoints of a segment of an obstacle are always visible to each other. This leads us to the definition of a visibility graph. The definition of a visibility graph is well described in Choset et al. (2005, Sec. 5.1.1). Here, we give a simple version of the definition.



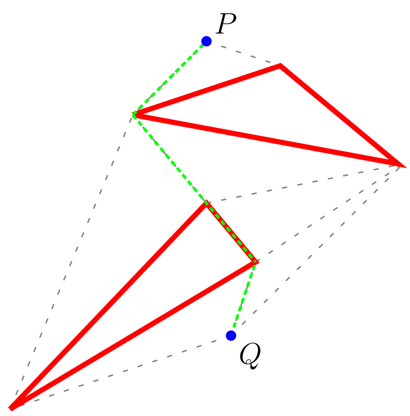
(a) Visibility graph for segment obstacles



(b) Visibility graph for polygonal obstacles



(c) Shortest Path for (a)



(d) Shortest Path for (b)

Figure 7.1: The thin dashed gray lines represent the edges of the visibility graph for the problem. The thick dashed green lines represent the shortest path from P to Q.

Definition 7.0.3. *Visibility Graph*
The visibility graph of a source node (P), goal node (Q), and the obstacle set $\mathcal{S}$ is a graph whose nodes are P, Q, and the obstacle nodes. Nodes v_1 and v_2 are joined by an edge if v_1 and v_2 are either mutually visible or if $\overline{v_1 v_2}$ is an edge of some obstacle.

Using the visibility graph method to solve shortest path problems is well studied (Ghosh, 2007; O'Rourke, 1987). Let $\mathcal{S}$ be an obstacle set consisting of a set of edges and n be the total number of endpoints of edges in $\mathcal{S}$. Then, the total number of nodes in the visibility graph is $n + 2$, when the source and goal nodes are not obstacle nodes. The number of edges of the visibility graph of $\mathcal{S}$ is therefore bounded by $\binom{n+2}{2}$. The direct method for constructing the visibility graph, is by checking whether the segment connecting two nodes intersects any segments of an obstacle. For a particular node v, there are $O(n^2)$ checks to determine which nodes are visible from v. This straightforward (naive) method takes $O(n^3)$ time and it is the first algorithm given by Lozano-Pérez and Wesley (1979). They also showed that, by using Dijkstra's shortest graph path algorithm, the shortest path problem for polygonal obstacles can be solved in $O(n^2)$ time. Lee (1978) and Sharir and Schorr (1986) designed the first efficient algorithm that runs in $O(n^2 \log n)$ time and takes $O(n^2)$ storage space. Later Asano et al. (1986) and Welzl (1985) developed an $O(n^2)$ time algorithm for this problem which is worst-case optimal since the largest possible size of a visibility graph can be $O(n^2)$. Edelsbrunner and Guibas (1989) then improved the working storage to $O(n)$ by using the technique of topologically sweeping an arrangement of lines that allows all nodes to be visited in a consistent ordering.

If the visibility graph is sparse, for example, when there are many densely arranged obstacles, the number of edges ($|E|$) in the visibility graph may be much smaller than its worst-case size of $O(n^2)$. Therefore, output-sensitive algorithms that depend on $|E|$, the size of the visibility graph have been developed. Hershberger (1987) presented an $O(|E|)$ algorithm for constructing the visibility graph of a simple polygon. Sudarshan and Pandu Rangan (1990) proposed an $O(|E| \log^2 n)$ algorithm for computing sparse visibility graph of a polygon with holes that needs $O(n)$ storage space. At the same time, Ghosh and Mount (1991) published an algorithm that takes $O(|E| + n \log n)$ time and $O(|E|+n)$ storage space for the same problem. Although the algorithm is claimed to be optimal in time, the authors remarked on the high complexity of its implementation due to the technique used in locating the visible segments through fast traversal of the enhanced visibility graph. Overmars and Welzl (1988) also gave an output-sensitive algorithm that computes the visibility graph for a set of disjoint polygonal obstacles. It runs in $O(|E| \log n)$ time and uses $O(n)$ storage space. This algorithm is relatively simpler to implement and runs efficiently, although it is not optimal. More efficient algorithms are known for special cases such as parallel line segment obstacles, rectangular obstacles, or when the triangulation of the obstacle space is given (Lee and Preparata, 1984; Storer and Reif, 1994; Guibas et al., 1987; Henzinger et al., 1997; Kapoor et al., 1997).

We have described the shortest path problems and given some background on how the visibility graph is used to solve this problem. In our work, we applied the idea in Overmars and Welzl (1988) to construct the visibility graph. Some modifications were needed to adapt their algorithm to our somewhat different problem. In the subsequent sections, we will give their algorithm and explain the method clearly. We will also highlight the changes we have made in the procedure.

The rest of the chapter is organized as follows. In Section 7.1 we illustrate the technique for sorting all other nodes by counter-clockwise order around a node. This rotation tree method is first proposed by Welzl (1985), then presented in detail in Overmars and Welzl (1988). In Section 7.2 we describe how the visibility graph for disjoint segment obstacles is constructed using the rotation tree technique. In Section 7.3 we extend the method to construct the visibility graph for disjoint polygonal obstacles. We then describe a shortest path algorithm, Dijkstra's algorithm, in the last section of this chapter.

7.1 Rotation Trees

In this section we present Overmars and Welzl's technique (1988) for sorting the order of other nodes by angle around a particular node. We will show that the sorted angular orders for all nodes can be computed in $O(n^2)$ time. Based on this, the slopes of all segments between a node and all other nodes can be given in the sorted angular order around that node. We will then show that the visibility of this node can be computed in $O(n)$ time.

Edelsbrunner and Guibas (1989) have solved the problem by using dualization and topologically sweeping method in optimal $O(n^2)$ time and $O(n)$ storage. However, Overmars and Welzl (1988) proposed a rather simple method with the same complexity without using dualization. The core idea of their method is to maintain a tree on the set of nodes during one rotational or angular sweep and report for each node p, all the nodes to the right of p in counter clockwise order.

Let $\mathcal{S}$ be a finite set of disjoint line segments in the plane and V denote the finite set of endpoints of the line segments in $\mathcal{S}$. It should be noted that, the nodes in V are assumed to be in general position, that is,

1. no lines through pairs of nodes in V are parallel,
2. no three nodes in V are collinear,
3. no vertical line contains two nodes in V.

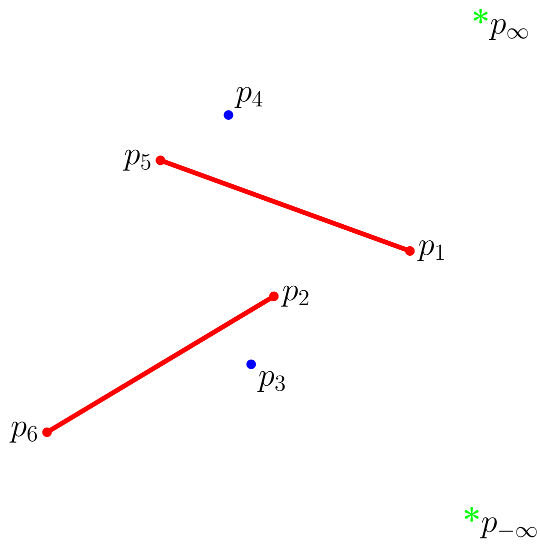


Figure 7.2: Example configuration of nodes and obstacles.

Here, we define two nodes $p_\infty = (x^*, +\infty)$ and $p_{-\infty} = (x^*, -\infty)$ where x^* is an x-coordinate larger than that of all nodes in V. For illustration purposes, see Figure 7.2. Next, we give a formal definition for a rotation tree.

Definition 7.1.1. *(Overmars and Welzl, 1988, Sec. 2) For a direction d between $-\pi/2$ and $\pi/2$, the rotation tree RT_d following direction d is an ordered tree with node set $V \cup \{p_\infty, p_{-\infty}\}$ where every node p (except p_∞) has exactly one outgoing edge. If q is the first node encountered when looking in direction d from p, and rotating counter clockwise, then this outgoing edge is $\overline{pq}$.*

For any direction d, since q is the first node encountered from p, the slope of the directed line edge $\overline{pq}$ cannot be smaller than d. A sub-tree in RT_d is a set of connected edges (ordered by slope) starting from a node growing to the right, such that the slope of any edge is always larger than the slope of the previous edge. This means that the left most edge is the edge with the smallest slope in a sub-tree. p_∞ is always the root of the rotation tree and no edge in the tree has the slope larger than $\pi/2$. See Figure 7.3 for an example. Note that all sub-trees are directed towards the root.

The idea used here is similar to any sweep line algorithm that uses a horizontal or vertical line moving in a fixed direction to sweep the plane. The difference here

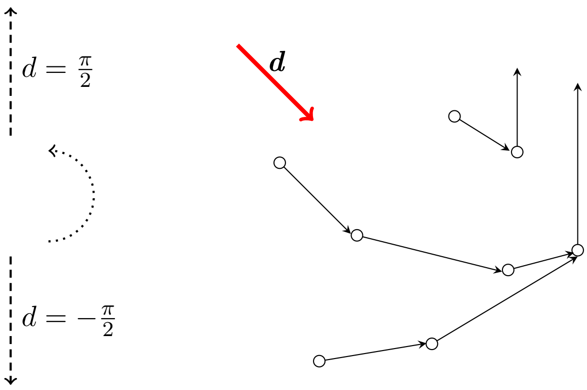


Figure 7.3: A rotation tree following direction d with 3 maximal sub-trees.

is a direction line is rotating to sweep the plane. The rotational sweep is started for $d = -\pi/2$, a direction pointing downward (negative y direction). Then, d is increased while updating the rotation tree RT_d until $d = \pi/2$ (the sweep proceeds in counter clockwise direction). During the sweeping process, RT_d is updated by removing and inserting an edge when determining the next *correct* node in the tree. By *correct*, we mean, for a given direction d, q is the next *correct* node for p if q and p satisfy the definition in Definition 7.1.1. At the end of the process, all edges will be detected and reported.

Here, we give some notations on the rotation tree structure before we further describe the procedure. A rotation tree consists of a set of nodes and edges that has the following properties:

- The top most node in the tree is assigned as root node and has no parent.
- Each node has zero or more sons but has at most one parent.
- A leaf node is a terminal node that has no sons.
- Nodes with the same parent are siblings.

Refer to Figure 7.4, A is the root node that has two sons B and C. B is the parent of D and E. F, G and H are siblings and also sons of C. Leaf nodes of this example are D, E, F, G and H. The directed arrows in Figure 7.4 show the relationship between nodes, used in the data structure of the algorithm. Note for example, edge AC is bidirectional meaning that A has a right most son C and C has A as its parent. Edge AB is unidirectional meaning that B has A as its parent but B is not the right most son of A.

The basic properties of a rotation tree are as follows.

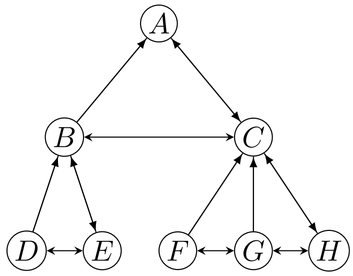


Figure 7.4: A rotation tree structure.

Property 7.1.2. *Basic properties (Overmars and Welzl, 1988, Property 1)*

1. *p_∞ is the root of the tree.*
2. *The left most edge of a set of connecting edges has the smallest slope.*
3. *For every edge $\overline{pq}$, $-\frac{\pi}{2} \leq slope(\overline{pq}) \leq \frac{\pi}{2}$.*
4. *If $\overline{pq}$ and $\overline{qr}$ are edges then $slope(\overline{qr}) \geq slope(\overline{pq})$.*

Therefore, the path from a node in the tree to the root rode is a convex curve of increasing slope.

Property 7.1.3. *Order Property (Overmars and Welzl, 1988, Property 2)*
Let $p, q,$ and r be nodes in the tree where $\overline{pq}$ is an edge. Let this setting be the current state of the tree. If $slope(\overline{pq}) \leq slope(\overline{pr}) \leq \pi/2$, then either r lies on the path from p to the root or it is in a left sub-tree of this path. That means r precedes p in the previous state of the tree. On the other hand, r succeeds p in the previous state of the tree if $-\pi/2 \leq slope(\overline{pr}) \leq slope(\overline{pq})$ in the current state of the tree.

If we remove an edge $\overline{pq}$ from the tree, the next node in counterclockwise order around p, z can be found on the path from p to the root node or in a left subtree of this path.

Property 7.1.4. *Cone Property (Overmars and Welzl, 1988, Property 3)*
Let $\overline{pq}$ and $\overline{rs}$ be edges in the tree such that $slope(\overline{pq}) \leq slope(\overline{pr}) \leq slope(\overline{rs})$. Then no node in V lies in the cone which is the intersection of the open halfplane to the right of $\overline{rs}$ and the closed halfplane to the left of $\overline{pr}$.

With the above properties of a rotation tree, we give two lemmas that determine the procedure of rotational sweeping.

Lemma 7.1.5. *(Overmars and Welzl, 1988, Lemma 2.1)*
Let RT be a rotation tree on V and let $\overline{pq}$ be the right most left leaf-edge in RT. If $\overline{pq}$ is replaced by $\overline{pz}$ where z is the next node after q, in counter clockwise order around p, then the resulting tree is a rotation tree.

A chain in a rotation tree can be defined as a sequence of edges $E_0, E_1, E_2, \ldots, E_i$ such that E_{j+1} starts where E_j ends and all edges except for E_i are the right most incoming edges of their endpoint.

Lemma 7.1.6. *(Overmars and Welzl, 1988, Lemma 2.2)*
Let RT be a rotation tree on V and let $\overline{pq}$ be the left most edge of q ($q \neq p_\infty$). Let r be the parent of q and let z' be the left brother of q, provided it exists. Then the next node z around p (after q) is the tangent (from the right) from p to the chain ending in $\overline{z'r}$, or, if z' does not exist, $z = r$.

At any stage of the rotational sweep, the edge of the right most leaf in the tree that is left most son of its parent is determined and replaced.

7.1.1 Data structures and algorithm

The detailed algorithm will be presented in this section. Before that, we describe the data structure and present a few functions with their descriptions that can be implemented in $O(1)$ time. For each node in V, six fields are stored in the data structure. They are the x and y coordinates of the nodes and four pointers to its parent, left brother, right brother and its right most son.

The following functions manipulating these data structures are needed. More detail is given here than in the published papers.

- **RightBrother**(p):
  Returns the right brother of p.
  (NULL if p has no right brother, i.e. p is the right most son of its parent)

- **LeftBrother**(p):
  Returns the left brother of p.
  (NULL if p has no left brother, i.e. p is the left most son of its parent)

- **Parent**(p):
  Returns the parent of p.
  (NULL if p has no parent, i.e. p is a root node)

- **RightMostSon**(p):
  Returns the right most son of p.
  (NULL if p has no son, i.e. p is a leaf node)

- **AddRightMost**(p, q):
  Inserts p as right most son to q.
  Side effects:
  - **if** RightMostSon(q) **not** NULL **then**
    $t :=$ RightMostSon(q);
    LeftBrother(p) $:= t$;
    RightBrother(t) $:= p$;
    else
    LeftBrother(p) :=NULL;
    end
  - RightMostSon(q) $:= p$;
  - Parent(p) $:= q$;
  - RightMostSon(p) := NULL;
  - RightBrother(p) := NULL;

- **AddLeftOf**(p, q):
  Inserts p as left brother of q.
  Side effects:
  - **if** LeftBrother(q) **not** NULL **then**
    $t :=$ LeftBrother(q);
    LeftBrother(p) $:= t$;
    RightBrother(t) $:= p$;
    else
    LeftBrother(p) := NULL;
    end
  - LeftBrother(q) $:= p$;
  - Parent(p) := Parent(q);
  - RightBrother(p) $:= q$;
  - RightMostSon(p) := NULL;

- **Remove**(p):
  Removes p from the tree.
  Side effects:

- **if** RightBrother(p) **not** NULL **then**
  $t :=$ RightBrother(p);
  LeftBrother(t) := NULL;
  else
  RightMostSon(Parent(p)) := NULL;
  end
- Parent(p) := NULL;
- LeftBrother(p) := NULL;
- RightBrother(p) := NULL;
- RightMostSon(p) := NULL;

- **Sort**(V):
  Returns the set of nodes in V, sorted in decreasing x coordinate order. In the case of equal x coordinates, sort them by decreasing y coordinate.

- **LeftTurn**(p, q, r):
  For $r = p_\infty$, returns **true** when p lies on the left of q or when both p and q lie on the same vertical line and p is below q, i.e. p is after q in **Sort**(V).
  For $r \neq p_\infty$, returns **true** if r lies to the left of the directed line $\overline{pq}$.
  Returns **false** if r lies on or to the right of the directed line $\overline{pq}$.
  (Note that at any stage of the procedure, **LeftTurn** will not be called whenever $p = p_\infty$ or $p_{-\infty}$, $q = p_\infty$ or $p_{-\infty}$, and $r = p_{-\infty}$)

In this algorithm, we need a data structure to store all the nodes that are left most sons of their parents and process the right most node first. In this case, the stack data structure is used where the element can be stored and retrieved in a *last-in*, *first-out* manner (Cormen et al., 2001). The usual operations such as **Pop**, **Push**, **Top**, **InitStack**, and **EmptyStack** are incorporated in the algorithm.

Algorithm 7 below summarizes the rotational plane sweep method. First, the set of nodes is sorted by decreasing x-coordinate. Lines $2-5$ build the initial rotation tree when $d = -\pi/2$. At this stage, $p_{-\infty}$ has p_∞ as its parent and every node in V has $p_{-\infty}$ as its parent. The left most son of $p_{-\infty}$ is the node in V with the largest x coordinate and the right most son of $p_{-\infty}$ is the node with smallest x coordinate. To begin the loop, the stack is initialized and the first node in the sorted list is pushed into the stack. The first node is the right most node in V and the left most leaf in the initial rotation tree. While the stack is not empty, Lines $7-33$ form the main loop that determines the node to be removed from the stack and inserted in the tree. Lines $8-10$ assign p as the current node to be considered, p_r as the right brother of p and q as the parent of

Algorithm 7 *VisibleEdgeRotTree*: Constructing visible edges using rotational plane sweep method for sorting pairs of nodes in counter clockwise order.

Input: V which contains $(n-2)$ segment endpoints, source node v_s and goal node v_g.
Output: All visible edges between nodes in V **or** $\binom{n}{2}$ pairs of nodes sorted by the slope of their connecting lines when Lines $11-13$ are commented out.

```
1:  Sort(V);
2:  AddRightMost(p_{-∞}, p_∞);
3:  for i = 1 to n do
4:     AddRightMost(p_i, p_{-∞});
5:  end for
6:  InitStack; Push(p_1);
7:  while NOT EmptyStack do
8:     p := Pop;
9:     p_r := RightBrother(p);
10:    q := Parent(p);
11:    if q ≠ p_{-∞} then
12:       Visible(p, q);
13:    end if
14:    z := LeftBrother(q);
15:    Remove(p);
16:    if z = NULL or NOT LeftTurn(p, z,Parent(z)) then
17:       AddLeftOf(p, q);
18:    else
19:       while RightMostSon(z) ≠ NULL AND LeftTurn(p,RightMostSon(z), z)
          do
20:          z := RightMostSon(z);
21:       end while
22:       AddRightMost(p, z);
23:       if z = Top then
24:          z := Pop;
25:       end if
26:    end if
27:    if LeftBrother(p) = NULL AND Parent(p) ≠ p_∞ then
28:       Push(p);
29:    end if
30:    if p_r ≠ NULL then
31:       Push(p_r);
32:    end if
33: end while
```

p. The directed line $\overline{pq}$ will be passed into the function **Visible**(p, q) which will decide whether q is visible to p and report it as an visible edge if they are mutually visible. We will describe this function in the next section.

To update the tree, there are two cases to consider. In the first case, if there is no node on the left of q or on the left of the directed $\overline{pq}$, then p is added as a son to the parent of its parent q. p will now be the left brother of q (Lines $16-17$). Otherwise, the loop will walk along the chain to find the next *correct* node, z, that is the tangent node from p. When z is found, p is added to be the right most son of z (Lines $19-22$). Lines $23-25$ check if z is on the stack. Since z is now the parent of p, it is no longer a leaf node. Therefore it should be removed from the stack. Lastly, the stack is adapted where p is pushed into the stack if p does not have a left brother (Lines $27-29$). If p has a right brother p_r, p_r should be pushed into the stack too (Lines $30-32$).

A step-by-step illustration of updating the rotation tree based on the configuration in Figure 7.2 is given in Appendix A.

The correctness of the algorithm follows from Lemmas 7.1.5 and 7.1.6. The following theorem summarize this result. The proof given is a much more detailed version of that outlined by Overmars and Welzl (1988).

Theorem 7.1.7. *(Overmars and Welzl, 1988, Theorem 2.3)*
Given a set of n nodes in the plane, we can, for each node p, sort the other nodes by angle around p in $O(n^2)$ time.

Proof. Since there are n nodes, we will have exactly $\binom{n}{2}$ distinct edges to consider the visibility of. Let us first define a connection between two nodes, p and $q \neq p_{-\infty}$ to be an actual edge, and a connection between a node p and the phantom node $p_{-\infty}$ to be a phantom edge.

Firstly, consider the number of passes through the main ***while*** *loop.*

We claim that in Algorithm 7, the main **while** loop (Lines $7-33$) is performed once for each actual edge, and once for each phantom edge.

To justify this, we count the number of passes through the main loop by considering the output edges, $\overline{pq}$ where q is the parent of p in the rotation tree (Line 12). Note that the rotation tree is updated by sweeping the plane with a direction line d with angles $[-\frac{\pi}{2}, \frac{\pi}{2}]$. At the initial stage of the rotation tree, $p_{-\infty}$ is assigned to be the parent for every node. These phantom edges ($\overline{pp_{-\infty}}$) have angles $-\frac{\pi}{2}$ and are considered first in the loop. By the end of these initial n passes, the parent of each node p will no longer be $p_{-\infty}$.

Then, the actual edges are considered in groups with the same parent. Within a group they are considered in order of increasing slope from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. In the algorithm, once edge $\overline{pq}$ is considered, p is moved to the left of q in the rotation tree and therefore $\overline{pq}$ is not considered again in lines $8-12$. Therefore, there are at most $\binom{n}{2} + n$ passes through the main loop. Excluding the work in the inner **while** loop (Lines 19 – 21), each pass through the main loop requires $\mathcal{O}(1)$ work. So the total work done excluding the work in the inner **while** loop (Lines 19 – 21) is $\mathcal{O}(n^2)$.

Now consider the work of all passes through the inner ***while*** *loop (Lines* 19 – 21*).*

Once the actual edge $\overline{pq}$ is considered, the rotation tree will be updated such that the next edges to be processed are in order of increasing slope. More precisely, there are two cases to consider here. First, if there is no node on the left of $\overline{pq}$ (meaning that there is no node on the left of q in the rotation tree), p will be moved to be on the left of q and execution continues at line 27. In the second '**else**' case where there exists a node on the left of $\overline{pq}$, the inner **while** loop (Lines 19 – 21) is executed. The loop will walk along the chain, that is the subtree to the left of q in the rotation tree to search for the node that is tangent to p. When the node is found, p will be added as right most son of this node and become part of the chain. This chain is on the left of q. Thus, p will be moved to be on the left of q in both cases.

Note that nodes that are on the right of p will never move back to the left of p while updating the rotation tree. Hence when $\overline{pq}$ is considered, it will never be considered again because p is moved and q is now on the right of p in the rotation tree.

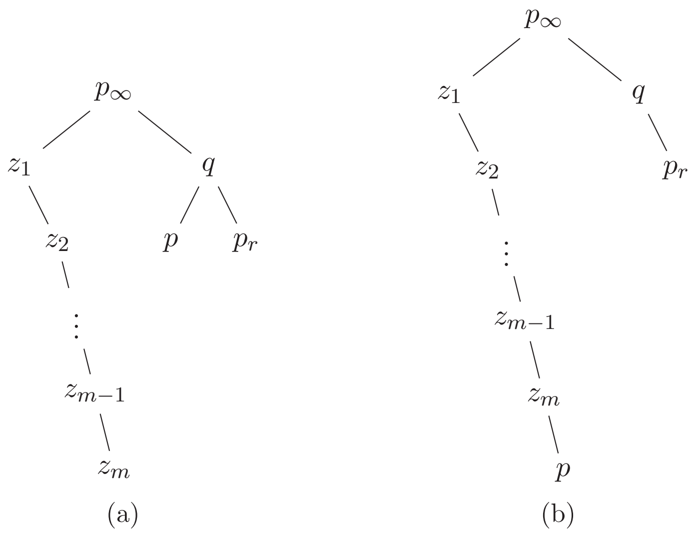


Figure 7.5: Once edge $\overline{pq}$ is considered in this setting, shown in (a), p is then moved to the left of q, as shown in (b).

To count the work done in lines 16 – 26, we consider the $k-$th pass through the main **while** loop. Suppose in the corresponding execution of lines 19 – 21, we generate $z_1, z_2, \ldots, z_m$ in turn by calling **LeftTurn**(p, **RightmostSon**(z), z) for $1 \le i < m$ to search for the node that is tangent to p, that is z_m. (See Figure 7.5(a).) Then, p will be added as the right most son of z_m and become part of the chain that contains z_m. (See

Figure 7.5(b).) Therefore edges $\overline{p\,z}_1, \overline{p\,z}_2, \ldots, \overline{p\,z}_{m-1}$ will not be considered again in any passes because when $\overline{p\,z}_m$ is considered later, p will be moved to the left and never will be added as a son of $z_1, z_2, \ldots$ or z_{m-1}.

In this pass, the work in the inner **while** loop (Lines 19−21) is $\mathcal{O}(1) + Cf(k)$ where C is a constant and $f(k)$ is $m-1$ if m, the number of z_i's generated in the inner **while** loop, is greater than 1, and 0 otherwise. Since edges $\overline{p\,z}_1, \overline{p\,z}_2, \ldots, \overline{p\,z}_{m-1}$ are considered at most once in the execution of the inner **while** part of the algorithm, and the sum of $f(k)$ is bounded by the total number of distinct edges, which is $\binom{n}{2}$, the total work in the inner **while** loop over all passes through the main loop is $\mathcal{O}(n^2)$.

Therefore, combining the two counts above, the total work of the algorithm is $\mathcal{O}(n^2)$.

$\square$

7.2 Visibility Graph Construction for Disjoint Segment Obstacles

In this section, we will present a method for constructing the visibility graph between two nodes of a set of line segment obstacles. Let $\mathcal{S}$ be the finite set of line segment obstacles and V be the finite set of n nodes which are the endpoints of the segment obstacles. We obtain $\binom{n}{2}$ line segments sorted by slope in counter clockwise order using the method described in the previous section. Let $D = \{\boldsymbol{d}_i\}$, where $1 \leq i \leq \binom{n}{2}$ be the set of sorted slopes, hereafter denoted as the set of finite directions. Now we have to identify those line segments that are also a visible edge.

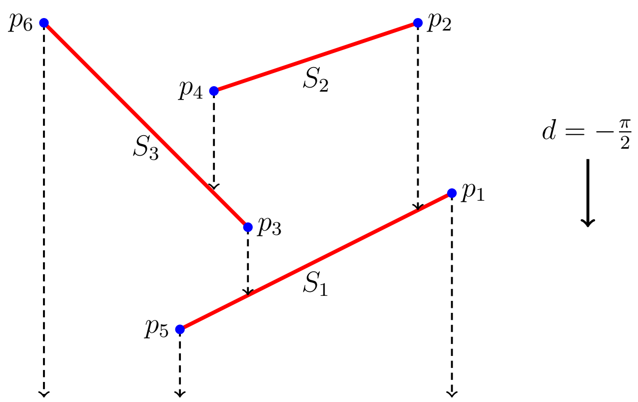
Following the method in Overmars and Welzl (1988), let $\mathbf{Vis}_{\boldsymbol{d}}(p)$ be the vision of p in direction $\boldsymbol{d}$. That means $\mathbf{Vis}_{\boldsymbol{d}}(p)$ is the first obstacle segment hit by the ray emanating from p in direction $\boldsymbol{d}$. For any direction $\boldsymbol{d} = [-\pi/2, \pi/2]$, we denote $\boldsymbol{d}_{-}$ be the direction immediately preceding $\boldsymbol{d}$, and $\boldsymbol{d}_{+}$ be the direction immediately succeeding $\boldsymbol{d}$. To initialize the structure, we set $\boldsymbol{d}_0 = -\pi/2$. Thus, we can determine $\mathbf{Vis}_{\boldsymbol{d}_0}(p)$ for each node in V, which gives the obstacle segment immediately below it. For example, the table in Figure 7.6 shows the initial vision for each node in the configuration. If there is no such segment, $\mathbf{Vis}_{\boldsymbol{d}_0}(p) = 0$.

In the context of our problem, we omit the assumption of all nodes in V must be in general position. In other words, we allow nodes to be collinear and any vertical line can intersect more than one node. Moreover, set V not only contains the endpoints of segment obstacles, but also the source and goal nodes. Thus, we have to modify the definition of $\mathbf{Vis}_{\boldsymbol{d}_0}(p)$ to cater for this problem. If the ray emanating from p in the vertical downward direction hits an endpoint q of a segment S_i,

$$\mathbf{Vis}_{\boldsymbol{d}_0}(p) = \begin{cases} \mathbf{Vis}_{\boldsymbol{d}_0}(r), & \text{if segment } S_i \text{ is vertical and} \\ & r \text{ is the lower endpoint of segment } S_i; \\ \mathbf{Vis}_{\boldsymbol{d}_0}(q), & \text{if } q \text{ is the right endpoint of segment } S_i; \\ i, & \text{if } q \text{ is the left endpoint of segment } S_i. \end{cases} \tag{7.1}$$

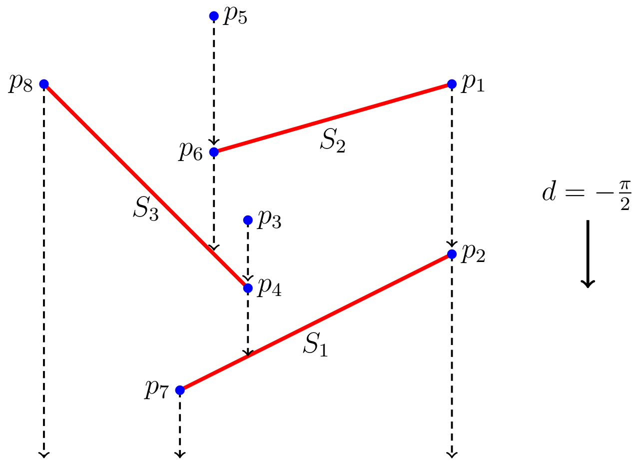
See Figure 7.7 for an example.

It requires $O(n^2)$ time to compute initial $\mathbf{Vis}_{\boldsymbol{d}}(p)$ for $\boldsymbol{d} = -\pi/2$ since we scan through each segment obstacle for every node in V. Recall that the rotational plane sweep algorithm will returns pairs of nodes sorted by slope of their connecting line in counter clockwise order. These pairs of nodes will be passed into the function **Visible**(p, q) to determine their visibility. Let p and q be the two nodes to be considered,



p	p_1	p_2	p_3	p_4	p_5	p_6
$\mathbf{Vis}_{\boldsymbol{d}_0}(p)$	0	1	1	3	0	0

Figure 7.6: The initial structure for $\mathbf{Vis}_{\boldsymbol{d}}(p)$. The entries 1 and 3 denote the obstacle segments S_1 and S_3.



p	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
$\mathbf{Vis}_{\boldsymbol{d}_0}(p)$	0	0	1	1	2	3	0	0

Figure 7.7: The modified initial structure for $\mathbf{Vis}_{\boldsymbol{d}}(p)$. The entries 1, 2 and 3 denote the obstacle segments S_1, S_2 and S_3.

we will report $\overline{pq}$ as a visible edge only if p and q are mutually visible. In the function, $\mathbf{Vis}_{\boldsymbol{d}}(p)$ will be updated while $\boldsymbol{d}$ sweeps in counter clockwise order from $\boldsymbol{d} = -\pi/2$. At the beginning, $\mathbf{Vis}_{\boldsymbol{d}}(p)$ remain unchanged until we sweep through the first detected pair of nodes. We will now show all the possible cases to determine if q is visible to p.

- Case (1): If p and q are endpoints of the same line segment in $\mathcal{S}$, then they are visible to each other. The vision of p remains unchanged. See Figure 7.8(a).
- Case (2): If q is an endpoint of $\mathbf{Vis}_{\boldsymbol{d}}(p)$, then q must be succeeding the other endpoint of $\mathbf{Vis}_{\boldsymbol{d}}(p)$ in counter clockwise order. p and q are visible to each other. Thus, the vision of p will now be the vision of q (see Figure 7.8(b)).
- Case (3): If q lies in front of $\mathbf{Vis}_{\boldsymbol{d}}(p)$ from p, then $\overline{pq}$ does not intersect $\mathbf{Vis}_{\boldsymbol{d}}(p)$. Hence, p and q are mutually visible. If q is the endpoint of a line segment in $\mathcal{S}$, the vision of p will now be that segment of q (see Figure 7.8(c)). If q is the source or goal node (without segment), the vision of p remains unchanged.
- Case (4): If q lies behind $\mathbf{Vis}_{\boldsymbol{d}}(p)$ from p, then $\overline{pq}$ intersects $\mathbf{Vis}_{\boldsymbol{d}}(p)$. Thus, p and q are not visible to each other. In this case, the vision of p remains unchanged (see Figure 7.8(d)).

Cases (1), (2) and (3) imply that $\overline{pq}$ is a visible edge for the visibility graph. Here, we define some operations used in the **Visible**(p, q) function. These operations can all be performed in $O(1)$ time. Then, function **Visible**(p, q) will be given. Note that this function is the modified version of function **Handle**(p, q), given in Overmars and Welzl (1988).

- **Segment**(p):
  Returns the index of segment which p is an endpoint of.
  (NULL if p is the source or goal node)
- **Other**(p):
  Returns the other endpoint of **Segment**(p).
  (NULL if p is the source or goal node)
- **Before**(p, q, S_i):
  Returns **true** if q lies before segment S_i from the position of p.
  (Note that **Before**(p, q, S_i) is always **true** when $\mathbf{Vis}_{\boldsymbol{d}}(p) = 0$, i.e. $S_i = 0$.)
- **Report**(p, q):
  Report $\overline{pq}$ as an edge of the visibility graph, VG.

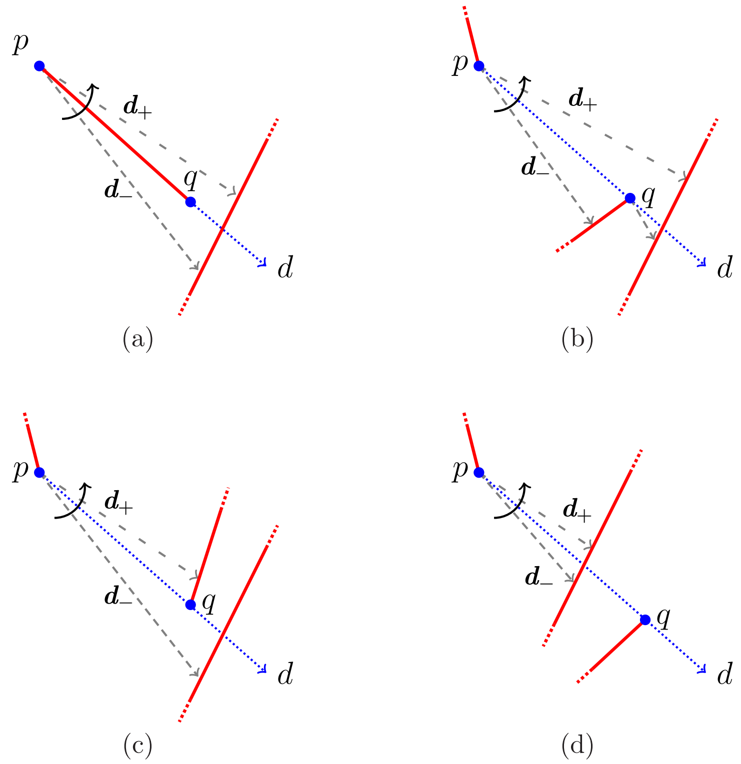


Figure 7.8: Rotational sweep transitions in counter clockwise order: cases determining the visibility of p and q. The densely dashed gray line $\boldsymbol{d}_-$ denotes the direction immediately preceding $\boldsymbol{d}$ (blue dotted line), and loosely dashed gray line, $\boldsymbol{d}_+$ denotes the direction immediately succeeding $\boldsymbol{d}$.

Algorithm 8 Function *Visible*(p, q)

if $q =$ **Other**(p) **then**
    Report(p, q);
else if Segment(q)=**Vis**$_{\boldsymbol{d}_-}(p)$ **then**
    Vis$_{\boldsymbol{d}_+}(p)$:= **Vis**$_{\boldsymbol{d}_-}(q)$;
    Report(p, q);
else if Before$(p, q,$ **Vis**$_{\boldsymbol{d}_-}(p))$ **then**
    if Segment(q) **not** NULL **then**
        Vis$_{\boldsymbol{d}_+}(p)$:= **Segment**(q);
    end if
    Report(p, q);
end if

Here, we give two lemmas that show the correctness of the method.

Lemma 7.2.1. *(Ghosh, 2007, Lemma 5.3.1)*
Let $\boldsymbol{d}_{-}$ be the direction of of q from p and $\boldsymbol{d}_{+}$ be the next direction after $\boldsymbol{d}_{-}$ in counter clockwise order around p. If q is an endpoint of $\boldsymbol{Vis}_{\boldsymbol{d}_{-}}(p)$ provided $\boldsymbol{Vis}_{\boldsymbol{d}_{-}}(p)$ exists, then $\boldsymbol{Vis}_{\boldsymbol{d}_{+}}(p) = \boldsymbol{Vis}_{\boldsymbol{d}_{-}}(q)$.

Lemma 7.2.2. *(Ghosh, 2007, Lemma 5.3.2)*
Let $D = \{d_1, d_2, \ldots, d_n\}$ be the set of n directions. The order of rotation for the directions in D can be computed in $O(n^2)$ time.

The proofs of Lemma 7.2.1 and Lemma 7.2.2 are given in Ghosh (2007). These lemmas were also given by Welzl (1985). These lemmas gives the following theorem.

Theorem 7.2.3. *(Overmars and Welzl, 1988, Theorem 3.1)*
The visibility graph of a set $\mathcal{S}$ of disjoint segment obstacles with n segments in total can be computed in $O(n^2)$ time and $O(n)$ working storage.

Hence, given a set $\mathcal{S}$ of line segments as obstacles, a source node (v_s) and a goal node (v_g), the visibility graph (VG) can be constructed via:

- initialize $\mathbf{Vis}_{\boldsymbol{d}_0}(p)$ for all $p \in V$
- VG=*VisibleEdgeRotTree*$(\mathcal{S}, v_s, v_g)$.

7.3 Visibility Graph Construction for Polygonal Obstacles

The visibility graph for a set of polygonal obstacles can be constructed by an algorithm similar to that given in Section 7.2. The difference is that the vision function $\mathbf{Vis}_{\boldsymbol{d}}(p)$ and the visibility function *Visible*(p, q) must be changed. Here we outline the construction as given by Alt and Welzl (1988). The main procedure remains the same. We need to first initialize $\mathbf{Vis}_{\boldsymbol{d}}(p)$ for every node in V. Then process every sorted pair of nodes and determine their visibility.

Let $\mathcal{S}$ be the finite set of segments of polygonal obstacles and V be the finite set of nodes which are the endpoints of the segments. The vision of a node p, $\mathbf{Vis}_{\boldsymbol{d}}(p)$ in a given direction $\boldsymbol{d}$ is defined as follows.

1. If the ray emanating from p in direction $\boldsymbol{d}$ does not intersect any obstacles, then $\mathbf{Vis}_{\boldsymbol{d}}(p) = 0$.
2. If the ray emanating from p in direction $\boldsymbol{d}$ enters the obstacle of which p is a node, then $\mathbf{Vis}_{\boldsymbol{d}}(p) = -1$.
3. If the ray emanating from p in direction $\boldsymbol{d}$ first intersects S_i, a segment of another obstacle, then $\mathbf{Vis}_{\boldsymbol{d}}(p) = i$.

We initialize $\mathbf{Vis}_{\boldsymbol{d}}(p)$ for all p in V at $\boldsymbol{d}_0 = -\pi/2$ to begin the algorithm. Again, we need extra definition for $\mathbf{Vis}_{\boldsymbol{d}}(p)$ in order to consider nodes that are not in the general position. If the ray emanating from p in the vertical downward direction hits an obstacle node q, then q must be an endpoint of two connecting segments S_i and S_j in counter clockwise order. Therefore,

$$\mathbf{Vis}_{\boldsymbol{d}_0}(p) = \begin{cases} \mathbf{Vis}_{\boldsymbol{d}_0}(q), & \text{if } q \text{ is the right endpoint of both segments } S_i \text{ and } S_j; \\ & \text{(See Figure 7.9(a).)} \\ i, & \text{otherwise. (See Figure 7.9(b) and (c).)} \end{cases} \tag{7.2}$$

Note that segment S_i must be the nearest segment to node p. For example, see Figure 7.10.

Similar to the method of constructing the visibility graph for disjoint segment obstacles, we will process the pairs of nodes (p, q), obtained from the rotational plane sweep algorithm. To distinguish the current function from the previous function (for disjoint segment obstacles), we name it as *VisiblePolygonal*(p, q). Let us now define

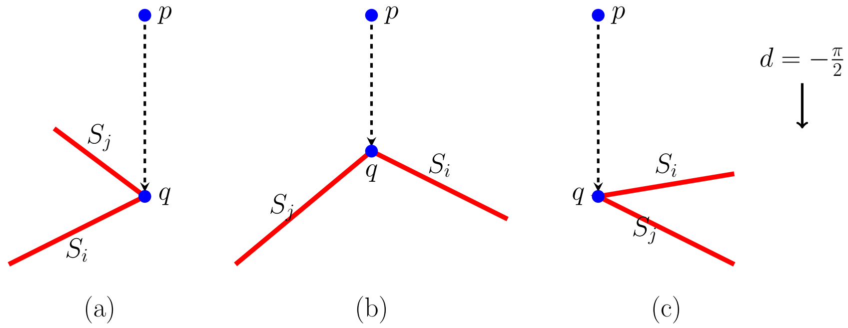
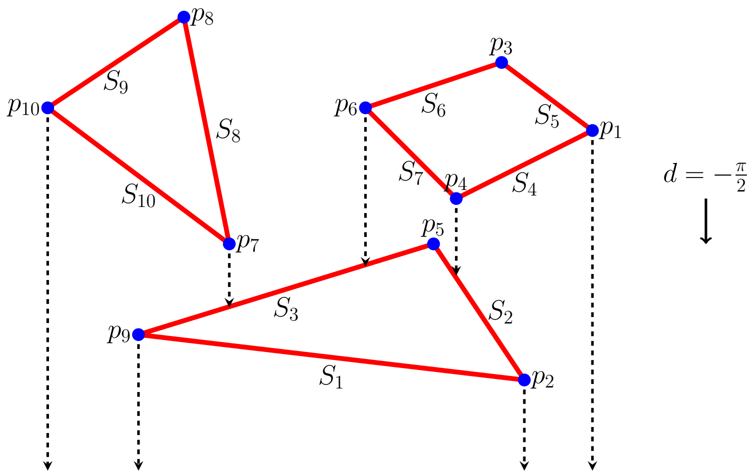


Figure 7.9: Cases for $\mathbf{Vis}_{\boldsymbol{d}}(p)$ when q is an obstacle node.



p	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
$\mathbf{Vis}_{d_0}(p)$	0	0	-1	2	-1	3	3	-1	0	0

Figure 7.10: The initial structure for $\mathbf{Vis}_{\boldsymbol{d}}(p)$ with polygonal obstacles. The entries 2 and 3 in the table denote segment S_2 and S_3.

some notation and operations used in the *VisiblePolygonal*(p, q) function which we will describe later. If p is a node of any polygonal obstacle, then p is a node connecting two segments of that polygonal obstacle, see Figure 7.11.

- **SegIn**(p):
  Looking in counter clockwise order around p, **SegIn**(p) is the first segment encountered before entering the obstacle of which p is a node.
  SegIn(p) returns the index of this segment.
  (NULL if p is the source or goal node)

- **SegOut**(p):
  Looking in counter clockwise order around p, **SegOut**(p) is the last segment encountered before exiting the obstacle of which p is a node.
  SegOut(p) returns the index of this segment.
  (NULL if p is the source or goal node)

For example, let the polygonal obstacle consists of line segments S_1, S_2, S_3, S_4 and S_5, in counter clockwise order (Figure 7.11). Then **SegIn**$(p) = 1$ and **SegOut**$(p) = 5$ where 1 and 5 are the index of segment S_1 and S_5.

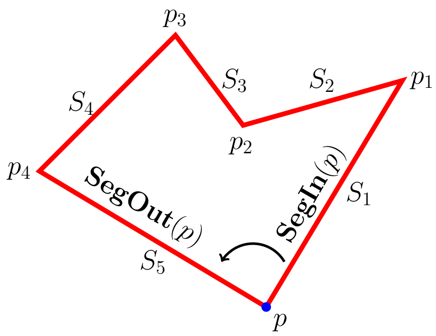


Figure 7.11: **SegIn**(p) and **SegOut**(p) of node p in a polygon.

- **OtherSegIn**(p):
  Returns the other endpoint of **SegIn**(p) that is not p.
  (NULL if p is the source or goal node)

- **OtherSegOut**(p):
  Returns the other endpoint of **SegOut**(p) that is not p.
  (NULL if p is the source or goal node)

- **Before**(p, q, S_i):
  Returns **true** if q lies before segment S_i from the position of p.
  (Note that **Before**(p, q, s) is always **true** when $S_i = 0$, i.e. $\mathbf{Vis}_{\boldsymbol{d}}(p) = 0$.

- **Report**(p, q):
  Report $\overline{pq}$ as an edge of the visibility graph, VG.

Let p and q be the two nodes to be considered, we will report $\overline{pq}$ as a visible edge only if p and q are mutually visible. In the algorithm, $\mathbf{Vis}_{\boldsymbol{d}}(p)$ will be updated while $\boldsymbol{d}$ sweeps from $-\pi/2$ to $\pi/2$. Here, we will describe all the cases to determine if q is visible to p.

- Case (1): If $\mathbf{Vis}_{\boldsymbol{d}_-}(p) = -1$ and q is **OtherSegOut**(p), then $\overline{pq}$ is a line segment in S. In this case, q is the other endpoint of **SegOut**(p). Thus, p and q are visible to each other. However, the vision of p changes according to the configuration of the obstacle. We look at the current vision of q and there are two possibilities as follows:
$$\mathbf{Vis}_{\boldsymbol{d}_+}(p) = \begin{cases} \mathbf{SegOut}(q), & \text{if } \mathbf{Vis}_{\boldsymbol{d}_-}(q) = -1; \\ \mathbf{Vis}_{\boldsymbol{d}_-}(q), & \text{otherwise.} \end{cases}$$
  See Figure 7.12(a) and 7.12(b) for illustration.

- Case (2): If $\mathbf{Vis}_{\boldsymbol{d}_-}(p) = S_i$ and q is an endpoint of $\mathbf{Vis}_{\boldsymbol{d}_-}(p)$, then $\overline{pq}$ does not intersect S_i but meets S_i at q (see Figure 7.12(c) and 7.12(d)). Hence, p and q are visible to each other. Similar to Case (1), there the vision of p changes as follows:
$$\mathbf{Vis}_{\boldsymbol{d}_+}(p) = \begin{cases} \mathbf{SegOut}(q), & \text{if } \mathbf{Vis}_{\boldsymbol{d}_-}(q) = -1; \\ \mathbf{Vis}_{\boldsymbol{d}_-}(q), & \text{otherwise.} \end{cases}$$

- Case (3): If q is **OtherSegIn**(p), that means $\overline{pq}$ is a line segment in $\mathcal{S}$. They are therefore visible to each other. Since $\overline{pq}$ is **SegIn**(p), the vision of p now changes to -1 (see Figure 7.12(e)).

- Case (4): If $\mathbf{Vis}_{\boldsymbol{d}_-}(p) \neq -1$ and q lies in front of S_i from p, then $\overline{pq}$ does not intersect S_i. p and q are visible to each other. The vision of p changes to be **SegOut**(q). See Figure 7.12(g).

- Case (5): If $\mathbf{Vis}_{\boldsymbol{d}_-}(p) = -1$ and q is not **OtherSegOut**(p), then q must be one of the obstacle nodes not immediately succeeding p. Therefore, p and q are not visible to each other. The ray emanating from p to q still within its obstacle. The vision of p remains unchanged (see Figure 7.12(f)).

- Case (6): If $\mathbf{Vis}_{d_-}(p) \neq -1$ and q lies behind S_i from p, then $\overline{pq}$ intersects S_i where $i = \mathbf{Vis}_{d_-}(p)$. Thus, q is not visible to p. However, the vision of p remains unchanged (see Figure 7.12(h)).

Cases (1) to (4) imply that $\overline{pq}$ is a visible edge for the visibility graph. Here, we will give the function for *VisiblePolygonal*(p, q). Note that operations such as **SegIn**(p), **SegOut**(p), **OtherSegIn**(p), **OtherSegOut**(p) and **Before**(p, q, s) can all be performed in $O(1)$ time.

Algorithm 9 Function *VisiblePolygonal*(p, q)

```
if Vis_d1(p) = −1 AND q = OtherSegOut(p) then
    if Vis_d1(q) = −1 then
        Vis_d2(p) := SegOut(q);
    else
        Vis_d2(p) := Vis_d1(q);
    end if
    Report(p, q);
else if Vis_d1(p) ≠ −1 AND q is an endpoint of Vis_d1(p) then
    if Vis_d1(q) = −1 then
        Vis_d2(p) := SegOut(q);
    else
        Vis_d2(p) := Vis_d1(q);
    end if
    Report(p, q);
else if q = OtherSegIn(p) then
    Vis_d2(p) := −1;
    Report(p, q);
else if Vis_d1(p) ≠ −1 AND Before(p, q, Vis_d1(p))  then
    Vis_d2(p) := SegOut(q);
    Report(p, q);
end if
```

Hence, given a finite set $\mathcal{S}$ of polygonal line segments as obstacles, a source node (v_s) and a goal node (v_g), the visibility graph (VG) can be constructed via:

- initialize $\mathbf{Vis}_{d_0}(p)$ for all $p \in V$
- VG=*VisibleEdgeRotTree*$(\mathcal{S}, v_s, v_g)$ with Line 12 of Algorithm 7 replaced by *VisiblePolygonal*(p, q) (see Algorithms 7 and 9).

Following Theorem 7.2.3, we give the following theorem:

Theorem 7.3.1. *The visibility graph of a set $\mathcal{S}$ of disjoint polygonal obstacles with n segments in total can be computed in $O(n^2)$ time and $O(n)$ working storage.*

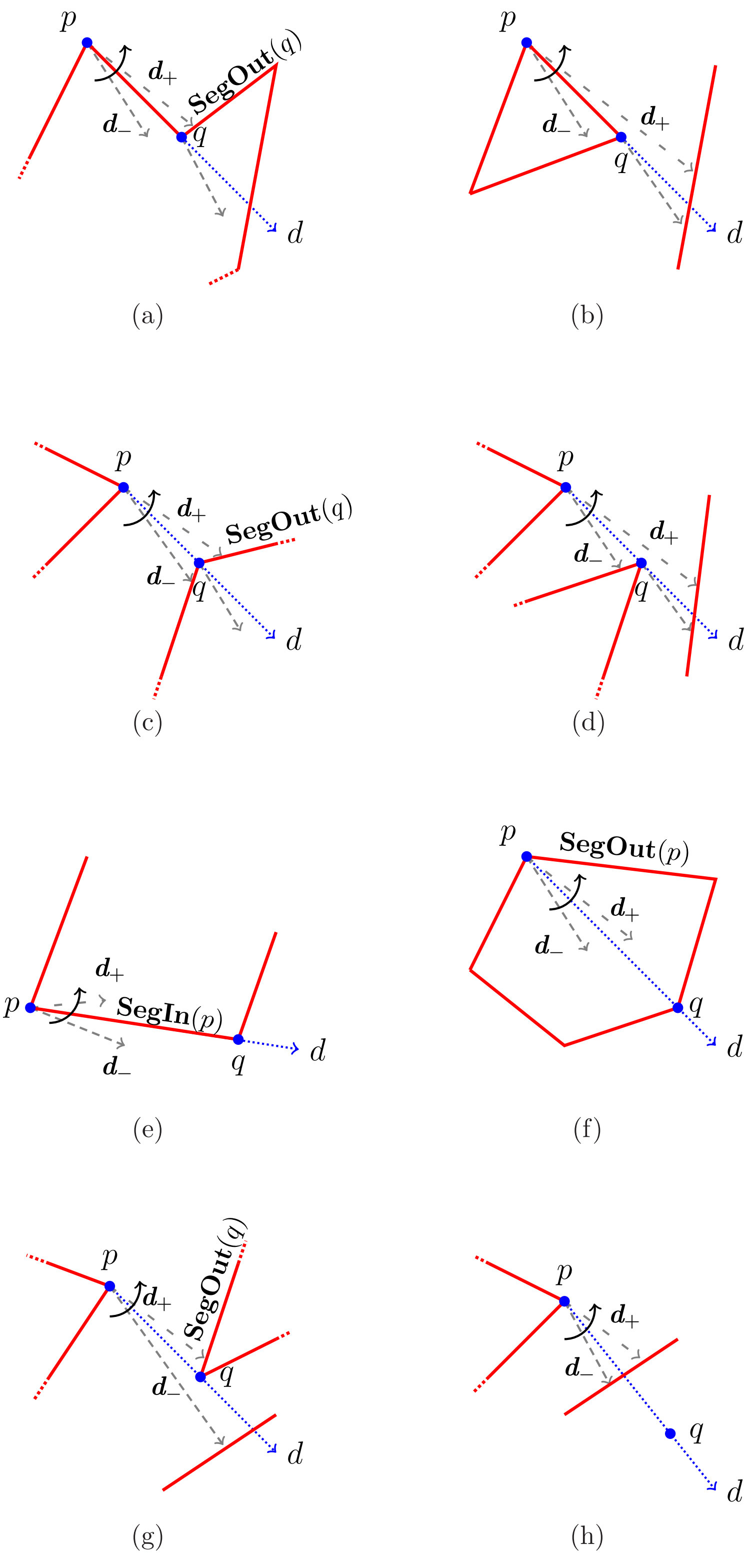


Figure 7.12: Rotational sweep transitions in counter clockwise order: cases determining the visibility of p and q for polygonal obstacles configuration. The densely dashed gray line $\boldsymbol{d}_-$ denotes the direction immediately preceding $\boldsymbol{d}$ (blue dotted line), and loosely dashed gray line, $\boldsymbol{d}_+$ denotes the direction immediately succeeding $\boldsymbol{d}$.

7.4 Dijkstra's Algorithm

In this section, we will describe a shortest path algorithm used in our problem. There are numerous algorithms available for finding shortest paths efficiently (Champandard, 2002). These algorithms are also known as path planning algorithms which can be constructed based on a weighted graph. Recall that a graph, $G = (V, E)$ always consists of a set of nodes, V and a set of edges, E. Let us now give the definition of a weighted graph and its shortest path weight.

Definition 7.4.1. *Weighted Graph*
A weighted graph is a graph, $G = (V, E)$ in which each edge, $\overline{v_i v_j}$, is assigned a real value, $w(\overline{v_i v_j})$, called the weight of $\overline{v_i v_j}$.

Definition 7.4.2. *Shortest Path Weight*
Given a weighted graph, $G = (V, E)$ with weight function w, the weight of the path $P = \langle v_1, v_2, \ldots v_k \rangle$ is the sum of the weights of its subsequent edges:

$$w(P) = \sum_{i=1}^{k-1} w(\overline{v_i v_{i+1}}).$$

A shortest path weight from v_1 to v_j is defined as:

$$MinDist(v_j) = \begin{cases} \text{MIN}\{w(P)\}, & \textit{if there is a path } P \textit{ from } v_1 \textit{ to } v_j; \\ \infty, & \textit{otherwise.} \end{cases}$$

From the above definitions, a shortest path between two nodes in a weighted graph is a path between the nodes which minimizes the sum of weights of the path edges. A typical weighted graph can be viewed as a road map (see Figure 7.13). Nodes in the graph represent cities or intersections and edges represent roads connecting the cities. The weight of edges can be distance, time or cost involved in traveling between the cities. Hence, a shortest path problem can now be modeled as: How to compute a shortest route (distance) from one city to another?

Generally, shortest path algorithms can be categorized into three major variants:

- **Single-Source Shortest Path**
  This algorithm finds a shortest path from a specified source node to all other nodes. Dijkstra's algorithm (Dijkstra, 1959) is one of the most popular and celebrated algorithms for solving the single-source shortest path problem. It is widely used in the fields of computer science and operations research. The main constraint of this

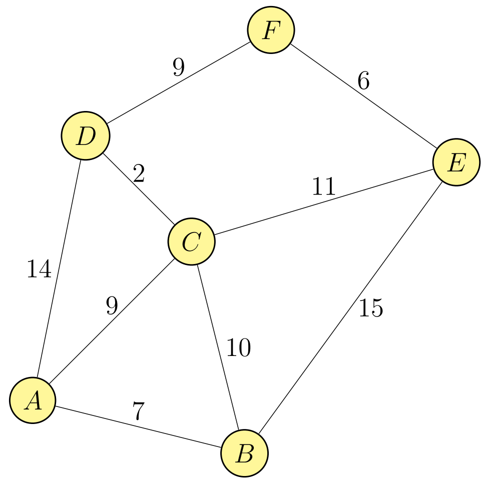


Figure 7.13: Example of a weighted graph.

algorithm is that all the edge weights must be positive. In the case where negative weighted edges may be involved, the Bellman-Ford algorithm can be used.

- **Single-pair Shortest Path**
  This algorithm finds a shortest path between a specified source and goal (destination) node. This problem can be solved with the single-source shortest path algorithm where the algorithm is terminated when a shortest path from the source to the goal node is found. This is the approach we have used.

- **All-pairs Shortest Path**
  This algorithm finds a shortest path from each node to every other node. Clearly, this algorithm is computationally more expensive. One can solve this problem by performing a single-source algorithm, once for every node as the source node. However this is often impractical to implement. For example, using Dijkstra's algorithm to solve the problem gives an $O(n^3)$ running time. If the Bellman-Ford algorithm is used (for a graph with some negative weights), the time complexity can be $O(n^4)$ (Cormen et al., 2001).

In our problem, we will use the constructed visibility graph (demonstrated in the previous section) to form a shortest path problem. We represent the visibility graph as a weighted graph where the weight of each edge is the distance between the nodes of that edge. Since the weight cannot be negative, it is natural to use Dijkstra's algorithm to solve the shortest path problem. In the next section, we will give a detailed description of Dijkstra's algorithm.

7.4.1 Description and algorithm

Dijkstra (1959) has proposed two solutions for graph search problems. They are the minimum weight spanning tree algorithm and the shortest path algorithm. Dijkstra's algorithm for solving the shortest path problem is a greedy algorithm that is often applied by itself or as a subroutine in another algorithm.

The algorithm works by updating two sets of array: the pending (unvisited) list, Q and a shortest distance list, $MinDist$. Q denotes the list of nodes that have not been considered or visited. $MinDist$ denotes a shortest distance of a node from the source node (see Definition 7.4.2). For example, let the source node be v_s. $MinDist(v_g)$ returns the value which is a shortest distance from v_s to v_g. The algorithm repeatedly considers nodes in the Q list and update its $MinDist$ from the source node. The algorithm will be terminated when Q list is empty.

- Step 1 : Initialization - Each node is assigned a shortest distance value, zero for the source node and infinity for all other nodes. All nodes are marked as unvisited and stored in the Q list. Set the source node as the current node to consider.
- Step 2 : For the current node, detect all its unvisited neighboring nodes and calculate their tentative $MinDist$. For example, let v_i be the current node with $MinDist(v_i) = 10$ and has an unvisited neighboring node v_j with $w(\overline{v_i v_j}) = 5$. Then, a possible total distance from v_i to v_j is $10 + 5 = 15$. If this value is less than the previously recorded value of $MinDist(v_j)$, then $MinDist(v_j)$ should be updated.
- Step 3 : When $MinDist$ for all the unvisited neighboring nodes of the current node have been updated, the current node is marked as visited. It is removed from the Q list and will not be selected again. $MinDist$ of the current node is therefore final and optimal.
- Step 4 : Select the node in Q list with least $MinDist$ to be the current node (**ExtractMin**(Q)). Repeat Step 2 and 3.
- Step 5 : The process is continued until the Q list is empty. An empty Q list implies that all nodes are visited and considered. At this stage, $MinDist$ at each node is final and optimal.

Next, we will give the detailed algorithm used in our problem. Note that the goal node is given and the algorithm is terminated once the goal node has been considered. Lines $11 - 13$ of Algorithm 10 should be removed if a shortest path to all other nodes from the source node is desired.

Algorithm 10 *Dijkstra*(G, v_s, v_g)

Input: $G = (V, E)$, source node v_s and goal node v_g.
Output: $MinDist(v_g)$.

Initialize
$V := \{V, v_s, v_g\}$;
$Q := V \setminus \{v_s\}$;
for all $v_i \in V$ **do**
  $MinDist(v_i) := \infty$;
end for
$MinDist(v_s) := 0$;
while $Q \neq \emptyset$ **do**
  $u :=$ **ExtractMin**(Q);
  Remove u from Q;
  if $u = v_g$ **then**
    Break;
  end if
  for t, each unvisited neighbor of u **do**
    $temp := MinDist(u) + w(\overline{ut})$;
    if $temp < MinDist(t)$ **then**
      $MinDist(t) := temp$;
    end if
  end for
end while

Theorem 7.4.3. *Correctness of Dijkstra's algorithm (Cormen et al., 2001, Theorem 24.6)*
Given a weighted graph $G = (V, E)$ with nonnegative edge weights and a source node v_s, Dijkstra's algorithm terminates with a shortest distance ($MinDist(v)$) from v_s for all nodes $v \in V$.

A thorough treatment of Dijkstra's algorithm is given in Cormen et al. (2001, Chap. 24).

7.4.2 Example of Dijkstra's algorithm

Here we will give a step-by-step illustration of Dijkstra's algorithm based on the example of Figure 7.13. This is a weighted graph with 6 nodes, $V = \{A, B, C, D, E, F\}$. The procedure of finding a shortest distance from node A to F will be shown. Let the weight for each edge be the distance between the two nodes. For example, $w(\overline{BE}) = 15$ implies that the distance between B and E is 15. Let the source node be A. In the initial step, $MinDist$ of A is set to 0 and for the rest of the nodes, $MinDist$ is set to infinity. Further description at each step will be given in the caption of each sub-figure. In the illustrations, the value of $MinDist$ is labeled beside each nodes and will be updated at each step. Unvisited nodes in Q are marked with yellow circle and visited nodes are marked with red square. In each step, red dashed lines represent edges connected to a visited node. These edges should not be considered in the subsequent step. For more animations (demos with Java applet) of Dijkstra's algorithm, the reader is referred to Pryor et al. (2007) and Ikeda (2000).

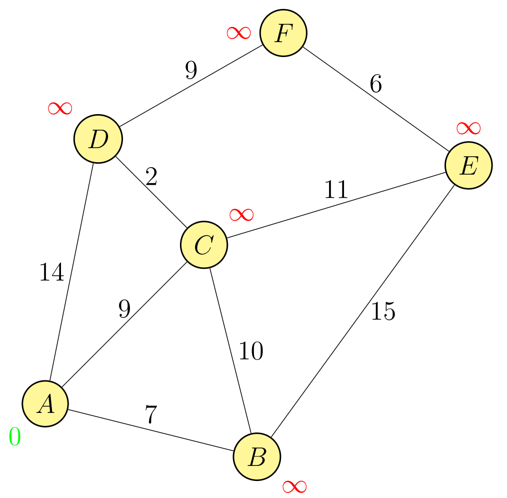
7.4.3 Complexity of Dijkstra's algorithm

In this section, we discuss the complexity of Dijkstra's algorithm which often relies on the following constants:

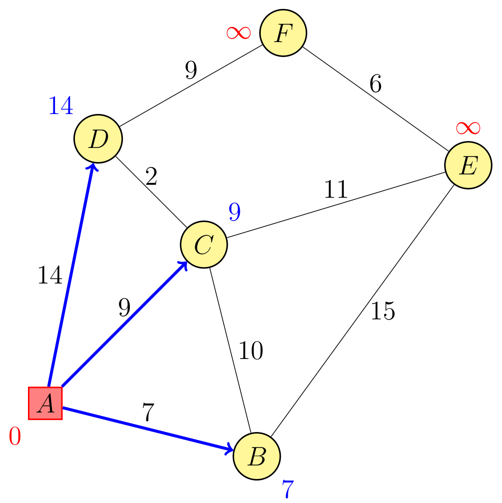
- $n = |V|$, the number of nodes in the graph,
- $m = |E|$, the number of edges in the graph,
- C, the maximum absolute value of an edge.

The performance of Dijkstra's algorithm mainly depends on the data structure implemented for Q (Cormen et al., 2001). Recall that Q is a min priority queue that should maintain the node with the least $MinDist$ to be selected and removed. A direct and simple implementation of a priority queue is to store the nodes in Q using an ordinary array. **ExtractMin**(Q) runs a linear search through all the nodes in Q. Therefore each **ExtractMin** operation takes $O(n)$ time. A total time in the **while** loop takes $O(n^2)$ time. Since there are m number of edges to be checked in the **for** loop, this gives a $O(n^2 + m) = O(n^2)$ total running time of the algorithm with array implementation.

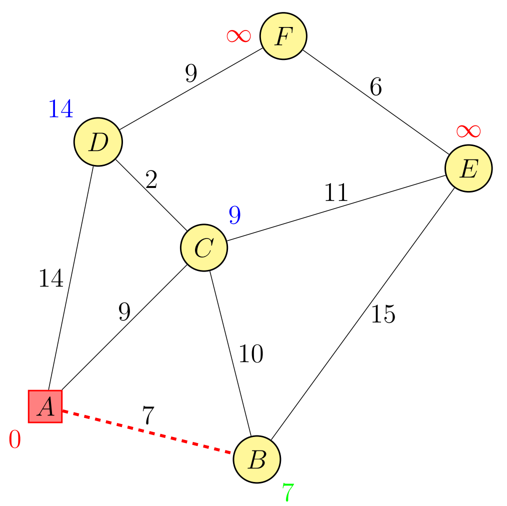
If the given graph is sparse, a binary heap can be used for a more efficient implementation of Q. In this case, the binary heap can be built in $O(n)$ time. The **ExtractMin** operations take $O(\log n)$ time and there are n such operations. With binary heap implementation, the checking steps in the **for** loop takes $O(m \log n)$ time. Hence the algorithm requires a total running time of $O((n + m) \log n)$.



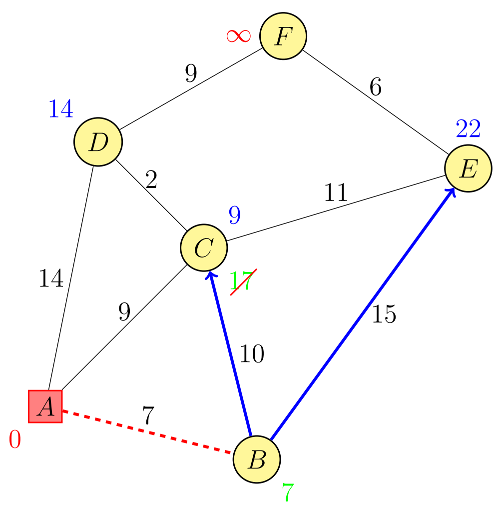
(a) This is the initial step where $MinDist$ of all nodes are assigned a value '∞' except for the source node. The $MinDist$ from the source node to itself is 0. At this stage, all nodes are unvisited.



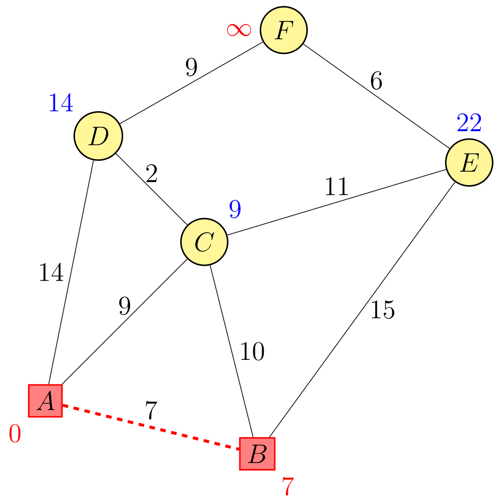
(b) Starting at the source node A, consider all its unvisited neighboring nodes, B, C and D. $MinDist$ of these nodes are updated where $MinDist(B) = 7$, $MinDist(C) = 9$ and $MinDist(D) = 14$.



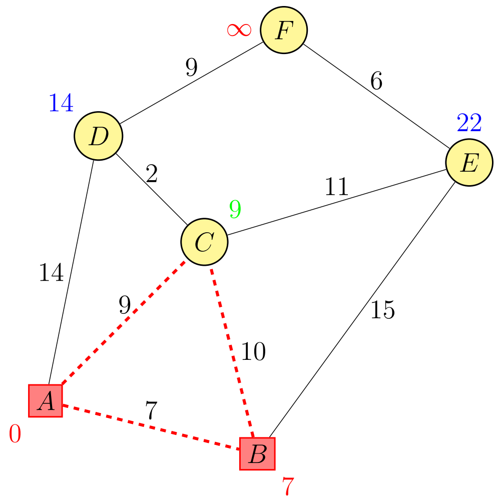
(c) Since all the neighboring nodes of A have been considered, A is marked as visited and it is deleted from Q list. A will not be considered again in the later stage and $MinDist(A)$ is now final. The node with the least $MinDist$ value in Q, B is selected to be considered next. Note that the edge between A and B is not available as A is already visited.



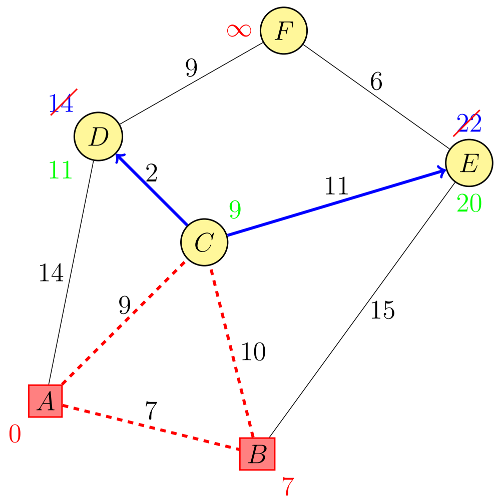
(d) B is the current node to consider. Its unvisited neighboring nodes are C and E. $MinDist(C)$ is checked and the value 9 retained since $9 < 17$. $MinDist(E)$ is updated to 22.



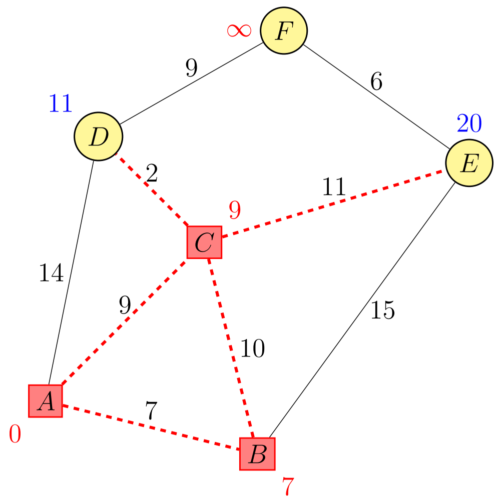
(e) Since all the neighboring nodes of B have been considered, B is marked as visited and it is deleted from Q list. $MinDist(B) = 7$ is now final.



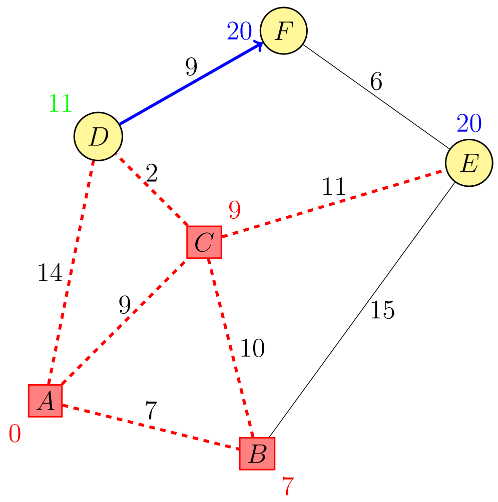
(f) The next node in Q list to be considered is C. Since A and B are visited nodes, the edge connecting them to C is not available.



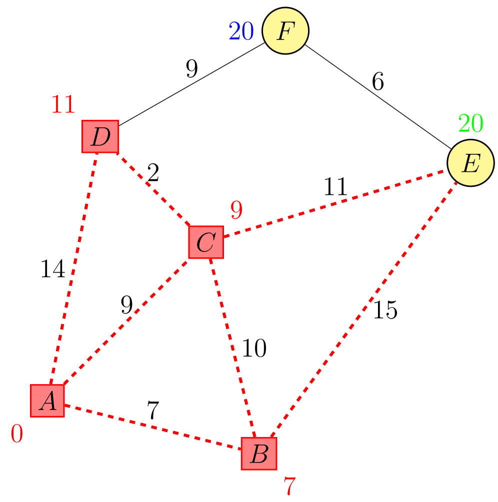
(g) Consider all the unvisited neighboring nodes of C, D and E. $MinDist(D)$ is updated to 11 since $11 < 14$. $MinDist(E)$ is updated to 20 since $20 < 22$.



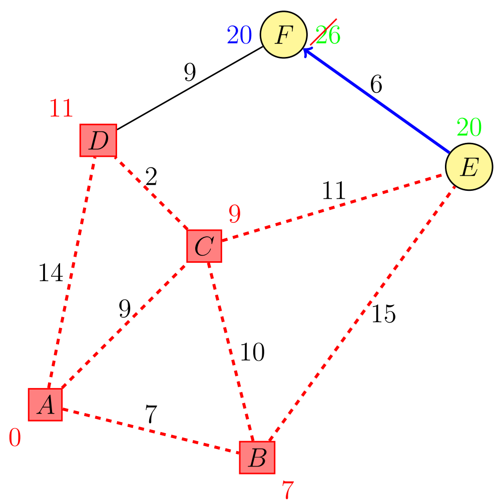
(h) Since all the neighboring nodes of C have been considered, C is marked as visited and it is deleted from Q list. $MinDist(C) = 9$ is now final.



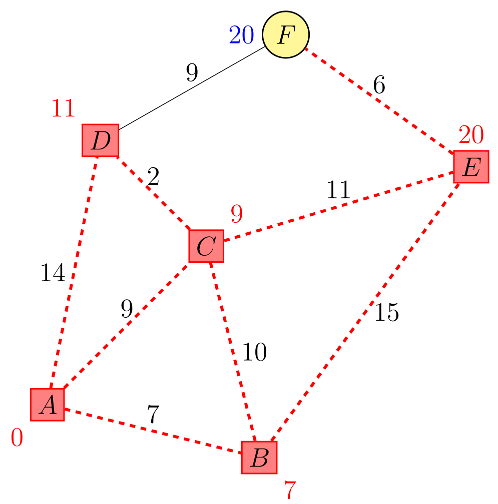
(i) The next node in Q list to be considered is D. F is the only unvisited neighboring node of D. $MinDist(F)$ is updated to 20.



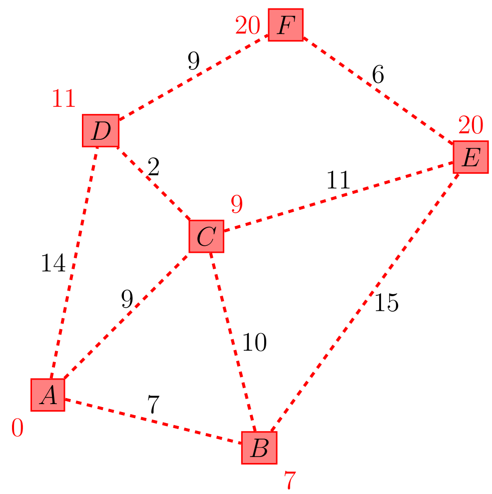
(j) D is now marked as visited and it is deleted from Q list. $MinDist(D) = 11$ is now final. At this stage, $Q = \{E, F\}$ are the unvisited nodes. Both $MinDist(E)$ and $MinDist(F)$ have the same value 20. So, either one can be selected for consideration.



(k) Let E be the node to be considered. The only unvisited neighboring node of E is F. $MinDist(F)$ remains the same since $20 < 26$.



(l) E is now marked as visited and $MinDist(E) = 20$ is now final.



(m) The last node in the Q list is F. F has no unvisited neighboring node. Therefore, $MinDist(F) = 20$ now final. F is deleted from Q list and the algorithm terminates when Q is empty. A shortest distance from A to F is 20 with the shortest path $ACDF$. As indicated earlier, a shortest distances from A to all other nodes (value of $MinDist$) are also found at the end of the algorithm. For instance, a shortest distance from A to D is 11.

Figure 7.14: Dijkstra's algorithm illustrations.

Chapter 8

Test Cases

In this chapter, we will present several test cases with different configurations of faults and centers. We will show that the application of the shortest path problem to scattered data geodesic interpolation gives expected results. The approximations used were all of the kernel based form

$$s(x) = p(x) + \sum_{i=1}^{N} \lambda_i \, \phi\big(\rho(x, x_i)\big), \tag{8.1}$$

where ρ is some metric for $\mathbb{R}^2$, p is a polynomial, and $\phi : [0, \infty) \to \mathbb{R}$. We performed the interpolation in MATLAB using both geodesic shortest distance $d(a, b)$ and direct Euclidean distance $r = \|a - b\|_2$. Direct Euclidean distance will give an interpolation that does not take faults into account. Test cases were of two types. One considered disjoint segments as faults and the other considered connected segments as faults. The interpolant s (solution) of each test case will be illustrated in Figures 8.1 to 8.10.

8.1 Disjoint Segments as Faults

All the test cases were carried out on a $[0, 100] \times [0, 100]$ regular uniform grid (evaluation points) with N interpolation points (centers). For interpolation, we choose the basis function $\phi(r) = (1 - r)^4(4r + 1)$ which is positive definite on $\mathbb{R}^3$ and has C^2 continuity. The support of ϕ can be scaled to adapt the interpolation to scattered data of different densities. We set the radius of support of ϕ to be $R = 50$, hence $r = t/R$, denotes the scaled distance. There are two main computations in our algorithm. The first computation calculates the coefficients $\boldsymbol{\lambda}$ by solving the interpolation equations

$$s(x_i) = f(x_i), \quad 1 \le i \le N. \tag{8.2}$$

In the case of no polynomial part, these equations can be rewritten as a linear system

$$\begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NN} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{pmatrix} \tag{8.3}$$

or

$$\mathbf{A}\boldsymbol{\lambda} = f. \tag{8.4}$$

Note that the ϕ_{ij}'s are defined by

$$\phi_{ij} = \begin{cases} \phi(d(x_i, x_j)/R), & \text{for geodesic shortest distance,} \\ \phi(\|x_i - x_j\|_2/R), & \text{for direct Euclidean distance.} \end{cases} \tag{8.5}$$

The second computation evaluates the interpolant s at evaluation points using the previously obtained coefficients $\boldsymbol{\lambda}$. Algorithm 11 describes the process of computing the interpolant s for a given set of faults, we call it geodesic interpolation.

Figures 8.1 to 8.4 show the results for four different configurations using disjoint segments to represent faults. In each case, the original input data is shown in subfigure (a). The input data consists of faults (segment endpoints), interpolation points and their respective point values or heights. Subfigures (b) and (d) show the contour lines and surface plot of the geodesic interpolant s using shortest distance not crossing the faults. It can be seen that, the faults were reproduced and clearly visible. There is no direct influence of data values across the faults. Indeed the geodesic interpolant is discontinuous across the faults but smooth elsewhere. Further, the influence of data values decays with geodesic distance as evaluation points move around corners of the faults. They were compared to those interpolated using direct Euclidean distance (without faults) as shown in subfigures (c) and (e). For interpolation using direct Euclidean distance, both Lines $6, 7$ and $19, 20$ are replaced with $d(v_s, v_g) = \|v_s - v_g\|_2$.

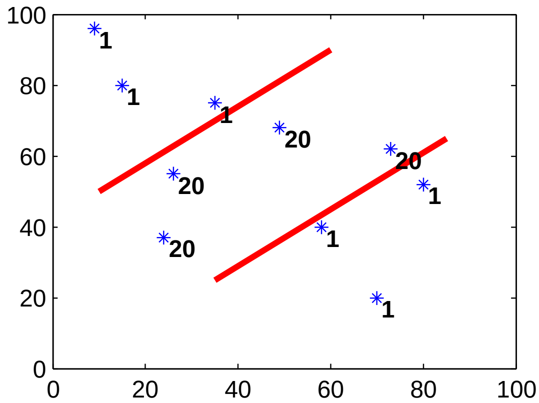
With four different configurations, we have shown that the method we described in Section 7.1.1 and 7.2 works well without any restriction on the position of data points and segment endpoints. Note that the method given in Overmars and Welzl (1988) assumed that all segment endpoints must be in general position as stated in Section 7.1. For example, Test case (i) consists of two segments and they are parallel (see Figure 8.1). Test case (iii) consists of segments that are vertical and parallel. Moreover, three or more endpoints of the segments are collinear too (see Figure 8.3).

Algorithm 11 Geodesic Interpolation

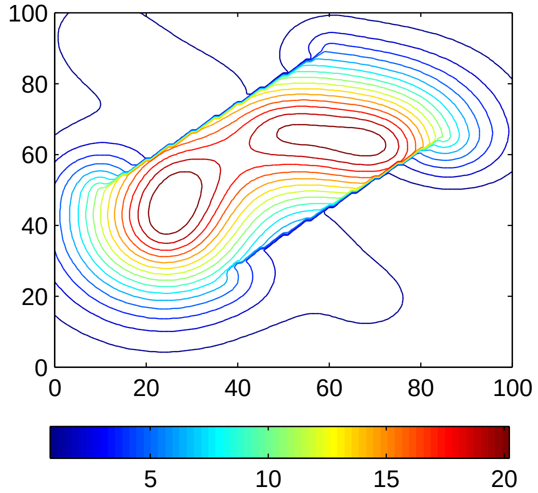
Input: Interpolation data, (x_i, f_i) for $1 \leq i \leq N$,
Segments $S = \{S_1, S_2, \ldots, S_m\}$, and ϕ, type of basis function.

Output: Interpolant $s(x)$.

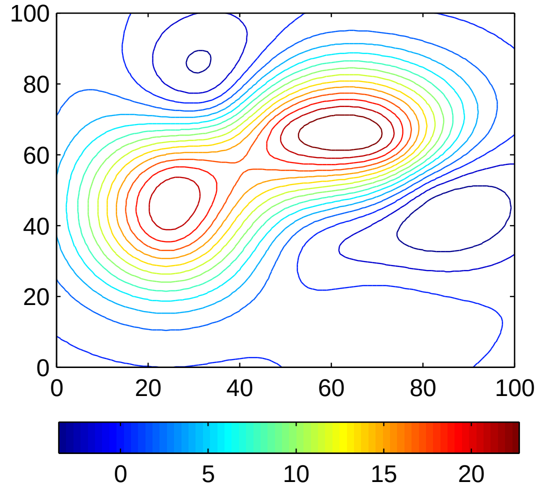
Compute matrix $\mathbf{A}$:
Initialize $\mathbf{Vis}_{\boldsymbol{d}_0}(x_i)$ for $1 \leq i \leq N$
for $1 \leq i \leq N$ **do**
    for $1 \leq j \leq N$ **do**
        $v_s = x_i\,; \quad v_g = x_j\,;$
        $VG{=}\mathit{VisibleEdgeRotTree}(S, v_s, v_g);$
        $d(v_s, v_g) = \mathit{Dijkstra}(VG, v_s, v_g);$
        $\phi_{ij} = \phi(d(v_s, v_g)/R);$
    end for
end for
$\mathbf{A}{=}[\phi_{ij}]$ for $1 \leq i, j \leq N$
$\boldsymbol{\lambda} = \mathbf{A} \setminus f;$
Define evaluation points x_{ij} on grid $0 \leq i, j \leq 100$
for $0 \leq i, j \leq 100$ **do**
    $v_s = x_{ij}\,;$
    $s_{ij} = 0;$
    for $0 \leq k \leq N$ **do**
        $v_g = x_k\,;$
        $VG{=}\mathit{VisibleEdgeRotTree}(S, v_s, v_g);$
        $d(v_s, v_g) = \mathit{Dijkstra}(VG, v_s, v_g);$
        $\phi_k = \phi(d(v_s, v_g)/R);$
        $s_{ij} = s_{ij} + \lambda_k\, \phi_k\,;$
    end for
end for



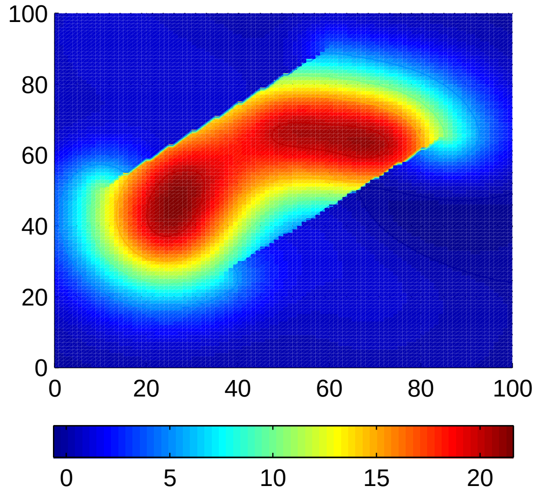
(a) Input Data



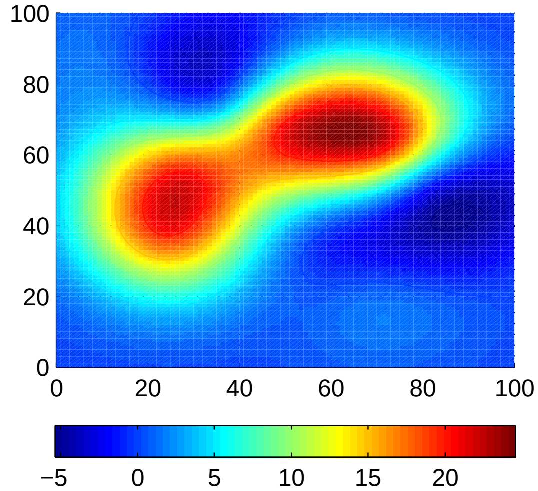
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

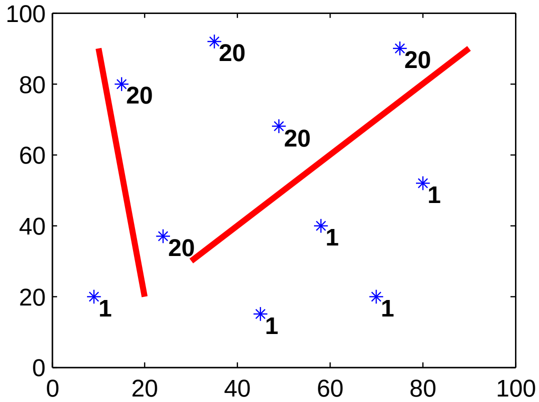


(c) Surface plot (X-Y view) of interpolant s with faults.

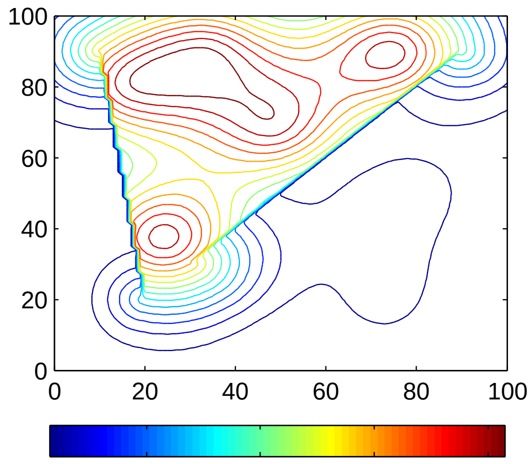


(d) Surface plot (X-Y view) of interpolant s without faults.

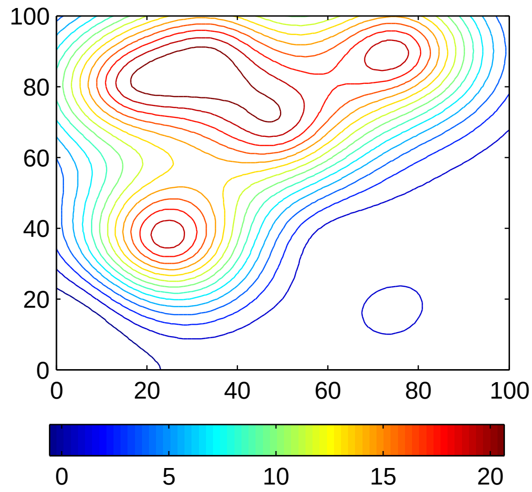
Figure 8.1: Test Case (i).



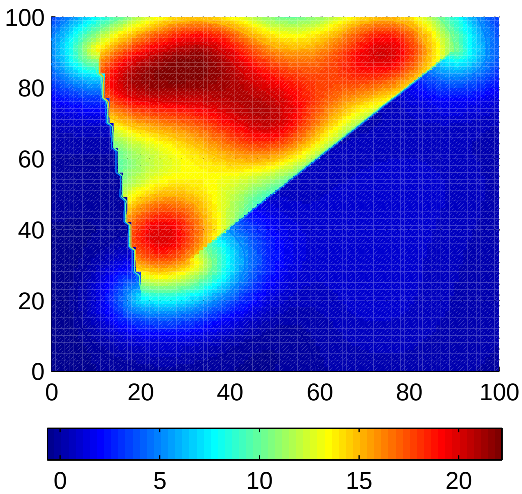
(a) Input Data



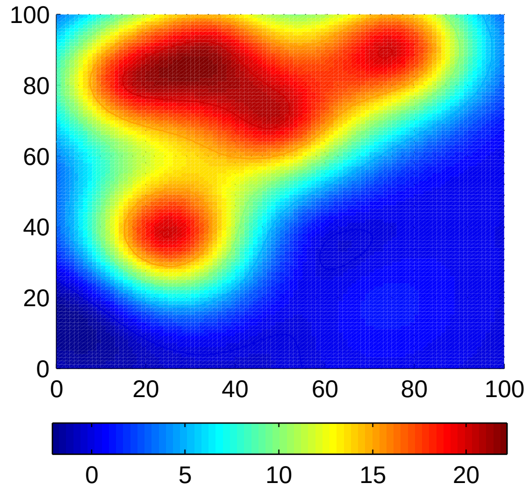
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

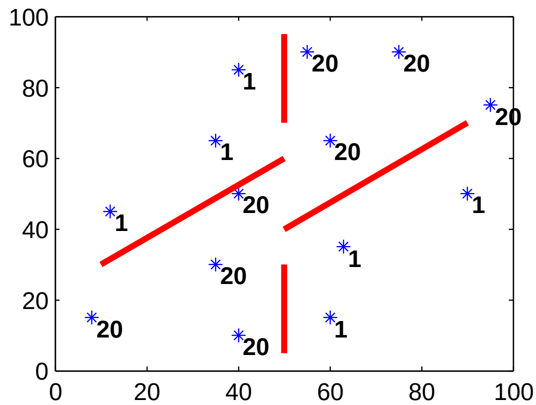


(c) Surface plot (X-Y view) of interpolant s with faults.

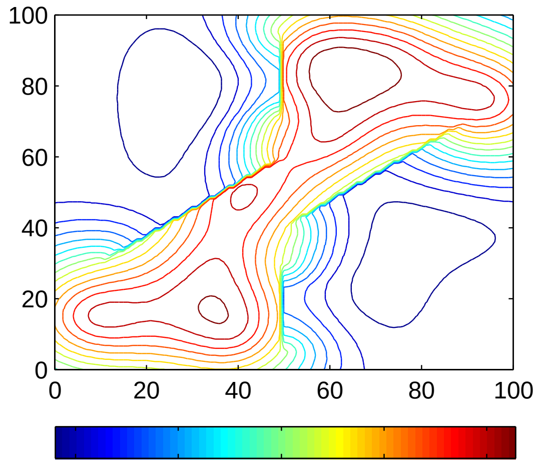


(d) Surface plot (X-Y view) of interpolant s without faults.

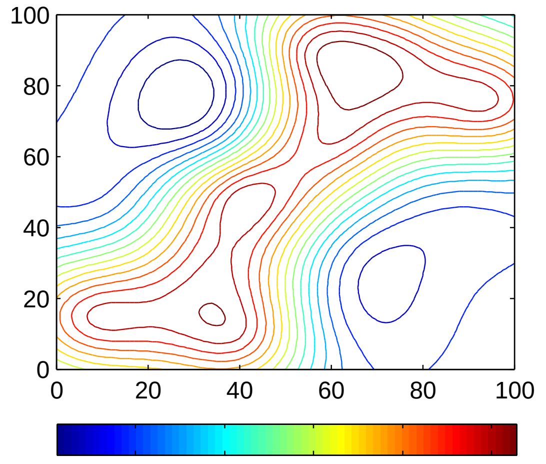
Figure 8.2: Test Case (ii).



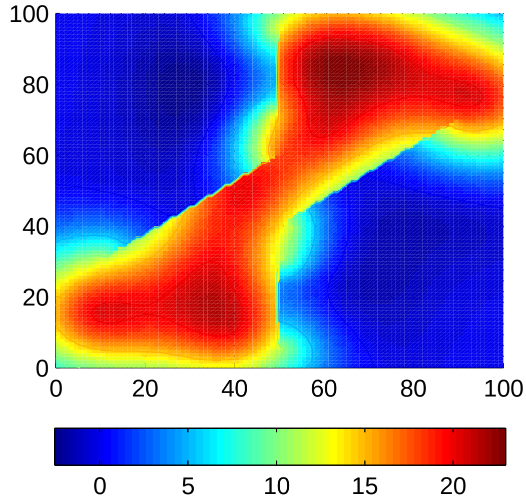
(a) Input Data



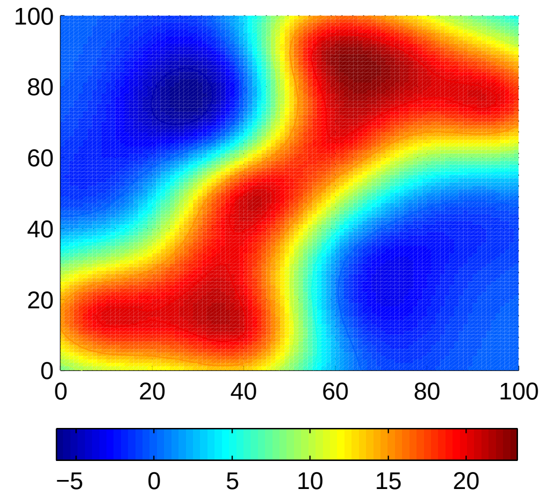
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

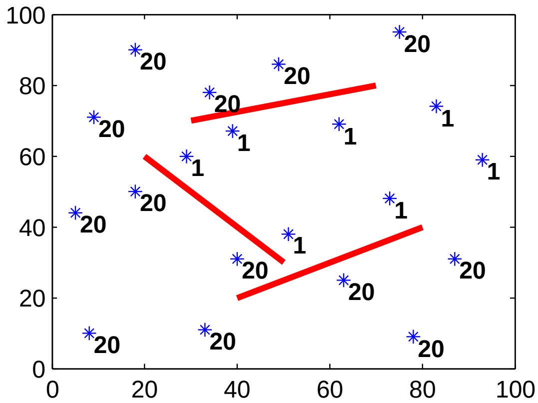


(c) Surface plot (X-Y view) of interpolant s with faults.

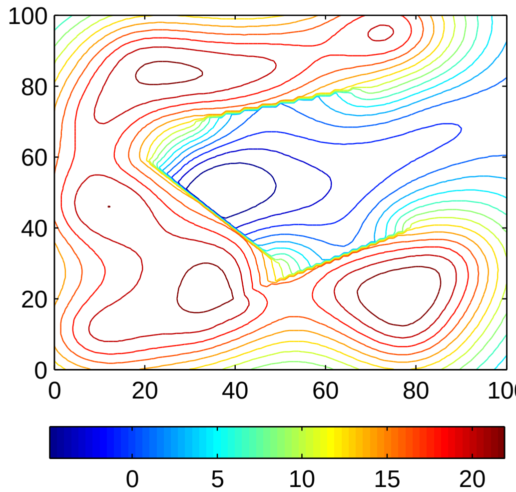


(d) Surface plot (X-Y view) of interpolant s without faults.

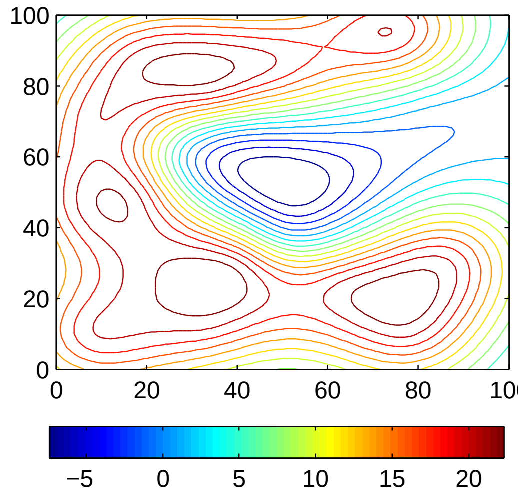
Figure 8.3: Test Case (iii).



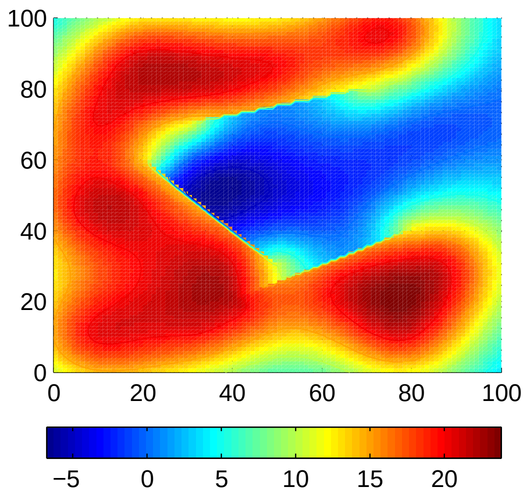
(a) Input Data



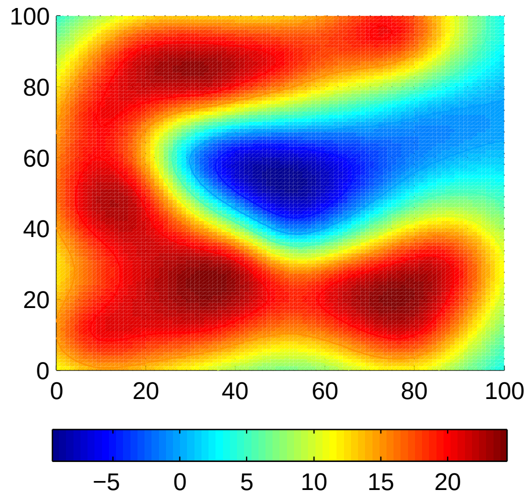
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.



(c) Surface plot (X-Y view) of interpolant s with faults.



(d) Surface plot (X-Y view) of interpolant s without faults.

Figure 8.4: Test Case (iv).

8.2 Connected Segments as Faults

Earlier in Section 7.3, we have given a method to construct the visibility graph of polygonal obstacles. In this section, we will describe how the method is applied to perform interpolation with connected segments as faults. Reasonably, a set of connected segments can be viewed as a simple polygon with no area or area ≈ 0 (see Figure 8.5).

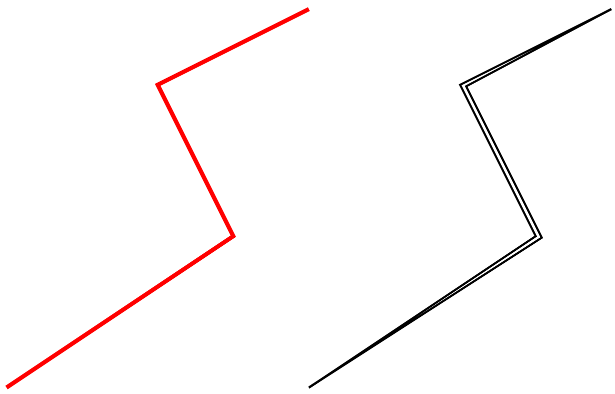


Figure 8.5: A set of connected segments represented by a narrow polygon.

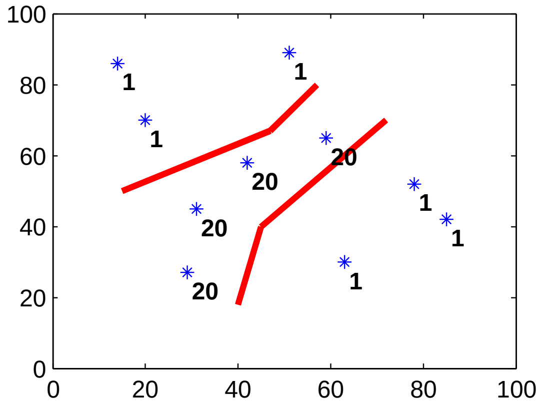
In other words, we will have m narrow polygons for a set of m connected segments. Figure 8.6(a) shows an example of two sets of connected segments. With this setting, constructing the visibility graph for these connected segments is equivalent to constructing the visibility graph for polygonal obstacles. Hence Algorithm 11 can be used to perform geodesic interpolation for connected segments as faults, with changes made on Lines 6 and 19 (constructing the visibility graph). Refer to Algorithm 7 *VisibleEdgeRotTree*, Line 12 is replaced by routine *VisiblePolygonal* (Algorithm 9).

Figures 8.6 to 8.10 show the result of five different configurations using connected segments as faults. Comparisons between interpolation using shortest distance avoiding and crossing the faults are presented. We again observe that, discontinuities across the faults are significant.

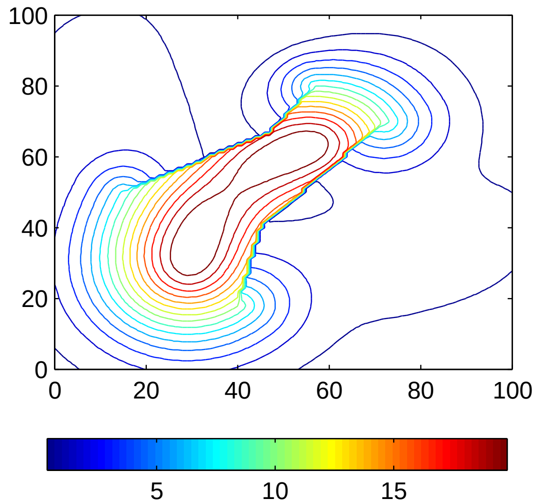
For a specified configuration, we also carried out interpolation using the same basis function $\phi(r) = (1-r)^4(4r+1)$ with varying support radius R. The results are shown in Figure 8.11. When R is small, the basis functions are too localized and generally the interpolation is a poor approximation in between data points (see Figure 8.11(b)). It is clear that the quality of approximation can be improved by increasing the support.

For comparison purposes, we show interpolated surfaces of the same configuration using Wendland functions of smoothness C^0, C^2, and C^4, at $R = 50$, see Figures 8.12(a)–(c). Figure 8.12(a) clearly shows C^0 peaks of the interpolant. Moreover, the Gaussian,

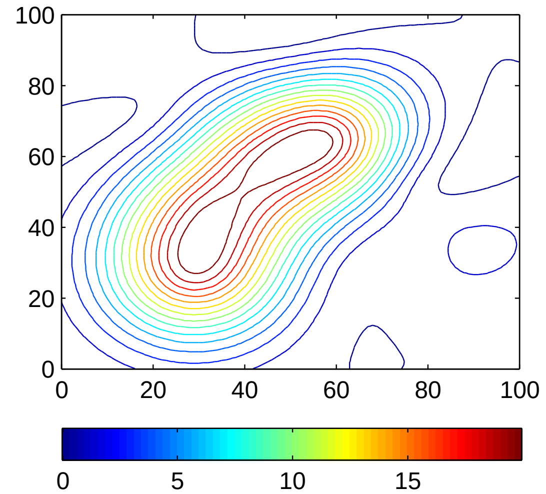
$\phi(r) = e^{-\alpha r^2}$ is positive definite on every $\mathbb{R}^d$ for all positive parameters α. Therefore, we interested to see the behavior of interpolant when using Gaussian as basis function, see Figure 8.12$(d)-(f)$. These calculations are carried out with different values of the scale parameter α. If the parameter is large, the Gaussian becomes sharply peaked. This observation is similar to the case for Wendland function when a smaller radius of support was used.



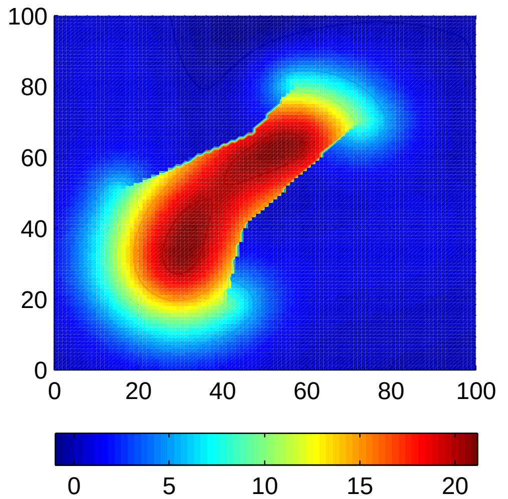
(a) Input Data



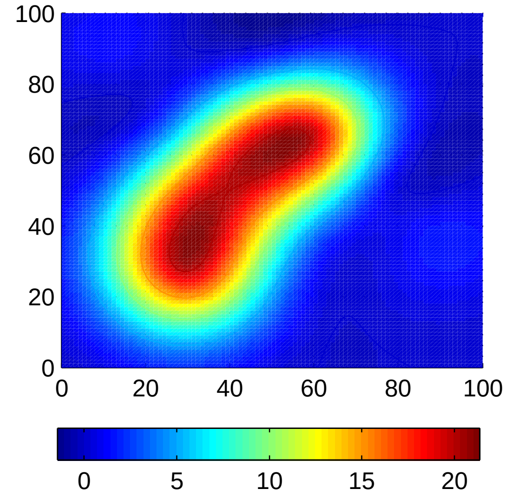
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

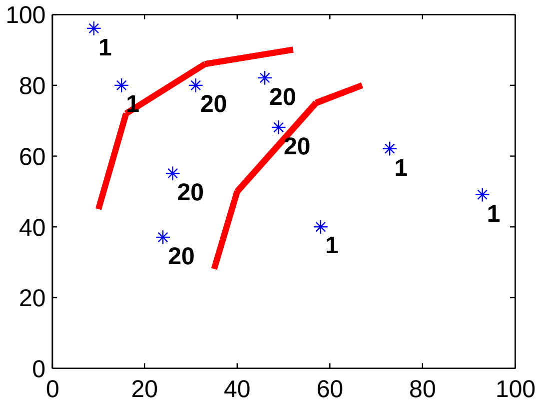


(c) Surface plot (X-Y view) of interpolant s with faults.

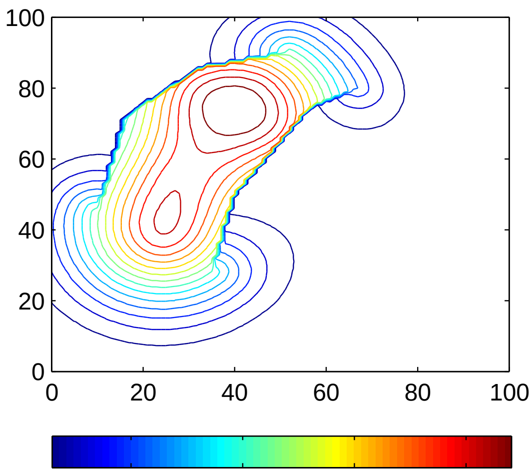


(d) Surface plot (X-Y view) of interpolant s without faults.

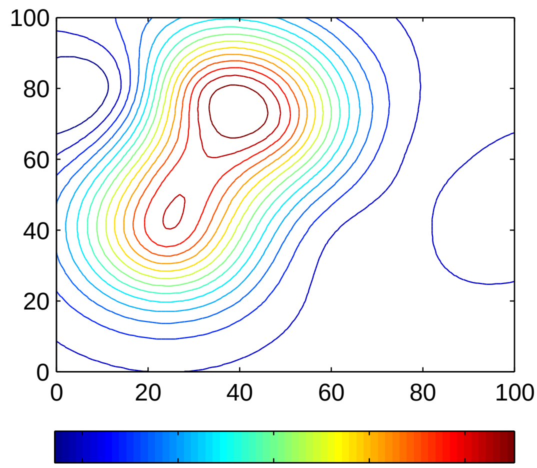
Figure 8.6: Test Case (v).



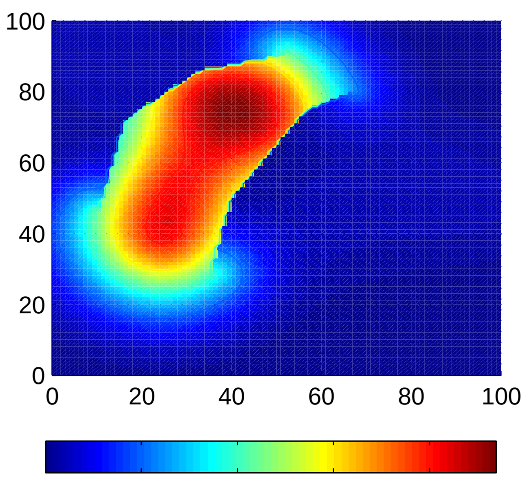
(a) Input Data



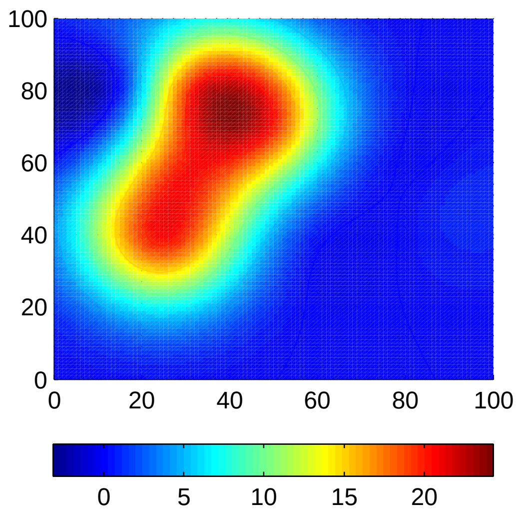
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

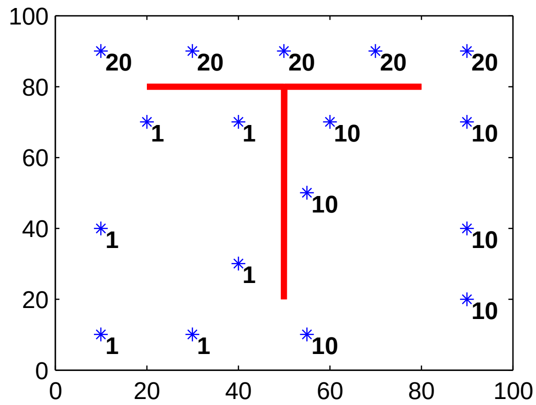


(c) Surface plot (X-Y view) of interpolant s with faults.

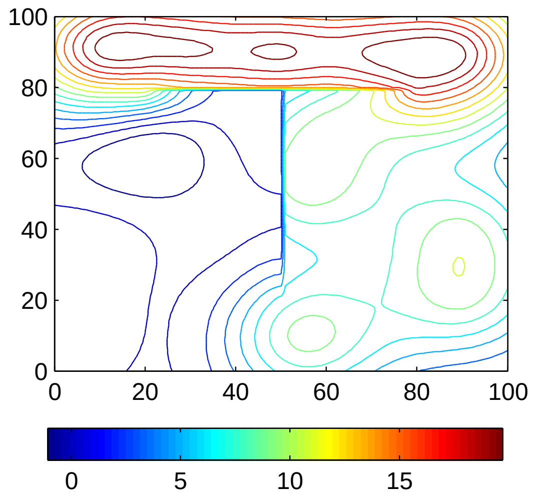


(d) Surface plot (X-Y view) of interpolant s without faults.

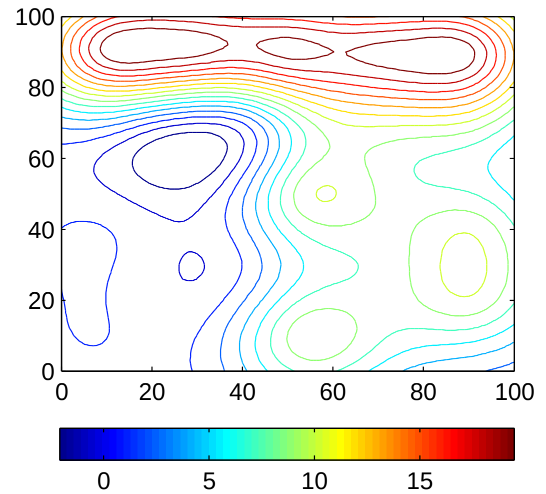
Figure 8.7: Test Case (vi).



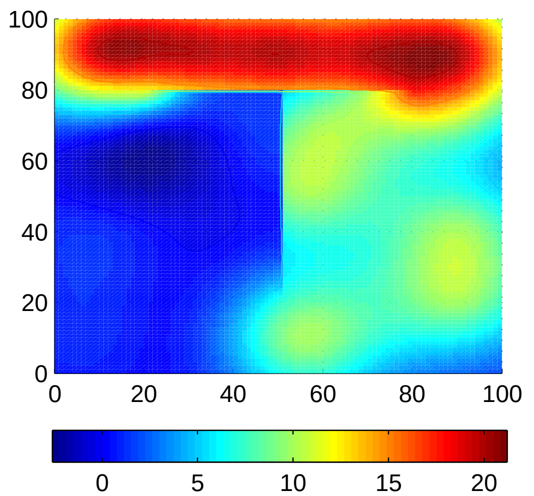
(a) Input Data



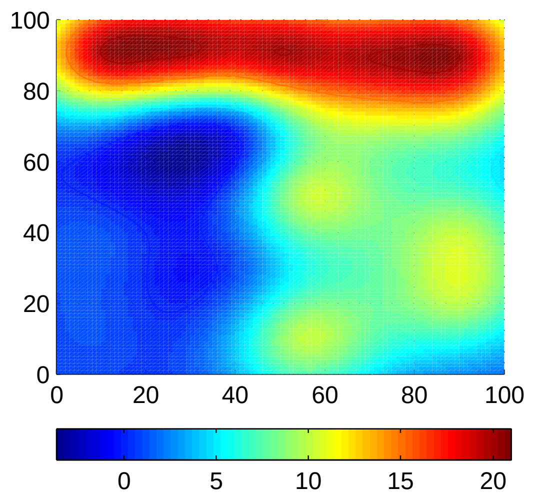
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

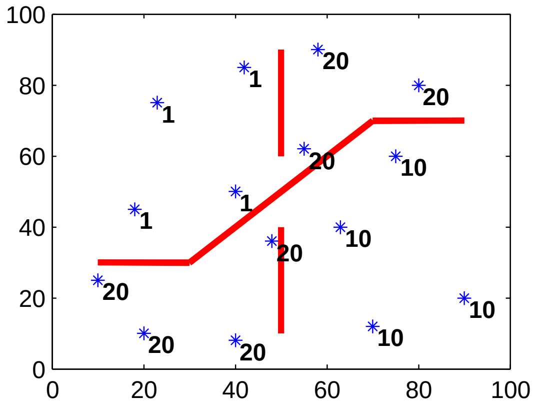


(c) Surface plot (X-Y view) of interpolant s with faults.

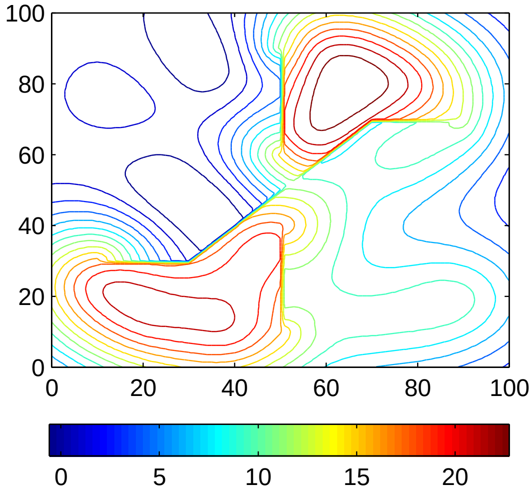


(d) Surface plot (X-Y view) of interpolant s without faults.

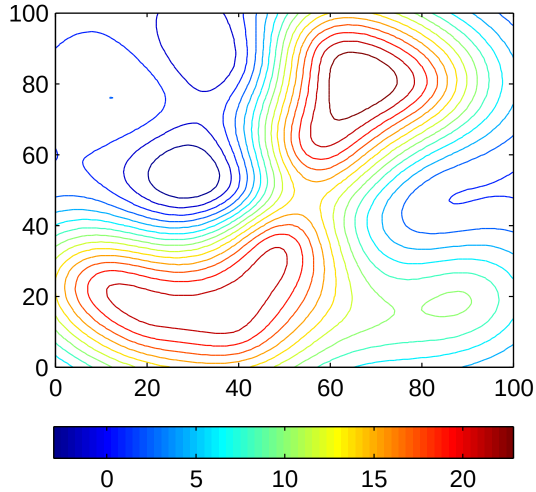
Figure 8.8: Test Case (vii).



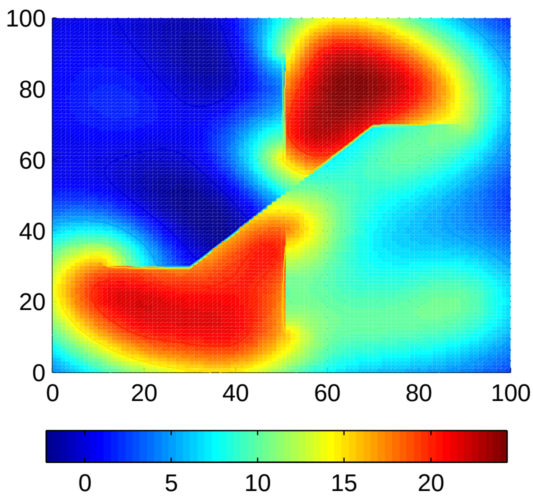
(a) Input Data



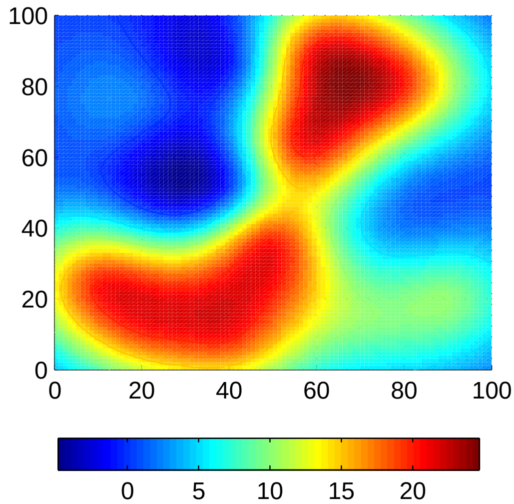
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

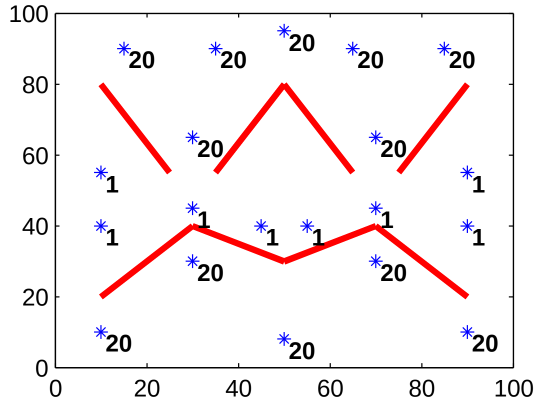


(c) Surface plot (X-Y view) of interpolant s with faults.

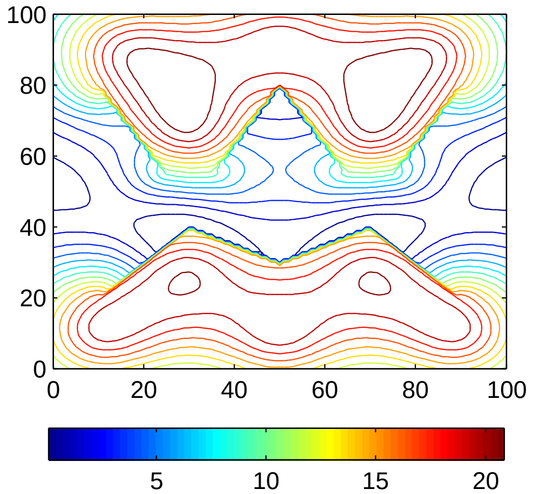


(d) Surface plot (X-Y view) of interpolant s without faults.

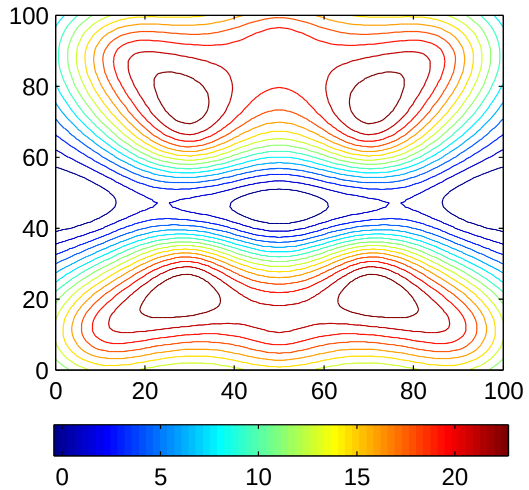
Figure 8.9: Test Case (viii).



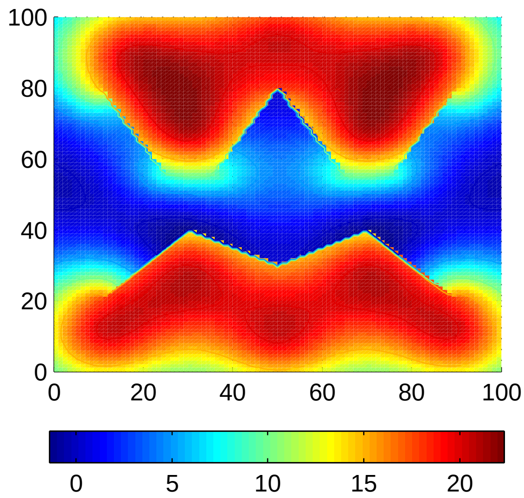
(a) Input Data



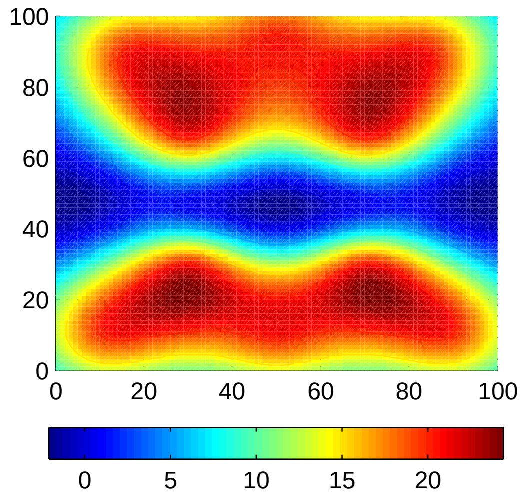
(a) Contour lines of interpolant s with faults.



(b) Contour lines of interpolant s without faults.

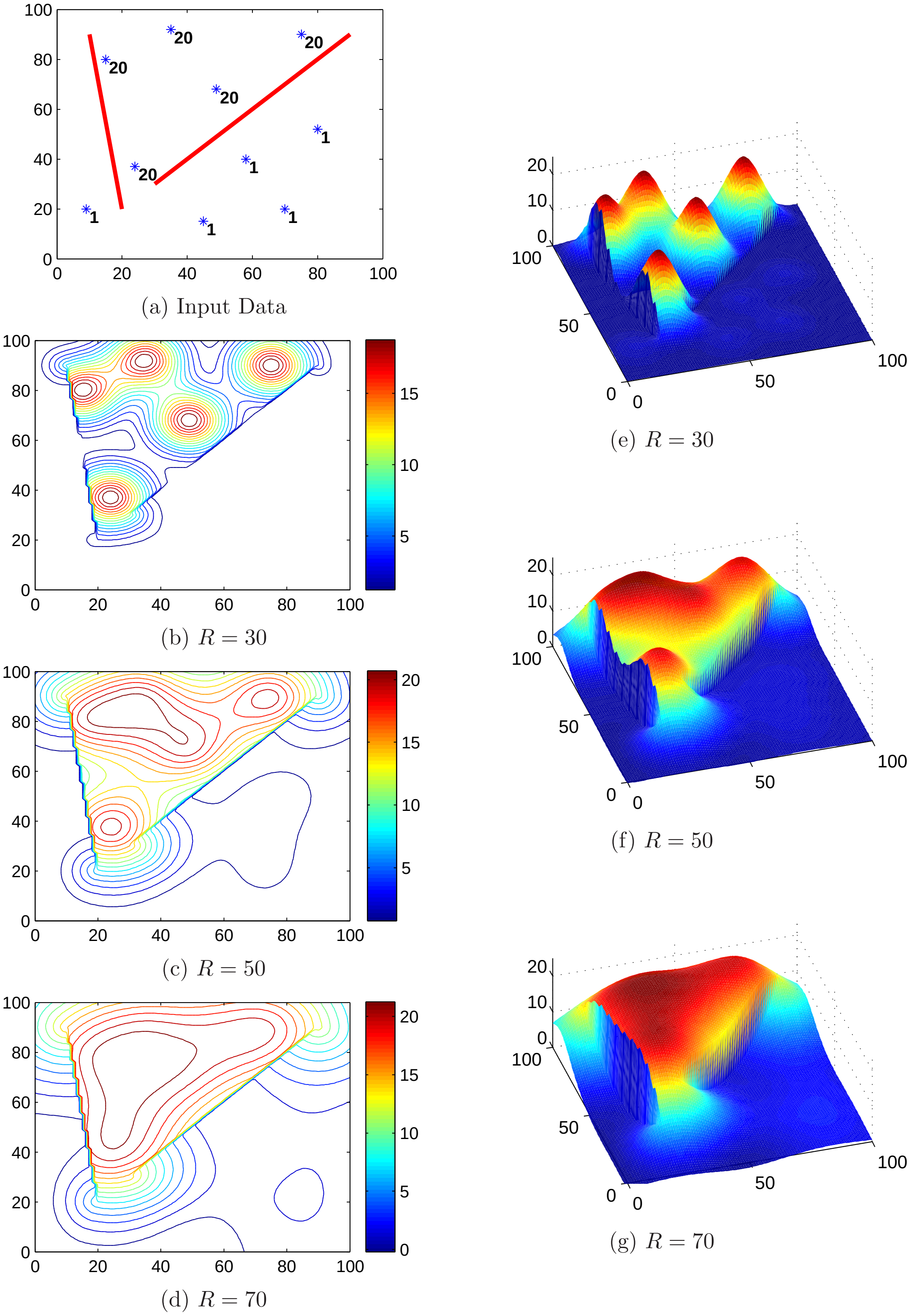


(c) Surface plot (X-Y view) of interpolant s with faults.



(d) Surface plot (X-Y view) of interpolant s without faults.

Figure 8.10: Test Case (ix).



(a) Input Data

(b) $R = 30$

(c) $R = 50$

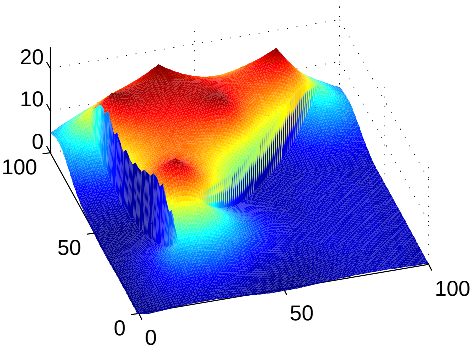
(d) $R = 70$

(e) $R = 30$

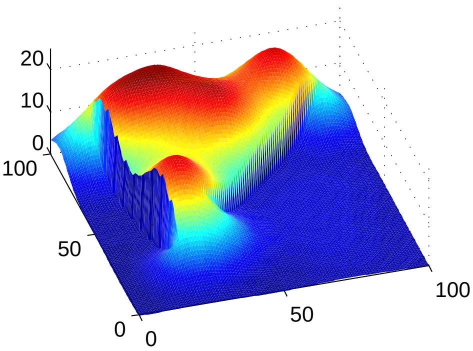
(f) $R = 50$

(g) $R = 70$

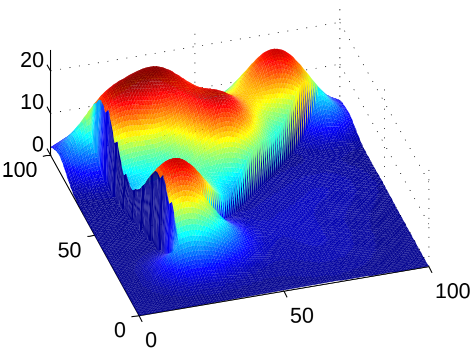
Figure 8.11: Subfigure (a) shows a given configuration of centers and faults. Subfigures $(b) - (d)$ show the contour lines of interpolant s at different radius of support (R) and the corresponding surface plot are shown in subfigures $(e) - (g)$.



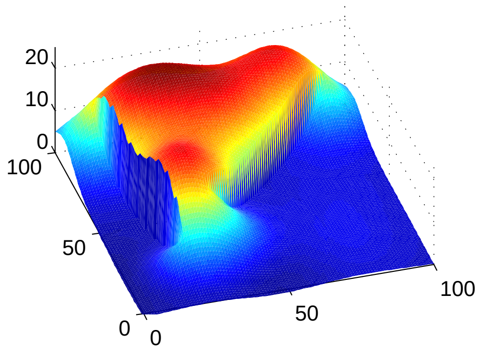
(a) $\phi(r) = (1-r)^2$ with C^0 smoothness.



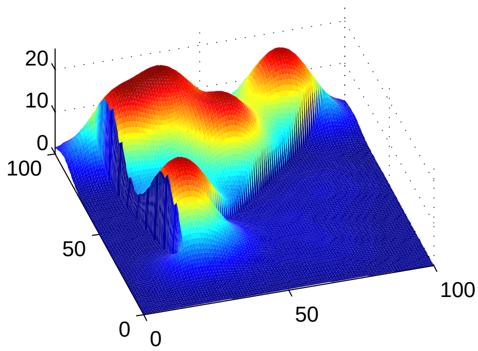
(b) $\phi(r) = (1-r)^4(4r+1)$ with C^2 smoothness.



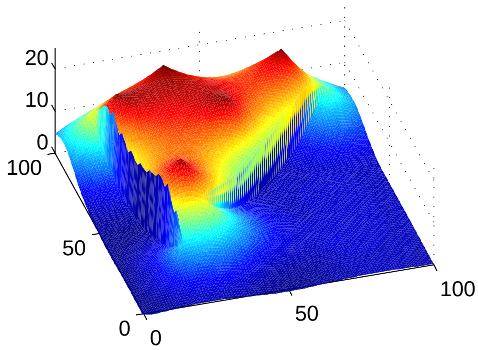
(c) $\phi(r) = (1-r)^6(35r^2+18r+3)$ with C^4 smoothness.



(d) $\phi(r) = e^{-\alpha r^2}$ with $\alpha = 0.2$.



(e) $\phi(r) = e^{-\alpha r^2}$ with $\alpha = 0.15$.



(f) $\phi(r) = e^{-\alpha r^2}$ with $\alpha = 0.1$.

Figure 8.12: Subfigures $(a) - (c)$ show surface plots of interpolant s using different Wendland functions of C^0, C^2 and C^4 smoothness respectively. Subfigures $(d) - (f)$ show surface plots of interpolant s using Gaussian function at different scale parameter.

8.3 Conclusions and Future Directions

We have described a method that fits a surface to scattered data with the presence of obstacles. This method involves constructing the visibility graph and solving the shortest path problem for the graph. As observed in various test cases presented in the last section, the results showed correct behaviour. The faults are clearly reproduced, and the cross fault influence is small.

Although the results are as expected, we admit that there are drawbacks in our method. The current implementation is in MATLAB and the algorithm is by no means optimal. For even reasonably small problems, the compute time is so large as to preclude the use of this method as a standard approach. To ensure its practicality, we suggest a few possible way to optimize and improve the current method.

- Implementing the current method in C will be our next task. We are interested in recording some timings for this implementation in C. Based on our experience in MATLAB, more time is required to construct the visibility graph than to solve the shortest path problem using Dijkstra's algorithm. We shall investigate further on this matter.

- For visibility graph construction, while much effort has been made to derive the complexity (lower bound and worst-case optimal) of an algorithm, no timings have been reported in the literature to date. Besides, we are likely to get advantage from more efficient algorithms such as output sensitive algorithms. These algorithms are much faster when the obstacles are formed by many configurations of line segments because the visibility graph constructed will be sparse (less edges).

  The main issue in our current method is the large number of visibility graphs generated in the interpolation process. For all evaluation points, part of the visibility graph is generated and used repeatedly. One way to avoid this redundancy is to pre-compute and store part of the visibility graph for later usage.

- For solving the shortest path problem, improvement can be done by using a Fibonacci heap based priority queue to efficiently extract the closest unvisited node in Dijkstra's algorithm. Various other techniques to speed-up Dijkstra's algorithm are known, see for example Wagner and Willhalm (2007).

  Also, we are interested in applying the bidirectional (Luby and Ragde, 1989) search speed-up technique to our case. Instead of carrying out a unidirectional search from the source node towards the goal node (original Dijkstra's algorithm), the bidirectional search carries out two searches simultaneously: one search from the source node and another search from the goal node (backward search). The algorithm can then be terminated when one node has been assigned to be a visited node by both searches. In a complete directed graph with n nodes, Luby and Ragde (1989)'s bidirectional search algorithm has the expected running time $\mathcal{O}(\sqrt{n}\log n)$ with the condition where the adjacency lists at each node are sorted by length and the length of each edge is chosen independently from the exponential distribution. In the problem at hand, it is reasonable to expect that with a bidirectional approach some work associated with a source will not need to be redone when only the target changes.

- I believe that the examples given show the approach to interpolation with obstacles detailed in these 3 chapters works. The approach has passed the "proof of concept" stage. The obstacle is the massive computational cost. For this reason, I am interested in exploring massively parallel implementations of the shortest path algorithm (Singh and Khare, 2012), for example using NVIDIA's CUDA (Martín et al., 2009; Harish and Narayanan, 2007) to solve shortest path problem. The work of Madduri et al. (2007) using multithread parallel computer also shows a remarkable speedup.

Appendix A

Rotation tree illustrations

Here we illustrate the operations of Algorithm 7 based on the example in Figure A.1. The rotation tree is updated at each step. To begin the process, all the nodes are sorted by **Sort**(V) that gives $V = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. When the tree is initialized at $d = -\pi/2$, $p_{-\infty}$ has p_∞ as its parent and every node in V has $p_{-\infty}$ as its parent (see Figure A.2(a)). The subsequent steps of updating the rotation tree is shown in Figure A.2. At the end of the process, all the nodes have p_∞ as their parent. All sons of p_∞ are arranged in increasing x-coordinate order from left to right. The node with smallest x-coordinate is the left most node.

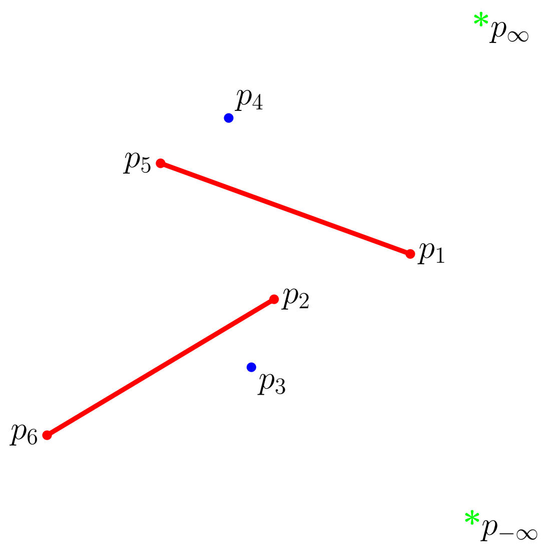
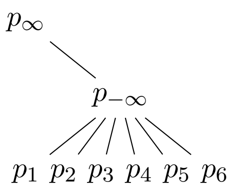
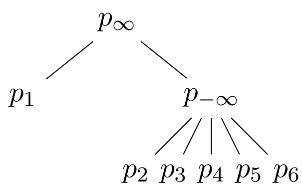


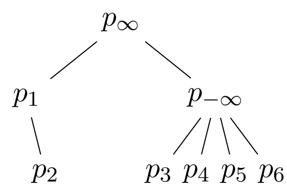
Figure A.1: Example



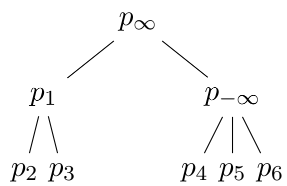
(a) Initial tree. At this step, $p = p_1$ and $q = p_{-\infty}$. Since $q = p_{-\infty}$, the visibility of $\overline{pq}$ will not be considered. p is then removed from tree and the tree is updated.



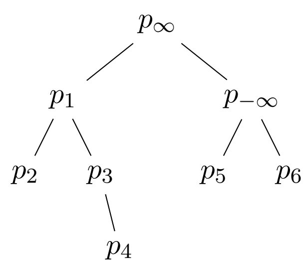
(b) At this step, $p = p_2$ and $q = p_{-\infty}$.



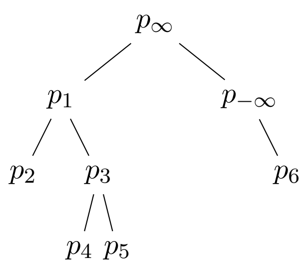
(c) At this step, $p = p_3$ and $q = p_{-\infty}$.



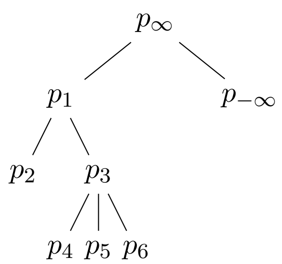
(d) At this step, $p = p_4$ and $q = p_{-\infty}$.



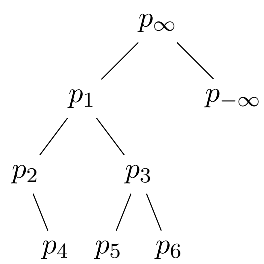
(e) At this step, $p = p_5$ and $q = p_{-\infty}$.



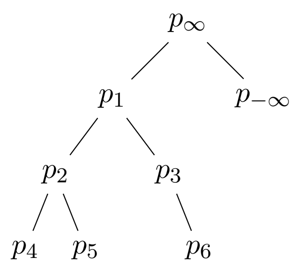
(f) At this step, $p = p_6$ and $q = p_{-\infty}$.



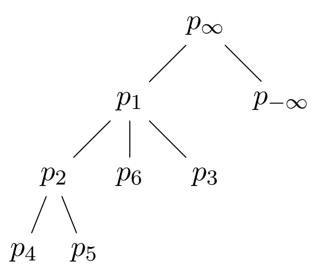
(g) At this step, $p = p_4$ and $q = p_3$. The visibility of $\overline{pq}$ is determined and recorded.



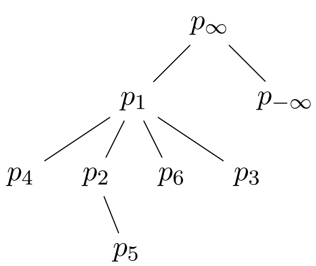
(h) At this step, $p = p_5$ and $q = p_3$.



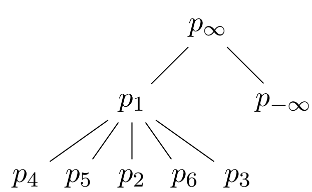
(i) At this step, $p = p_6$ and $q = p_3$.



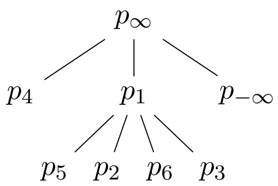
(j) At this step, $p = p_4$ and $q = p_2$.



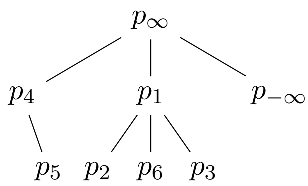
(k) At this step, $p = p_5$ and $q = p_2$.



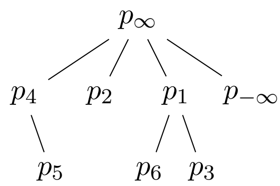
(l) At this step, $p = p_4$ and $q = p_1$.



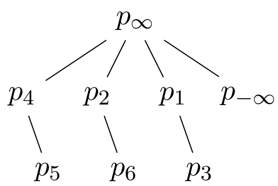
(m) At this step, $p = p_5$ and $q = p_1$.



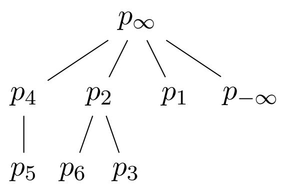
(n) At this step, $p = p_2$ and $q = p_1$.



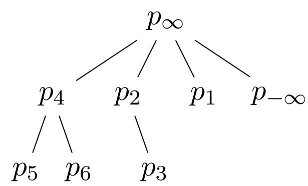
(o) At this step, $p = p_6$ and $q = p_1$.



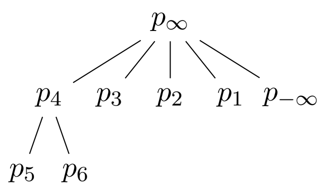
(p) At this step, $p = p_3$ and $q = p_1$.



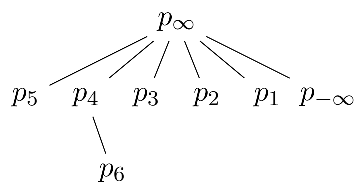
(q) At this step, $p = p_6$ and $q = p_2$.



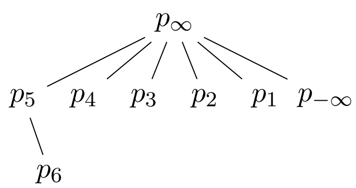
(r) At this step, $p = p_3$ and $q = p_2$.



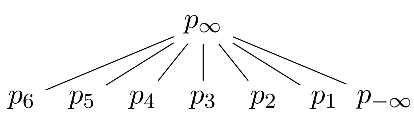
(s) At this step, $p = p_5$ and $q = p_4$.



(t) At this step, $p = p_6$ and $q = p_4$.



(u) At this step, $p = p_6$ and $q = p_5$.



(v) The final step of the rotation tree.

Figure A.2: Illustration of the rotational plane sweep procedure, showing the updated rotation tree at each step.

Bibliography

R.K. Ahuja, K. Mehlhorn, J. Orlin, and R.E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990.

P. Alfeld. Scattered data interpolation in three or more variables. *Mathematical methods in computer aided geometric design*, pages 1–34, 1989.

H. Alt and E. Welzl. Visibility graphs and obstacle-avoiding shortest paths. *Mathematical Methods of Operations Research*, 32(3):145–164, 1988.

N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421. ACM, 1998.

I. Amidror. Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of electronic imaging*, 11:157, 2002.

T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.

M. Attene and M. Spagnuolo. Automatic surface reconstruction from point sets in space. In *Computer Graphics Forum*, volume 19, pages 457–465. Wiley Online Library, 2000.

M. Attene, B. Falcidieno, and M. Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22(3):181–193, 2006.

R.E. Barnhill. Surfaces in computer aided geometric design: A survey with new results. *Computer Aided Geometric Design*, 2(1-3):1–17, 1985.

R.K. Beatson and L. Greengard. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs*, pages 1–37, 1997.

R.K. Beatson and G.N. Newsam. Fast evaluation of radial basis functions: I. *Computers & Mathematics with Applications*, 24(12):7–19, 1992.

R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned gmres iteration. *Advances in Computational Mathematics*, 11(2):253–270, 1999.

R.K. Beatson, J.B. Cherrie, and D.L. Ragozin. Fast evaluation of radial basis functions: Methods for four-dimensional polyharmonic splines. *SIAM journal on mathematical analysis*, 32(6):1272–1310, 2001a.

R.K. Beatson, W.A. Light, and S. Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM Journal on Scientific Computing*, 22(5):1717–1740, 2001b.

S.D. Billings, R.K. Beatson, and G.N. Newsam. Interpolation of geophysical data using continuous global surfaces. *Geophysics*, 67(6):1810, 2002a.

S.D. Billings, G.N. Newsam, and R.K. Beatson. Smooth fitting of geophysical data using continuous global surfaces. *Geophysics*, 67(6):1823, 2002b.

S. Bischoff and L. Kobbelt. Ellipsoid decomposition of 3D-models. *3DPVT Proceedings*, pages 480–488, 2002a.

S. Bischoff and L. Kobbelt. Towards robust broadcasting of geometry data. *Computers & Graphics*, 26(5):665–675, 2002b.

J. Bloomenthal, C. Bajaj, J. Blinn, M.P. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. Introduction to implicit surfaces. 1997.

R.P. Boas. *Entire functions*, volume 5. Academic Press, 1954.

D. Brujic, I. Ainsworth, and M. Ristic. Fast and accurate nurbs fitting for reverse engineering. *The International Journal of Advanced Manufacturing Technology*, pages 1–10, 2010.

M.D. Buhmann. Radial basis functions. *Acta Numerica 2000*, 9(1):1–38, 2000.

M.D. Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge Univ Pr, 2003.

J.C. Carr, W.R. Fright, and R.K. Beatson. Surface interpolation with radial basis functions for medical imaging. *Medical Imaging, IEEE Transactions on*, 16(1):96–107, 1997.

J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

J.C. Carr, R.K. Beatson, B.C. McCallum, W.R. Fright, T.J. McLennan, and T.J. Mitchell. Smooth surface reconstruction from noisy range data. *ACM GRAPHITE*, 3:119–126, 2003.

A. J. Champandard. Path-planning from start to finish, June 2002. URL `http://ai-depot.com/articles/path-planning-from-start-to-finish/`.

W. Cheney and W. Light. A course in approximation theory, brooks, 2000.

K.S.D. Cheng, W. Wang, H. Qin, K.Y.K. Wong, H. Yang, and Y. Liu. Design and analysis of optimization methods for subdivision surface fitting. *IEEE Transactions on Visualization and Computer Graphics*, pages 878–890, 2007.

J.B. Cherrie, R.K. Beatson, and G.N. Newsam. Fast Evaluation of Radial Basis Functions: Methods for Generalized Multiquadrics in Rˆ n. *SIAM Journal on Scientific Computing*, 23(5):1549–1571, 2002.

P.N. Chivate and A.G. Jablokow. Review of surface representations and fitting for reverse engineering. *Computer Integrated Manufacturing Systems*, 8(3):193–204, 1995.

H. Choset, KM Lynch, S. Hutchinson, G. Kantor, W. Burgard, LE Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. The MIT Press, 2005.

E. Cohen, T. Lyche, and R. Riesenfeld. Discrete b-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 14(2):87–111, 1980.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms, 2nd Edition*. The MIT press, 2001.

M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc, 2008.

A. De Boer, MS Van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & structures*, 85(11-14):784–795, 2007.

T.K. Dey and P. Kumar. A simple provable algorithm for curve reconstruction. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 893–894. Society for Industrial and Applied Mathematics, 1999.

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

J. Dongarra and F. Sullivan. Guest editors' introduction: The top 10 algorithms. *Computing in Science and Engineering*, 2(1):22–23, 2000.

W.H. Du and F.J.M. Schmitt. On the g1 continuity of piecewise bézier surfaces: a review with new results. *Computer-Aided Design*, 22(9):556–573, 1990.

Y. Duan, L. Yang, H. Qin, and D. Samaras. Shape reconstruction from 3d and 2d data using pde-based deformable surfaces. *Computer Vision-ECCV 2004*, pages 238–251, 2004.

M. Eck and H. Hoppe. Automatic reconstruction of b-spline surfaces of arbitrary topological type. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 325–334. ACM, 1996.

H. Edelsbrunner and L.J. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38(1):165–194, 1989.

T. El Abbass, C. Jallouli, Y. Albouy, and M. Diament. A comparison of surface fitting algorithms for geophysical data. *Terra Nova*, 2(5):467–475, 1990.

W. Eric, L. Grimson, and T. Pavlidis. Discontinuity detection for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 30(3):316–330, 1985.

G. Farin. A history of curves and surfaces in cagd. *Handbook of Computer Aided Geometric Design, G. Farin, J. Hoschek and M.-S. Kim, eds., Elsevier*, pages 1–21, 2002a.

G.E. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Pub, 2002b.

M.S. Floater and A. Iske. Multistep scattered data interpolation using compactly supported radial basis functions. *Journal of Computational and Applied Mathematics*, 73 (1-2):65–78, 1996.

M.S. Floater and M. Reimers. Meshless parameterization and surface reconstruction. *Computer Aided Geometric Design*, 18(2):77–92, 2001.

S. Flöry. Fitting curves and surfaces to point clouds in the presence of obstacles. *Computer Aided Geometric Design*, 26(2):192–202, 2009.

D. Forsey and D. Wong. Multiresolution surface reconstruction for hierarchical b-splines. In *Graphics Interface*, pages 57–64. Canadian Information Processing Society, 1998.

R. Franke and G.M. Nielson. Scattered data interpolation and applications: A tutorial and survey. *Geometric Modelling: Methods and Their Application*, pages 131–160, 1991.

G. Gallo and S. Pallottino. Shortest path algorithms. *Annals of Operations Research*, 13(1):1–79, 1988.

S.K. Ghosh. *Visibility algorithms in the plane*. Cambridge Univ Pr, 2007.

S.K. Ghosh and D.M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20:888, 1991.

H. Gonzalez, J. Han, X. Li, M. Myslinska, and J.P. Sondag. Adaptive fastest path computation on a road network: A traffic mining approach. In *Proceedings of the 33rd international conference on Very large data bases*, pages 794–805. VLDB Endowment, 2007.

L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

A. Gregory, A. State, M.C. Lin, D. Manocha, and M.A. Livingston. Interactive surface decomposition for polyhedral morphing. *The Visual Computer*, 15(9):453–470, 1999.

L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

H. Hagen, G. Nielson, and Y. Nakajima. Surface design using triangular patches. *Computer aided geometric design*, 13(9):895–904, 1996.

S.P. Han. A globally convergent method for nonlinear programming. *Journal of optimization theory and applications*, 22(3):297–309, 1977.

P. Harish and P. Narayanan. Accelerating large graph algorithms on the gpu using cuda. *High Performance Computing–HiPC 2007*, pages 197–208, 2007.

E. Hartmann. Parametric g n blending of curves and surfaces. *The Visual Computer*, 17(1):1–13, 2001.

M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

J. Hershberger. Finding the visibility graph of a simple polygon in time proportional to its size. In *Proceedings of the third annual symposium on Computational geometry*, pages 11–20. ACM, 1987.

J. Hershberger and S. Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 71–78. ACM, 1992.

H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 295–302. ACM, 1994.

K. Hormann. From scattered samples to smooth surfaces. *Proc. of Geometric Modeling and Computer Graphics*, 2003.

T. Hrycak and V. Rokhlin. An improved fast multipole algorithm for potential fields. *SIAM Journal on Scientific Computing*, 19(6):1804–1826, 1998.

K. Ikeda. Dijkstra's algorithm, June 2000. URL `http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml`.

R. Jacob, M. Marathe, and K. Nagel. A computational study of routing algorithms for realistic transportation networks. *Journal of Experimental Algorithmics (JEA)*, 4(6), 1999.

X. Jin, S. Liu, C.C.L. Wang, J. Feng, and H. Sun. Blob-based liquid morphing. *Computer Animation and Virtual Worlds*, 16(3-4):391–403, 2005.

S. Kapoor, SN Maheshwari, and J.S.B. Mitchell. An efficient algorithm for euclidean shortest paths among polygonal obstacles in the plane. *Discrete & Computational Geometry*, 18(4):377–383, 1997.

Y Kotliarov. rbb3select, March 2005. URL `http://www.mathworks.com/matlabcentral/fileexchange/7191-rbb3select`.

G.S. Kumar, P.K. Kalra, and S.G. Dhande. Curve and surface reconstruction from points: an approach based on self-organizing maps. *Applied Soft Computing*, 5(1): 55–66, 2004.

M.K. Langton. *Radial Basis Functions Applied to Integral Interpolation, Piecewise Surface Reconstruction and Animation Control*. PhD thesis, University of Canterbury, 2009.

J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research*, 18(11):1119, 1999.

N. Leal, E. Leal, and J.W. Branch. Simple method for constructing nurbs surfaces from unorganized points. *Proceedings of the 19th International Meshing Roundtable*, pages 161–175, 2010.

D.T. Lee. Proximity and reachability in the plane. Technical Report R-831, Dept. Electrical Engineering, University of Illinois, Urbana, IL, 1978.

D.T. Lee and F.P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

I.K. Lee. Curve reconstruction from unorganized points. *Computer Aided Geometric Design*, 17(2):161–177, 2000.

J.P. Lewis, F. Pighin, and K. Anjyo. Scattered data interpolation and approximation for computer graphics. In *ACM SIGGRAPH ASIA 2010 Courses*, page 2. ACM, 2010.

S.P. Lim and H. Haron. Surface reconstruction techniques: a review. *Artificial Intelligence Review*, pages 1–20, 2012.

D. Liu and J. Hoschek. Gc1 continuity conditions between adjacent rectangular and triangular bézier surface patches. *Computer-Aided Design*, 21(4):194–200, 1989.

L. Liu, C. Bajaj, J.O. Deasy, D.A. Low, and T. Ju. Surface reconstruction from non-parallel curve networks. In *Computer Graphics Forum*, volume 27, pages 155–163. Wiley Online Library, 2008.

S. Liu, X. Jin, C.C.L. Wang, and K. Hui. Ellipsoidal-blob approximation of 3D models and its applications. *Computers & Graphics*, 31(2):243–251, 2007.

Y. Liu and W. Wang. A revisit to least squares orthogonal distance fitting of parametric curves and surfaces. *Advances in Geometric Modeling and Processing*, pages 384–397, 2008.

Y. Liu, H. Pottmann, and W. Wang. Constrained 3d shape reconstruction using a combination of surface fitting and registration. *Computer-Aided Design*, 38(6):572–583, 2006.

S. Lodha and J. Warren. Bezier representation for quadric surface patches. *Computer-Aided Design*, 22(9):574–579, 1990.

W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Siggraph Computer Graphics*, 21(4):163–169, 1987.

T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

M. Luby and P. Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(1):551–567, 1989.

L. Ma and Q. Peng. Smoothing of free-form surfaces with bezier patches. *Computer aided geometric design*, 12(3):231–249, 1995.

W. Ma and J.P. Kruth. Nurbs curve and surface fitting for reverse engineering. *The International Journal of Advanced Manufacturing Technology*, 14(12):918–927, 1998.

Y. Ma, S. Soatto, and J. Košecká. *An invitation to 3-D vision: From images to geometric models*. Springer Verlag, 2004.

K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak. An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. In *Workshop on Algorithm Engineering and Experiments (ALENEX), New Orleans, LA*, 2007.

P. Martín, R. Torres, and A. Gavilanes. Cuda solutions for the sssp problem. *Computational Science–ICCS 2009*, pages 904–913, 2009.

F.C.M. Menandro, C.F.L. Neto, H.J.S.G. Meira, and R.P.R. Bastos. Compactly supported radial basis functions for function interpolation: comparative behaviour. *Mathematical Foundations of MEF and Meshless Methods (B)*, XXIX(47):4733–4752, 2010.

C.A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2(1):11–22, 1986.

M.J. Milroy, C. Bradley, G.W. Vickers, and D.J. Weir. G1 continuity of b-spline surface patches in reverse engineering. *Computer-Aided Design*, 27(6):471–478, 1995.

J.S.B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3(1):83–105, 1991.

J.S.B. Mitchell. Shortest paths among obstacles in the plane. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 308–317. ACM, 1993.

J.S.B. Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.*, 6:309–332, 1996.

J.S.B. Mitchell. Geometric shortest paths and network optimization. *Handbook of computational geometry*, 334:633–702, 2000.

B.S. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *ACM SIGGRAPH 2005 Courses*, page 78. ACM, 2005.

P. Mullen, F. De Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez. Signing the unsigned: Robust surface reconstruction from raw pointsets. In *Computer Graphics Forum*, volume 29, pages 1733–1741. Wiley Online Library, 2010.

S. Muraki. Volumetric shape description of range data using blobby model. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 227–235. ACM, 1991.

Y. Ohtake, A. Belyaev, and H.P. Seidel. 3D scattered data interpolation and approximation with multilevel compactly supported RBFs. *Graphical Models*, 67(3):150–165, 2005.

W.E. Ong, R.K. Beatson, and C.J. Price. Reconstruction with blobby shapes. *ANZIAM Journal*, 52(0):C596–C611, 2011.

J. O'Rourke. *Art gallery theorems and algorithms*. Oxford University Press Oxford, 1987.

M.H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 164–171. ACM, 1988.

L. Piegl. On nurbs: a survey. *computer Graphics and Applications, IEEE*, 11(1):55–71, 1991.

D. Potts, G. Steidl, and A. Nieslony. Fast convolution with radial kernels at nonequispaced knots. *Numerische Mathematik*, 98(2):329–351, 2004.

M. Powell. A fast algorithm for nonlinearly constrained optimization calculations. *Numerical analysis*, pages 144–157, 1978a.

M.J.D. Powell. The convergence of variable metric methods for nonlinearly constrained optimization calculations. *Nonlinear programming*, 3(0):27–63, 1978b.

M.J.D. Powell. Radial basis functions for multivariable interpolation: a review. In *Algorithms for approximation*, pages 143–167. Clarendon Press, 1987.

M.J. Pratt. Smooth parametric surface approximations to discrete data. *Computer Aided Geometric Design*, 2(1-3):165–171, 1985.

J. Pryor, K. Kinahan, and J. Chinneck. Example networks1: Dijkstra's algorithm for shortest route problems, 2007. URL `http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html`.

V.V. Savchenko, A.A. Pasko, O.G. Okunev, and T.L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. In *Computer Graphics Forum*, volume 14, pages 181–188. Wiley Online Library, 1995.

R. Schaback. Creating surfaces from scattered data using radial basis functions. *Mathematical methods for curves and surfaces*, pages 477–496, 1995.

L.L. Schumaker. Fitting surfaces to scattered data. *Approximation theory II*, pages 203–268, 1982.

A. Sharf, T. Lewiner, G. Shklarski, S. Toledo, and D. Cohen-Or. Interactive topology-aware surface reconstruction. *ACM Transactions on Graphics (TOG)*, 26(3):43–es, 2007.

M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15:193, 1986.

X. Shi, T. Wang, P. Wu, and F. Liu. Reconstruction of convergent g1 smooth b-spline surfaces. *Computer Aided Geometric Design*, 21(9):893–913, 2004.

B. Siciliano, L. Sciavicco, and L. Villani. *Robotics: modelling, planning and control*. Springer Verlag, 2009.

P.D. Simari and K. Singh. Extraction and remeshing of ellipsoidal representations from mesh data. In *Proceedings of Graphics Interface 2005*, page 168, 2005.

D.P. Singh and N. Khare. A study of different parallel implementations of single source shortest path algorithms. *International Journal of Computer Applications*, 54(10): 26–30, 2012.

S.S. Sinha and B.G. Schunck. A two-stage algorithm for discontinuity-preserving surface reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14 (1):36–55, 1992.

I. Söderkvist. Introductory overview of surface reconstruction methods. Technical Report 10, Luleå University of Technology, Department of Mathematics, 1999.

J.A. Storer and J.H. Reif. Shortest paths in the plane with polygonal obstacles. *Journal of the ACM (JACM)*, 41(5):982–1012, 1994.

S. Sudarshan and C. Pandu Rangan. A fast algorithm for computing sparse visibility graphs. *Algorithmica*, 5(1):201–214, 1990.

I. Tobor, P. Reuter, and C. Schlick. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. *Journal of WSCG 2004*, 12(3):467–474, 2004a.

I. Tobor, P. Reuter, and C. Schlick. Multi-scale reconstruction of implicit surfaces with attributes from large unorganized point sets. In *Shape Modeling Applications, 2004. Proceedings*, pages 19–30. IEEE, 2004b.

G. Turk and J.F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics (TOG)*, 21(4):855–873, 2002.

T. Varady, R.R. Martin, and J. Cox. Reverse engineering of geometric models–an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

J. Vida, R.R. Martin, and T. Varady. A survey of blending methods that use parametric surfaces. *Computer-Aided Design*, 26(5):341–365, 1994.

D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. *STACS 2007*, pages 23–36, 2007.

J.G. Wang and G.R. Liu. A point interpolation meshless method based on radial basis functions. *International Journal for Numerical Methods in Engineering*, 54(11):1623–1648, 2002.

Y. Wang, D. Wang, and AM Bruckstein. On variational curve smoothing and reconstruction. *Journal of Mathematical Imaging and Vision*, 37(3):183–203, 2010.

V. Weiss, L. Andor, G. Renner, and T. Várady. Advanced surface fitting techniques. *Computer Aided Geometric Design*, 19(1):19–42, 2002.

E. Welzl. Constructing the visibility graph for n-line segments in o (n2) time. *Information Processing Letters*, 20(4):167–171, 1985.

H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics*, 4(1):389–396, 1995.

H. Wendland. On the smoothness of positive definite and radial functions. *Journal of Computational and Applied Mathematics*, 101(1):177–188, 1999.

H. Wendland. *Scattered data approximation*, volume 17. Cambridge Univ Pr, 2005.

S.M. Wong, Y.C. Hon, and M.A. Golberg. Compactly supported radial basis functions for shallow water equations. *Applied mathematics and computation*, 127(1):79–101, 2002.

Z. Wu. Compactly supported positive definite radial functions. *Advances in Computational Mathematics*, 4(1):283–292, 1995.

N. Yarvin and V. Rokhlin. Generalized gaussian quadratures and singular value decompositions of integral operators. *SIAM Journal on Scientific Computing*, 20:699–718, 1998.

J. Ye, X. Bresson, T. Goldstein, and S. Osher. A fast variational method for surface reconstruction from sets of scattered points. *CAM Report*, pages 10–01, 2010.

F. You, Q. Hu, Q. Lu, and Y. Yao. Study on the algorithm of surface reconstruction of defective bone based on delaunay triangulation. In *7th Asian-Pacific Conference on Medical and Biological Engineering*, pages 250–254. Springer, 2008.

F.B. Zhan and C.E. Noon. Shortest path algorithms: an evaluation using real road networks. *Transportation Science*, 32(1):65–73, 1998.

H.K. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and nonparametric shape reconstruction from unorganized data using a variational level set method. *Computer Vision and Image Understanding*, 80(3):295–314, 2000.

K. Zhou, M. Gong, X. Huang, and B. Guo. Highly parallel surface reconstruction. Technical Report 53, Microsoft Research Asia, 2008.