Building Interpolating and Approximating Implicit Surfaces Using Moving Least Squares

Chen Shen

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-14
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-14.html

January 12, 2007

Building Interpolating and Approximating Implicit Surfaces Using Moving Least Squares

by

Chen Shen

M.S. (Chinese Academy of Science, China) 2000

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor James F. O'Brien, Chair
Professor Jonathan Shewchuk
Professor Carlo H. Squin
Professor Sara McMains

Fall 2006

The dissertation of Chen Shen is approved.

Chair	Date
	Date
	Date
	Date

University of California, Berkeley

Fall 2006

Building Interpolating and Approximating Implicit Surfaces Using Moving Least Squares

by

Chen Shen

Abstract

Building Interpolating and Approximating Implicit Surfaces Using Moving Least Squares

by

Chen Shen

Doctor of Philosophy in Computer Sciences

University of California, Berkeley

Professor James F. O'Brien, Chair

This dissertation addresses the problems of building interpolating or approximating implicit surfaces from a heterogeneous collection of geometric primitives like points, polygons, and spline/subdivision surface patches. The user can choose to generate a surface that exactly interpolates the geometric elements, or a surface that approximates the input by smoothing away features smaller than some user-specified size. The implicit functions are represented using a scattered data interpolation formulation known as moving least-squares with constraints at input points or integrated over the parametric domain of each polygon or surface patch. This dissertation also proposes an improved technique for enforcing normal constraints that overcomes undesirable oscillatory behavior produced by previous methods. Multiple points, polygons and surface patches can be blended together by a single implicit function whose isosurface is a manifold envelope that either interpolates or approximates the original input, even when self-intersections, holes, or other defects are present. With an iterative procedure for ensuring that the implicit surface tightly encloses the input elements, the resulting clean, manifold surface can then be used for generating volume meshes for finite elements, manufacturing rapid prototyping models, and other applications that require manifold surfaces.

Professor James F. O'Brien
Dissertation Committee Chair

To my wife, Kun,

for your love and support.

Contents

List of Figures

List of Tables

Acknowledgements

While any error in this manuscript is certainly my own, any success I have had with this dissertation is the result of the excellent support, advice and guidance I have received from countless sources. I thank them here individually and collectively. If I have overlooked anyone I apologize.

I owe my foremost gratitude to my advisor, James O'Brien for providing tremendous guidance and support over the last six years. He has always been a source of inspiration and motivation for me. He has excellent insight at suggesting research problems that are well-motivated and also challenging. My interactions with him have greatly shaped my perspectives about research. I am especially thankful for his support and encouragement during the initial stage of my PhD program. I am also grateful to Jonathan Shewchuk for giving me the major ideas in this dissertation and the other members of my committee, Carlo Sequin and Sara McMains, for their guidance over the years.

The best thing about my PhD at Berkeley was the opportunity to do research with bright and energetic students. I would like to thank Tolga, Bryan, Adam, Jordan, Hayley, Pushkar, Ravi, Xiaofeng and all other colleagues who made research at Berkeley an enjoyable experience. I would also like to thank the other members of the Berkeley graphics group, faculty and students, for their insightful suggestions and helpful comments. I am also grateful to my friends and colleagues at Pixar Animation Studios, for providing me the opportunity of learning the real production world.

I owe my greatest thanks to my parents, my sister and her family for their support and encouragement. They have always been there when needed and this could not have been accomplished without them.

I dedicate this dissertation to Kun, my wife and best friend. It has been a long journey that at times seemed without end, but you were always there to offer support and encouragement. I could not have done this without you and would not have wanted to try. You inspire me in ways that you will never know.

Chapter 1

Introduction

From fascinating cinematic special effects to navigating robots along collision-free paths in chaotic environments, from analyses of the stresses in aerospace structures during flight to rapid prototyping machines that behave like three-dimensional printers, underlying these and many other applications are geometric models of the objects under study. Geometric Modeling is the branch of Computer Science concerned with the efficient acquisition, representation, manipulation, reconstruction and analysis of 3-dimensional geometric models on a computer and it is an important foundation of many applications covering a wide collection of areas including computer graphics, CAD/CAM, scientific visualization, medical imaging, multimedia and entertainment.

The representation of complex surfaces has been one of the core fields of geometric modeling. There are many different ways to represent a surface. Using geometric primitives like points, polygons and spline/subdivision surface patches is common in computer graphics.

With the evolution of modern 3D digital photography and 3D scanning systems for acquisition of complex real-world objects, point primitives have received growing attention. Point sampled objects do not have to store or maintain globally consistent topological information. Therefore they are flexible to handle highly complex or dynamically changing shapes. However constructing a continuous representation from discrete point samples is an inevitable step before a point based object can be fully utilized for rendering or simulation.

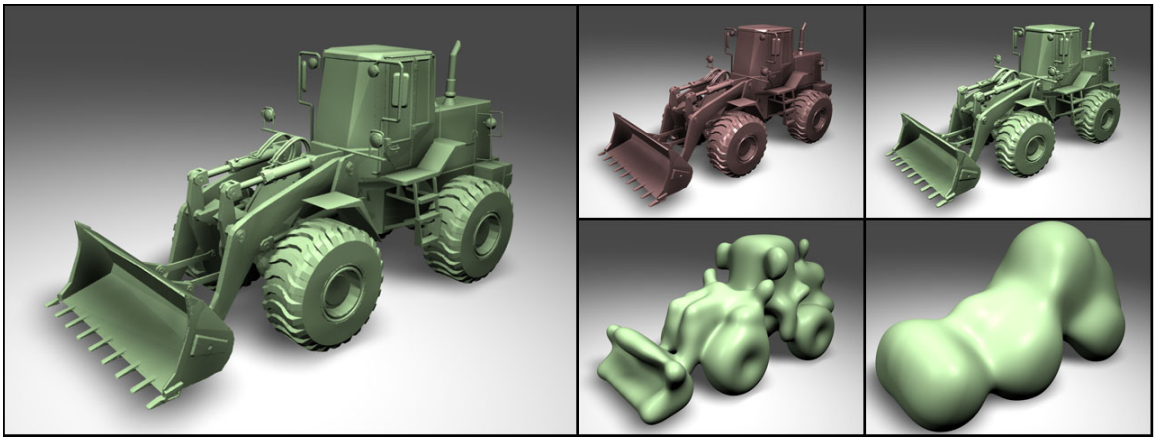

Figure 1.1. Interpolating and approximating surfaces (green) generated from polygonal original (brown).

Polygonal models occur ubiquitously in graphics applications. They are easy to render, easy to compute with, and a vast array of tools have been developed for creating and manipulating polygon data. Unfortunately, polygonal data sets often contain problems, such as holes, gaps, T-junctions, self-intersections and non-manifold structure, that make them unsuitable for many purposes other than rendering. Even when a polygonal data set does define a closed, manifold surface, other difficulties such as excessive detail or bad-aspect-ratio polygons, can preclude many uses. Data sets containing these problems are so common that the term "polygon soup" has evolved for describing arbitrary collections of polygons that carry no warranties concerning their structure.

Parametric surface like NURBS used in a CAD system or subdivision surfaces used in character animation provides a compact and smooth representation of complex geometries. Although higher order continuity of parametric surfaces gives designers a freedom to model naturally curved objects with a smaller number of primitives, it might not be possible to model a complicated object with a single parametric surface and stitching multiple parametric pieces together is a common practice. Similar to "polygon soup", a collection of parametric patches is fine for the purpose of rendering, but not suitable for applications that require watertight manifold surfaces.

This dissertation provides a tool that can transform arbitrary geometric data represented as a collection of elements like points, polygons or spline/subdivision surface patches

into a more useful form. We address this task with a method for generating implicit surfaces that can interpolate or approximate a set of geometric elements. The user controls how closely the surface approximates the input by selecting a minimum feature size. Geometric details or topological structures below this size tend to be smoothed away. Setting the minimum feature size to zero forces exact interpolation of the input. Additionally, if desired, we can cause an approximating surface to fit tightly around the input while still ensuring that the input are completely enclosed by the implicit surface. Figure 1.1 shows interpolating and approximating surfaces generated from a complex polygonal model with sharp edges and many small features.

An interpolating surface will exactly interpolate the input elements, but it will also extend to fill gaps and holes. The resulting surface will be "watertight" and generally manifold. This implicit function can then be used directly for a variety of applications, such as inside-outside tests, that are better suited to implicit representations. Alternatively, a clean polygonal model can be extracted and used for applications that require such clean polygonal input.

Approximating surfaces will naturally smooth out geometric features of the input data. Because we are using an implicit representation, topological structures of the input surfaces can also be smoothed away. This behavior makes the method suitable as part of a model simplification process when combined with an appropriate polygonization algorithm.

We can also force the approximating surface to stay "tight" around the original input while still smoothing away details and ensuring that all the original geometric elements fall inside the approximating surface. This capacity allows us to generate a family of increasingly smooth approximations that eventually converge to a circumscribing ellipsoid. Among other uses, these simplified shapes can be used for efficient simulation envelopes.

Our algorithm makes use of a scattered-data interpolation method known as *moving least-squares*, commonly abbreviated MLS. The function defining our implicit surfaces is specified by the moving least-squares solution to a set of constraints that would force the function to a given value over the surface region of each geometric element, and that would

over the same region also force the function's upward gradient to match the element's outward normal. For input point elements, both conditions are specified by simple point constraints. For polygonal and parametric inputs, integrated constraints are used over the input, and normals constraints directly effect the function's gradient. The degree of approximation is controlled by simply adjusting the least-squares weighting function, but the tightness of the surface and the requirement that the input elements fall inside the implicit surface both depend on a procedure for adjusting the constraint values at each element.

The moving least-squares method has been used by other graphics researchers to define a surface as the fixed-point of an iterative parametric fit procedure — For example see Alexa et al. (2001). Other than using the same mathematical tool, that approach and this one are unrelated. Unfortunately, those surfaces are often referred to simply as *MLS Surfaces* which may cause some confusion with the method described here. We suggest that the term *implicit moving least-squares surface*, or *IMLS Surface* be used to describe our method.

Our approach is, however, closely related to implicit methods based on partition-of-unity interpolants. (For example see Ohtake et al. (2003).) Partition-of-unity and moving least-squares interpolants use different notation, but they are fundamentally alike. One key difference between our formulation and prior ones is that our integrated constraints differ significantly from collections of point constraints. We also use improved normal and approximation procedures, which are applicable to point constraints as well as to our integrated constraints.

The primary components of our algorithms are

- A scattered data interpolation scheme that, in addition to simple point constraints, allows integrated constraints over polygons and parametric patches.
- A method for enforcing true normal constraints that does not produce undesirable oscillatory behavior.
- An adjustment procedure that causes the implicit surface to fit tightly around the

input models while still ensuring that the input is completely enclosed by the implicit surface.

- A hierarchical fast evaluation scheme that makes the method practical for large data sets.

Chapter 2

Background

Interpolating and approximating implicit surfaces draw upon two areas of modeling: implicit surfaces and scattered data interpolation. In this chapter I briefly review work in these two sub-areas. Interpolating and approximating implicit surfaces are not new to graphics, and in this chapter I also describe earlier published methods of creating interpolating and approximating implicit surfaces. Other than using implicit approaches, some of the applications that can be addressed with our method have also been addressed with other explicit approaches that are discussed at the close of this chapter.

2.1 Implicit Surfaces

There are two main techniques for representing surfaces in geometric modeling and computer graphics — parametric and implicit. Parametric representations typically define a surface as a set points $\boldsymbol{p}(s,t)$, i.e.

$$\boldsymbol{p}(s,t) = (\boldsymbol{x}(s,t)) \tag{2.1}$$

The bold letter $\boldsymbol{x}$ represents a point in the domain , and in this dissertation I will use bold letters to represent such points, both in 2D and 3D.

Implicit representations typically define a surface as the zero contour of a function

$F(\boldsymbol{p}) = 0$, i.e.

$$F(\boldsymbol{p}) = F(\boldsymbol{x}) = 0 \tag{2.2}$$

Implicit methods provide mathematical tractability and are becoming extremely useful for modeling operations such as blending, sweeping, metamorphosis, intersections, boolean operations, and even image rendering.

In this section, I will take a brief tour through some of the key concepts of three major lines of work in implicit methods: algebraic surfaces, blobby objects and functional representations.

2.1.1 Algebraic Surfaces

Algebraic methods describe surfaces as implicit polynomials, i.e. $F(\boldsymbol{x})$ is a polynomial in $\boldsymbol{x}$. These are typically low degree $(2, 3, 4)$ polynomials, and the most popular ones are quadratic implicit (degree 2) surfaces, often referred to as "quadrics". If a surface is simple enough, it may be described by a single polynomial expression. Much of work has been devoted to this approach. Taubin (1993) and Keren and Gotsman (1998) are good starting points in this area. Fitting an algebraic surface to a given collection of points has attracted a great deal of attention. Unfortunately it is not always possible to interpolate all of the data points, so error minimizing techniques are used. Surfaces may also be constructed by piecing together many separate algebraic surface patches, and the chapters by Bajaj and Rockwood in Bloomenthal (1997) are a good introduction to these surfaces. It is easier to create complex surfaces using a collection of algebraic patches rather than using a single algebraic surface. But the tradeoff is that a good deal of extra effort is required to create smooth joins across patch boundaries.

2.1.2 Blobby Methods

The idea of using blobby objects originates from physical ball-and-stick models for molecules. From physics, we know that electron clouds around each atom are not spherical, but rather are distorted by the electron clouds around other atoms. To generate iso-surfaces

(those with identical electron densities), one would therefore have to consider the effects from all neighboring atoms.

The computation of exact iso-surfaces is expensive, and several good approximations have been made. Blinn (1982) used exponentially decaying (with distance) fields created by each atom, and defined the iso-surface as those points where a "density" function D equals some threshold amount T, i.e.

$$F(\boldsymbol{x}) = D(\boldsymbol{x}) - T \tag{2.3}$$

where the function D took the form

$$D(\boldsymbol{x}) = \sum_i b_i \exp^{-a_i r_i^2} \tag{2.4}$$

The value T is the iso-surface threshold, and it specifies one particular surface from a family of nested surfaces that are defined by the sum of Gaussians. When the centers of two blobby spheres are close enough to one another, the implicit surface appears as though the two spheres have melted together. The exponential term is a simple Gaussian bump centered at r_i, with height b_i and standard deviation a_i. Different effects of "blobbiness" can be achieved for the same arrangement of atoms by adjusting the a_i and b_i parameters. Similar methods were also developed independently by Nishimura et. al. for use in the LINKS project [Nishimura et al. (1983)]. Evaluating an exponential function is computationally expensive, so some authors have used piecewise polynomial expressions instead of exponentials to define these blobby sphere functions like Wyvill et. al. used in Wyvill et al. (1986) to create "soft objects" by distributing field sources in space and computing a field value at each point of space. A greater variety of shapes can be created with the blobby approach by using ellipsoidal rather than spherical functions and there are others ways to achieve different blending.

2.1.3 Functional Methods

The function representation (or F-rep) defines a whole geometric object by a single real continuous function of several variables as $F(\boldsymbol{x}) \geq 0$. F-rep is an attempt to step to a

more general modeling scheme using real functions. Functions are not restricted very much - they only have to be at least C^0 continuous. The function can be defined by a formula or by an evaluation procedure. In this sense, F-rep combines many different models like classic implicits, skeleton based implicits, set-theoretic solids, sweeps, volumetric objects, parametric and procedural models (see survey in Pasko et al. (1995)).

2.2 Scattered Data Interpolation and Approximation

Scattered data interpolation and approximation is a recent, fast growing research area. It deals with the problem of reconstructing an unknown function from given scattered data. Naturally, it has many applications, such as terrain modeling, surface reconstruction, fluid-structure interaction, the numerical solution of partial differential equations, kernel learning, and parameter estimation, to name a few. Moreover, these applications come from such different fields as applied mathematics, computer science, geology, biology, engineering, and even business studies.

In practical applications over a wide field of study one often faces the problem of reconstructing an unknown function f from a finite set of discrete data. These data consist of data sites $\boldsymbol{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ and data values $f_i = f(\boldsymbol{x}_i), 1 \leq i \leq N$, and the reconstruction has to approximate the data values at the data sites. In other words, a function s is sought that either *interpolates* the data, i.e. that satisfies $s(\boldsymbol{x}_i) = f_i, 1 \leq i \leq N$, or at least *approximates* the data, $s(\boldsymbol{x}_i) \approx f_i$. The latter case is in particular important if the data contain noise.

2.2.1 Shepard's Method

One of the earliest algorithms in this field was based on inverse distance weighting of data and known as Shepard's method [Shepard (1968)]. Shepard defined a C^0-continuous interpolation function as the weighted average of the data, with the weights being inversely proportional to distance. This technique suffers from several shortcomings, including cusps, corners, and flat spots at the data points, as well as undue influence of points which are

far away. Furthermore, it is a global method requiring all the weights to be recomputed if any data point is added, removed, or modified. Franke and Nielson introduced the modified quadratic Shepard's method [Franke and Nielson (1980)] to address these deficiencies and produce C^1-continuous interpolation.

2.2.2 Radial Basis Functions

Another popular approach to scattered data interpolation is to define the interpolation function as a linear combination of radially symmetric basis functions, each centered at a data point. The unknown coefficients for the basis functions are determined by solving a linear system of equations. The coefficient matrix might be full, and, for large data sets, it may become poorly conditioned and require preconditioning [Dyn et al. (1986)]. Popular choices for the basis functions include Gaussian, multiquadratics [Hardy (1971)], and shifted log [Nielson (1993)]. Hardy's multiquadratics are among the most successful and applied methods due to its simplicity, well-conditioned property, and intuitive results.

2.2.3 Thin Plate Splines

Thin plate splines are derived by minimizing the integral of the curvatures square over the domain among the interpolation functions of the scattered points. They are widely used due to their visually pleasing results and stability for large data sets. Although they are usually formulated as the solution to a variational problem, Duchon has shown thin plate splines to be derived from radial basis functions [Duchon (1975)]. Thin-plate interpolation is often used in the computer vision domain, where there are often sparse surface constraints [Grimson (1983) and Terzopoulos (1988)]. Recently, thin plate splines have been used to generate smooth warp functions for image warping and morphing [Litwinowicz and Williams (1994)].

2.2.4 Finite Element Methods

Finite element methods involve creating some type of optimal triangulation on the set of data points to delimit local neighborhoods over which surface patches are defined. These patches are constrained to interpolate the original data. There are several criteria suggested by Lawson (1977) to derive optimal triangulations in which long thin triangles with small angles are avoided. Piecewise linear approximation over the triangulation is not smooth, achieving only C^0 -continuity. The most common C^1 method uses the Clough-Tocher triangular interpolant [Clough and Tocher (1965)]. Triangulation methods, however, are sensitive to data distribution, i.e., long thin triangles cannot always be avoided. Shewchuk (1997) introduces a technique for generating unstructured meshes of triangles or tetrahedra suitable for use in the finite element method or other numerical methods for solving partial differential equations.

2.2.5 Meshless Methods

Meshless methods originate from computational mechanics. Extremely large deformations make the conventional computational methods such as finite element, finite volume or finite difference methods no longer suitable for simulation purpose since the underlying structure of these methods which originates from their reliance on a mesh is not well suited to the treatment of discontinuities which do not coincide with the original mesh lines. The objective of meshless methods is to eliminate at least part of this structure by constructing the approximation entirely in terms of nodes without connectivities. The earliest work is the smooth particle hydrodynamics (SPH) method [Lucy (1977)], who used it for modeling astrophysical phenomena without boundaries such as exploding starts and dust clouds. A parallel path to constructing meshless approximations is the use of moving least squares (MLS) approximations. Nayroles and Villon (1992) first used MLS in a Galerkin method known as the diffuse element method (DEM). Belytschko and Gu (1994) refined and modified the method to EFG, element-free Galerkin. Duarte and J.T.Oden (1996) recognized

that partitions of unity are specific instances of MLS methods. Belytschko et al. (1996) provides a complete discussion of the relationships between these different formulations.

Approximation by moving least squares has its origin in the early paper [Lancaster and Salkauskas (1981)] by Lancaster and Salkauskas from 1981 with special cases going back to McLain (1974) and McLain (1976) to Shepard (1968). It is interesting to see that Farwig remarked in Farwig (1986) that if the least squares problem varies in x, computing the global approximation is generally very time-consuming. With the development of fast computers and efficient data structures for the data sites, this statement no longer holds, and the moving least squares approximation has in recent times attracted attention again.

2.3 Related Work on Implicit Surfaces

The work most closely related to ours appears in Ohtake et al. (2003). They use a partition-of-unity method to build a function whose zero-set passes through, or near, a set of input points. Using a formulation originally proposed by Turk and O'Brien (1999), they place zero constraints at each input point, and they also place a pair of additional non-zero point constraints offset in the inward and outward normal directions. To keep the method feasible for large data sets, they use a fast hierarchical evaluation scheme. The partition-of-unity formulation they use and the moving least-squares formulation that we start with are essentially identical: they both belong to a family of meshless interpolation methods that also includes the element-free Galerkin method and smoothed particle hydrodynamics as what we have discussed in the previous section. The two most significant difference between our work and Ohtake et al. (2003) are that we use integrated polygon constraints, and that we use a significantly improved method for enforcing normal constraints. We also describe a different hierarchical evaluation scheme and an iterative method for generating useful approximating surfaces.

The moving least-squares interpolation was also used to develop the non-linear projection method used in Alexa et al. (2001), Alexa et al. (2003), and Fleishman et al. (2003). This projection method defines a surface based on a set of points, but the moving least-

squares fit is used as part of a non-linear projection that differs substantially from the implicit-surface based method described here.

The technique of defining a surface implicitly using a function constrained to match a set of input points is fairly widespread. In Savchenko et al. (1995), Turk and O'Brien (1999), Carr et al. (2001), and Turk and O'Brien (2002) the function is represented using globally supported radial splines. This class of functions has the nice property that one can make definite statements about a solution's global behavior. These radial splines have also been used to match polygon data by Yngve and Turk (2002). While they were able to achieve results that roughly matched the input polygons, the resulting implicit surfaces still deviated substantially from the input. Different, locally supported functions were used in both Muraki (1991) and Morse et al. (2001) for fitting an implicit surface to clouds of point data. In addition to representing function as sums of continuous basis functions, Museth et al. (2002) and Zhao and Osher (2002) have used level-set methods for fitting surfaces to point clouds. Other function representations include signed-distance functions, Cohen-Or et al. (1998), and medial axes Bittar et al. (1995).

2.4 Explicit Mesh Processing

Some of the applications that can be addressed with our method have also been addressed with other methods. An enormous amount of work has been done on smoothing explicit representations of polygonal models, two early examples of which include Taubin (1995) and Desbrun et al. (1999). Work in that sub-area is now quite advanced and methods are available that can preserve sharp features while still smoothing away noise. (For a single recent example, see Jones et al. (2003).) We can also generate envelopes around input objects and similar ideas have been explored in Cohen et al. (1996) and Keren and Gotsman (1998). The problem of rectifying polygonal models has been investigated in Nooruddin and Turk (2003). In Nooruddin and Turk (2000) the same researchers also looked at methods for removing unwanted interior structure from a polygon model.

Chapter 3

Moving Least-Squares Basics

The primary tool used in this research is a scattered data interpolation method known as moving least-squares. With this method we can create an implicit surface that either interpolates or approximates a given geometry model. This chapter defines the basic mathematical descriptions of moving least-squares, compares it with the standard stationary least-squares and indicates other applications.

3.1 Standard Least-Squares

Least-Squares (or standard stationary least-squares) is a mathematical optimization technique which, when given a series of measured data, attempts to find a function which closely approximates the data (a "best fit"). It attempts to minimize the sum of the squares of the ordinate differences (called residuals) between values generated by the function and corresponding measures in the data.

Assume that we have N points located in 1D space at positions x_i, $i \in [1 \dots N]$, and we would like to build a function f that approximates the values y_i at those points ($f(x_i) \approx y_i$).

To attain this goal, we suppose that the function f is of a particular form containing some parameters which need to be determined. For instance, suppose that it is linear, meaning that $f(x) = c_0 + c_1\, x$, c_0 and c_1 are not yet known. We now seek the values of c_0

and c_1 that minimize the sum of the squares of the residuals (R^2):

$$R^2 = \sum_{i=1}^{N} [y_i - f(x_i)]^2 \tag{3.1}$$

This explains the name least-squares. For a linear fit, $f(c_0, c_1) = c_0 + c_1\, x$, we have

$$R^2(c_0, c_1) = \sum_{i=1}^{N} [y_i - (c_0 + c_1\, x_i)]^2 \tag{3.2}$$

To minimize the residuals,

$$\frac{\partial(R^2)}{\partial c_0} = -2 \sum_{i=1}^{N} [y_i - (c_0 + c_1\, x_i)] = 0 \tag{3.3}$$

and

$$\frac{\partial(R^2)}{\partial c_1} = -2 \sum_{i=1}^{N} [y_i - (c_0 + c_1\, x_i)]\, x_i = 0 \tag{3.4}$$

These lead to the equations

$$N\, c_0 + c_1 \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i \tag{3.5}$$

$$c_0 \sum_{i=1}^{N} x_i + c_1 \sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} x_i\, y_i \tag{3.6}$$

In matrix form,

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i\, y_i \end{bmatrix} \tag{3.7}$$

so,

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i\, y_i \end{bmatrix} \tag{3.8}$$

$$= \frac{1}{N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2} \begin{bmatrix} \sum_{i=1}^{N} y_i \sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i\, y_i \\ N \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i \end{bmatrix} \tag{3.9}$$

Now we have the fit function as

$$f(x) = \frac{\sum_{i=1}^{N} y_i \sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i y_i}{N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2} + \frac{N \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{N \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2}\, x \tag{3.10}$$

Figure 3.1 shows the plot of function $f(x)$ based on Equation (3.10) to a set of sample points at x_i with values y_i

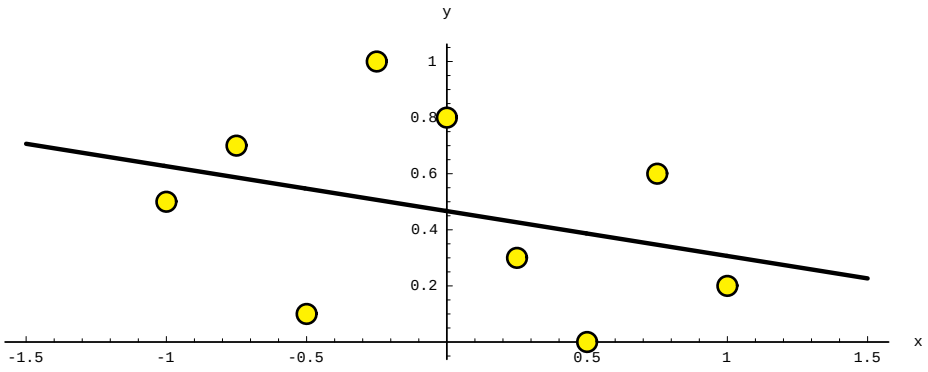

Figure 3.1. Linear least-squares fit to a set of sample points at x_i with values y_i

3.2 Moving Least-Squares

In the standard least-squares, the fit is based on the minimization of a sum of the squares of the residuals and each individual residual contributes equally to the over all minimization so that the solution is constant over the entire space and can only approximate the samples by balancing between all of them.

Different from this kind of global fit, moving least-squares allow the fit to change locally depending on where we evaluate the function so that the solution varies with x. To make the least-squares function move, the key idea is to have sample points contribute differently to the fit. We do so by weighting each individual residual of Equation (3.1) by $w(x - x_i)$, where $w(r)$ is some distance weighting function, which gives us

$$R^2 = \sum_{i=1}^{N} w(x - x_i) \, [y_i - f(x_i)]^2 \tag{3.11}$$

For a linear fit, $f(c_0, c_1) = c_0 + c_1 \, x$, we have

$$R^2(c_0(x), c_1(x)) = \sum_{i=1}^{N} w(x - x_i) \, [y_i - (c_0 + c_1 \, x_i)]^2 \tag{3.12}$$

It leads to the equations

$$c_0 \sum_{i=1}^{N} w(x - x_i) + c_1 \sum_{i=1}^{N} w(x - x_i) \, x_i = \sum_{i=1}^{N} w(x - x_i) \, y_i \tag{3.13}$$

$$c_0 \sum_{i=1}^{N} w(x - x_i) \, x_i + c_1 \sum_{i=1}^{N} w(x - x_i) \, x_i^2 = \sum_{i=1}^{N} w(x - x_i) \, x_i \, y_i \tag{3.14}$$

In matrix form,

$$\begin{bmatrix} \sum_{i=1}^{N} w(x-x_i) & \sum_{i=1}^{N} w(x-x_i)\, x_i \\ \sum_{i=1}^{N} w(x-x_i)\, x_i & \sum_{i=1}^{N} w(x-x_i)\, x_i^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} w(x-x_i)\, y_i \\ \sum_{i=1}^{N} w(x-x_i)\, x_i\, y_i \end{bmatrix} \tag{3.15}$$

so,

$$\begin{aligned} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} &= \begin{bmatrix} \sum_{i=1}^{N} w(x-x_i) & \sum_{i=1}^{N} w(x-x_i)\, x_i \\ \sum_{i=1}^{N} w(x-x_i)\, x_i & \sum_{i=1}^{N} w(x-x_i)\, x_i^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{N} w(x-x_i)\, y_i \\ \sum_{i=1}^{N} w(x-x_i)\, x_i\, y_i \end{bmatrix} \\ &= \frac{1}{\sum_{i=1}^{N} w(x-x_i) \sum_{i=1}^{N} w(x-x_i)\, x_i^2 - (\sum_{i=1}^{N} w(x-x_i)\, x_i)^2} \cdot \\ &\quad \begin{bmatrix} \sum_{i=1}^{N} w(x-x_i)\, y_i \sum_{i=1}^{N} w(x-x_i)\, x_i^2 - \sum_{i=1}^{N} w(x-x_i)\, x_i \sum_{i=1}^{N} w(x-x_i)\, x_i\, y_i \\ \sum_{i=1}^{N} w(x-x_i) \sum_{i=1}^{N} w(x-x_i)\, x_i y_i - \sum_{i=1}^{N} w(x-x_i)\, x_i \sum_{i=1}^{N} w(x-x_i)\, y_i \end{bmatrix} \end{aligned} \tag{3.16}$$

Now we have the fit function as

$$\begin{aligned} f(x) &= \frac{\sum_{i=1}^{N} w(x-x_i)\, y_i \sum_{i=1}^{N} w(x-x_i)\, x_i^2 - \sum_{i=1}^{N} w(x-x_i)\, x_i \sum_{i=1}^{N} w(x-x_i)\, x_i y_i}{\sum_{i=1}^{N} w(x-x_i) \sum_{i=1}^{N} w(x-x_i)\, x_i^2 - (\sum_{i=1}^{N} w(x-x_i)\, x_i)^2} \\ &\quad + \\ &\quad \frac{\sum_{i=1}^{N} w(x-x_i) \sum_{i=1}^{N} w(x-x_i)\, x_i y_i - \sum_{i=1}^{N} w(x-x_i)\, x_i \sum_{i=1}^{N} w(x-x_i)\, y_i}{N \sum_{i=1}^{N} w(x-x_i)\, x_i^2 - (\sum_{i=1}^{N} w(x-x_i)\, x_i)^2}\, x \end{aligned} \tag{3.17}$$

3.3 Weight Functions

By selecting an appropriate weight function, a variety of interpolating or approximating behaviors can be achieved, even with low order basis functions. In general, a weight function that approaches $+\infty$ at zero will cause interpolation. Approximation can be achieved by using a weight function with a finite function value at zero. We use the weight function

$$w(r) = \frac{1}{(r^2 + \epsilon^2)} \quad . \tag{3.18}$$

which can provide both interpolating and approximating behavior by adjusting the parameter ϵ. When ϵ is set to zero, the moving least-squares function will exactly interpolate sample points (constraint values). When ϵ is set to a non-zero value the weighting function is no longer singular at zero, and the moving least-squares function interpolates constraint values only approximately.

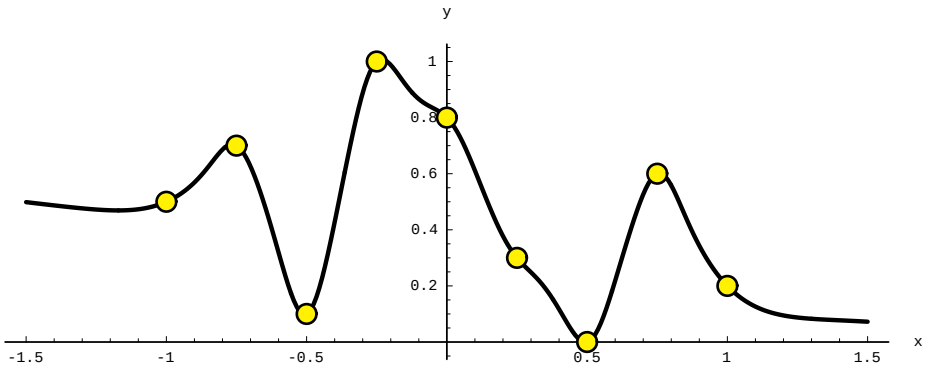

Figure 3.2. Moving least-squares fit with a linear basis function and an interpolating weight function to a set of sample points at x_i with values y_i

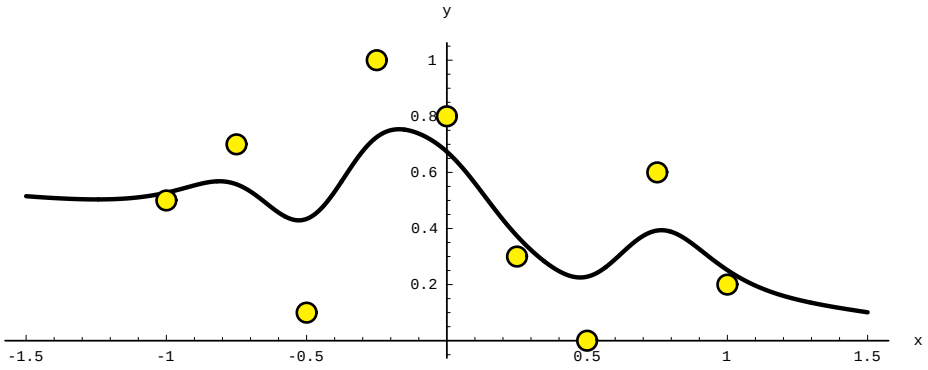

Figure 3.3. Moving least-squares fit with a linear basis function and an approximating weight function to a set of sample points at x_i with values y_i

Figure 3.2 shows the plot of the function $f(x)$ based on Equation (3.17) and Equation (3.18) with $\epsilon = 0$ to the same set of sample points used in the previous discussion of standard least-squares.

Figure 3.3 shows the plot of the function $f(x)$ based on the same Equation (3.17) and Equation (3.18) but with $\epsilon = 0.1$.

By adjusting ϵ, we can achieve a variety of approximating behaviors. In general, a larger ϵ tends to provide a smoother approximation while a smaller ϵ gives us a closer approximation to the original samples.

Other inverse distance functions like

$$w(r) = \frac{1}{(r^2 + \epsilon^2)^d} \quad . \tag{3.19}$$

where d is a positive integer can provide both interpolation and approximation. A larger d gives us more interpolating behaviors.

Gaussian function,

$$w(r) = a\, e^{-r^2/b^2} \tag{3.20}$$

for some real constants $a > 0$ and b, is another good candidate of weight functions for achieving approximating behaviors. Unfortunately, Gaussian weight function cannot provide interpolation since it does not approach $+\infty$ at zero. But in practice, we can use a good combination of a and b to achieve a nearly interpolating behavior.

3.4 Basis Functions

So far, we suppose that the function f we are fitting is of a particular form (a linear polynomial). An immediate generalization of using the linear function f is to fit the data points to a model that is not just a linear combination of 1 and x (namely $c_0 + c_1\, x$), but rather a linear combination of any M specified functions of x. For example, the functions could be $1, x, x^2, \ldots, x^{(M-1)}$, in which case their general linear combination,

$$f(x) = c_0 + c_1\, x + c_2\, x^2 + \cdots + c_{M-1}\, x^{M-1} \tag{3.21}$$

is a polynomial of degree $M - 1$. The general form of this kind of model is

$$f(x) = \sum_{i=0}^{M-1} c_i\, b_i(x) \tag{3.22}$$

where $b_0(x), b_1(x), \ldots, b_{M-1}(x)$ are arbitrary fixed functions of x, called the basis functions. In the previous discussion, we used $1, x$ as the basis functions.

Even simpler, we can use just 1 to form a constant basis function. Figure 3.4 shows the plot of using a constant basis function with the moving least-squares formulation and an interpolating weight function.

The difference between Figure 3.2 and Figure 3.4 tells us that moving least-squares fit the sample points locally. Near each sample point, the curve in Figure 3.2 behaves like a

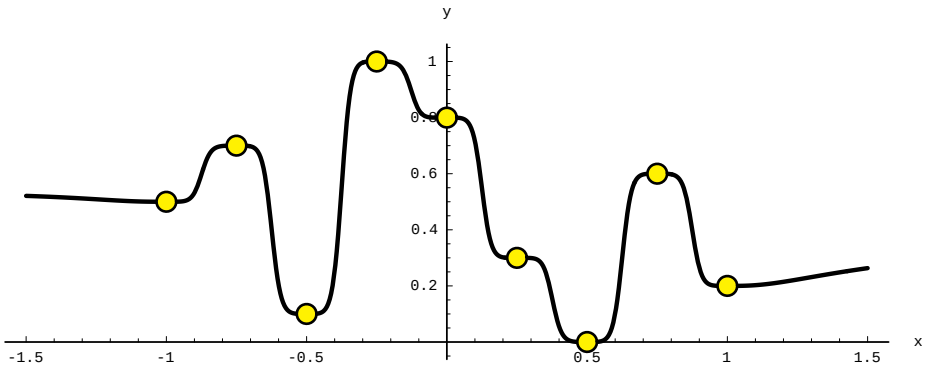

Figure 3.4. Moving least-squares fit with a constant basis function and an interpolating weight function to a set of sample points at x_i with values y_i

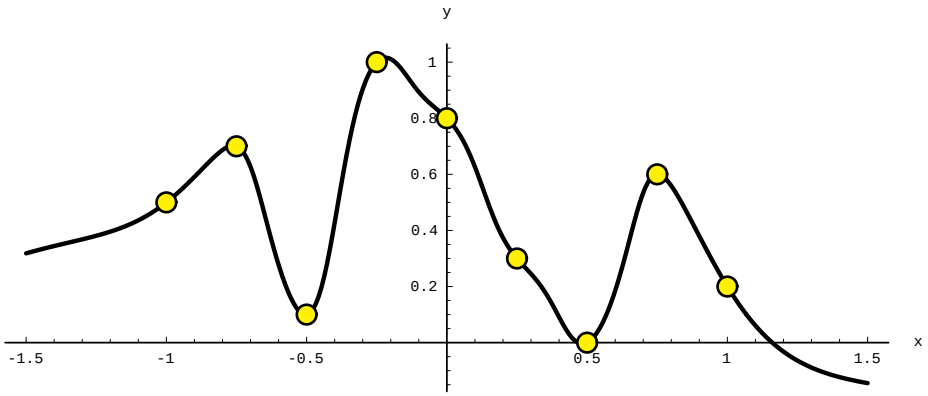

Figure 3.5. Moving least-squares fit with a quadratic basis function and an interpolating weight function to a set of sample points at x_i with values y_i

straight line with some slope (which is determined by the unknown coefficient c_1) while the curve in Figure 3.4 goes horizontally pass through the samples.

Figure 3.5 shows the plot of using a higher order basis function (a quadratic form $1, x, x^2$) with an interpolating weight function. The curve passes through the sample points in a smoother way.

Choosing higher order basis functions is not always a better idea since the computation of using a higher order basis function is much more expensive. The moving least-squares formulation with a constant basis function has only one unknown coefficient so that it requires inversion of a single scalar value

$$\frac{1}{\sum_{i=1}^{N} w(x - x_i)} \tag{3.23}$$

that is cheap to compute. A linear case from our previous discussion and Equation (3.17) has two unknown coefficients and requires inversion of a $2x2$ matrix

$$\begin{bmatrix} \sum_{i=1}^{N} w(x-x_i) & \sum_{i=1}^{N} w(x-x_i)\, x_i \\ \sum_{i=1}^{N} w(x-x_i)\, x_i & \sum_{i=1}^{N} w(x-x_i)\, x_i^2 \end{bmatrix}^{-1} \tag{3.24}$$

which is slightly more expensive. A quadratic version shown in Figure 3.5 needs to invert a 3 by 3 matrix

$$\begin{bmatrix} \sum_{i=1}^{N} w(x-x_i) & \sum_{i=1}^{N} w(x-x_i)\, x_i & \sum_{i=1}^{N} w(x-x_i)\, x_i^2 \\ \sum_{i=1}^{N} w(x-x_i)\, x_i & \sum_{i=1}^{N} w(x-x_i)\, x_i^2 & \sum_{i=1}^{N} w(x-x_i)\, x_i^3 \\ \sum_{i=1}^{N} w(x-x_i)\, x_i^2 & \sum_{i=1}^{N} w(x-x_i)\, x_i^3 & \sum_{i=1}^{N} w(x-x_i)\, x_i^4 \end{bmatrix}^{-1} \tag{3.25}$$

In general, the moving least-squares formulation with a polynomial basis function of degree $M-1$ in one-dimension (or say order M) requires inversion of a $(M+1) \times (M+1)$ matrix and has M unknown coefficients to solve.

3.5 General Matrix Form

So far, we have discussed the moving least-squares formulation with a particular weight function and a few low order basis functions for a simple 1D case. Now we extend it to a general matrix form.

Assume that we have N points at positions $\boldsymbol{p}_i$, $i \in [1 \dots N]$, and we would like to build a function f that approximates the values ϕ_i at those points ($f(\boldsymbol{p}_i) \approx \phi_i$).

For a standard least-squares fit we would solve

$$\begin{bmatrix} \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_1) \\ \vdots \\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_N) \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix} \quad , \tag{3.26}$$

where $\boldsymbol{b}(\boldsymbol{x})$ is the vector of basis functions we use for the fit, and $\boldsymbol{c}$ is the unknown vector of coefficients. Unless this system is under-constrained, it can be resolved efficiently using the method of normal equations and solving an $M \times M$ matrix, where M is the number of

basis functions (*i.e.*, the lengths of $\boldsymbol{b}$ and $\boldsymbol{c}$). For example, if we wished to fit a plane in 3D space we would choose $\boldsymbol{b}(\boldsymbol{x}) = [1, x, y, z]$, or simply $\boldsymbol{b}(\boldsymbol{x}) = [1]$ if we just wished to fit a constant. The resulting function would be evaluated with

$$f(\boldsymbol{x}) = \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{c} \quad . \tag{3.27}$$

For the moving least-squares formulation, we allow the fit to change depending on where we evaluate the function so that $\boldsymbol{c}$ varies with $\boldsymbol{x}$. We do so by weighting each row of Equation (3.26) by $w(\|\boldsymbol{x} - \boldsymbol{p}_i\|)$, where $w(r)$ is some distance weighting function, which gives us

$$\begin{bmatrix} w(\boldsymbol{x}, \boldsymbol{p}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{x}, \boldsymbol{p}_N) \end{bmatrix} \begin{bmatrix} \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_1) \\ \vdots \\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_N) \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} w(\boldsymbol{x}, \boldsymbol{p}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{x}, \boldsymbol{p}_N) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix} \tag{3.28}$$

Giving matrices names and explicitly noting their dependence on $\boldsymbol{x}$, Equation (3.28) becomes

$$\boldsymbol{W}(\boldsymbol{x})\, \boldsymbol{B}\, \boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{W}(\boldsymbol{x})\, \boldsymbol{\phi} \quad . \tag{3.29}$$

The resulting normal equations are

$$\boldsymbol{B}^{\mathsf{T}}\ (\boldsymbol{W}(\boldsymbol{x}))^2\ \boldsymbol{B}\, \boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{B}^{\mathsf{T}}\ (\boldsymbol{W}(\boldsymbol{x}))^2\ \boldsymbol{\phi} \tag{3.30}$$

and we can evaluate the fit function's value using

$$f(\boldsymbol{x}) = \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{H}^{-1}\, \boldsymbol{B}^{\mathsf{T}}\ (\boldsymbol{W}(\boldsymbol{x}))^2\ \boldsymbol{\phi} \quad , \tag{3.31}$$

where

$$\boldsymbol{H} = \boldsymbol{B}^{\mathsf{T}}\ (\boldsymbol{W}(\boldsymbol{x}))^2\ \boldsymbol{B} \quad . \tag{3.32}$$

3.6 Derivatives

The spatial derivatives with respect to $\boldsymbol{x}$ of the fit function can be evaluated using

$$\begin{aligned} f'(\boldsymbol{x}) = (\boldsymbol{b}^{\mathsf{T}})'(\boldsymbol{x})\, \boldsymbol{H}^{-1}\, \boldsymbol{B}^{\mathsf{T}}\ (\boldsymbol{W}(\boldsymbol{x}))^2\ \boldsymbol{\phi}\ - \\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{H}^{-1} \boldsymbol{H}' \boldsymbol{H}^{-1}\, \boldsymbol{B}^{\mathsf{T}}\ (\boldsymbol{W}(\boldsymbol{x}))^2\ \boldsymbol{\phi}\ + \\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{H}^{-1}\, \boldsymbol{B}^{\mathsf{T}}\ ((\boldsymbol{W}(\boldsymbol{x}))^2)'\, \boldsymbol{\phi} \end{aligned} \tag{3.33}$$

where

$$\boldsymbol{H}' = \boldsymbol{B}^{\mathsf{T}} \left((\boldsymbol{W}(\boldsymbol{x}))^2 \right)' \boldsymbol{B} \quad , \tag{3.34}$$

and the derivative of $(\boldsymbol{W}(\boldsymbol{x}))^2$ is obtained by simply taking the derivative of the squared weighting function along the matrix's diagonal.

Notice that the second term in Equation (3.33) is based on the following relation:

$$\left(\boldsymbol{H}^{-1}\right)' = -\boldsymbol{H}^{-1} \cdot \boldsymbol{H}' \cdot \boldsymbol{H}^{-1} \tag{3.35}$$

The following equations explain why this relation is true,

$$\boldsymbol{H} \cdot \boldsymbol{H}^{-1} = I \tag{3.36}$$

$$\left(\boldsymbol{H} \cdot \boldsymbol{H}^{-1}\right)' = I' = 0 \tag{3.37}$$

$$\boldsymbol{H}' \cdot \boldsymbol{H}^{-1} + \boldsymbol{H} \cdot (\boldsymbol{H}^{-1})' = 0 \tag{3.38}$$

$$\left(\boldsymbol{H}^{-1}\right)' = -\boldsymbol{H}^{-1} \cdot \boldsymbol{H}' \cdot \boldsymbol{H}^{-1} \tag{3.39}$$

Chapter 4

Implicit Moving Least-Squares

With the mathematical tools of moving least-squares for solving the scattered data interpolation problem in hand, we now turn our attention to creating implicit surfaces.

Our main goal is to build a continuous representation of a given geometry model with discrete samples. Let's first examine a 2D example, then we will extend our discussion into a real 3D problem.

Assume the yellow dots in Figure 4.1 on the left are samples from a closed curve and we want to reconstruct the curve just from those sample points to some shape like the one shown on the right.

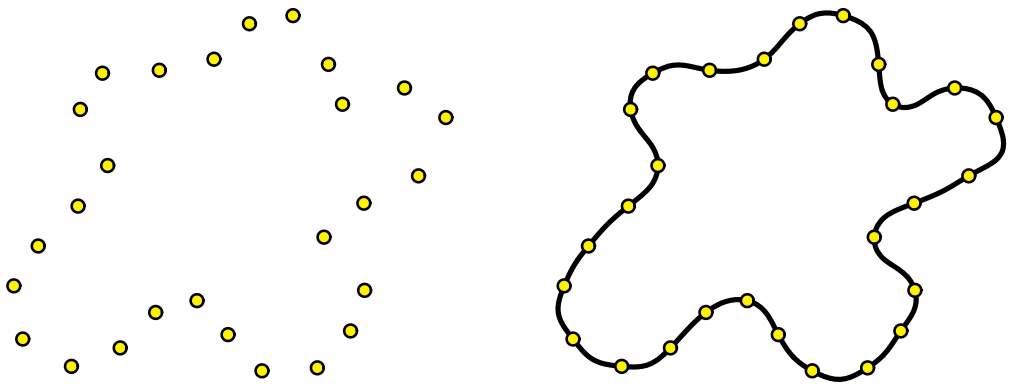

Figure 4.1. 2D illustration of the basic problem we want to solve: reconstruct the geometry model from a set of samples

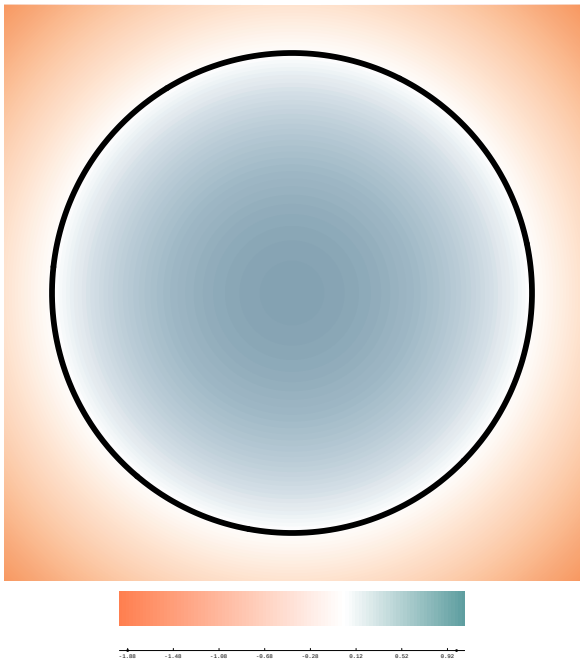

Figure 4.2. Contour plot of the implicit function defined by a unit circle

There are several different ways to achieve this goal. For example, we can construct a linear approximation of the curve by connecting adjacent sample points with line segments. Finding the adjacency relationship is not a trivial problem. A robust algorithm needs to handle all kinds of cases. This is known as an explicit scheme. Now let's look at the problem from a different perspective using an implicit method.

4.1 2D Implicit Curves

An implicit curve in 2D is defined by an implicit function, a continuous scalar-valued function over the domain $\mathbf{R}^2$. The implicit curve of such a function is the locus of points at which the function takes on the value zero. For example, a unit circle may be defined using the implicit function

$$f(\boldsymbol{x}) = 1 - \|\boldsymbol{x}\| \tag{4.1}$$

for points $\boldsymbol{x} \in \mathbf{R}^2$. Points on the circle are those locations at which $f(\boldsymbol{x}) = 0$. This implicit function takes on positive values inside the circle and is negative outside, as will be the convention in this dissertation.

Figure 4.2 shows the contour plot of the implicit function defined as Equation (4.1). We use a color mapping scheme that maps positive values to blue, negative values to red, and zero to white. The black curve highlights where the zero crossing (the unit circle) is.

Similar to the way we define the implicit function to a unit circle, if we can construct a function $f(\boldsymbol{x})$ that has zero values at the sample points (the yellow dots shown in Figure 4.1 on the left) and positive values in side the given curve and negative outside, then we can extract the zero crossing which will be the curve we are looking for.

Assume an implicit function $f(\boldsymbol{x})$ represents the unknown curve, we have the sample points ($\boldsymbol{p}_i$, $i \in [1 \ldots N]$) on the curve shown as the yellow dots in Figure 4.1 on the left with their implicit function values (ϕ_i) as zero. Applying our mathematical tool of moving least-squares, we can solve the curve fitting problem as a scattered data interpolation.

Recall Equation (3.31) in the previous chapter

$$f(\boldsymbol{x}) = \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{x})\ \boldsymbol{H}^{-1}\ \boldsymbol{B}^{\mathsf{T}}\ \left(\boldsymbol{W}(\boldsymbol{x})\right)^2\ \boldsymbol{\phi} \quad , \tag{4.2}$$

If we only use those N sample points $\boldsymbol{p}_i$, $i \in [1 \ldots N]$, we get a trivial solution as $f(\boldsymbol{x}) = 0$ since $\boldsymbol{\phi} = [0, \ldots, 0]^{\mathsf{T}}$. This leads to an implicit function with zero values everywhere which is not the one we seek for. We understand that if we attempt to define a curve by only requiring it to take a given value on it, we will not obtain useful results.

4.2 Pseudo-Normal Constraints

To avoid getting a trivial solution, we need more points with positive/negative constraints to define the off curve information. This is the basic idea of pseudo-normal constraints which has been widely used in building implicit surfaces from scattered surface samples. Previous researchers, for example Ohtake et al. (2003), have implemented pseudo-normal constraints with a technique originally suggested by Turk and O'Brien (1999). This technique places a zero constraint at a point on the curve, a positive constraint offset sightly inside the curve, and a negative one slightly outside as illustrated in Figure 4.3

Unfortunately, this approach does not work as well as one might like. The additional

constraints influence the function's gradient only crudely, and they can cause undesirable oscillatory behavior as the evaluation point moves away from the surface. This behavior is illustrated in the contour plot as Figure 4.4. With the color mapping function, we can see that the pseudo-normal constraints method keeps the inside/outside information correctly and produces a curve (as shown on the right) that passes through all the sample points. However, if we take a closer look, we can easily see those dimples and lumps around the constrains and the extracted curve is not as smooth as what we expect.

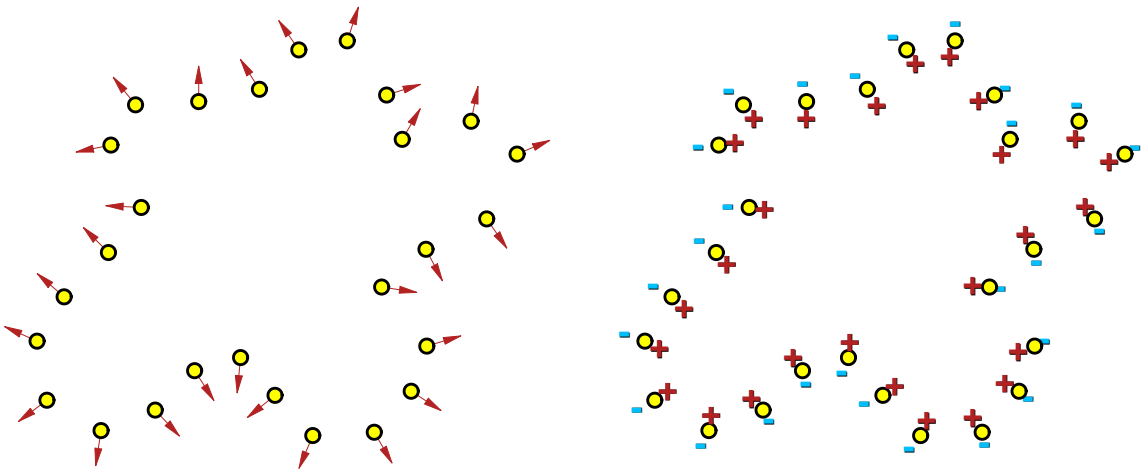

Figure 4.3. Pseudo-normal constraints technique places a zero constraint at a point on the curve, a positive constraint offset sightly inside the curve along the normal direction, and a negative one slightly outside. Left: a set of sample points with normal information; Right: point constraints setup with pseudo-normal constraints method.

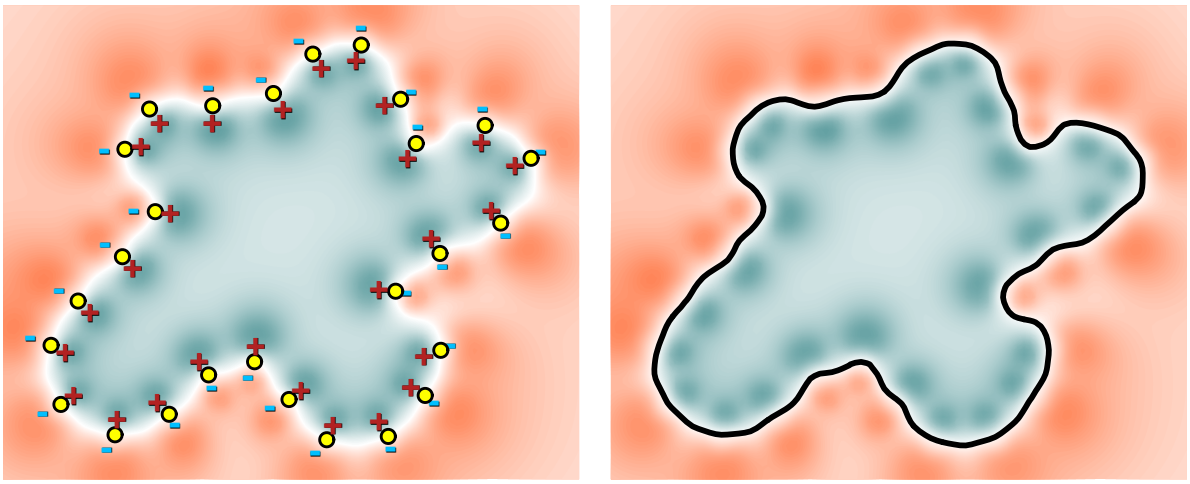

Figure 4.4. Contour plot with the pseudo normal constraints method. Left: contour plot with zero, positive and negative constraints. Right: contour plot with the extracted curve.

A zoom-in view of the cross section near the constraints in Figure 4.5 explains why we see those dimples and lumps in the contour plot. We can clearly see the oscillatory behavior away from the constraints.

The oscillatory behavior occurs because when the distance between the evaluation point and the surface point is much larger than the offset distance, the inside and outside constraints effectively cancel each other out. Even if only outside (or only inside) constraints are used, they will still effectively merge to a single average valued constraint far away. Heuristics, such as those described by Ohtake et al. (2003), can suppress some of the spurious behavior, but the value of the function far from the surface will not be useful. Furthermore, these quasi-normal constraints cause severe problems when used with the approximation procedure described later.

4.3 True-Normal Constrains

The oscillatory behavior of the pseudo-normal constraints method comes from the way we use the normal information by putting additional inside/outside point constraints. To avoid the undesirable oscillation, we need a better way to use the normals.

A point constraint associated with a normal vector gives us information about the whole space instead of just those three points (the original point plus two offset points). Figure 4.6 shows the contour plot of a single point constraint with a normal vector. The whole space has a well defined behavior that the upper half space has increasing values and the other half space has decreasing values.

Assume we have a point constraint at $\boldsymbol{p}$ with a normal $\hat{\boldsymbol{n}}$. We define the following shape

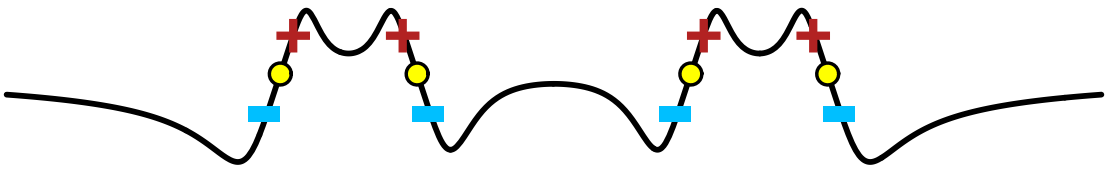

Figure 4.5. A 1D cross section showing the height field generated from the pseudo-normal constraints.

function S that can precisely represent the space plotted in Figure 4.6. Any point $\boldsymbol{x}$ in the space even far away from $\boldsymbol{p}$ can be predicated well by function S.

$$S(\boldsymbol{x}) = \phi + (\boldsymbol{x} - \boldsymbol{p})^{\mathsf{T}} \hat{\boldsymbol{n}} \tag{4.3}$$

$$= \psi_0 + \psi_x x + \psi_y y \quad , \tag{4.4}$$

where ϕ is the constraint value and ψ_{0k}, ψ_x and ψ_y are resulting polynomial coefficients

One of our key innovations is to impose normal constraints by forcing the interpolating function to behave like a prescribed function (a shape function), for example S in Equation (4.3), in the neighborhood of the constraint point, as opposed to a prescribed constant value (ϕ). In other words, instead of using the moving least-squares method to blend between constant values (ϕ_i) associated with each point, we blend between functions (S_i) associated with them. The fit from Equation (3.28) becomes

$$\begin{bmatrix} w(\boldsymbol{x}, \boldsymbol{p}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{x}, \boldsymbol{p}_N) \end{bmatrix} \begin{bmatrix} \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_1) \\ \vdots \\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_N) \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} w(\boldsymbol{x}, \boldsymbol{p}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{x}, \boldsymbol{p}_N) \end{bmatrix} \begin{bmatrix} S_1(\boldsymbol{x}) \\ \vdots \\ S_N(\boldsymbol{x}) \end{bmatrix} \tag{4.5}$$

where $S_i(\boldsymbol{x})$ is defined as

$$S_i(\boldsymbol{x}) = \phi_i + (\boldsymbol{x} - \boldsymbol{p}_i)^{\mathsf{T}} \hat{\boldsymbol{n}}_i \tag{4.6}$$

$$= \psi_{0i} + \psi_{xi} x + \psi_{yi} y \quad , \tag{4.7}$$

for each constraint point $\boldsymbol{p}_i$ with normal $\hat{\boldsymbol{n}}_i$.

Figure 4.6. Contour plot of a single point constraint with a normal vector. The whole space has a well defined behavior that the upper half space has increasing values and the other half space has decreasing values.

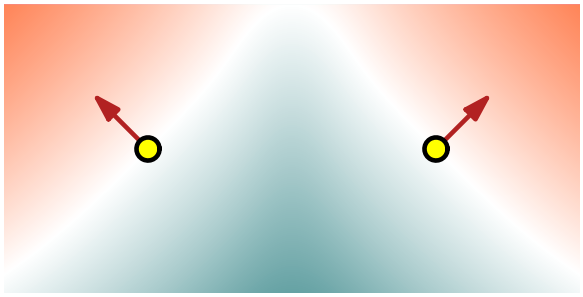

Figure 4.7. Contour plot of two point constraints each with a normal vector by using the moving least-squares formulation with interpolating functions.

We now apply moving least-squares formulation with interpolating S functions to two point constraints shown in Figure 4.6 each with a normal vector. In our experiment, a simple constant basis function ($\boldsymbol{b}(\boldsymbol{x}) = [1]$) works very well with the new normal constraints. Equation (4.5) simplifies to

$$\begin{bmatrix} w(\boldsymbol{x}, \boldsymbol{p}_1) \\ \vdots \\ w(\boldsymbol{x}, \boldsymbol{p}_i) \end{bmatrix} c_0 = \begin{bmatrix} w(\boldsymbol{x}, \boldsymbol{p}_1) & & \\ & \ddots & \\ & & w(\boldsymbol{x}, \boldsymbol{p}_N) \end{bmatrix} \begin{bmatrix} S_1(\boldsymbol{x}) \\ \vdots \\ S_N(\boldsymbol{x}) \end{bmatrix} \tag{4.8}$$

which has the very intuitive interpretation that the interpolating function's value at $\boldsymbol{x}$ is simply the weighted average of the values at $\boldsymbol{x}$ predicted by each of the $S_k(\boldsymbol{x})$.

Figure 4.7 shows the contour plot of such a blending function. We notice that this method exhibits little undesirable oscillation.

Interpolating between these functions reduces to simply interpolating the ψ coefficients just as we would normally interpolate a constant value. In the special case where $\hat{\boldsymbol{n}}_k = 0$, the normal constraints are exactly equivalent to the original value constraints. As a result we can easily mix constraints with and without normals.

When we apply the new moving least-squares formula Equation (4.8) to the curve extracting problem in Figure 4.1, we get a much nicer contour plot shown in Figure 4.8 without any dimples or lumps and the extracted zero level contour gives us a smoother curve passing through all the sample points.

Look at the zoom-in view of the cross section near the constraints in Figure 4.9, we

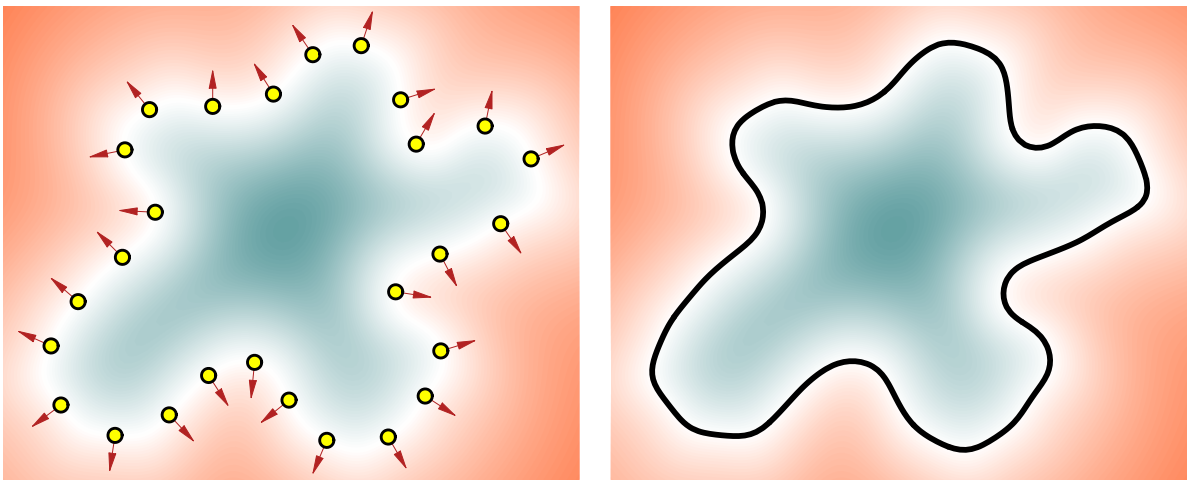

Figure 4.8. Contour plot with the true-normal constraints method. Left: contour plot with normal constraints. Right: contour plot with the extracted curve.

don't see those undesirable oscillations. The extended view tells us the far field behavior is stable.

If we put pseudo-normal constraints and true-normal constraints methods side by side, we can easily see the improvement as shown in Figure 4.10

In addition to being useful with moving least-squares, this approach should also work with other interpolation methods such as, for example, the radial splines used in Carr et al. (2001). Because the magnitude of the normal constraint grows linearly as the evaluation point moves away, the weighting function should fall off faster than linear.

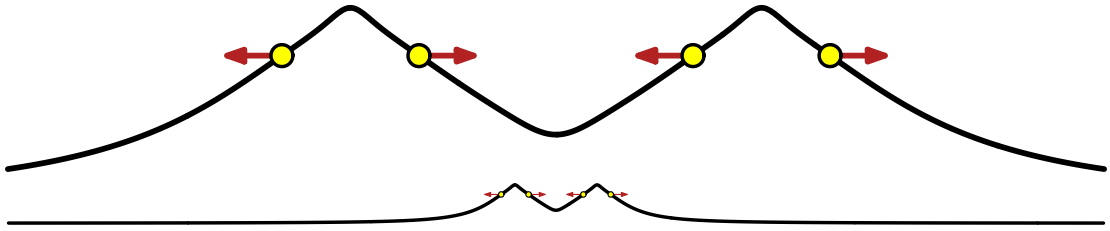

Figure 4.9. A 1D cross section showing the height field generated from the true-normal constraints. The first (top) image shows the result near the constraints, and the arrows indicate the outward normal directions. The second shows an expanded view demonstrating far-field behavior.

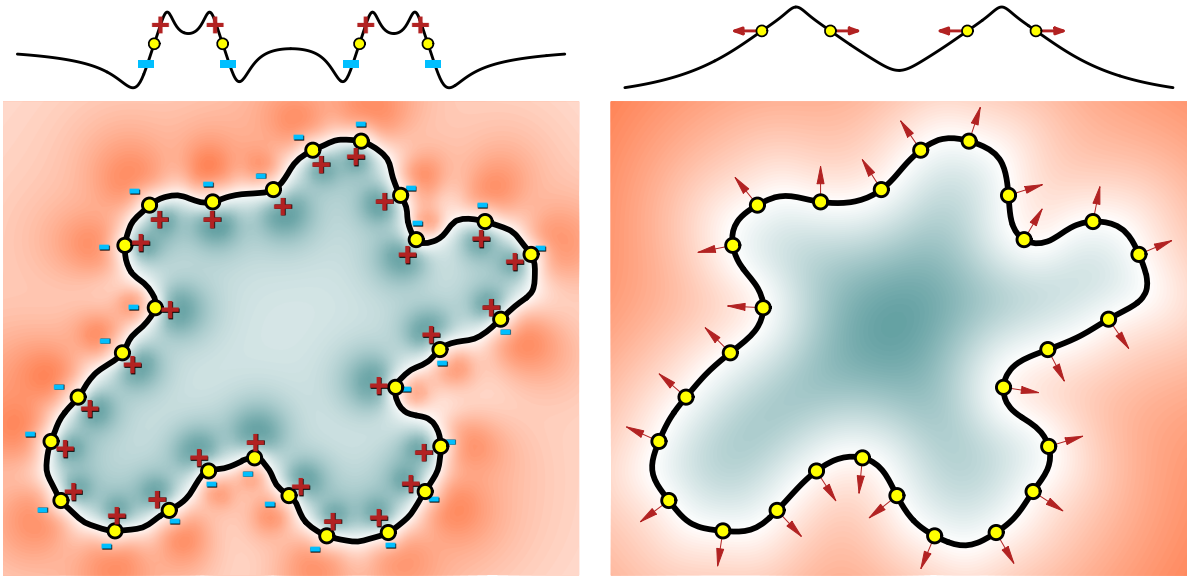

Figure 4.10. Comparison between pseudo-normal constraints method and our new true-normal constraints method. Top row figures show the cross section in a zoom-in view of each method. Bottom row figures show the contour plot with constraints setup and the extracted curves. Left column shows the results of the pseudo-normal constraints method. Right column shows the results of the true-normal constraints method.

4.4 Application: Surface Reconstruction from Range Scan Data

With the true normal constrains, the moving least square method works very well for point clouds interpolation including applications like surface reconstruction from range scan data.

Figure 4.11 shows the surface reconstruction results from real range scan data by using moving least-squares method with true-normal constraints.

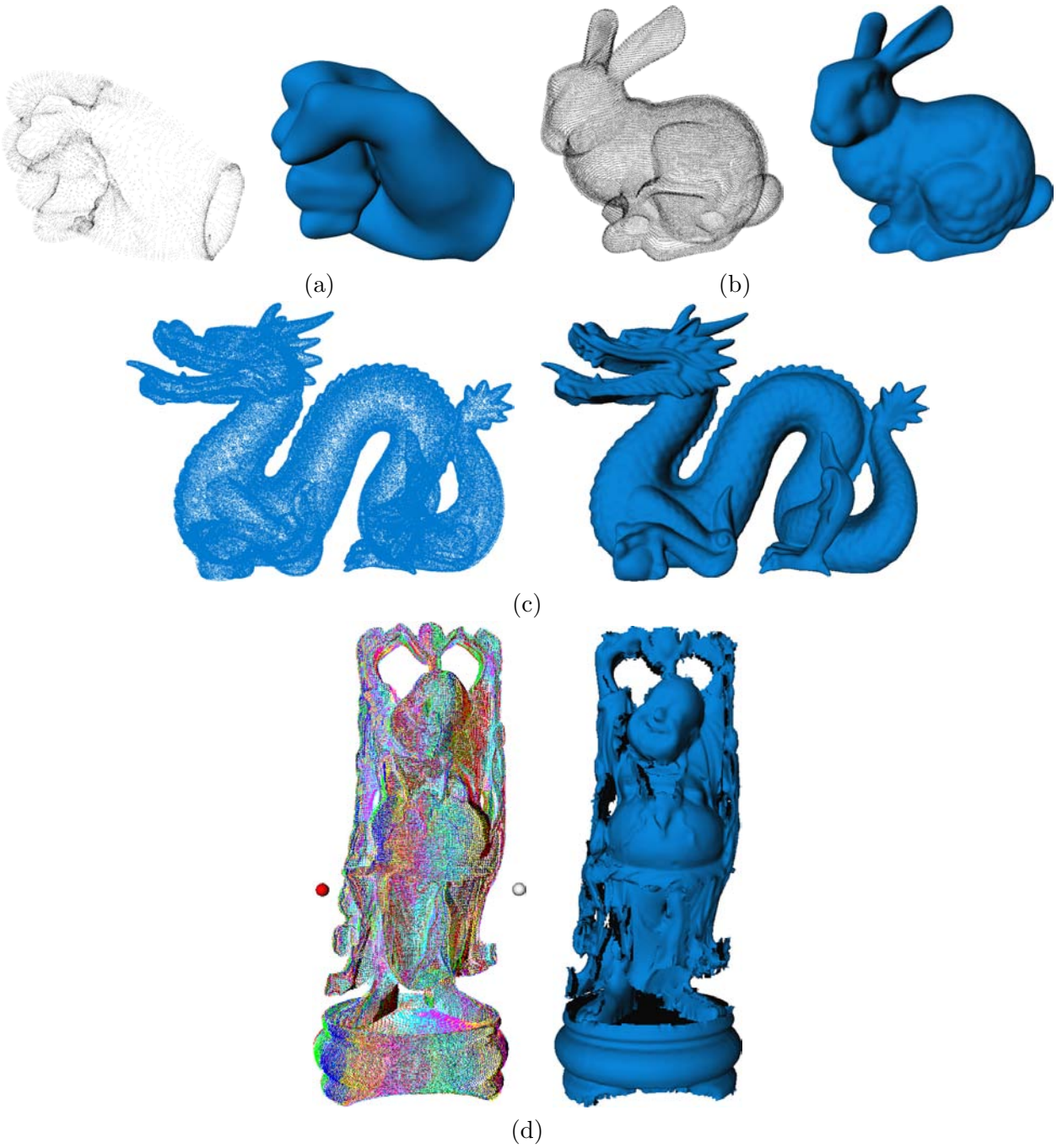

Figure 4.11. Surface Reconstruction from Range Scan Data. Left image of each example is the point clouds processed from range images. Surface normals are constructed from by analyzing the local neighborhood of each sample point with the given range scanner information. Right image shows the reconstructed surfaces. The last row of the Buddha example shows the blending between multiple scans. The different colors used in the left image of (d) represents different range images from multiple scanners.

Chapter 5

Integrating Moving Least-Squares

Although the formulation in the previous section works well for point constraints, the input data we are concerned with consists of polygons and parametric surface patches, and for each of these polygons or patches we want to constrain the fit function over its entire surface. If we were not interested in interpolating the polygons, we could approximate the desired effect with point constraints scattered over the surface of each polygon/patch. Aside from potentially requiring a very large number of points that require expensive computation,

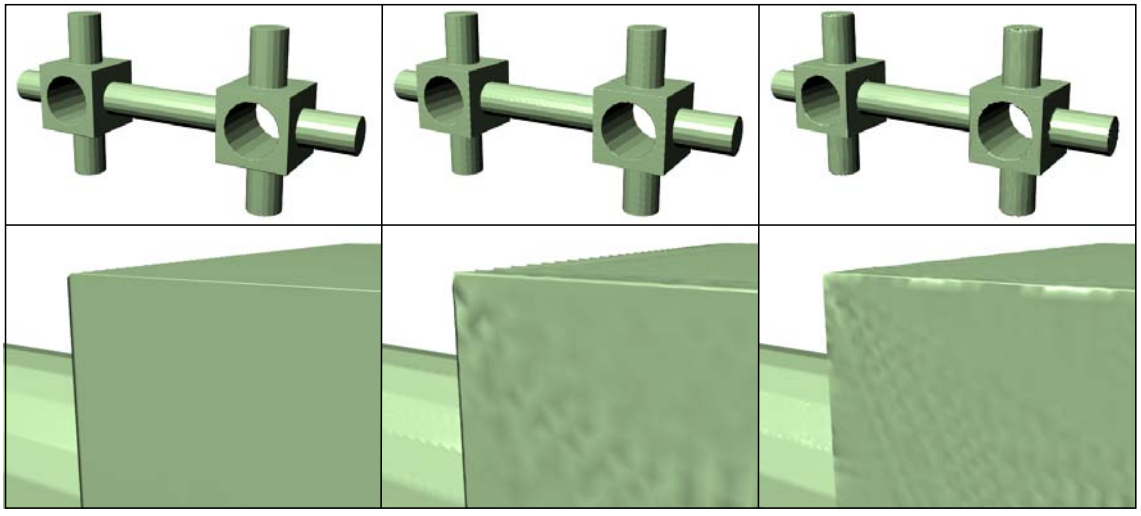

Figure 5.1. The column on the left shows the result from our method. The middle and right columns show the results generated with different densities of scattered points using the moving least-squares formulation with the true-normal constraints described in the previous section.

scattered point constraints work reasonably well for approximating surfaces. However, interpolating surfaces and surfaces that approximate closely will show undesirable bumps and dimples corresponding to the point locations. (See Figure 5.1.) In particular, bumps and dimples will occur unless ϵ is substantially larger than the spacing between points.

5.1 Problems of Finite Point Constraints

If we look at a 2D plot shown in Figures 5.2 and 5.3, we can understand better why point constraints method produces bumpy looking surfaces in the 3D example in Figure 5.1.

From Figure 5.2, we can clearly see those undesirable bumps and dimples in between point constraints. In the region where two line segments meet, this problem is much more severe since nearby normal constraints do not agree with each other. Near the flat region, however, even though the close-by normal constraints do agree with each other, bumps and dimples still appear since the far-away samples contribute contradicting constraints.

When we increase the sample density, like what Figure 5.3 shows, the problem becomes less severe. But if we look closely with a zoomed-in view at the region (shown in the red box) in between sample points, we can still see slight bumps and dimples.

5.2 Achieving Infinite Point Constraints

From Figure 5.2 and Figure 5.3, we understand that increasing the density of point constraints will improve the fit with smaller bumps and dimples, but with a finite number of constraints there is still room left for the curve to wiggle up and down.

To achieve good results, what we would like to do is to scatter an infinite number of points continuously across the surface of each polygon. Notice that Equation (3.30) can be rewritten as an explicit summation over a set of point constraints,

$$\left(\sum_{i=1}^{N} w^2(\boldsymbol{x}, \boldsymbol{p}_i)\, \boldsymbol{b}(\boldsymbol{p}_i)\, \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}_i) \right) \boldsymbol{c}(\boldsymbol{x}) = \sum_{i=1}^{N} w^2(\boldsymbol{x}, \boldsymbol{p}_i)\, \boldsymbol{b}(\boldsymbol{p}_i)\, \phi_i \tag{5.1}$$

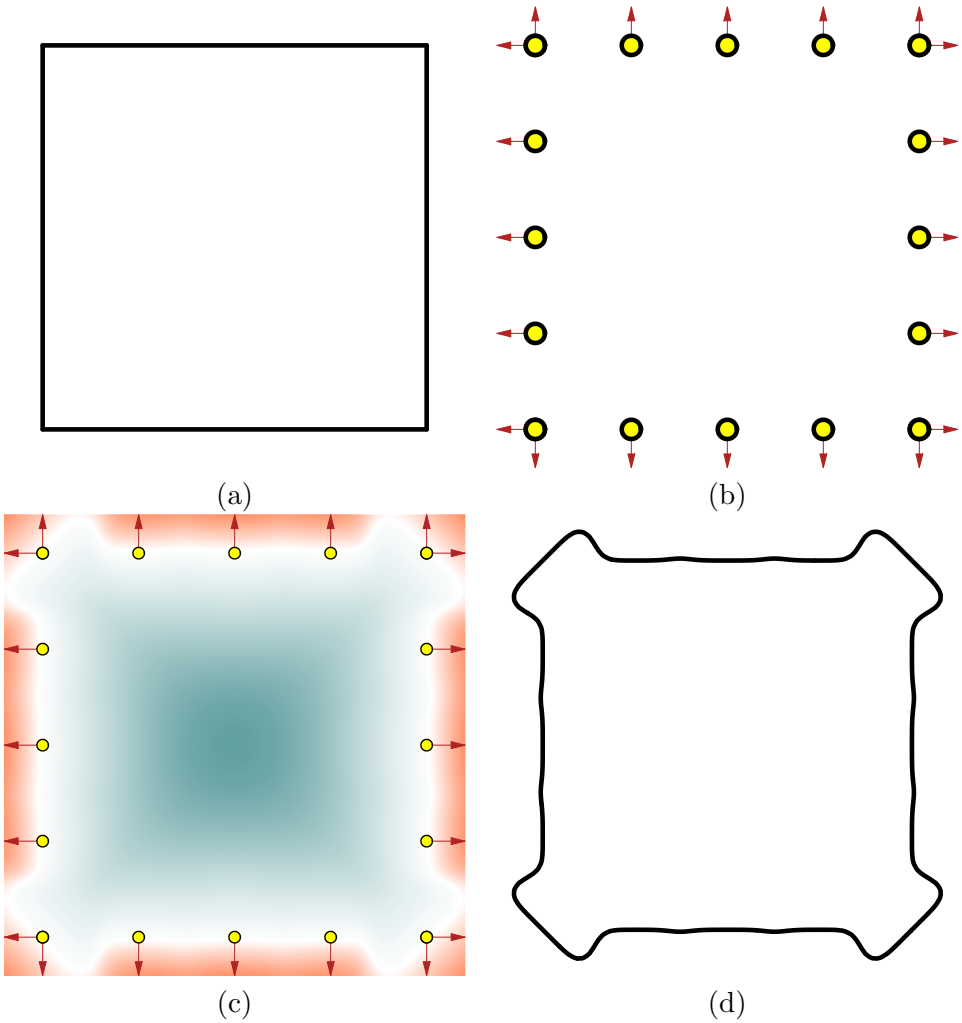

Figure 5.2. Point constraints method on a 2D shape with 4 line segments. Each line segment has 5 point samples. (a) the input shape; (b) point samples with true-normal constraints. At the corners of the square shape, there are two point samples with different normals overlapping together; (c) contour plot using the moving least squares formulation with true-normal constraints method; (d) extracted zero level contour;

In this form it becomes clear how we can apply constraints continuously over each element (line segment in 2D or polygon/patch in 3D).

For a data set of K element, let Ω_k, $k \in [1 \dots K]$, be the domain for each of K input elements. The parenthesized term of Equation (5.1) and the term on the right become integrals over all the points on the elements and we have

$$\left(\sum_{k=1}^{K} \boldsymbol{A}_k \right) \boldsymbol{c}(\boldsymbol{x}) = \sum_{k=1}^{K} \boldsymbol{a}_k \tag{5.2}$$

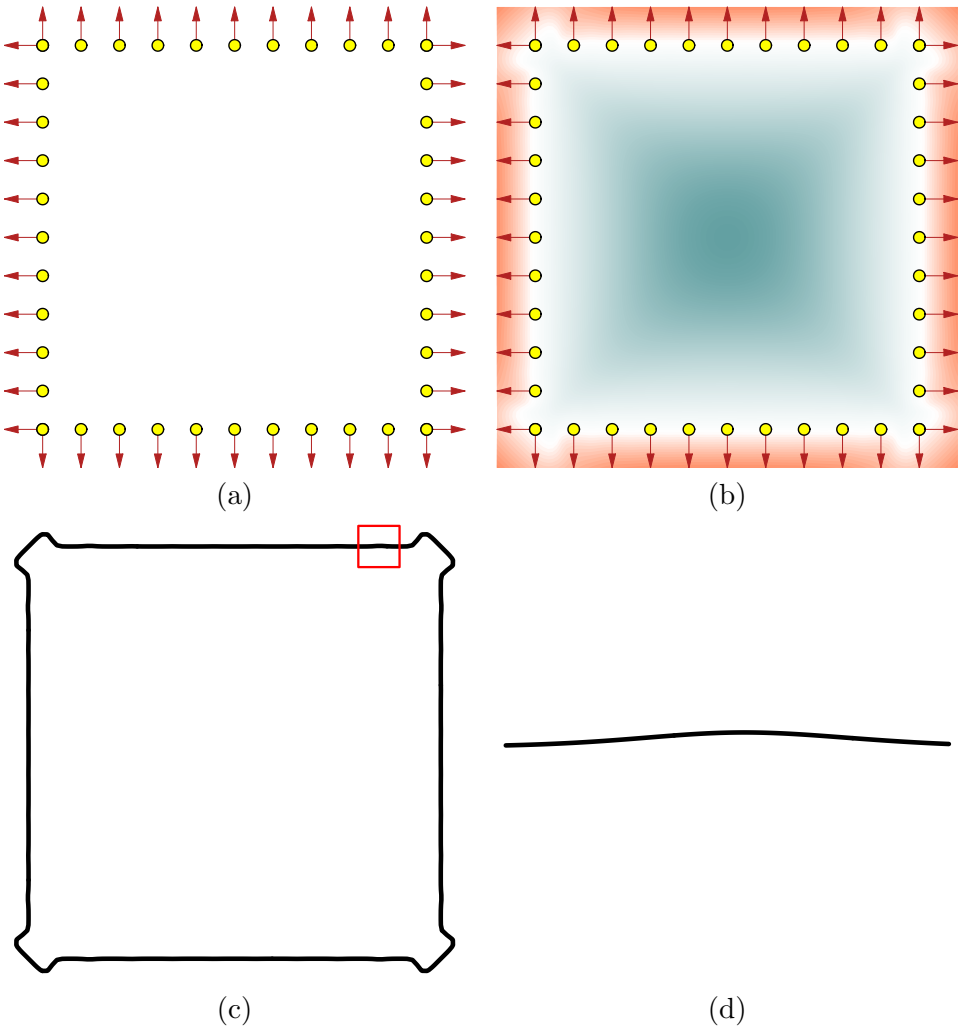

Figure 5.3. Point constraints method on a 2D shape with 4 line segments. Each line segment has 10 point samples. (a) point samples with true-normal constraints. At the corners of the square shape, there are two point samples with different normals overlapping together; (b) contour plot using the moving least squares formulation with true-normal constraints method; (c) extracted zero level contour. The red box shows the zoom-in region; (d) zoom-in view the extracted zero level contour;

where $\boldsymbol{A}_k$ and $\boldsymbol{a}_k$ are defined by

$$\boldsymbol{A}_k = \int_{\Omega_k} w^2(\boldsymbol{x}, \boldsymbol{p})\, \boldsymbol{b}(\boldsymbol{p})\, \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}) \;\; \mathrm{d}\boldsymbol{p} \quad , \tag{5.3}$$

$$\boldsymbol{a}_k = \int_{\Omega_k} w^2(\boldsymbol{x}, \boldsymbol{p})\, \boldsymbol{b}(\boldsymbol{p})\, \phi_k \;\; \mathrm{d}\boldsymbol{p} \quad , \tag{5.4}$$

$\boldsymbol{p}$ is the integration variable ranging over the element, and ϕ_k is the constraint value which we assume is either constant or varies polynomially over each element. For later use,

it is convenient to define terms with the weighting function omitted:

$$\widetilde{\boldsymbol{A}}_k = \int_{\Omega_k} \boldsymbol{b}(\boldsymbol{p})\, \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p}) \;\; \mathrm{d}\boldsymbol{p} \quad , \tag{5.5}$$

$$\tilde{\boldsymbol{a}}_k = \int_{\Omega_k} \boldsymbol{b}(\boldsymbol{p})\, \phi_k \;\; \mathrm{d}\boldsymbol{p} \quad . \tag{5.6}$$

The integrals will be infinite when $\epsilon = 0$ and the evaluation point $\mathbf{x}$ lies precisely on an element. In this case, we skip the least-squares step and simply set $f(\mathbf{x})$ to the value ϕ_k dictated by the element. It is possible that two elements intersect at a point where their constraints disagree, in which case f will approach some intermediate value.

Computing these integrals is conceptually straightforward. Each entry of the matrix $\boldsymbol{b}\,\boldsymbol{b}^{\mathsf{T}}$ and the vector $\boldsymbol{b}$ is a polynomial in $\boldsymbol{p}$, the weight function we have chosen is a rational polynomial in $\boldsymbol{p}$, and each of the components of the matrices can, of course, be computed independently.

5.3 Integration Over Line Segments

We first look at how to set up integrations in a 2D case over line segments. From our experiments, we found that using low order basis functions (for example a constant basis function as $\boldsymbol{b}(\boldsymbol{x}) = [1]$) with proper weight functions and the true normal constraints, we can achieve a good implicit function. In the following sections, we consider the constant basis function. For higher order basis functions, the derivation for integration is similar.

5.3.1 Function Values

With a constant basis function and a linear shape function, moving least squares formula for point clouds $\boldsymbol{p}_i$ as shown in Equation (5.1) becomes

$$c(\boldsymbol{x}) = \frac{\sum_i w_i^2(\boldsymbol{x}, \boldsymbol{p}_i)\, S(\boldsymbol{x}, \boldsymbol{p}_i)}{\sum_i w_i^2(\boldsymbol{x}, \boldsymbol{p}_i)} \tag{5.7}$$

where the shape function S_i is defined as

$$S_i(\boldsymbol{x}, \boldsymbol{p}_i) = \phi_{\boldsymbol{p}_i} + \boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{x} - \boldsymbol{p}_i) = \phi_{\boldsymbol{p}_i} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p}_i + \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{x} \tag{5.8}$$

and the weight function as

$$w_i(\boldsymbol{x}, \boldsymbol{p}_i) = \frac{1}{\|\boldsymbol{x} - \boldsymbol{p}_i\|^2 + \epsilon^2} \tag{5.9}$$

A line segment L_i ($[\boldsymbol{p}_{ia}, \boldsymbol{p}_{ib}]$) with ϕ_{ia} and ϕ_{ib} at the end points ($\boldsymbol{p}_{ia}$ and $\boldsymbol{p}_{ib}$) can be parameterized as a function of t ($t \in [0, 1]$). We assume the constraint function value varys along the line segment linearly. For any given $\boldsymbol{p} \in L_i$,

$$\boldsymbol{p} = (1-t)\,\boldsymbol{p}_{ia} + t\,\boldsymbol{p}_{ib} = (\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia})\,t + \boldsymbol{p}_{ia} \tag{5.10}$$

$$\phi\boldsymbol{p} = (1-t)\,\phi_{ia} + t\,\phi_{ib} = (\phi_{ib} - \phi_{ia})\,t + \phi_{ia} \tag{5.11}$$

For a data set of N line segments, let $L_i, i \in [1 \ldots N]$, be the ith input line segments. The MLS formula for point clouds can be extended to a formula for a set of line segments as follows,

$$c(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} \int_{L_i} w^2(\boldsymbol{x}, \boldsymbol{p})\, S(\boldsymbol{x}, \boldsymbol{p})\, \mathrm{d}\boldsymbol{p}}{\sum_{i=1}^{N} \int w^2(\boldsymbol{x}, \boldsymbol{p})\, \mathrm{d}\boldsymbol{p}} \tag{5.12}$$

with

$$\begin{aligned}
w(\boldsymbol{x}, t) &= w(\boldsymbol{x}, \boldsymbol{p}) \\
&= \frac{1}{\|\boldsymbol{x} - \boldsymbol{p}\|^2 + \epsilon^2} \\
&= \frac{1}{\|\boldsymbol{x} - ((\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia})\,t + \boldsymbol{p}_{ia})\|^2 + \epsilon^2} \\
&= \frac{1}{T_{i2}\,t^2 + T_{i1}\,t + T_{i0}} \\
&= \frac{1}{T_{i2}\left(t^2 + \frac{T_{i1}}{T_{i2}}\,t + \frac{T_{i0}}{T_{i2}}\right)} \\
&= \frac{1}{T_{i2}\left[\left(t + \frac{T_{i1}}{2T_{i2}}\right)^2 + \left(\frac{T_{i0}}{T_{i2}} - \frac{T_{i1}^2}{4T_{i2}^2}\right)\right]} \\
&= \frac{C_i}{(t + A_i)^2 + B_i}
\end{aligned} \tag{5.13}$$

and

$$\begin{aligned} S(\boldsymbol{x},t) &= S(\boldsymbol{x},\boldsymbol{p}) \\ &= \phi\boldsymbol{p} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p} + \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{x} \\ &= [(\phi_{ib} - \phi_{ia}) - \boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia})]\, t + [\phi_{ia} - \boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{p}_{ia} - \boldsymbol{x})] \\ &= PT_{i1}\, t + PT_{i0} \end{aligned} \tag{5.14}$$

where

$$T_{i2} = \|\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia}\|^2 = \|\boldsymbol{p}_{iba}\|^2 \tag{5.15}$$

$$T_{i1}(\boldsymbol{x}) = 2(\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia})^{\mathsf{T}} \cdot (\boldsymbol{p}_{ia} - \boldsymbol{x}) = 2\boldsymbol{p}_{iba}^{\mathsf{T}} \cdot \boldsymbol{p}_{iax} \tag{5.16}$$

$$T_{i0}(\boldsymbol{x}) = \|\boldsymbol{p}_{ia} - \boldsymbol{x}\|^2 + \epsilon^2 = \|\boldsymbol{p}_{iax}\|^2 + \epsilon^2 \tag{5.17}$$

$$A_i(\boldsymbol{x}) = \frac{T_{i1}}{2T_{i2}} \tag{5.18}$$

$$B_i(\boldsymbol{x}) = \frac{T_{i0}}{T_{i2}} - \frac{T_{i1}^2}{4T_{i2}^2} \tag{5.19}$$

$$C_i = \frac{1}{T_{i2}} \tag{5.20}$$

and

$$PT_{i1} = (\phi_{ib} - \phi_{ia}) - \boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia}) = \phi_{ib} - \phi_{ia} \tag{5.21}$$

$$PT_{i0}(\boldsymbol{x}) = \phi_{ia} - \boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{p}_{ia} - \boldsymbol{x}) = \phi_{ia} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p}_{iax} \tag{5.22}$$

where $\boldsymbol{n}_i$ is the normal vector of the ith line segment. $\boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia}) = 0$ since $\boldsymbol{n}_i$ is perpendicular to $\boldsymbol{p}_{ib} - \boldsymbol{p}_{ia}$.

We define,

$$c(\boldsymbol{x}) = \frac{Num(\boldsymbol{x})}{Den(\boldsymbol{x})} \tag{5.23}$$

where

$$Num(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, l_i \int_0^1 \frac{PT_{i1}\, t + PT_{i0}}{[(t + A_i)^2 + B_i]^2}\, \mathrm{d}t \tag{5.24}$$

$$Den(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, l_i \int_0^1 \frac{1}{[(t + A_i)^2 + B_i]^2}\, \mathrm{d}t \tag{5.25}$$

$$\tag{5.26}$$

l_i is the length of the ith line segment and we define

$$I_{i0}(\boldsymbol{x}) = \int_0^1 \frac{1}{[(t+A_i)^2+B_i]^2}\,\mathrm{d}t \tag{5.27}$$

$$\tag{5.28}$$

$$= \frac{\frac{\sqrt{B_i}\,(-A_i\,(1+A_i)+B_i)}{\left({A_i}^2+B_i\right)\left((1+A_i)^2+B_i\right)} - \arctan\left[\frac{A_i}{\sqrt{B_i}}\right] + \arctan\left[\frac{1+A_i}{\sqrt{B_i}}\right]}{2\,{B_i}^{3/2}} \tag{5.29}$$

$$I_{i1}(\boldsymbol{x}) = \int_0^1 \frac{t}{[(t+A_i)^2+B_i]^2}\,\mathrm{d}t \tag{5.30}$$

$$\tag{5.31}$$

$$= \frac{(1+A_i)\,\sqrt{B_i} + A_i\,((1+A_i)^2+B_i)\left(\arctan\left[\frac{A_i}{\sqrt{B_i}}\right] - \arctan\left[\frac{1+A_i}{\sqrt{B_i}}\right]\right)}{2\,{B_i}^{3/2}\,((1+A_i)^2+B_i)} \tag{5.32}$$

so,

$$Den(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\,l_i\,I_{i0}(\boldsymbol{x}) \tag{5.33}$$

$$Num(\boldsymbol{x}) = \sum_{i-1}^{N} C_i^2\,l_i\,(PT_{i1}\,I_{i1}(\boldsymbol{x}) + PT_{i0(\boldsymbol{x})}\,I_{i0}(\boldsymbol{x})) \tag{5.34}$$

We can summarize the computation for the integration over a set of line segments as the following pseudo-code

double MLSLineSegmentIntegrationValue($\boldsymbol{x}$, $\boldsymbol{p}_{ia}$, $\boldsymbol{p}_{ib}$, $\boldsymbol{n}_i$) {

$$
\begin{aligned}
T_{i2} &= \|\boldsymbol{p}_{iba}\|^2 \\
T_{i1} &= 2\boldsymbol{p}_{iba}^{\mathsf{T}} \cdot \boldsymbol{p}_{iax} \\
T_{i0} &= \|\boldsymbol{p}_{iax}\|^2 + \epsilon^2 \\
A_i &= \frac{T_{i1}}{2T_{i2}} \\
B_i &= \frac{T_{i0}}{T_{i2}} - \frac{T_{i1}^2}{4T_{i2}^2} \\
C_i &= \frac{1}{T_{i2}} \\
PT_{i1} &= \phi_{ib} - \phi_{ia} \\
PT_{i0} &= \phi_{ia} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p}_{iax} \\
ARC &= \arctan\left[\frac{A_i}{\sqrt{B_i}}\right] - \arctan\left[\frac{1+A_i}{\sqrt{B_i}}\right] \\
I_{i0} &= \frac{\frac{\sqrt{B_i}\,(-A_i\,(1+A_i)+B_i)}{\left({A_i}^2+B_i\right)\left((1+A_i)^2+B_i\right)} - ARC}{2\,{B_i}^{3/2}} \\
I_{i1} &= \frac{(1+A_i)\sqrt{B_i} + A_i\left((1+A_i)^2+B_i\right)ARC}{2\,{B_i}^{3/2}\left((1+A_i)^2+B_i\right)} \\
Den &= \sum_i C_i^2\, l_i\, I_{i0} \\
Num &= \sum_i C_i^2\, l_i\,(PT_{i1}\, I_{i1} + PT_{i0}\, I_{i0}) \\
\text{return } & \frac{Num}{Den}
\end{aligned}
$$

}

When we apply the above calculation to the problem shown in the previous section, we can achieve a much better result (shown as Figure 5.4) than what the point constraints method can provide us (comparison as shown in Figure 5.5).

5.3.2 Function Derivatives

$c(\boldsymbol{x}) = \frac{Num(\boldsymbol{x})}{Den(\boldsymbol{x})}$ has the derivative

$$\boldsymbol{c}'(\boldsymbol{x}) = \frac{Num'(\boldsymbol{x})\, Den(\boldsymbol{x}) - Num(\boldsymbol{x})\, Den'(\boldsymbol{x})}{Den^2(\boldsymbol{x})} \tag{5.35}$$

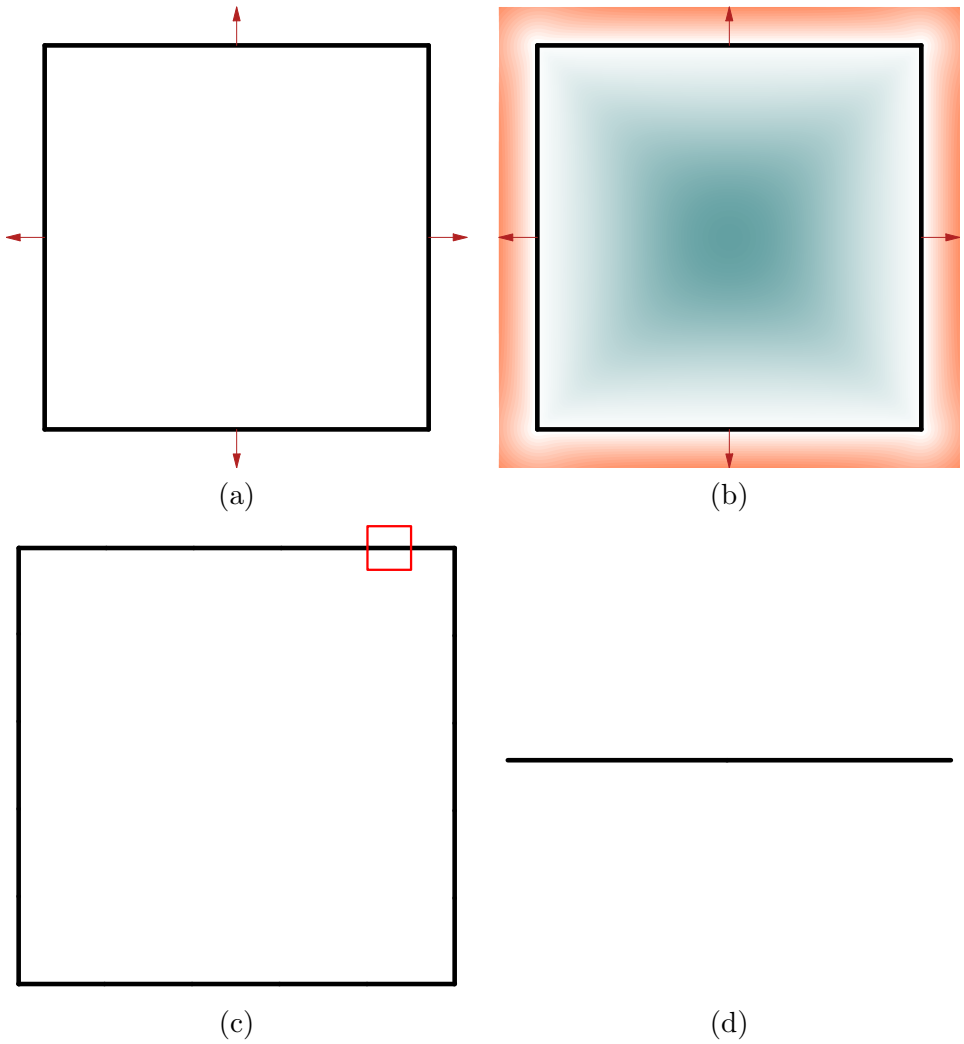

Figure 5.4. Integrating moving least squares constraints over line segments (a) Input constraints: 4 line segments, each has a normal vector; (b) contour plot by integrating the moving least squares true-normal constraints over the input line segments; (c) extracted zero level contour. The red box shows the zoom-in region; (d) zoom-in view the extracted zero level contour;

where

$$Den(\boldsymbol{x}) = \sum_{i=1}^{N} \int_{L_i} \frac{1}{(\|\boldsymbol{x}-\boldsymbol{p}\|^2+\epsilon^2)^2}\,\mathrm{d}\boldsymbol{p} \tag{5.36}$$

$$Num(\boldsymbol{x}) = \sum_{i=1}^{N} \int_{L_i} \frac{\phi\boldsymbol{p}-\boldsymbol{n}_i^{\mathsf{T}}\cdot\boldsymbol{p}+\boldsymbol{n}_i^{\mathsf{T}}\cdot\boldsymbol{x}}{(\|\boldsymbol{x}-\boldsymbol{p}\|^2+\epsilon^2)^2}\,\mathrm{d}\boldsymbol{p} \tag{5.37}$$

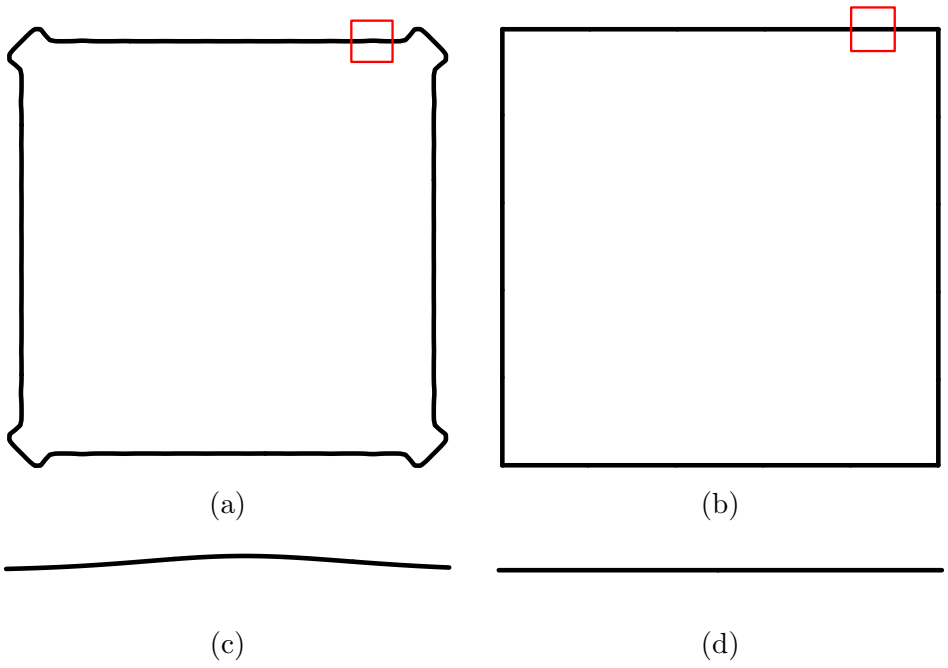

Figure 5.5. Comparison between scattered point constraints method and integrating constraints method. Left column shows the results from scattering 10 points on each line segment and solving standard point constraints; Right column shows the results from integrating constraints over line segments; Top row shows the whole extracted zero level contours for both methods; Bottom row shows the zoom-in views of the same selected region.

and

$$\begin{aligned}
\boldsymbol{Den}'(\boldsymbol{x}) &= \sum_{i=1}^{N} \int_{L_i} \frac{4\,(\boldsymbol{p}-\boldsymbol{x})}{(\|\boldsymbol{x}-\boldsymbol{p}\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p} \\
&= \sum_{i=1}^{N} 4\,C_i^3\,l_i \int_0^1 \frac{\boldsymbol{p}_{iba}\,t+\boldsymbol{p}_{iax}}{[(t+A_i)^2+B_i]^3}\,\mathrm{d}t \qquad (5.38) \\
\boldsymbol{Num}'(\boldsymbol{x}) &= \sum_{i=1}^{N} \int_{L_i} \frac{\boldsymbol{n}_i\,(\|\boldsymbol{x}-\boldsymbol{p}\|^2+\epsilon^2)+4\,(\boldsymbol{p}-\boldsymbol{x})\,\phi\boldsymbol{p}}{(\|\boldsymbol{x}-\boldsymbol{p}\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p} \\
&= \sum_{i=1}^{N} \left(C_i^2\,l_i \int_0^1 \frac{\boldsymbol{n}_i}{[(t+A_i)^2+B_i]^2}\,\mathrm{d}t + 4\,C_i^3\,l_i \int_0^1 \frac{(\boldsymbol{p}_{iba}\,t+\boldsymbol{p}_{iax})\,(PT_{i1}\,t+PT_{i0})}{[(t+A_i)^2+B_i]^3}\,\mathrm{d}t \right) \qquad (5.39)
\end{aligned}$$

We define

$$
\begin{aligned}
D_{i0} &= \int_0^1 \frac{1}{[(t+A_i)^2+B_i]^3}\,\mathrm{d}t \\
&= \frac{1}{8\,{B_i}^{5/2}}\Bigg(-\frac{A_i\,\sqrt{B_i}\,(3\,{A_i}^2+5\,B_i)}{({A_i}^2+B_i)^2}+\frac{(1+A_i)\,\sqrt{B_i}\,(3\,(1+A_i)^2+5\,B_i)}{((1+A_i)^2+B_i)^2}- \\
&\quad 3\,\arctan\big[\frac{A_i}{\sqrt{B_i}}\big]+3\,\arctan\big[\frac{1+A_i}{\sqrt{B_i}}\big]\Bigg) \qquad (5.40)\\
D_{i1} &= \int_0^1 \frac{t}{[(t+A_i)^2+B_i]^3}\,\mathrm{d}t \\
&= \frac{1}{8\,{B_i}^{5/2}}\Bigg(\big(\sqrt{B_i}\,(3\,{A_i}^2\,(1+A_i)^3+(1+A_i)\,(2+3\,A_i+6\,{A_i}^2)\,B_i+ \\
&\quad (4+3\,A_i)\,{B_i}^2)\big)/\big(({A_i}^2+B_i)\,((1+A_i)^2+B_i)^2\big)+ \\
&\quad 3\,A_i\,\arctan\big[\frac{A_i}{\sqrt{B_i}}\big]-3\,A_i\,\arctan\big[\frac{1+A_i}{\sqrt{B_i}}\big]\Bigg) \qquad (5.41)\\
D_{i2} &= \int_0^1 \frac{t^2}{[(t+A_i)^2+B_i]^3}\,\mathrm{d}t \\
&= \frac{1}{8\,{B_i}^{5/2}}\Bigg(\frac{\sqrt{B_i}\,(-3\,A_i\,(1+A_i)^3+B_i-A_i\,(3+4\,A_i)\,B_i-{B_i}^2)}{((1+A_i)^2+B_i)^2}- \\
&\quad (3\,{A_i}^2+B_i)\left(\arctan\big[\frac{A_i}{\sqrt{B_i}}\big]-\arctan\big[\frac{1+A_i}{\sqrt{B_i}}\big]\right)\Bigg) \qquad (5.42)
\end{aligned}
$$

so,

$$
\begin{aligned}
\boldsymbol{Den}'(\boldsymbol{x}) &= 4\sum_{i=1}^{N} C_i^3\,l_i,(\boldsymbol{p}_{iba}\,D_{i1}+\boldsymbol{p}_{iax}\,D_{i0}) \qquad (5.43)\\
\boldsymbol{Num}'(\boldsymbol{x}) &= \sum_{i=1}^{N} l_i\,(C_i^2\,\boldsymbol{n}_i\,I_{i0}+4\,C_i^3\,(\boldsymbol{DPT}_{i2}\,D_{i2}+\boldsymbol{DPT}_{i1}\,D_{i1}+\boldsymbol{DPT}_{i0}\,D_{i0})) \qquad (5.44)\\
& \qquad (5.45)
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{DPT}_{i2} &= \boldsymbol{p}_{iba}\,PT_{i1} \qquad (5.46)\\
\boldsymbol{DPT}_{i1} &= \boldsymbol{p}_{iba}\,PT_{i0}+\boldsymbol{p}_{iax}\,PT_{i1} \qquad (5.47)\\
\boldsymbol{DPT}_{i0} &= \boldsymbol{p}_{iax}\,PT_{i0} \qquad (5.48)
\end{aligned}
$$

We can summarize the computation for the derivative as the following pseudo-code

Vector2D MLSLineSegmentIntegrationDerivative($\boldsymbol{x}$, $\boldsymbol{p}_{ia}$, $\boldsymbol{p}_{ib}$, $\boldsymbol{n}_i$) {

$$
\begin{aligned}
T_{i2} &= \|\boldsymbol{p}_{iba}\|^2 \\
T_{i1} &= 2\boldsymbol{p}_{iba}^{\mathsf{T}} \cdot \boldsymbol{p}_{iax} \\
T_{i0} &= \|\boldsymbol{p}_{iax}\|^2 + \epsilon^2 \\
A_i &= \frac{T_{i1}}{2T_{i2}} \\
B_i &= \frac{T_{i0}}{T_{i2}} - \frac{T_{i1}^2}{4T_{i2}^2} \\
C_i &= \frac{1}{T_{i2}} \\
PT_{i1} &= \phi_{ib} - \phi_{ia} \\
PT_{i0} &= \phi_{ia} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p}_{iax} \\
\boldsymbol{DPT}_{i2} &= \boldsymbol{p}_{iba}\, PT_{i1} \\
\boldsymbol{DPT}_{i1} &= \boldsymbol{p}_{iba}\, PT_{i0} + \boldsymbol{p}_{iax}\, PT_{i1} \\
\boldsymbol{DPT}_{i0} &= \boldsymbol{p}_{iax}\, PT_{i0} \\
D_{i0} &= \frac{1}{8\, {B_i}^{5/2}} \Bigg(- \frac{A_i \sqrt{B_i}\,(3\,{A_i}^2 + 5\,B_i)}{({A_i}^2 + B_i)^2} + \frac{(1+A_i)\sqrt{B_i}\,(3\,(1+A_i)^2 + 5\,B_i)}{((1+A_i)^2 + B_i)^2} - \\
&\quad 3\,\arctan\Big[\frac{A_i}{\sqrt{B_i}}\Big] + 3\,\arctan\Big[\frac{1+A_i}{\sqrt{B_i}}\Big] \Bigg) \\
D_{i1} &= \frac{1}{8\, {B_i}^{5/2}} \Bigg(\Big(\sqrt{B_i}\,(3\,{A_i}^2\,(1+A_i)^3 + (1+A_i)\,(2 + 3\,A_i + 6\,{A_i}^2)\,B_i + \\
&\quad (4 + 3\,A_i)\,{B_i}^2) \Big) \Big/ \Big(({A_i}^2 + B_i)\,((1+A_i)^2 + B_i)^2 \Big) + \\
&\quad 3\,A_i\,\arctan\Big[\frac{A_i}{\sqrt{B_i}}\Big] - 3\,A_i\,\arctan\Big[\frac{1+A_i}{\sqrt{B_i}}\Big] \Bigg) \\
D_{i2} &= \frac{1}{8\, {B_i}^{5/2}} \Bigg(\frac{\sqrt{B_i}\,(-3\,A_i\,(1+A_i)^3 + B_i - A_i\,(3 + 4\,A_i)\,B_i - {B_i}^2)}{((1+A_i)^2 + B_i)^2} - \\
&\quad (3\,{A_i}^2 + B_i) \left(\arctan\Big[\frac{A_i}{\sqrt{B_i}}\Big] - \arctan\Big[\frac{1+A_i}{\sqrt{B_i}}\Big] \right) \Bigg) \\
\boldsymbol{Den}'(\boldsymbol{x}) &= 4 \sum_{i=1}^{N} C_i^3\, l_i, (\boldsymbol{p}_{iba}\, D_{i1} + \boldsymbol{p}_{iax}\, D_{i0}) \\
\boldsymbol{Num}'(\boldsymbol{x}) &= \sum_{i=1}^{N} l_i\, (C_i^2\, \boldsymbol{n}_i\, I_{i0} + 4\, C_i^3\, (\boldsymbol{DPT}_{i2}\, D_{i2} + \boldsymbol{DPT}_{i1}\, D_{i1} + \boldsymbol{DPT}_{i0}\, D_{i0})) \\
\text{return} \quad & \frac{Num'(\boldsymbol{x})\, Den(\boldsymbol{x}) - Num(\boldsymbol{x})\, Den'(\boldsymbol{x})}{Den^2(\boldsymbol{x})}
\end{aligned}
$$

}

5.4 Integration Over Triangles

A triangle region $\triangle_i(\boldsymbol{p}_{i0}, \boldsymbol{p}_{i1}, \boldsymbol{p}_{i2})$ with ϕ_{i0}, ϕ_{i1}, ϕ_{i2} at the vertices can be parameterized as,

$$\begin{aligned} \boldsymbol{p} &= (1-s-t)\,\boldsymbol{p}_{i0} + s\,\boldsymbol{p}_{i1} + t\,\boldsymbol{p}_{i2} \\ &= \boldsymbol{p}_{i0} + (\boldsymbol{p}_{i1} - \boldsymbol{p}_{i0})\,s + (\boldsymbol{p}_{i2} - \boldsymbol{p}_{i0})\,t \qquad (5.49) \\ \phi\boldsymbol{p} &= \phi_{i0} + (\phi_{i1} - \phi_{i0})\,s + (\phi_{i2} - \phi_{i0})\,t \qquad (5.50) \end{aligned}$$

5.4.1 Function Values

With the normal vector $\boldsymbol{n}_i$, the shape function of the moving least squares formulation for this triangle is

$$\begin{aligned} S(\boldsymbol{x}, \boldsymbol{p}) &= (\phi\boldsymbol{p} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p}) + \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{x} \\ &= \phi_0 + \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{x} \qquad (5.51) \end{aligned}$$

We separate the evaluation point $\boldsymbol{x}$ from other terms so that we can achieve an approximation for a fast evaluation that will be discussed in the next chapter. For a data set of N triangles, let Ω_i, $i \in [1 \ldots N]$, be the ith input triangle. We have,

$$c(\boldsymbol{x}) = \frac{Num_0(\boldsymbol{x}) + \boldsymbol{x}^{\mathsf{T}} \cdot \boldsymbol{Num}_{\boldsymbol{x}}(\boldsymbol{x})}{Den(\boldsymbol{x})} \qquad (5.52)$$

where

$$\begin{aligned} Den(\boldsymbol{x}) &= \sum_{i=1}^{N} \int_{\Omega_i} w^2(\boldsymbol{x}, \boldsymbol{p})\, \mathrm{d}\boldsymbol{p} \qquad (5.53) \\ Num_0(\boldsymbol{x}) &= \sum_{i=1}^{N} \int_{\Omega_i} w^2(\boldsymbol{x}, \boldsymbol{p})\, \phi_0\, \mathrm{d}\boldsymbol{p} \qquad (5.54) \\ \boldsymbol{Num}_{\boldsymbol{x}}(\boldsymbol{x}) &= \sum_{i=1}^{N} \boldsymbol{n}_i \int_{\Omega_i} w^2(\boldsymbol{x}, \boldsymbol{p})\, \mathrm{d}\boldsymbol{p} \qquad (5.55) \end{aligned}$$

with

$$\begin{aligned} w(\boldsymbol{x},s,t) &= w(\boldsymbol{x},\boldsymbol{p}_i) \\ &= \frac{1}{\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2} \\ &= \frac{1}{\|\boldsymbol{x}-(\boldsymbol{p}_{i0}+(\boldsymbol{p}_{i1}-\boldsymbol{p}_{i0})\,s+(\boldsymbol{p}_{i2}-\boldsymbol{p}_{i0})\,t)\|^2+\epsilon^2} \\ &= \frac{1}{T_{i2}\,t^2+T_{i1}(s)\,t+T_{i0}(s)} \\ &= \frac{1}{T_{i2}\left(t^2+\frac{T_{i1}(s)}{T_{i2}}\,t+\frac{T_{i0}(s)}{T_{i2}}\right)} \\ &= \frac{1}{T_{i2}\left[\left(t+\frac{T_{i1}(s)}{2T_{i2}}\right)^2+\left(\frac{T_{i0}(s)}{T_{i2}}-\frac{T_{i1}^2(s)}{4T_{i2}^2}\right)\right]\,|} \\ &= \frac{C_i}{(t+A_i(s))^2+B_i(s)} \end{aligned} \tag{5.56}$$

and

$$\begin{aligned} \phi_0 &= \phi_{\boldsymbol{p}}-\boldsymbol{n}_i^{\mathsf{T}}\cdot\boldsymbol{p} \\ &= \phi_{i0}+(\phi_{i1}-\phi_{i0})\,s+(\phi_{i2}-\phi_{i0})\,t-\boldsymbol{n}_i^{\mathsf{T}}\cdot(\boldsymbol{p}_{i0}+(\boldsymbol{p}_{i1}-\boldsymbol{p}_{i0})\,s+(\boldsymbol{p}_{i2}-\boldsymbol{p}_{i0})\,t) \\ &= (\phi_{i20}-\boldsymbol{n}_i^{\mathsf{T}}\cdot\boldsymbol{p}_{\boldsymbol{i20}})\,t+\phi_{i0}+\phi_{i10}\,s-\boldsymbol{n}_i^{\mathsf{T}}\cdot\boldsymbol{p}_s(s) \\ &= PT_{i1}\,t+PT_{i0}(s) \end{aligned} \tag{5.57}$$

where

$$T_{i2} = \|\boldsymbol{p}_{i2}-\boldsymbol{p}_{i0}\|^2=\|\boldsymbol{p}_{\boldsymbol{i20}}\|^2 \tag{5.58}$$

$$\begin{aligned} T_{i1}(\boldsymbol{x},s) &= 2(\boldsymbol{p}_{i2}-\boldsymbol{p}_{i0})^{\mathsf{T}}\cdot((\boldsymbol{p}_{i0}+(\boldsymbol{p}_{i1}-\boldsymbol{p}_{i0})\,s)-\boldsymbol{x}) \\ &= 2\boldsymbol{p}_{\boldsymbol{i20}}{}^{\mathsf{T}}\cdot(\boldsymbol{p}_s(s)-\boldsymbol{x}) \end{aligned} \tag{5.59}$$

$$T_{i0}(\boldsymbol{x},s) = \|(\boldsymbol{p}_{i0}+(\boldsymbol{p}_{i1}-\boldsymbol{p}_{i0})\,s)-\boldsymbol{x}\|^2+\epsilon^2=\|\boldsymbol{p}_s(s)-\boldsymbol{x}\|^2+\epsilon^2 \tag{5.60}$$

$$A_i(s)(\boldsymbol{x}) = \frac{T_{i1}(s)}{2T_{i2}} \tag{5.61}$$

$$B_i(s)(\boldsymbol{x}) = \frac{T_{i0}(s)}{T_{i2}}-\frac{T_{i1}^2(s)}{4T_{i2}^2} \tag{5.62}$$

$$C_i = \frac{1}{T_{i2}} \tag{5.63}$$

$$\boldsymbol{p}_s(s) = \boldsymbol{p}_{i0}+(\boldsymbol{p}_{i1}-\boldsymbol{p}_{i0})\,s \tag{5.64}$$

and

$$PT_{i1} = \phi_{i20} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p_{i20}} = \phi_{i20} \tag{5.65}$$

$$PT_{i0}(s) = \phi_{i0} + \phi_{i10}\, s - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{p}_s(s) \tag{5.66}$$

so,

$$Num_0(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, 2\, Area_i \int_0^1 \int_0^{1-s} \frac{\phi_0}{[(t + A_i(s))^2 + B_i(s)]^2}\, \mathrm{d}t\, \mathrm{d}s \tag{5.67}$$

$$\boldsymbol{Num_x}(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, 2\, Area_i\, \boldsymbol{n}_i \int_0^1 \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2}\, \mathrm{d}t\, \mathrm{d}s \tag{5.68}$$

$$Den(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, 2\, Area_i \int_0^1 \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2}\, \mathrm{d}t\, \mathrm{d}s \tag{5.69}$$

We define

$$\begin{aligned} I_{i0}(s) &= \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2} \mathrm{d}t \\ &= \frac{-\frac{A_i(s)\sqrt{B_i(s)}}{A_i(s)^2 + B_i(s)} + \frac{\sqrt{B_i(s)}\,(1 + A_i(s) - s)}{B_i(s) + (1 + A_i(s) - s)^2} - \arctan\left[\frac{A_i(s)}{\sqrt{B_i(s)}}\right] + \arctan\left[\frac{1 + A_i(s) - s}{\sqrt{B_i(s)}}\right]}{2\, B_i(s)^{3/2}} \end{aligned} \tag{5.70}$$

$$\begin{aligned} I_{i1}(s) &= \int_0^{1-s} \frac{t}{[(t + A_i(s))^2 + B_i(s)]^2} \mathrm{d}t \\ &= \frac{1}{2\, B_i(s)^{3/2}} \left(\sqrt{B_i(s)} - \frac{\sqrt{B_i(s)}\,(A_i(s) + A_i(s)^2 + B_i(s) - A_i(s)\, s)}{B_i(s) + (1 + A_i(s) - s)^2} + \right. \\ &\quad \left. A_i(s)\, \arctan\left[\frac{A_i(s)}{\sqrt{B_i(s)}}\right] - A_i(s)\, \arctan\left[\frac{1 + A_i(s) - s}{\sqrt{B_i(s)}}\right] \right) \end{aligned} \tag{5.71}$$

so,

$$Den(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, 2\, Area_i \int_0^1 I_{i0}(s)\, \mathrm{d}s \tag{5.72}$$

$$Num_0(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, 2\, Area_i \int_0^1 \left(PT_{i1}\, I_{i1}(s) + PT_{i0}(s)\, I_{i0}(s)\right) \mathrm{d}s \tag{5.73}$$

$$\boldsymbol{Num_x}(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^2\, 2\, Area_i\, \boldsymbol{n}_i \int_0^1 I_{i0}(s)\, \mathrm{d}s \tag{5.74}$$

5.4.2 Function Derivatives

Similar as the line segments case, formula

$$c(\boldsymbol{x}) = \frac{Num_0(\boldsymbol{x}) + \boldsymbol{x}^\mathsf{T} \cdot \boldsymbol{Num_x}(\boldsymbol{x})}{Den(\boldsymbol{x})} \tag{5.75}$$

has the derivative as

$$\begin{aligned} \boldsymbol{c'}(\boldsymbol{x}) &= \frac{\left(\boldsymbol{Num_0}'(\boldsymbol{x}) + \boldsymbol{Num_x}(\boldsymbol{x}) + \left(\boldsymbol{x}^\mathsf{T} \cdot \boldsymbol{NUM_x}'(\boldsymbol{x})_{3\times 3}\right)^\mathsf{T}\right) Den(\boldsymbol{x})}{Den(\boldsymbol{x})^2} - & (5.76) \\ & \frac{\left(Num_0(\boldsymbol{x}) + \boldsymbol{x}^\mathsf{T} \cdot \boldsymbol{Num_x}(\boldsymbol{x})\right) \boldsymbol{Den}'(\boldsymbol{x})}{Den(\boldsymbol{x})^2} & (5.77) \end{aligned}$$

We define

$$\begin{aligned} D_{i0}(s) &= \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^3}\,\mathrm{d}t \\ &= \frac{1}{8\,B_i(s)^{5/2}}\Bigg(- \frac{A_i(s)\,\sqrt{B_i(s)}\,(3\,A_i(s)^2 + 5\,B_i(s))}{\left(A_i(s)^2 + B_i(s)\right)^2} + \\ &\quad \frac{\sqrt{B_i(s)}\,(5\,B_i(s) + 3\,(1 + A_i(s) - s)^2)\,(1 + A_i(s) - s)}{\left(B_i(s) + (1 + A_i(s) - s)^2\right)^2} - \\ &\quad 3\,\arctan\left[\frac{A_i(s)}{\sqrt{B_i(s)}}\right] + 3\,\arctan\left[\frac{1 + A_i(s) - s}{\sqrt{B_i(s)}}\right]\Bigg) \end{aligned} \tag{5.78}$$

$$\begin{aligned} D_{i1}(s) &= \int_0^{1-s} \frac{t}{[(t + A_i(s))^2 + B_i(s)]^3}\,\mathrm{d}t \\ &= \frac{1}{8\,B_i(s)^{5/2}}\Bigg(\sqrt{B_i(s)}\,\Bigg(2 + \frac{A_i(s)^2}{A_i(s)^2 + B_i(s)} - \\ &\quad \frac{2\,B_i(s)\,(A_i(s) + A_i(s)^2 + B_i(s) - A_i(s)\,s)}{\left(1 + 2\,A_i(s) + A_i(s)^2 + B_i(s) - 2\,(1 + A_i(s))\,s + s^2\right)^2} - \\ &\quad \frac{3\,A_i(s)\,(1 + A_i(s) - s)}{1 + 2\,A_i(s) + A_i(s)^2 + B_i(s) - 2\,(1 + A_i(s))\,s + s^2}\Bigg) + \\ &\quad 3\,A_i(s)\,\arctan\left[\frac{A_i(s)}{\sqrt{B_i(s)}}\right] - 3\,A_i(s)\,\arctan\left[\frac{1 + A_i(s) - s}{\sqrt{B_i(s)}}\right]\Bigg) \end{aligned} \tag{5.79}$$

$$
\begin{aligned}
D_{i2}(s) &= \int_0^{1-s} \frac{t^2}{[(t+A_i(s))^2+B_i(s)]^3}\,\mathrm{d}t \\
&= \frac{1}{8\,B_i(s)^{5/2}}\Bigg(\Big(\sqrt{B_i(s)}\,(3\,A_i(s)^4 + B_i(s)\,(B_i(s)-(-1+s)^2) + \\
&\quad A_i(s)^2\,(4\,B_i(s)+9\,(-1+s)^2) - 9\,A_i(s)^3\,(-1+s) - \\
&\quad 3\,A_i(s)\,(B_i(s)+(-1+s)^2)\,(-1+s))\,(-1+s)\Big)\Big/ \\
&\quad (A_i(s)^2 + B_i(s) - 2\,A_i(s)\,(-1+s) + (-1+s)^2)^2 - \\
&\quad (3\,A_i(s)^2 + B_i(s))\left(\arctan\Big[\frac{A_i(s)}{\sqrt{B_i(s)}}\Big] - \arctan\Big[\frac{1+A_i(s)-s}{\sqrt{B_i(s)}}\Big]\right)\Bigg) \qquad (5.80)
\end{aligned}
$$

with

$$
\begin{aligned}
\boldsymbol{Den}'(\boldsymbol{x}) &= \sum_{i=1}^{N} \int \frac{4\,(\boldsymbol{p}_i - \boldsymbol{x})}{(\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p}_i \\
&= 4\,(\boldsymbol{dDen_0}(\boldsymbol{x}) - \boldsymbol{x}^{\mathsf{T}} \cdot dDen_x(\boldsymbol{x})) \qquad (5.81) \\
\boldsymbol{dDen_0}(\boldsymbol{x}) &= \sum_{i=1}^{N} \int \frac{\boldsymbol{p}_i}{(\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_{i=1}^{N} C_i^3\,2\,Area_i \int_0^1 \int_0^{1-s} \frac{\boldsymbol{p_{i20}}\,t + (\boldsymbol{p}_{i0} + \boldsymbol{p_{i10}}\,s)}{[(t+A_i(s))^2+B_i(s)]^3}\mathrm{d}t\,\mathrm{d}s \\
&= \sum_{i=1}^{N} C_i^3\,2\,Area_i \int_0^1 (\boldsymbol{p_{i20}}\,D_{i1}(s) + \boldsymbol{p}_s\,D_{i0}(s))\mathrm{d}s \\
&= \sum_{i=1}^{N} C_i^3\,2\,Area_i\,\boldsymbol{IdP_i}(x) \qquad (5.82) \\
dDen_x(\boldsymbol{x}) &= \sum_{i=1}^{N} \int \frac{1}{(\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_{i=1}^{N} C_i^3\,2\,Area_i \int_0^1 D_{i0}(s)\,\mathrm{d}s \\
&= \sum_{i=1}^{N} C_i^3\,2\,Area_i\,Id1(x) \qquad (5.83) \\
\boldsymbol{Num_0}'(\boldsymbol{x}) &= \sum_{i=1}^{N} \int \frac{4\,(\boldsymbol{p}_i - \boldsymbol{x})\,\phi_0}{(\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p}_i \\
&= 4\left(\sum_{i=1}^{N} \int \frac{\boldsymbol{p}_i\,\phi_0}{(\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p}_i - \boldsymbol{x}\sum_{i=1}^{N} \int \frac{\phi_0}{(\|\boldsymbol{x}-\boldsymbol{p}_i\|^2+\epsilon^2)^3}\,\mathrm{d}\boldsymbol{p}_i\right) \\
&= 4\,(\boldsymbol{dNum_{00}}(\boldsymbol{x}) - \boldsymbol{x}\,dNum_{0x}(\boldsymbol{x})) \qquad (5.84)
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{dNum_{00}}(\boldsymbol{x}) &= \sum_{i=1}^{N} C_i^3\, 2\, Area_i \int_0^1 \int_0^{1-s} \frac{(\boldsymbol{p_{i20}}\, t + \boldsymbol{p}_s(s))\,(PT_{i1}\, t + PT_{i0}(s))}{[(t + A_i(s))^2 + B_i(s)]^3}\, \mathrm{d}t\, \mathrm{d}s \\
&= \sum_{i=1}^{N} C_i^3\, 2\, Area_i \int_0^1 (DPT_{i2}\, D_{i2}(s) + DPT_{i1}(s)\, D_{i1}(s) + DT_{i0}(s)\, D_{i0}(s))\, \mathrm{d}s
\end{aligned} \tag{5.85}
$$

$$
\begin{aligned}
dNum_{0x}(\boldsymbol{x}) &= \sum_{i=1}^{N} C_i^3\, 2\, Area_i \int_0^1 \int_0^{1-s} \frac{PT_{i1}\, t + PT_{i0}(s)}{[(t + A_i(s))^2 + B_i(s)]^3}\, \mathrm{d}t\, \mathrm{d}s \\
&= \sum_{i=1}^{N} C_i^3\, 2\, Area_i \int_0^1 (PT_{i1}\, D_{i1}(s) + PT_{i0}(s)\, D_{i0}(s))\, \mathrm{d}s
\end{aligned} \tag{5.86}
$$

$$
\begin{aligned}
\boldsymbol{NUM'_x}(\boldsymbol{x})_{3\times 3} &= \sum_{i=1}^{N} \boldsymbol{n}_i^{\mathsf{T}} \cdot \left(\int \frac{4\,(\boldsymbol{p}_i - \boldsymbol{x})}{(\|\boldsymbol{x} - \boldsymbol{p}_i\|^2 + \epsilon^2)^3}\, \mathrm{d}\boldsymbol{p}_i \right)^{\mathsf{T}} \\
&= 4\,(\boldsymbol{dNUM_{x0}}(\boldsymbol{x})_{3\times 3} - \boldsymbol{x}^{\mathsf{T}} \cdot (\boldsymbol{dNUM_{xx}}(\boldsymbol{x}))^{\mathsf{T}})
\end{aligned} \tag{5.87}
$$

$$
\boldsymbol{dNUM_{x0}}(\boldsymbol{x})_{3\times 3} = \sum_{i=1}^{N} C_i^3\, 2\, Area_i\, (\boldsymbol{n}_i^{\mathsf{T}} \cdot (\boldsymbol{IdP_i}(\boldsymbol{x}))^{\mathsf{T}}) \tag{5.88}
$$

$$
\boldsymbol{dNUM_{xx}}(\boldsymbol{x}) = \sum_{i=1}^{N} C_i^3\, 2\, Area_i\, \boldsymbol{n}_i\, Id1(\boldsymbol{x}) \tag{5.89}
$$

and

$$
\boldsymbol{DPT_{i2}} = \boldsymbol{p_{i20}}\, PT_{i1} \tag{5.90}
$$

$$
\boldsymbol{DPT_{i1}} = \boldsymbol{p_{i20}}\, PT_{i0}(s) + \boldsymbol{p}_s\, PT_{i1} \tag{5.91}
$$

$$
\boldsymbol{DPT_{i0}} = \boldsymbol{p}_s\, PT_{i0}(s) \tag{5.92}
$$

5.4.3 Numerical Integration

Integrating constraints over line segments is a straightforward process since we have closed form solutions for one-dimensional integrals in Equation (5.12). Unfortunately, we have not been able to find closed form solutions of the two-dimensional integrals that are required for integration over two-dimensional domains like triangles and parametric patches.

In the above derivation, we use the closed form solutions for the one-dimensional integrals to calculate the inner layer of the two-dimensional integrals but have to leave the outer

layer in the equations above since we don't have closed form solutions for the integrals like

$$\begin{aligned} Den(\boldsymbol{x}) &= \sum_{i=1}^{N} C_i^2 \, 2 \, Area_i \int_0^1 \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2} \mathrm{d}t \, \mathrm{d}s \\ &= \sum_{i=1}^{N} C_i^2 \, 2 \, Area_i \int_0^1 I_{i0}(s) \, \mathrm{d}s \\ &= \sum_{i=1}^{N} C_i^2 \, 2 \, Area_i \int_0^1 \left(\frac{-\frac{A_i(s)\sqrt{B_i(s)}}{A_i(s)^2 + B_i(s)} + \frac{\sqrt{B_i(s)}\,(1 + A_i(s) - s)}{B_i(s) + (1 + A_i(s) - s)^2}}{2\, B_i(s)^{3/2}} \right. \\ &\quad \left. - \frac{\arctan\left[\frac{A_i(s)}{\sqrt{B_i(s)}}\right] - \arctan\left[\frac{1 + A_i(s) - s}{\sqrt{B_i(s)}}\right]}{2\, B_i(s)^{3/2}} \right) \mathrm{d}s \end{aligned} \tag{5.93}$$

The obvious solution to this problem would simply approximate the integrals using a standard quadrature method. Unfortunately, this solution performs poorly for the same reason that scattering point constraints does: unless the distance between quadrature points is significantly less than ϵ the resulting surface will have dimples and bumps. The culprit responsible for this behavior is the weighting function. Its singularity, or near singularity, at zero, causes severe problems for standard quadrature schemes. These difficulties extend to Monte-Carlo schemes, which explains the problems encountered with scattered points. The method we use is aware of the singular nature of the weighting function and it accounts for that contribution without under-weighting the contribution from the rest of the triangle.

Let $\boldsymbol{m}$ be the point in Ω_i that is closest to the evaluation point, $\boldsymbol{x}$. (See Figure 5.6.) If this point is on the interior of Ω_k, we split the triangle into three triangles each of which has $\boldsymbol{m}$ as one of its vertices. If the point lies on an edge, the triangle is split into two triangles. If the point lies on an existing vertex, the triangle is not split. The integral over the original triangle is the sum of integrals over each of these sub-triangles. Each sub-triangle has $\boldsymbol{m}$ as one of its vertices, and the other two vertices are denoted $\boldsymbol{v}^+$ and $\boldsymbol{v}^-$ such that $w(\boldsymbol{x}, \boldsymbol{v}^+) \geq w(\boldsymbol{x}, \boldsymbol{v}^-)$.

To compute the sub-triangle area integral we separate it into two successive one-dimensional integrals as shown in Figure 5.6. The outer one integrates s from 0 to 1 along the edge from $\boldsymbol{m}$ to $\boldsymbol{v}^-$ using a special numerical quadrature rule. The inner one

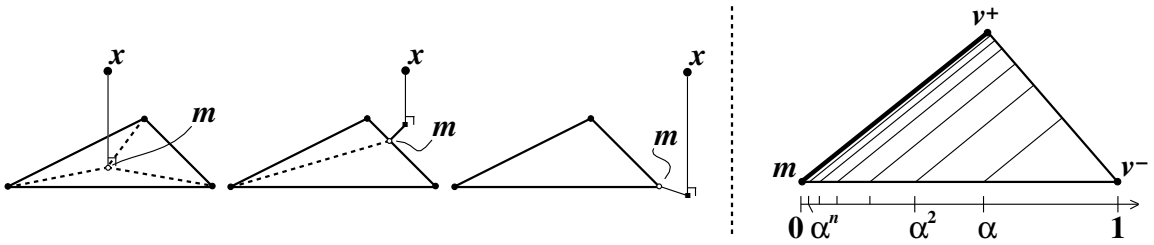

Figure 5.6. The quadrature scheme used over a triangle.

integrates t from 0 to $1 - s$ along the barycentric iso-lines that are parallel to the edge from $\boldsymbol{m}$ to $\boldsymbol{v}^+$, using the one-dimensional analytical solution.

A Simple Quadrature Scheme

With triangle splitting, we can make sure that the singularity or near-singularity is captured if the quadrature scheme always puts a quadrature point at $\boldsymbol{m}$, where the weight function has the largest value. Capturing the singularity point $\boldsymbol{m}$ dramatically reduces the numerical error for a standard quadrature scheme. However, a special quadrature rule can do better.

Wee are not solving a general numerical integration problem, but focusing on a particular weight function that has a known behavior with a maximum value at $\boldsymbol{m}$ and vanishing along the edge from $\boldsymbol{m}$ to $\boldsymbol{v}^-$, and therefore more accurate numerical integration quadrature scheme can use the Newton-Cotes trapezoidal rule with irregularly spaced samples. If the edge from $\boldsymbol{m}$ to $\boldsymbol{v}^-$ is parameterized from zero to one with zero corresponding to $\boldsymbol{m}$, the samples occur at $0, \alpha^n, \ldots, \alpha^1, \alpha^0$. We arbitrarily use $\alpha = 2/3$, and n is proportional to the logarithm of the edge length. The integral should be appropriately scaled by the sub-triangle area. This scheme captures the behavior near the potentially singular location, $\boldsymbol{m}$, without neglecting the rest of the triangle.

A Better Quadrature Scheme

The previous simple quadrature scheme is based on the assumption that the weight function has a fall-off behavior. Since we know the exact formula of the weight function, we can achieve a more elegant solution by transforming the integrants with singularities into flat ones using change of coordinates.

When we map the function we want to integrate onto a different space and use a compensating multiplicative factor, we want to make sure that the new function satisfies two properties:

- Accurate: The area under the new function is the same as the area under the original, badly behaved function.
- Well-behaved: the new function should be roughly constant over the domain, and thus can be integrated much more accurately than a function with a singularity.

To achieve such a mapping, for simplicity, let's first look at the 1D case. Suppose we want to integrate,

$$\int_{u_0}^{u_1} g(u)\mathrm{d}u \tag{5.94}$$

where g(u) has a near-singularity. Most of the area under the curve is near the near-singularity, and that's exactly the spot where most interpolation methods will be least accurate. So we'll transform the space to get rid of the near-singularity.

To achieve this, we define another function in a different parameter space,

$$u = u(j), \;\; j \in [j_0, j_1] \tag{5.95}$$

where $j_0 = u^{-1}(u_0)$ and $j_1 = u^{-1}(u_1)$. Our integral can be written,

$$\begin{aligned} \int_{u_0}^{u_1} g(u)\mathrm{d}u &= \int_{u^{-1}(u_0)}^{u^{-1}(u_1)} g(u(j))\mathrm{d}u(j) && (5.96)\\ &= \int_{j_0}^{j_1} g(u(j))\, u'(j)\mathrm{d}j && (5.97)\\ &= \int_{j_0}^{j_1} h(j)\mathrm{d}j && (5.98) \end{aligned}$$

We want $h(j) = g(u(j))u'(j)$ to be the new function and from the above equation we understand that the area under $h(j)$ is equal to the area under our original function $g(u)$. The question is, how do we choose $u(j)$ so the new function $h(j)$ satisfies the second property to be well-behaved and easy to integrate accurately? We want $h(j)$ to be roughly constant over the domain. One idea is to make $h(j)$ approximately equal to one.

$$h(j) = g(u(j))\, u'(j) \approx 1 \tag{5.99}$$

or,

$$u'(j) \approx \frac{1}{g(u(j))} \tag{5.100}$$

If we could solve the differential equation of $u'(j) = \frac{1}{g(u(j))}$, we'd probably be able to integrate $g(u)$ analytically in the first place, and we wouldn't be going to use numerical integration at all and wouldn't have all these troubles. Unfortunately, for a general form of $g(u)$, there isn't a analytical solution. However, for interpolation to work well, we don't need $h(j)$ to be exactly one. We just need it not to have near-singularities or other bad behavior. So instead of using the exact value of $g(u)$, we can approximate $g(u)$ by a function that can capture the near-singularities of $g(u)$ and has a closed-form integral. Consider what we use for the weight function that causes the near-singularities,

$$w(\boldsymbol{x}, \boldsymbol{p}) = \frac{1}{\|\boldsymbol{x} - \boldsymbol{p}\|^2 + \epsilon^2} \tag{5.101}$$

we can choose an approximation of $g(u)$ as

$$ga(u) = \frac{1}{c_2\, u^2 + c_1\, u + c_0} \tag{5.102}$$

where c_0, c_1 and c_2 are unknown coefficients that we need to solve to make the function $ga(u)$ capture the singularities of $g(u)$. To get c_0, c_1 and c_2, we sample three points on $g(u)$ around the near-singularity (as shown in Figure 5.7). u_m is the near-singularity point with the maximum function value $g(u_m)$. u_- and u_+ are two points near u_m with small offsets.

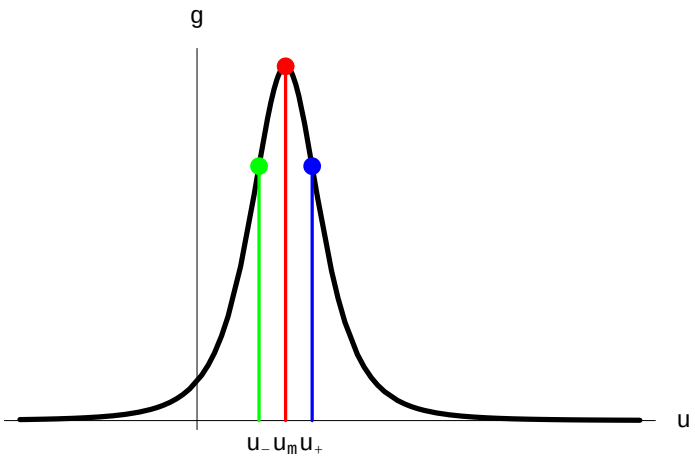

Figure 5.7. To achieve a good approximation of $g(u)$, we sample three points around the near-singularity at u_-, u_m and u_+.

With the three samples, we can solve the following linear system to get c_0, c_1 and c_2.

$$ga(u_-) = \frac{1}{c_2u_-^2 + c_1u_- + c_0} = g(u_-) \tag{5.103}$$

$$ga(u_m) = \frac{1}{c_2u_m^2 + c_1u_m + c_0} = g(u_m) \tag{5.104}$$

$$ga(u_+) = \frac{1}{c_2u_+^2 + c_1u_+ + c_0} = g(u_+) \tag{5.105}$$

or

$$\begin{bmatrix} u_-^2 & u_- & 1 \\ u_m^2 & u_m & 1 \\ u_+^2 & u_+ & 1 \end{bmatrix} \begin{bmatrix} c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} \frac{1}{g(u_-)} \\ \frac{1}{g(u_m)} \\ \frac{1}{g(u_+)} \end{bmatrix} \tag{5.106}$$

Since we have,

$$u'(j) = \frac{1}{ga(u(j))} \approx \frac{1}{g(u(j))} \tag{5.107}$$

so,

$$\frac{\mathrm{d}u(j)}{\mathrm{d}j} = \frac{1}{ga(u(j))} \tag{5.108}$$

$$ga(u(j))\,\mathrm{d}u(j) = \mathrm{d}j \tag{5.109}$$

$$\int ga(u(j))\,\mathrm{d}u(j) = \int \mathrm{d}j \tag{5.110}$$

$$\int \frac{1}{c_2u^2 + c_1u + c_0}\mathrm{d}u = j \tag{5.111}$$

Since there is a closed-form integral for $ga(u)$, we can solve j as a function of u.

$$j(u) = \int \frac{1}{c_2 u^2 + c_1 u + c_0} \mathrm{d}u \tag{5.112}$$

$$= \frac{2 \arctan\left[\frac{c_1 + 2\, u\, c_2}{\sqrt{-c_1^2 + 4\, c_0\, c_2}}\right]}{\sqrt{-c_1^2 + 4\, c_0\, c_2}} \tag{5.113}$$

Solving the inverse function, we get u as a function of j,

$$u(j) = \frac{-c_1 + \sqrt{-c_1^2 + 4\, c_0\, c_2}\, \tan\left[\frac{1}{2}\, j\, \sqrt{-c_1^2 + 4\, c_0\, c_2}\right]}{2\, c_2} \tag{5.114}$$

and

$$u'(j) = \frac{\sec\left[\frac{1}{2}\, j\, \sqrt{-c_1^2 + 4\, c_0\, c_2}\right]^2 (-c_1^2 + 4\, c_0\, c_2)}{4\, c_2} \tag{5.115}$$

Based on $h(j) = g(u(j))\, u'(j)$, we have

$$\int_{u_0}^{u_1} g(u)\mathrm{d}u = \int_{j_0}^{j_1} h(j)\mathrm{d}j$$

$$= \int_{j_0}^{j_1} g\left(\frac{-c_1 + \sqrt{-c_1^2 + 4\, c_0\, c_2}\, \tan\left[\frac{1}{2}\, j\, \sqrt{-c_1^2 + 4\, c_0\, c_2}\right]}{2\, c_2}\right)$$

$$\frac{\sec\left[\frac{1}{2}\, j\, \sqrt{-c_1^2 + 4\, c_0\, c_2}\right]^2 (-c_1^2 + 4\, c_0\, c_2)}{4\, c_2} \mathrm{d}j \tag{5.116}$$

Now let's look at an example of $g(u)$ with a particular form. Assume,

$$g(u) = \frac{1}{((u-a)^2 + b)^2} \tag{5.117}$$

where a controls where the near-singularity (the maximum function value) is and b controls how bad the near-singularity (or how big the maximum function value) is. Figure 5.8 shows the plots using $a = 1.0$ and $b = 0.01$ with $g(u)$, $ga(u)$ and $h(j)$ as,

$$g(u) = \frac{1}{\left(0.01 + (-1+u)^2\right)^2}$$

$$ga(u) = \frac{1}{0.0202 - 0.0402\, u + 0.0201\, u^2}$$

$$h(j) = \frac{0.0001 \sec[0.00141774\, j]^2}{\left(0.01 + (-1 + 1.5243 \times 10^{-9}(6.56041 \times 10^8 + 4.62735 \times 10^7 \tan[0.00141774\, j]))^2\right)^2}$$

From Figure 5.8, we can see that $g(u)$ is hard to integrate with a near-singularity at $u = 1.0$ whereas $h(j)$ is relatively much easier to integrate with a flat shape. If we use the

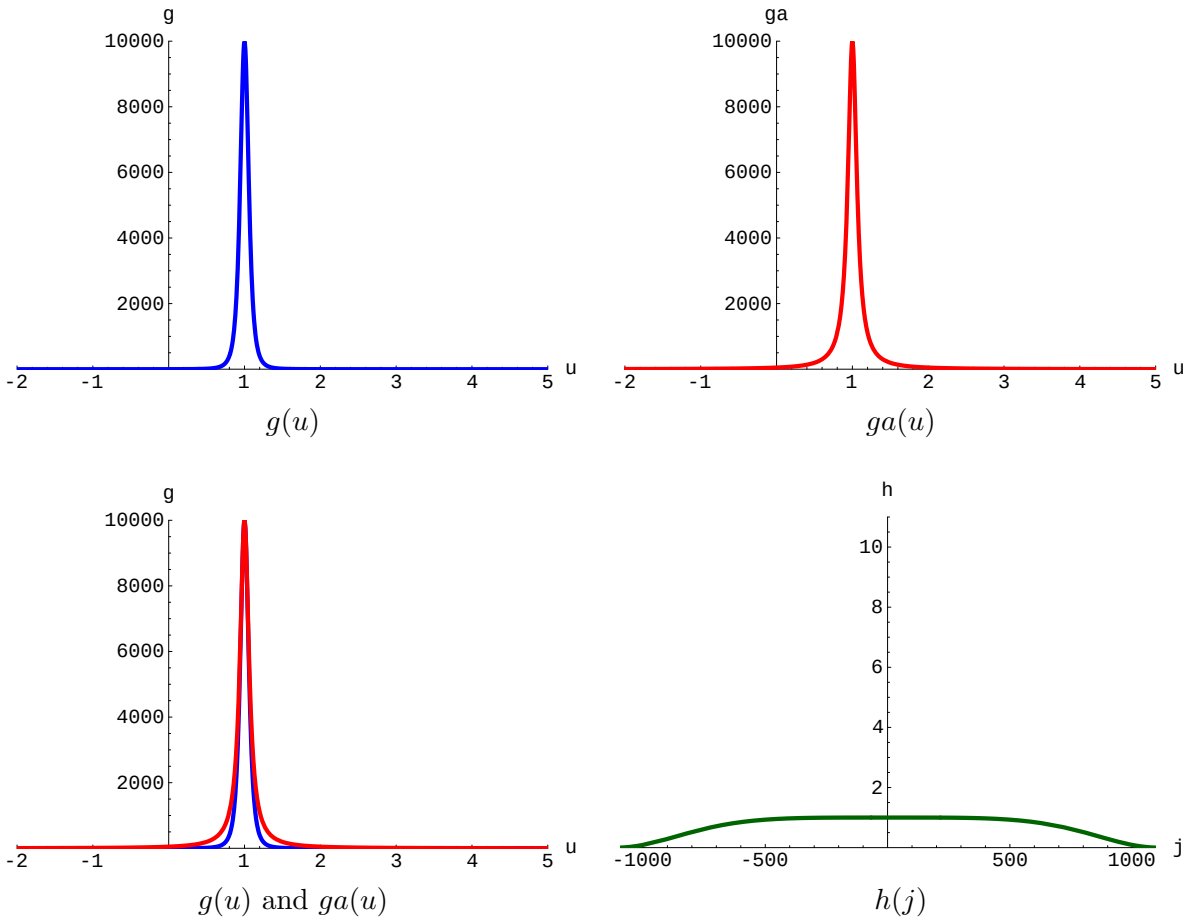

Figure 5.8. Plot of $g(u)$, $ga(u)$ and $h(j)$. Blue curve shows $g(u)$. Red curve shows $ga(u)$. Dark green curve shows $h(j)$.

most naive quadrature rule with uniform distribution of quadrature points on $g(u)$ as shown in Figure 5.9, we can see the numerical integration will have a huge numerical error since the near-singularity is not captured by any quadrature points. However, if we apply the same number of quadrature points on $h(j)$, we can achieve a much more accurate numerical integration since all parts between adjacent quadrature points are flat.

If we map those quadrature points on $h(j)$ back to $g(u)$'s space with the function $u(j)$ as shown in Figure 5.10, we have an interesting observation that most quadrature points automatically gather around the near-singularity point. This explains why the new scheme with transforming singularities can accurately integrate the original function in a much more efficient way.

Table 5.4.3 shows the numerical integration results for directly integrating $g(u)$ and

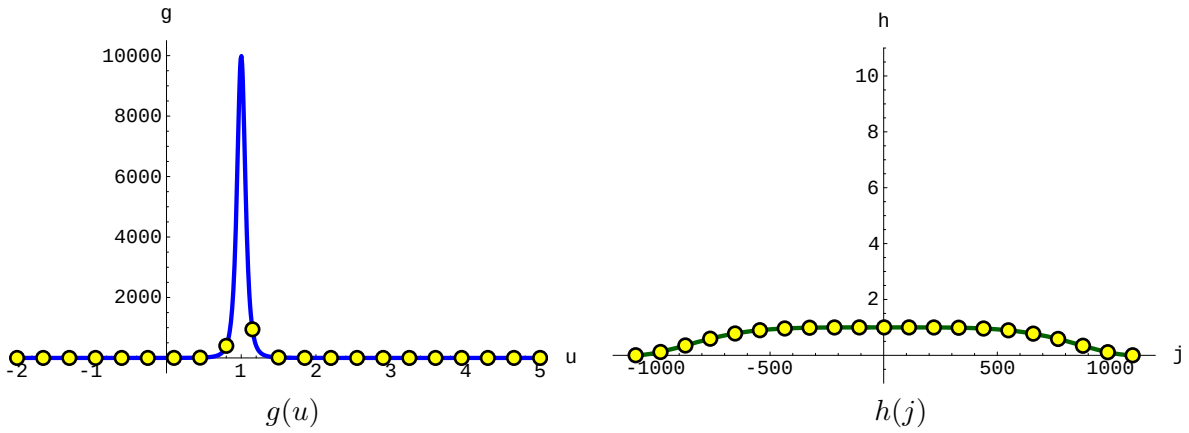

Figure 5.9. Applying 20 uniform distributed quadrature points on both $g(u)$ and $h(j)$. The near-singularity point in $g(u)$ is not captured by any quadrature points so that the numerical integration for $g(u)$ has a huge numerical error. $h(j)$ can be integrated by the same number of quadrature points more accurately since parts in between quadrature points on $h(j)$ are all flat.

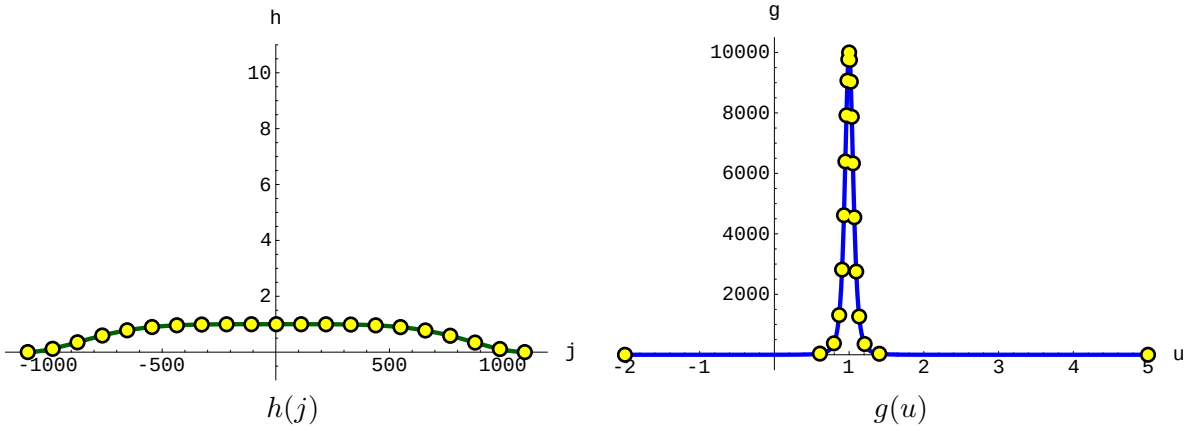

Figure 5.10. Map the same 20 uniform distributed quadrature points on $h(j)$ back to $g(u)$'s space with most quadrature points automatically gather around the near-singularity point.

integrating the transformed function $h(j)$. The error analysis tells us that integrating the transformed function $h(j)$ provides us a much higher numerical accuracy, even with a very small number of quadrature points. The true value of $g(u)$ comes from the closed-form integral of $g(u) = \frac{1}{((u-a)^2+b)^2}$ and all evaluation uses 8-digit precision. We use quasi Monte Carlo methods from MATHEMATICA for numerical integrations.

5.4.4 Integration Over Parametric Patches

With this better numerical integration quadrature scheme, we can integrate the moving least squares constraints over triangle meshes both accurately and efficiently. Now let's extend it to handle parametric patches.

A naive way to fit an interpolating or approximating implicit function with moving least squares to a parametric surface is to first tessellate the surface into polygons and then apply the technique from the above discussion to the tessellation. There are two drawbacks of this naive solution. First, the fit function only interpolates or approximates the polygons of the tessellation and is not necessarily close enough to the original parametric surface. Secondly, the interpolating surface of the tessellation will introduce sharp edges and corners corresponding to polygonal faces and the approximating surface, although it can be smooth, might be arbitrarily far way from the original parametric surface.

To achieve an accurate interpolation or approximation of a parametric surface, what we would like to do is to scatter an infinite number of points continuously across the parametric domain.

Given a parametric surface Ω

$$\boldsymbol{p}(u,v) = [X(u,v), Y(u,v), Z(u,v)] \tag{5.118}$$

the moving least squares formulation from the summation gives

$$f(\boldsymbol{x}) = c = \frac{\int_\Omega w(\boldsymbol{x}, \boldsymbol{p}(u,v))\, S(\boldsymbol{x}, \boldsymbol{p}(u,v))\, d\boldsymbol{p}}{\int_\Omega w(\boldsymbol{x}, \boldsymbol{p}(u,v))\, d\boldsymbol{p}} \tag{5.119}$$

If the parametric surface consists of K patches, let Ω_k, $k \in [1 \ldots K]$, be the kth patch in the (u,v) domain D_k, we have

$$f(\boldsymbol{x}) = c = \frac{\sum_{k=1}^{K} \int_{\Omega_k} w(\boldsymbol{x}, \boldsymbol{p}(u,v))\, S_i(\boldsymbol{x}, \boldsymbol{p}(u,v))\, d\boldsymbol{p}}{\sum_{k=1}^{K} \int_{\Omega_k} w(\boldsymbol{x}, \boldsymbol{p}(u,v)\, d\boldsymbol{p}} \tag{5.120}$$

or

$$f(\boldsymbol{x}) = c = \frac{\sum_{k=1}^{K} \int_0^1 \int_0^1 w(\boldsymbol{x}, \boldsymbol{p}(u,v))\, S_i(\boldsymbol{x}, u, v)\, J(u,v)\, du\, dv}{\sum_{k=1}^{K} \int_0^1 \int_0^1 w(\boldsymbol{x}, \boldsymbol{p}(u,v)\, J(u,v)\, du\, dv} \tag{5.121}$$

Where $J(u,v)$ is the Jacobi and

$$\begin{aligned} S(\boldsymbol{x},u,v) &= S(\boldsymbol{x},\boldsymbol{p}(u,v)) \\ &= \phi\boldsymbol{p} - \boldsymbol{n}(u,v)^{\mathsf{T}} \cdot \boldsymbol{p}(u,v) + \boldsymbol{n}(u,v)^{\mathsf{T}} \cdot \boldsymbol{x} \end{aligned}$$

Where $\boldsymbol{n}(u,v)$ is the surface normal at the point (u,v). One major difference between the polygonal case and the parametric patch case is that the normal vector on a single element is no longer constant on a parametric patch.

To compute the numerical integrals in the above equations, for a given patch Ω_k, we first get the linear approximation $\overline{\Omega_k}$ by tessellating Ω_k into smaller triangular regions. The corresponding (u,v) domain D_k has the same tessellation and the integral over the whole domain is the sum of integrals over all sub-domains.

$\overline{\Omega_k}$ is a good approximation for Ω_k when the parametric surface Ω_{k_i} ($i \in [1 \ldots N_k)$ in each corresponding triangular domain is almost flat. In this case, let $m(\boldsymbol{u}_m)$ ($\boldsymbol{u}_m = (u_m, v_m)$ be the closest point from the evaluation point $\boldsymbol{x}$ to the parametric surface Ω_{k_i} in one triangular domain D_{k_i} and $\overline{m(\boldsymbol{u}_{\overline{m}})}$ be the closest point from the evaluation point to the linear approximation $\overline{\Omega_{k_i}}$, we know that

$$\|\boldsymbol{u}_m - \boldsymbol{u}_{\overline{m}}\| \approx 0 \tag{5.122}$$

We use $m(\boldsymbol{u}_{\overline{m}})$ as an initial guess of the closest point from the evaluation point to the parametric surface. From this guess point, we search for a maximum integrant and its near offset points. Based on the algorithm given in the previous section, we transform the integrant with singularities into a relatively flat curve and numerically integrate along both s and t. Since $(u_{\overline{m}}, v_{\overline{m}})$ is a very good initial guess, the searching procedure is expected to converge quickly.

To compute the sub-domain integral we split the triangle region into three sub-triangles each of which has $\boldsymbol{u}_{\overline{m}}$ as one of its vertices. To compute the sub-triangle area integral we separate it into two successive one-dimensional integrals. In the case for integrating constraints over triangles, the inner integral has its closed form solution and the outer integral is computed by numerical integrations. Unfortunately, for a general parametric

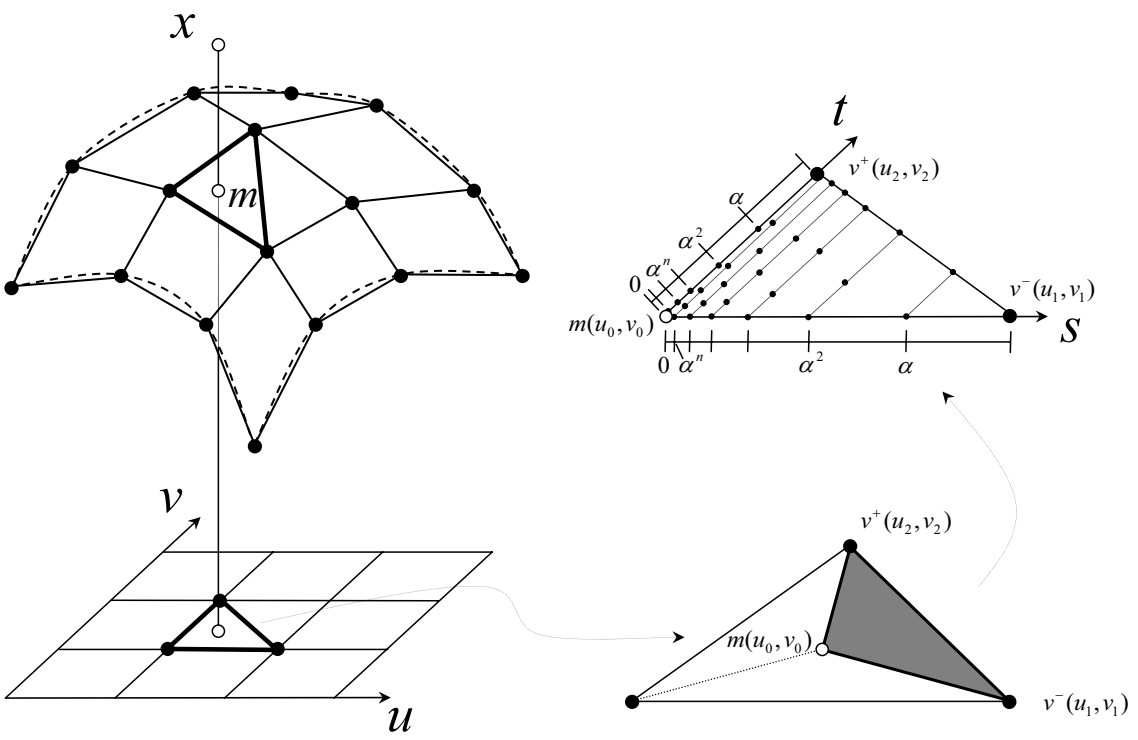

Figure 5.11. Tessellate the parametric surface into triangular regions. Find the closet point from the evaluation point to the triangle and split the triangle into three sub-triangles. Use the closet point as an initial guess for searching the maximum integrant value, transform singularities, and numerically integrate over both s and t directions for flattened curves.

surface, even the inner integral does not have a closed form solution. We have to use numerical integrations for both inner and outer integrals. We apply the powerful numerical integration quadrature scheme with transformed singularities on both s and t directions as shown in Figure 5.11.

Equation (5.121) turns to be

$$f(\boldsymbol{x}) = c = \frac{\sum_{k=1}^{K}\sum_{l=1}^{N_k}\int_0^1\int_0^{1-s} w(\boldsymbol{x},\boldsymbol{p}(u(s,t),v(s,t)))\,S_i(\boldsymbol{x})\,J(u(s,t),v(s,t))\,dt\,ds}{\sum_{k=1}^{K}\sum_{l=1}^{N_k}\int_0^1\int_0^{1-s} w(\boldsymbol{x},\boldsymbol{p}(u(s,t),v(s,t))\,J(u(s,t),v(s,t))\,dt\,ds} \tag{5.123}$$

where

$$u(s,t) = u_0 + (u_1 - u_0)\,s + (u_2 - u_0)\,t \tag{5.124}$$

$$v(s,t) = v_0 + (v_1 - v_0)\,s + (v_2 - v_0)\,t \tag{5.125}$$

and

$$
\begin{aligned}
J(u(s,t),v(s,t)) &= \|\boldsymbol{J_a} \times \boldsymbol{J_b}\| && (5.126)\\
\boldsymbol{J_a} &= \left[\frac{\partial X}{\partial u}\frac{\partial u}{\partial s} + \frac{\partial X}{\partial v}\frac{\partial v}{\partial s}\,,\ \frac{\partial Y}{\partial u}\frac{\partial u}{\partial s} + \frac{\partial Y}{\partial v}\frac{\partial v}{\partial s}\,,\ \frac{\partial Z}{\partial u}\frac{\partial u}{\partial s} + \frac{\partial Z}{\partial v}\frac{\partial v}{\partial s} \right]\\
&= \boldsymbol{p}_u\,(u1-u0)\ +\ \boldsymbol{p}_v\,(v1-v0) && (5.127)\\
\boldsymbol{J_b} &= \left[\frac{\partial X}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial X}{\partial v}\frac{\partial v}{\partial t}\,,\ \frac{\partial Y}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial Y}{\partial v}\frac{\partial v}{\partial t}\,,\ \frac{\partial Z}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial Z}{\partial v}\frac{\partial v}{\partial t} \right]\\
&= \boldsymbol{p}_u\,(u2-u0)\ +\ \boldsymbol{p}_v\,(v2-v0) && (5.128)
\end{aligned}
$$

For a bi-cubic B-Spline patch with control vertices $\boldsymbol{V}_{ij}$, we have

$$
\boldsymbol{p}(u,v) = [b_0(v)\ b_1(v)\ b_2(v)\ b_3(v)] \begin{bmatrix} \boldsymbol{V}_{00} & \boldsymbol{V}_{01} & \boldsymbol{V}_{02} & \boldsymbol{V}_{03} \\ \boldsymbol{V}_{10} & \boldsymbol{V}_{11} & \boldsymbol{V}_{12} & \boldsymbol{V}_{13} \\ \boldsymbol{V}_{20} & \boldsymbol{V}_{21} & \boldsymbol{V}_{22} & \boldsymbol{V}_{23} \\ \boldsymbol{V}_{30} & \boldsymbol{V}_{31} & \boldsymbol{V}_{32} & \boldsymbol{V}_{33} \end{bmatrix} \begin{bmatrix} b_0(u) \\ b_1(u) \\ b_2(u) \\ b_3(u) \end{bmatrix} \qquad (5.129)
$$

and

$$
\begin{aligned}
b_0(u) &= \frac{1}{6}(1 - 3u + 3u^2 - u^3) && (5.130)\\
b_1(u) &= \frac{1}{6}(4 - 6u^2 + 3u^3) && (5.131)\\
b_2(u) &= \frac{1}{6}(1 + 3u + 3u^2 - 3u^3) && (5.132)\\
b_3(u) &= \frac{1}{6}u^3 && (5.133)
\end{aligned}
$$

For a bi-cubic Bezier patch, the only difference is in the basis function,

$$
\begin{aligned}
b_0(u) &= (1-u)^3 && (5.134)\\
b_1(u) &= 3u(1-u)^2 && (5.135)\\
b_2(u) &= 3u^2(1-u) && (5.136)\\
b_3(u) &= u^3 && (5.137)
\end{aligned}
$$

The surface normal can be achieved by calculating the spatial gradient of $\boldsymbol{p}(u,v)$,

$$\begin{aligned}
\boldsymbol{p}_u &= \frac{\partial \boldsymbol{p}(u,v)}{\partial u} \\
&= [b_0(v)\ b_1(v)\ b_2(v)\ b_3(v)] \begin{bmatrix} \boldsymbol{V}_{00} & \boldsymbol{V}_{01} & \boldsymbol{V}_{02} & \boldsymbol{V}_{03} \\ \boldsymbol{V}_{10} & \boldsymbol{V}_{11} & \boldsymbol{V}_{12} & \boldsymbol{V}_{13} \\ \boldsymbol{V}_{20} & \boldsymbol{V}_{21} & \boldsymbol{V}_{22} & \boldsymbol{V}_{23} \\ \boldsymbol{V}_{30} & \boldsymbol{V}_{31} & \boldsymbol{V}_{32} & \boldsymbol{V}_{33} \end{bmatrix} \begin{bmatrix} b_0'(u) \\ b_1'(u) \\ b_2'(u) \\ b_3'(u) \end{bmatrix} \qquad (5.138) \\
\boldsymbol{p}_v &= \frac{\partial \boldsymbol{p}(u,v)}{\partial v} \\
&= \left[b_0'(v)\ b_1'(v)\ b_2'(v)\ b_3'(v)\right] \begin{bmatrix} \boldsymbol{V}_{00} & \boldsymbol{V}_{01} & \boldsymbol{V}_{02} & \boldsymbol{V}_{03} \\ \boldsymbol{V}_{10} & \boldsymbol{V}_{11} & \boldsymbol{V}_{12} & \boldsymbol{V}_{13} \\ \boldsymbol{V}_{20} & \boldsymbol{V}_{21} & \boldsymbol{V}_{22} & \boldsymbol{V}_{23} \\ \boldsymbol{V}_{30} & \boldsymbol{V}_{31} & \boldsymbol{V}_{32} & \boldsymbol{V}_{33} \end{bmatrix} \begin{bmatrix} b_0(u) \\ b_1(u) \\ b_2(u) \\ b_3(u) \end{bmatrix} \qquad (5.139)
\end{aligned}$$

and

$$\boldsymbol{n} = \frac{\boldsymbol{p}_u \times \boldsymbol{p}_v}{\|\boldsymbol{p}_u \times \boldsymbol{p}_v\|} \qquad (5.140)$$

For more complicated parametric patches like NURBS surfaces, we can follow the exact same algorithm but plug-in the evaluation procedure for the surface point $\boldsymbol{p}(u,v)$ and the surface normal $\boldsymbol{n}(u,v)$ with appropriate parametric formula. For Catmull-Clark subdivision surfaces, we can follow Stam (1998) that gives an exact evaluation scheme to get the surface point $\boldsymbol{p}(u,v)$ and the surface normal $\boldsymbol{n}(u,v)$.

$g(u)$	$\int_{u0}^{u1} g(u)\mathrm{d}u$	Max # Pts	Numerical Integration of $g(u)$		
			Integral	Error	%
$\frac{1}{((u-1.0)^2+0.01)^2}$	1570.7788	5	22.960611	1547.818182	98.53827%
		10	671.70054	899.0782487	57.23774%
		100	1568.3085	2.470319146	0.15727%
$\frac{1}{((u-1.0)^2+10^{-4})^2}$	1.570796×10^6	5	24.684638	1.5707716×10^6	99.99843%
		10	1380.0031	1.5694163×10^6	99.91215%
		100	150133.61	1.4206627×10^6	90.44220%
$\frac{1}{((u-1.0)^2+10^{-6})^2}$	1.570796×10^9	5	24.702897	1.5707963×10^9	100.00%
		10	1392.0949	1.5707949×10^9	99.9999%
		100	197164.41	1.5705991×10^9	99.98745%
$g(u)$	$\int_{u0}^{u1} g(u)\mathrm{d}u$	Max # Pts	Numerical Integration of $h(j)$		
			Integral	Error	%
$\frac{1}{((u-1.0)^2+0.01)^2}$	1570.778	5	1581.7585	10.97966087	0.69899%
		10	1571.8000	1.021243814	0.06502%
		100	1570.7881	0.0093311179	0.00059%
$\frac{1}{((u-1.0)^2+10^{-4})^2}$	1.570796×10^6	5	1.5774426×10^6	6646.280310	0.42312%
		10	1.5709031×10^6	106.82207	0.0068%
		100	1.5707973×10^6	0.95774363	0.00006%
$\frac{1}{((u-1.0)^2+10^{-6})^2}$	1.570796×10^9	5	1.5769831×10^9	6.1868039×10^6	0.39386%
		10	1.57081×10^9	12414.733	0.0007%
		100	1.5707964×10^9	96.003555	0.0000%

Table 5.1. Numerical integration with quasi Monte Carlo methods from MATHEMATICA. Comparisons between different integrants with near-singularities from small to large. Experiments with three different number of quadrature points $(5, 10, 100)$. Error is the absolute difference between the numerical integration value and the true value which comes from the close-form integral of $g(u)$. Column with symbol % shows the percentage error rate for each case. The top part shows the numerical interation of $g(u)$ and the bottom part shows the numerical integration of $h(j)$.

Chapter 6

Implementation Details

6.1 Interpolation and Approximation

When the weighting function parameter, ϵ, is set to zero, the moving least-squares function will exactly interpolate constraint values. If we follow the general approach described in Turk and O'Brien (1999) and Ohtake et al. (2003) of constraining the function to be zero at input elements (points, line segments, polygons or parametric patches), supplying

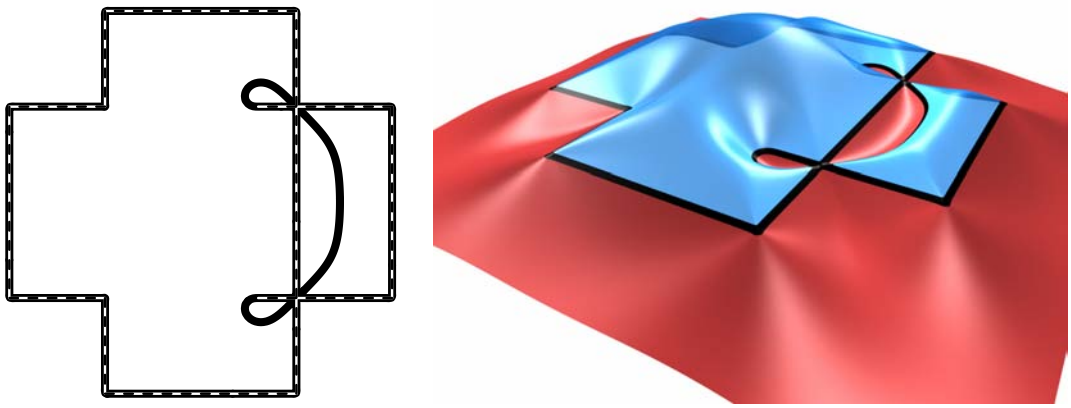

Figure 6.1. A two-dimensional example shows interpolating results. The left image shows input constraints as dotted lines and the contour as a solid line. The right image shows the resulting function as a height-field.

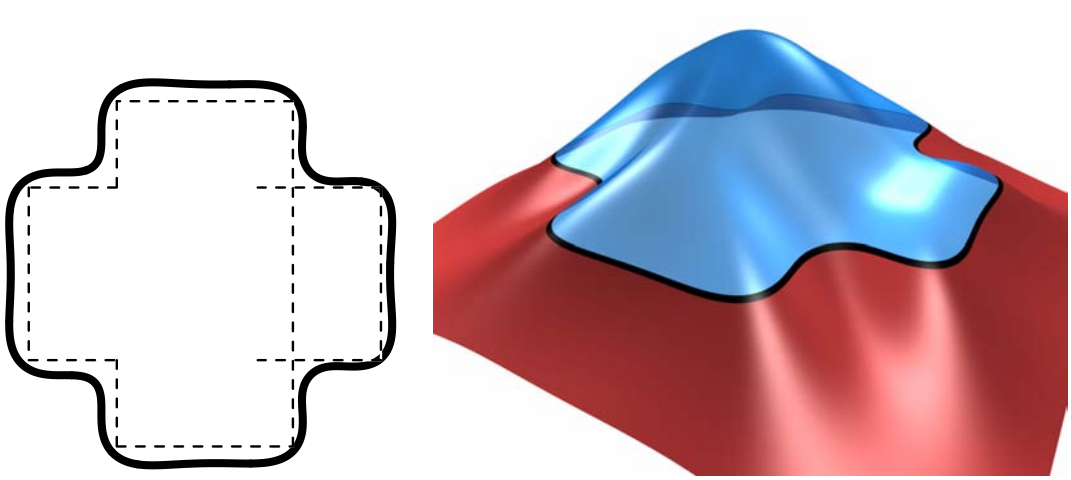

Figure 6.2. A two-dimensional example shows approximating results. The left image shows input constraints as dotted lines and the contour as a solid line. The right image shows the resulting function as a height-field.

appropriate normal constraints for each element, and extracting an iso-surface at zero, then the resulting implicit surface will pass through all of the input elements.

If the input collection of elements contains gaps or holes, then the surface will extend beyond the input elements to generate a closed surface. As with previous methods that accomplish hole filling using some form of implicit surface, there is no guarantee that the results will satisfy any particular criteria. However, we generally find that these extensions close gaps and holes in a useful fashion that produces results similar to what a human might have selected.

If the input surface self-intersects, then the interpolating surface will create some form of saddle at the intersections. This behavior is illustrated for a two-dimensional example in Figure 6.1.

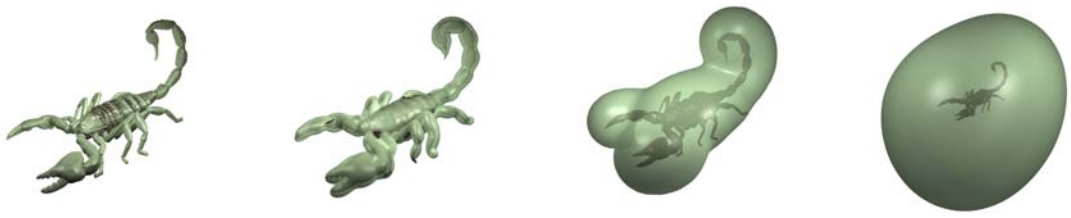

Figure 6.3. Approximate surfaces generated by simply applying different values of ϵ to a polygonal scorpion model. As ϵ is set to larger values, the approximating surface has the tendency to move away from the input data.

Figure 6.4. Approximate surfaces generated by only adjusting the surfaces to average values (no iterative adjustment)

When ϵ is set to a non-zero value the weighting function is no longer singular at zero, and the moving least-squares function interpolates constraint values only approximately. Examination of Equation (3.18) reveals that ϵ has the same units as distance. It corresponds to a feature size parameter: structures smaller than ϵ tend to be smoothed away by the approximation. The approximation behavior is illustrated for a two-dimensional example in Figure 6.2. The self-intersections from the input constraints are smoothed away by picking an appropriate ϵ.

While generating an approximate surface by simply setting ϵ to some non-zero value works well to an extent, it suffers from two problems. The first is that as ϵ is set to larger values, the approximating surface has the tendency to move away from the input data. For example, very large values of ϵ will smooth an object to a simple sphere-like shape, but the sphere radius may be several times the original object's circumradius. The second problem is that we cannot ensure that all the object's original vertices fall inside the implicit surface, and for some applications this guarantee is important. These problems are illustrated in Figure 6.3.

To correct the first problem we simply build a moving least-squares function with the desired ϵ, sample its average value over the surface of the input polygons, and then extract a surface at that iso-value. Although this procedure may at first appear to require substantial extra work, the additional work is actually not particularly significant. The majority of computation is spent extracting the iso-surface, and that task still only needs to be done once.

By adjusting the iso-value we achieve a surface (as shown in Figure 6.4) that, on

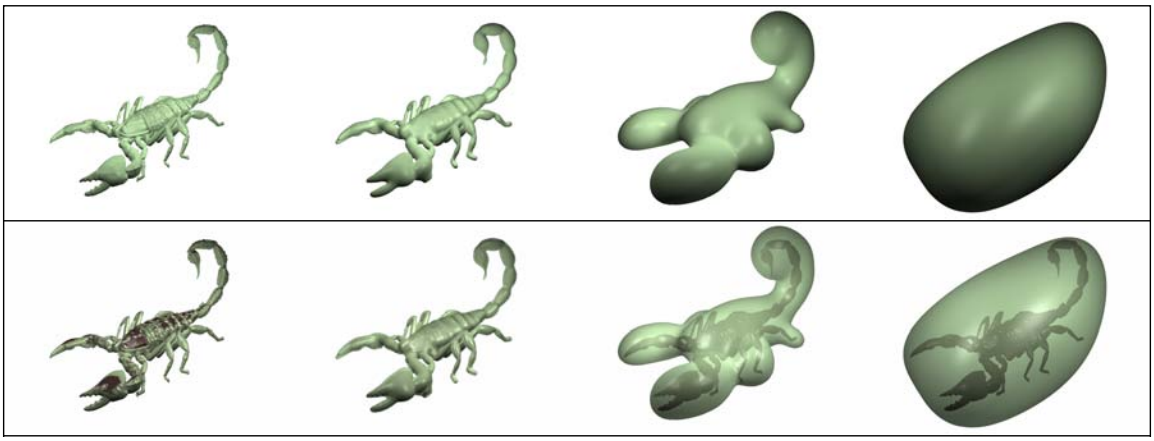

Figure 6.5. Approximate surfaces generated by only applying our iterative adjustment algorithm

average, stays close to the input data, but with this construction we expect that roughly half the original vertices will fall outside the surface. To ensure that original vertices lie inside the surface, we iteratively adjust the ϕ values assigned to the vertices.

Initially, the ϕ values associated with each vertex are all zero and the ϕ associated with each triangle is the constant zero as well. If a vertex, $\boldsymbol{v}$, protrudes outside the iso-surface (*i.e.*, $f(\boldsymbol{v}) > 0$), we adjust its ϕ value by $-\gamma\, f(\boldsymbol{v})$ where γ is an adjustment rate parameter between zero and one. Once the vertices of a triangle have been assigned different values, we linearly interpolate ϕ over the triangle when computing integrals. This adjustment process is done iteratively until no original vertices fall outside the iso-surface. The final surface is guaranteed to enclose all input vertices as illustrated in Figure 6.5. As with adjusting the iso-value, the majority of computation is still spent extracting the iso-surface, and that task still only needs to be done once.

Variations on this iterative procedure for adjusting the ϕ values could also be used to enforce other conditions. For example, it could be used to guarantee that all points are within some set distance of the iso-surface. Conditions could be tested at points other than the initial vertices, and the iterative procedure could also adjust the normal direction or magnitude associated with each constraint.

6.2 Fast Evaluation

Naïve implementation of the moving least-squares function would require work linear in the number of constraints for each function evaluation. For large data sets, this naïve approach is completely infeasible. A similar problem arises with the partition-of-unity method used in Ohtake et al. (2003). They address the problem using a hierarchical evaluation scheme that caches approximations based on local neighborhoods. Because partition-of-unity and moving least-squares methods are essentially equivalent methods, their hierarchical evaluation scheme could be used with our method as well. We have, however, implemented a different evaluation scheme which we describe briefly.

We observe that the primary expense for evaluating the moving least-squares function is the cost of computing the sums and integrals for Equation (5.2) repeated as following,

$$\left(\sum_{k=1}^{K} \boldsymbol{A}_k\right)\ \boldsymbol{c}(\boldsymbol{x}) = \sum_{k=1}^{K} \boldsymbol{a}_k$$

where $\boldsymbol{A}_k$ and $\boldsymbol{a}_k$ are defined by

$$\boldsymbol{A}_k = \int_{\Omega_k} w^2(\boldsymbol{x},\boldsymbol{p})\, \boldsymbol{b}(\boldsymbol{p})\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p})\ \ \mathrm{d}\boldsymbol{p} \quad ,$$

$$\boldsymbol{a}_k = \int_{\Omega_k} w^2(\boldsymbol{x},\boldsymbol{p})\, \boldsymbol{b}(\boldsymbol{p})\, \phi_k\ \ \mathrm{d}\boldsymbol{p} \quad ,$$

Were it not for the weighting function's dependence on $\boldsymbol{x}$, the terms would be constant and the summation would only need to be computed once. For terms that peak near the evaluation point, the weighting function changes rapidly. However, the weight function changes only slowly for far terms. We can approximate groups of the slowly changing far terms by first summing them and then multiplying by their average weight.

For our hierarchical scheme, we first store the input elements in a K-D tree where each element is stored at one of the leaf nodes. We then compute the unweighted integrals, Equations (5.5) and (5.6) repeated as following,

$$\widetilde{\boldsymbol{A}}_k = \int_{\Omega_k} \boldsymbol{b}(\boldsymbol{p})\ \boldsymbol{b}^{\mathsf{T}}(\boldsymbol{p})\ \ \mathrm{d}\boldsymbol{p} \quad ,$$

$$\tilde{\boldsymbol{a}}_k = \int_{\Omega_k} \boldsymbol{b}(\boldsymbol{p})\, \phi_k\ \ \mathrm{d}\boldsymbol{p} \quad .$$

for each element and store them, along with the element's axis-aligned bounding box, in the leaf nodes. The interior nodes store the unweighted sums of their children's integrals/sums, and a bounding box that encloses the union of their children's bounds. We also store an area-weighted "center of mass" for each node.

To evaluate the contribution of a subtree, we test the evaluation point to see if it falls outside the subtree's bounding box by a distance greater than λ times the box's diameter. If it does, we use the sums stored at the subtree's root node with a weight computed using the distance between the node's center of mass and the evaluation point. If the evaluation point is not sufficiently distant, we recursively test the node's children. Only when we find that a leaf node fails our distance test do we need to compute the weighted integral terms for that node.

This scheme was easy to implement using existing K-D tree collision detection code, and it allows us to work with models consisting of several hundred thousand triangles. The user can make a trade-off between speed and accuracy by adjusting λ.

6.2.1 Details of Hierarchical Fast Evaluation over Triangles

With the detailed descriptions about how we achieve integration over triangles as described in the previous chapter, the MLS formula can be approximated by moving the weight function outside the integration so that the integral can be pre-computed and stored for each triangle face to get a faster evaluation.

Function Values

For a set of triangles, the cut function can be approximated as,

$$\begin{aligned} c(\boldsymbol{x}) &= \frac{Num_0(\boldsymbol{x}) + \boldsymbol{x}^{\mathsf{T}} \cdot \boldsymbol{Num}_{\boldsymbol{x}}(\boldsymbol{x})}{Den(\boldsymbol{x})} \\ &\approx \frac{\widetilde{Num_0(\boldsymbol{x})} + \boldsymbol{x}^{\mathsf{T}} \cdot \widetilde{\boldsymbol{Num}_{\boldsymbol{x}}(\boldsymbol{x})}}{\widetilde{Den(\boldsymbol{x})}} \end{aligned} \tag{6.1}$$

where

$$\begin{aligned}
\widetilde{Den(\boldsymbol{x})} &= \sum_i \int_{\Omega_i} \widetilde{w^2(\boldsymbol{x},\boldsymbol{p}_i)}\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i \int_{\Omega_i} w^2(\boldsymbol{x},\boldsymbol{p}_{ic})\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i w^2(\boldsymbol{x},\boldsymbol{p}_{ic}) \int_{\Omega_i} 1\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i w^2(\boldsymbol{x},\boldsymbol{p}_{ic})\, I_1 \qquad (6.2) \\
\widetilde{Num_0(\boldsymbol{x})} &= \sum_i \int_{\Omega_i} \widetilde{w^2(\boldsymbol{x},\boldsymbol{p}_i)}\,\phi_0\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i w^2(\boldsymbol{x},\boldsymbol{p}_{ic}) \int_{\Omega_i} \phi_0\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i w^2(\boldsymbol{x},\boldsymbol{p}_{ic})\, I_{\phi_0} \qquad (6.3) \\
\widetilde{\boldsymbol{N}\boldsymbol{um}_{\boldsymbol{x}}(\boldsymbol{x})} &= \sum_i \boldsymbol{n}_i \int_{\Omega_i} \widetilde{w^2(\boldsymbol{x},\boldsymbol{p}_i)}\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i \boldsymbol{n}_i\, w^2(\boldsymbol{x},\boldsymbol{p}_{ic}) \int_{\Omega_i} 1\,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i \boldsymbol{n}_i\, w^2(\boldsymbol{x},\boldsymbol{p}_{ic})\, I_1 \\
&= \sum_i w^2(\boldsymbol{x},\boldsymbol{p}_{ic})\, \boldsymbol{n}\boldsymbol{I}_{\mathbf{1}} \qquad (6.4)
\end{aligned}$$

and

$$\begin{aligned}
I_1 &= \int_{\Omega_i} 1\,\mathrm{d}\boldsymbol{p}_i \\
&= 2\,Area_i \int_0^1 \int_0^{1-s} 1\,\mathrm{d}t\,\mathrm{d}s \\
&= Area_i \qquad (6.5) \\
I_{\phi_0} &= \int_{\Omega_i} \phi_0\,\mathrm{d}\boldsymbol{p}_i \\
&= \int_{\Omega_i} \phi_{\boldsymbol{p}_i} - \boldsymbol{n}\boldsymbol{i}^{\mathsf{T}} \cdot \boldsymbol{p}_i\,\mathrm{d}\boldsymbol{p}_i \\
&= \int_{\Omega_i} \phi_{\boldsymbol{p}_i}\,\mathrm{d}\boldsymbol{p}_i - \boldsymbol{n}\boldsymbol{i}^{\mathsf{T}} \cdot \int_{\Omega_i} \boldsymbol{p}_i\,\mathrm{d}\boldsymbol{p}_i \\
&= I_{\phi\boldsymbol{p}_i} - \boldsymbol{n}_i^{\mathsf{T}} \cdot \boldsymbol{I}\boldsymbol{p}_{\boldsymbol{i}} \qquad (6.6)
\end{aligned}$$

$$
\begin{aligned}
I_{\phi \boldsymbol{p}_i} &= 2\,Area_i \int_0^1 \int_0^{1-s} \phi_{i0} + (\phi_{i1} - \phi_{i0})\, s + (\phi_{i2} - \phi_{i0})\, t \,\mathrm{d}t \,\mathrm{d}s \\
&= 2\,Area_i \frac{1}{6} (\phi_{i0} + \phi_{i1} + \phi_{i2}) \\
&= Area_i\, \phi_{ic} \qquad (6.7) \\
\boldsymbol{I}_{\boldsymbol{p}_i} &= 2\,Area_i \int_0^1 \int_0^{1-s} \boldsymbol{p}_{i0} + (\boldsymbol{p}_{i1} - \boldsymbol{p}_{i0})\, s + (\boldsymbol{p}_{i2} - \boldsymbol{p}_{i0})\, t \,\mathrm{d}t \,\mathrm{d}s \\
&= 2\,Area_i \frac{1}{6} (\boldsymbol{p}_{i0} + \boldsymbol{p}_{i1} + \boldsymbol{p}_{i2}) \\
&= Area_i\, \boldsymbol{p}_{ic} \qquad (6.8) \\
\boldsymbol{nI_1} &= \boldsymbol{n}_i\, I_1 \qquad (6.9) \\
\phi_{ic} &= \frac{1}{3} (\phi_{i0} + \phi_{i1} + \phi_{i2}) \qquad (6.10) \\
\boldsymbol{p}_{ic} &= \frac{1}{3} (\boldsymbol{p}_{i0} + \boldsymbol{p}_{i1} + \boldsymbol{p}_{i2}) \qquad (6.11)
\end{aligned}
$$

Function Derivatives

The derivative of the cut function can be approximated as,

$$
\begin{aligned}
\boldsymbol{c}'(\boldsymbol{x}) &= \frac{\left(\boldsymbol{Num_0}'(\boldsymbol{x}) + \boldsymbol{Num_x}(\boldsymbol{x}) + \left(\boldsymbol{x}^\mathsf{T} \cdot \boldsymbol{NUM_x}'(\boldsymbol{x})_{3\times 3}\right)^\mathsf{T}\right) Den(\boldsymbol{x})}{Den^2(\boldsymbol{x})} - \\
&\quad \frac{(Num_0(\boldsymbol{x}) + \boldsymbol{x}^\mathsf{T} \cdot \boldsymbol{Num_x}(\boldsymbol{x}))\, \boldsymbol{Den}'(\boldsymbol{x})}{Den^2(\boldsymbol{x})} \\
&\approx \frac{\left(\widetilde{\boldsymbol{Num_0}'}(\boldsymbol{x}) + \widetilde{\boldsymbol{Num_x}}(\boldsymbol{x}) + \left(\boldsymbol{x}^\mathsf{T} \cdot \widetilde{\boldsymbol{NUM_x}'}(\boldsymbol{x})_{3\times 3}\right)^\mathsf{T}\right) \widetilde{Den(\boldsymbol{x})}}{\widetilde{Den(\boldsymbol{x})}^2} - \\
&\quad \frac{\left(\widetilde{Num_0}(\boldsymbol{x}) + \boldsymbol{x}^\mathsf{T} \cdot \widetilde{\boldsymbol{Num_x}}(\boldsymbol{x})\right) \widetilde{\boldsymbol{Den}'}(\boldsymbol{x})}{\widetilde{Den(\boldsymbol{x})}^2} \qquad (6.12)
\end{aligned}
$$

where

$$
\begin{aligned}
\widetilde{\boldsymbol{Den}'}(\boldsymbol{x}) &= 4\,(\widetilde{\boldsymbol{dDen_0}}(\boldsymbol{x}) - \boldsymbol{x}^\mathsf{T} \cdot \widetilde{dDen_x}(\boldsymbol{x})) \qquad (6.13) \\
\widetilde{\boldsymbol{dDen_0}}(\boldsymbol{x}) &= \sum_i w^3(\boldsymbol{x}, \boldsymbol{p}_{ic}) \int_{\Omega_i} \boldsymbol{p}_i \,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i w^3(\boldsymbol{x}, \boldsymbol{p}_{ic}) \boldsymbol{I}_{\boldsymbol{p}_i} \qquad (6.14) \\
\widetilde{dDen_x}(\boldsymbol{x}) &= \sum_i w^3(\boldsymbol{x}, \boldsymbol{p}_{ic}) \int_{\Omega_i} 1 \,\mathrm{d}\boldsymbol{p}_i \\
&= \sum_i w^3(\boldsymbol{x}, \boldsymbol{p}_{ic}) I_1 \qquad (6.15)
\end{aligned}
$$

$$\widetilde{\boldsymbol{Num_0'}}(\boldsymbol{x}) = 4\,(\widetilde{\boldsymbol{dNum_{00}}}(\boldsymbol{x}) - \boldsymbol{x}\,\widetilde{dNum_{0x}}(\boldsymbol{x})) \tag{6.16}$$

$$\begin{aligned}\widetilde{\boldsymbol{dNum_{00}}}(\boldsymbol{x}) &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic}) \int_{\Omega_i} \boldsymbol{p}_i\,\phi_0\,\mathrm{d}\boldsymbol{p}_i \\ &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic}) \boldsymbol{Ip_{\phi_0}} \end{aligned} \tag{6.17}$$

$$\begin{aligned}\widetilde{dNum_{0x}}(\boldsymbol{x}) &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic}) \int_{\Omega_i} \phi_0\,\mathrm{d}\boldsymbol{p}_i \\ &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic}) I_{\phi_0} \end{aligned} \tag{6.18}$$

$$\widetilde{\boldsymbol{NUM'_x}}(\boldsymbol{x})_{3\times3} = 4\,(\widetilde{\boldsymbol{dNUM_{x0}}}(\boldsymbol{x})_{3\times3} - \boldsymbol{x}^\mathsf{T}\cdot(\widetilde{\boldsymbol{dNUM_{xx}}}(\boldsymbol{x}))^\mathsf{T}) \tag{6.19}$$

$$\begin{aligned}\widetilde{\boldsymbol{dNUM_{x0}}}(\boldsymbol{x})_{3\times3} &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic})\,\boldsymbol{n}_i \int_{\Omega_i} \boldsymbol{p}_i\,\mathrm{d}\boldsymbol{p}_i \\ &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic})\,\boldsymbol{n}_i^\mathsf{T}\cdot\boldsymbol{Ip_i}^\mathsf{T} \\ &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic})\,\boldsymbol{nIp}_{i_{3\times3}} \end{aligned} \tag{6.20}$$

$$\begin{aligned}\widetilde{\boldsymbol{dNUM_{xx}}}(\boldsymbol{x}) &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic})\,\boldsymbol{n}_i \int_{\Omega_i} 1\,\mathrm{d}\boldsymbol{p}_i \\ &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic})\,\boldsymbol{n}_i\,I_1 \\ &= \sum_i w^3(\boldsymbol{x},\boldsymbol{p}_{ic})\,\boldsymbol{nI_1} \end{aligned} \tag{6.21}$$

$$\begin{aligned}\boldsymbol{Ip_{\phi_0}} &= 2\,Area_i\,(\int_{Bp_i} \boldsymbol{p}_i\,\phi_{\boldsymbol{p}_i}\,\mathrm{d}\boldsymbol{p}_i - \int_{\Omega_i} \boldsymbol{p}_i\,(\boldsymbol{n}_i^\mathsf{T}\cdot\boldsymbol{p}_i)\,\mathrm{d}\boldsymbol{p}_i) \\ &= \frac{1}{12}\,Area_i\Big((2\phi_{i0}+\phi_{i1}+\phi_{i2}-\boldsymbol{n}_i^\mathsf{T}\cdot(2\phi_{i0}+\phi_{i1}+\phi_{i2}))\,\boldsymbol{p}_{i0} + \\ &\quad (\phi_{i0}+2\phi_{i1}+\phi_{i2}-\boldsymbol{n}_i^\mathsf{T}\cdot(\phi_{i0}+2\phi_{i1}+\phi_{i2}))\,\boldsymbol{p}_{i1} + \\ &\quad (\phi_{i0}+\phi_{i1}+2\phi_{i2}-\boldsymbol{n}_i^\mathsf{T}\cdot(\phi_{i0}+\phi_{i1}+2\phi_{i2}))\,\boldsymbol{p}_{i2}\Big) \end{aligned} \tag{6.22}$$

In summary, the items that need to be pre-computed and stored in each leaf node of the hierarchical structure are I_1, I_{ϕ_0}, $\boldsymbol{nI_1}$, $\boldsymbol{Ip_i}$, $\boldsymbol{Ip_{\phi_0}}$, and $\boldsymbol{nIp}_{i_{3\times3}}$. They also need to be pre-summed for each inner node in a preprocessing step which only needs to be done once. The items that need to be gathered on the fly are $\widetilde{Den(\boldsymbol{x})}$ $\widetilde{Num_0(\boldsymbol{x})}$, $\widetilde{\boldsymbol{Num_x}}(\boldsymbol{x})$ $\widetilde{\boldsymbol{dDen_0}}(\boldsymbol{x})$, $\widetilde{dDen_x}(\boldsymbol{x})$ $\widetilde{\boldsymbol{dNum_{00}}}(\boldsymbol{x})$, $\widetilde{dNum_{0x}}(\boldsymbol{x})$, $\widetilde{\boldsymbol{dNUM_{x0}}}(\boldsymbol{x})_{3\times3}$, and $\widetilde{\boldsymbol{dNUM_{xx}}}(\boldsymbol{x})$

6.3 PreProcessing

Although the methods we have described in previous sections cope well with intersecting geometry and layers of internal structure, it may still be useful to first remove some of these polygons. In particular, our algorithm will happily produce surfaces corresponding to internal structures, even if only an exterior shell was desired. In these cases, we can preprocess the input to remove polygons that are not visible from the exterior using methods such as those in Nooruddin and Turk (2003) and Nooruddin and Turk (2000).

The normal constraints depend on consistently oriented normals. Unfortunately, many polygon models may have normals that randomly point inward or outward. We force normals on topological surfaces to point in a consistent direction. We also orient the normals of any exterior-visible polygon to point outward. If both sides of a triangle are exterior-visible then we set that triangles normal to zero.

Chapter 7

Results

This chapter presents results obtained using the methods I have developed for building interpolating and approximating implicit surfaces from a collection of geometry primitives like polygons and parametric patches. To visualize a two dimensional implicit function, we have shown contour plots as in Figure 4.10 and height fields as in Figure 6.1. In this chapter, we will show results of applying our algorithm to a variety of three dimensional models. Visualizing a three dimensional implicit function is hard and printing the visualization on paper is even harder. So the results shown in this chapter are zero-level iso-surfaces

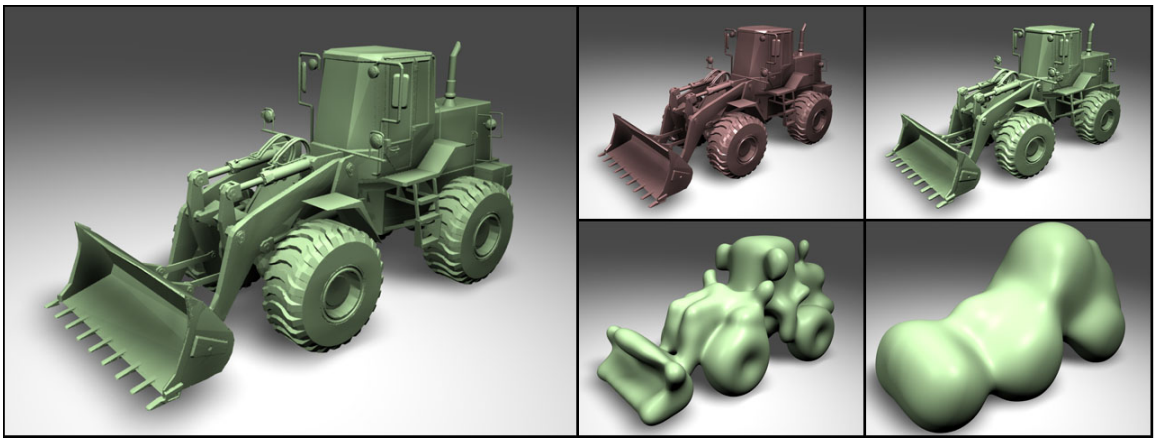

Figure 7.1. Interpolating and approximating surfaces (green) generated from polygonal original (brown).

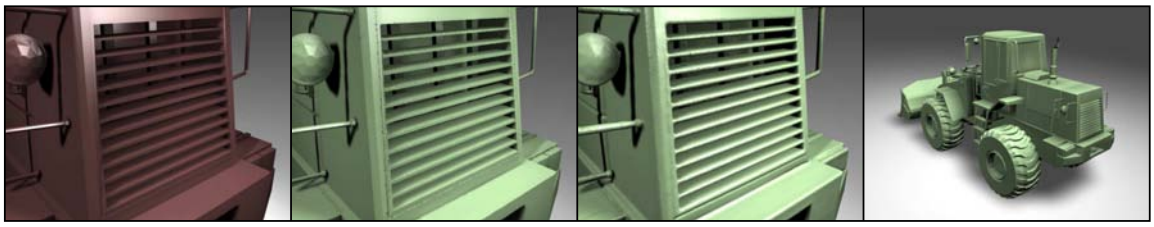

Figure 7.2. The far-left image shows a closeup view of the original polygons for the heavy-loader's back grill. The center-left image shows the resulting interpolating surface, and the center-right a slightly approximating one. The far-right image shows a rear view of the interpolating surface for the entire loader. The dented appearance near sharp edges is a polygonization artifact.

extracted from three dimensional implicit functions. In most applications, these zero-level iso-contours are the parts we care about in an entire three dimensional implicit function.

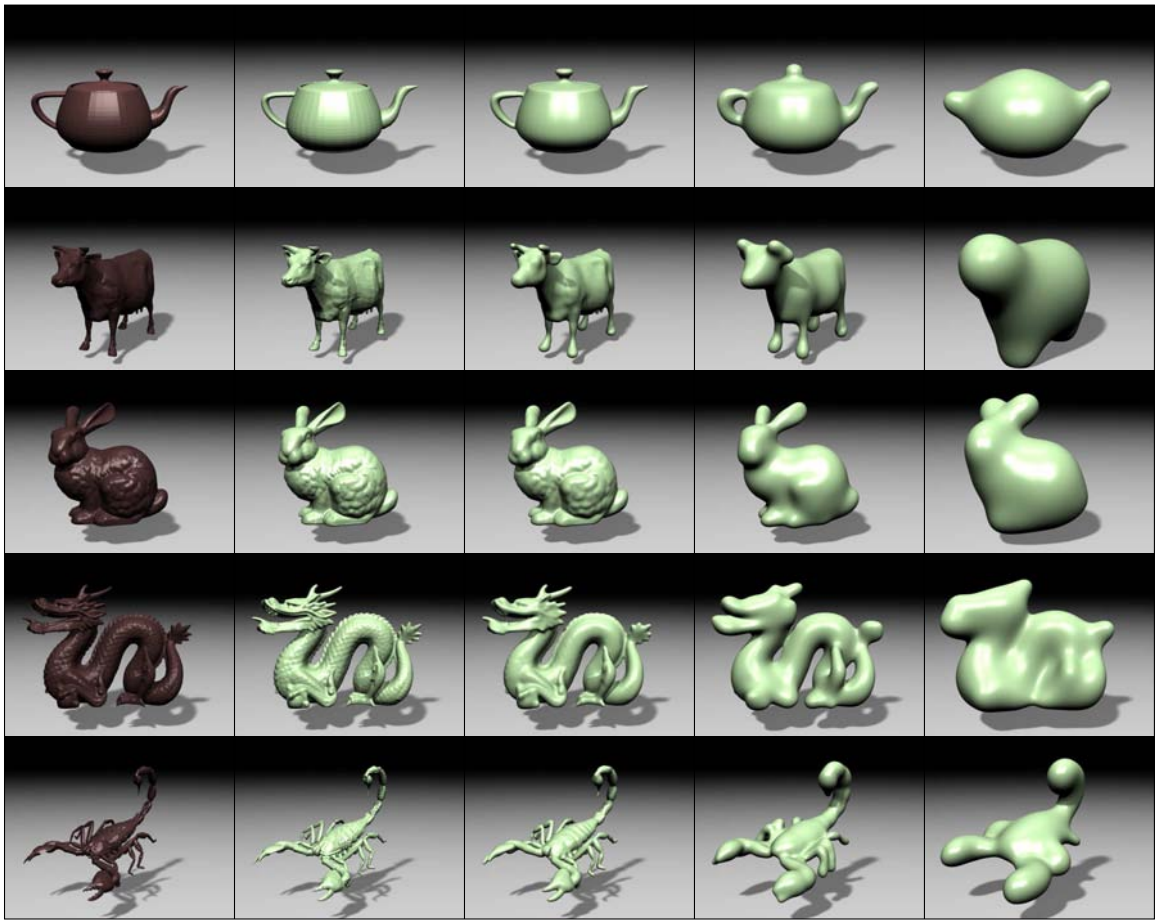

Figure 7.3. A collection of polygonal models processed with our algorithm. The objects shown in brown are the original polygonal models. Green objects are output from our algorithm. [Continued on next page.]

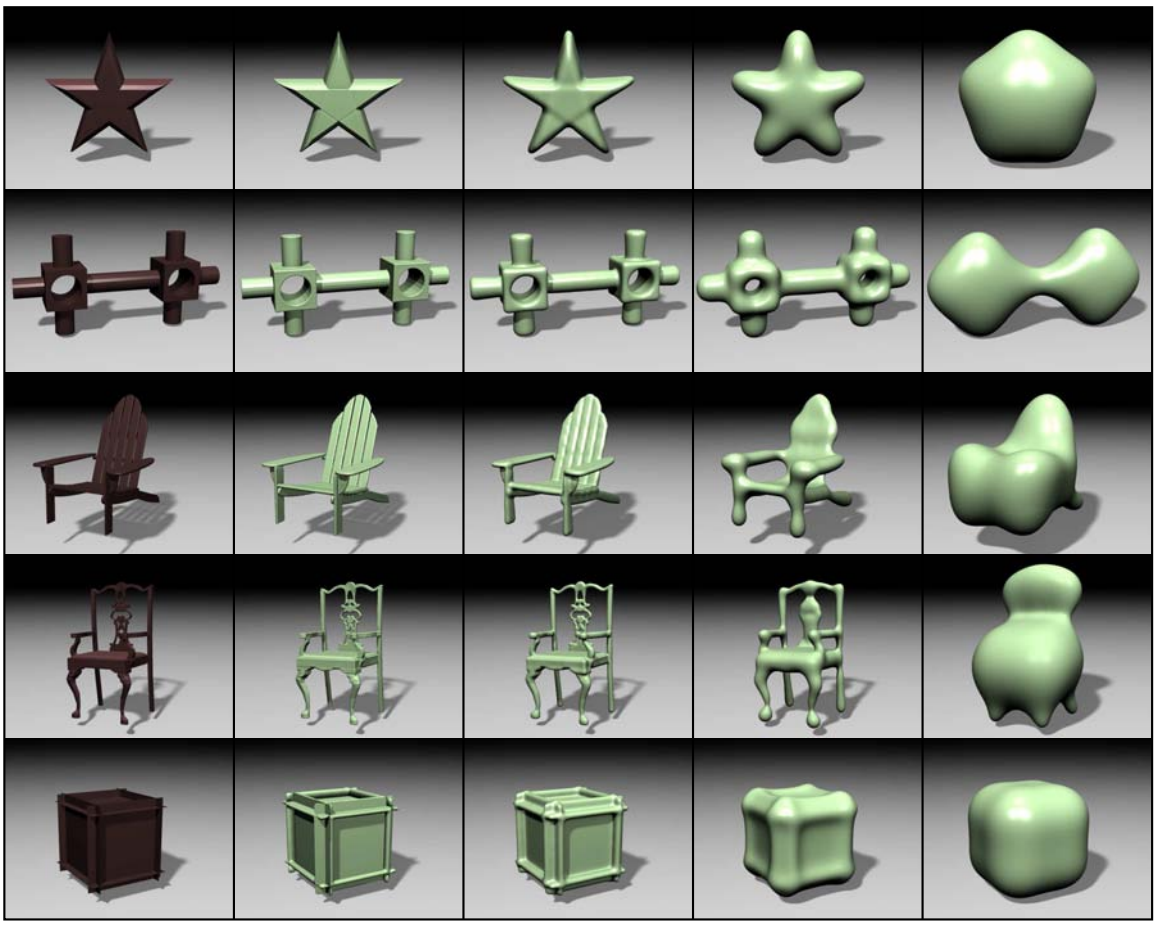

Figure 7.4. Additional polygonal models processed with our algorithm. [Continued from previous page.]

7.1 Polygon Soup

Figures 7.2, 7.3 and 7.4 show the result of applying our algorithm to a variety of three dimensional polygonal models using different values of ϵ. Most of these models contain holes, self intersections, non-manifold edges, and other defects. The objects shown in brown are the original polygonal models. Green objects are output from our algorithm. For sufficiently large ϵ, all objects converged to a circumscribing sphere.

Figure 7.5 shows closeup views of the heavy-loader's back grill.

Figure 7.5. The top left image shows a polygonized version the Utah teapot which contains holes (around lit and tip of spout), and intersecting parts (handle and spout with body). The top right image is a near-interpolating surface which fills the holes and removes intersecting surfaces. The bottom row contains images of a physical model build on a fuse-deposition machine.

7.2 Preprocessor for Rapid Prototyping Machines

As shown in Figure 7.5, we can use this algorithm as an effective preprocessor before sending a model to a rapid prototyping machine. The Utah teapot contains holes and self intersections that would cause the machine to produce garbage output. A tightly approximating implicit surface does not contain those problems and allows a successful build. Additionally, because building a solid teapot would waste material, it is desirable to include an inner surface. We generated the inner surface shown in the cut-away view by taking the same function used to create the outer surface and evaluating it again at lower value.

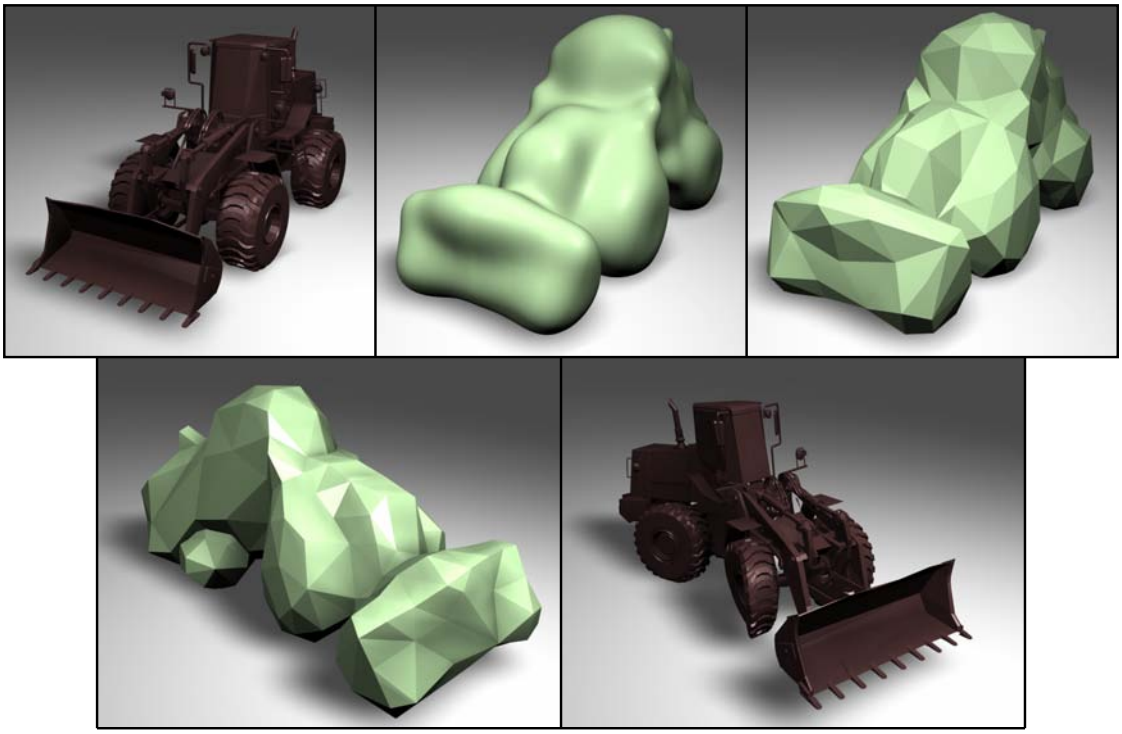

Figure 7.6. The heavy-loader shown top left contains many defects that make it unsuitable for simulation as a deformable object. The approximating surface, top center, fully encloses the original model. The tetrahedral finite-element model, top right, can be used as a simulation envelope to model the effect of an impact, lower left and lower right.

7.3 Simulation Envelopes

Deformable object simulations based on the finite element method have found widespread use in video games and film production. Unfortunately, self intersection, non-manifold edges, holes, and bad-aspect-ratio triangles, render most graphics models ill-suited for use as a finite element mesh. Even meshes that are free of these problems may contain far too many elements to be practical for simulation. We can still animate these objects by embedding them in a suitable, enclosing deformable mesh. As demonstrated by Figure 7.6, the tight, smooth, enclosing, surfaces that can be generated with our method make excellent simulation envelopes.

7.4 Parametric Patches

Figure 7.7 shows the result of applying our algorithm to a variety of three dimensional models containing parametric patches using a certain value of ϵ. The objects shown in red

Figure 7.7. A collection of models containing parametric patches processed with our algorithm. The objects shown in red are the original control polygons of the parametric models. Blue objects are output from our algorithm.

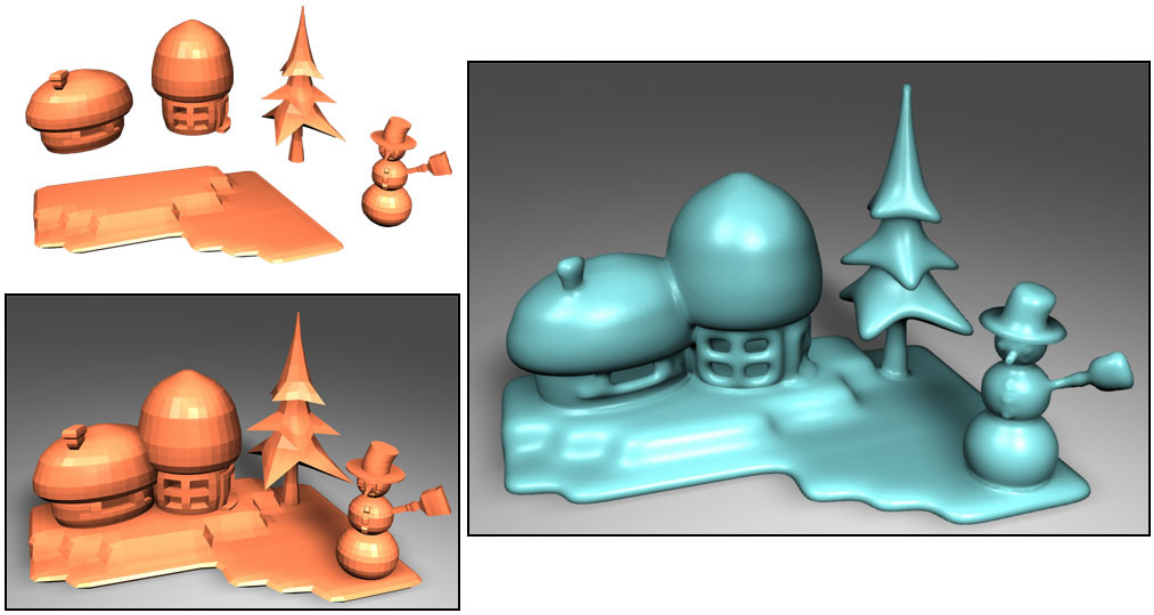

Figure 7.8. A set of parametric surfaces modeled separately, put together and processed with our algorithm to get a clean manifold surface. The object shown in red are the original control polygons of the parametric surfaces modeled separately and put together. The blue surface is the output from our algorithm and it is a clean manifold.

are the original control polygons of the parametric surfaces. Blue objects are output from our algorithm.

Figure 7.8 shows the result of combining multiple parametric models together to achieve one manifold surface. The object shown in red are the original control polygons of the parametric surfaces modeled separately and put together. The blue surface is the output from our algorithm and it is a clean manifold.

7.5 Discussion

Currently, we are using the polygonizer described in Bloomenthal (1994) for extracting iso surfaces. This marching cubes based technique works well for smooth surfaces, but extracting small features requires a very fine resolution and produces models with an inordinate number of polygons. Our polygonal models produce useful envelopes after being passed though surface simplification software (see Figure 7.6), but extracting them is time

consuming and requires substantial storage (See Table 7.1). We are currently considering better methods for surface extraction.

Marching cubes method suffers from a tendency to generate ill-shaped triangles. This problem has been improved to some degree by dual contouring which also provides adaptive contouring and preserves sharp boundaries. There are other techniques using Delaunay triangulation methods to generate provably good triangulations, but they are computationally more expensive.

In the application of rectifying defective polygonal models, it is common that a large portion of a defective model has good structures with nice shaped polygons. Turning the entire model into an implicit function and extracting an iso surface by marching cubes destroys the original structures and replaces them with much more ill-shaped triangles. It would be more efficient to have an algorithm that can use the input defective polygonal model to guide the process of marching cubes or other contouring schemes so that the output surface can preserve the original structure for the clean parts and extend new polygons for the defective regions.

Handling other types of geometric primitives would be a nice extension of this dissertation work. For example, lofting 3D curves has potentially promising applications for modeling from sketches. An ultimate goal of pursuing implicit representation would be converting all other different geometric representations into implicit surfaces and use implicit function as a central way to express shapes. Then contouring with primitives other than triangles for example quadrilaterals or subdivision patches would make it possible to have a general way to convert different geometric representations into each other.

Model	Fig.	P. In	ϵ	V. Out	Time	ϵ	V. Out	Time	ϵ	V. Out	Time	ϵ	V. Out	Time
Heavy Loader	7.2	37	0	2000	11:42	0.05	800	64:06	5	62	72:48	30	30	92:34
Teapot	7.3	6.3	0	1000	5:50	0.8	300	10:28	10	53	22:02	60	26	42:31
Cow	7.3	5.8	0	1000	5:37	0.8	300	8:04	7	61	23:35	120	23	58:19
Bunny	7.3	69	0	1500	8:13	0.4	400	19:34	10	50	41:36	60	28	72:23
Dragon	7.3	870	0	2000	12:23	0.6	400	82:21	7	46	89:54	30	21	97:04
Scorpion	7.3	78	0	1500	9:54	0.6	400	67:50	5	65	61:06	30	26	80:55
Star	7.4	0.05	0	1000	4:44	1	300	5:18	10	48	7:03	110	28	8:20
CSG Part	7.4	0.8	0	1000	4:45	0.8	300	5:54	7	60	8:50	120	29	20:21
Deck Chair	7.4	3.9	0	1000	5:01	0.8	300	8:29	10	55	27:06	120	30	52:36
Arm Chair	7.4	3.4	0	1000	4:56	0.4	300	7:53	7	62	28:28	120	28	41:09
Cube Shape	7.4	0.01	0	1000	4:44	0.8	300	5:01	10	45	3.06	80	25	1:52

Table 7.1. This table lists the computation times and ϵ parameter for the examples shown in Figures 7.2, 7.3 and 7.4. The columns P. In and V. Out list the number (in thousands) of polygons in the input model and the number (in thousands) of vertices in the output surface. We measure ϵ in thousandths of the diagonal length of the object's bounding box. Computation times (minutes:seconds) measure total time on a 3GHz P4 beginning with reading the input model and ending with writing the polygonized output surface. Column groups match the values used for the examples shown in the figures.

Bibliography

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. 2001. Point set surfaces. In *IEEE Visualization 2001*, 21–28.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. 2003. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (Jan.), 3–15.

Belytschko, Tl, Y. L., and Gu, L. 1994. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering 37*, 229–256.

Belytschko, T., Krongauz, Y., Organ, D., Fleming, M., and Krysl, P. 1996. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering 139*, 3–47. Special issue on meshless methods.

Bittar, E., Tsingos, N., and Gascuel, M.-P. 1995. Automatic reconstruction of unstructured 3d data: Combining a medial axis and implicit surfaces. *Proceedings of Eurographics 95*, 457–468.

Blinn, J. F. 1982. A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (July), 235–256.

Bloomenthal, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*. 324–349.

Bloomenthal, J., Ed. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc., San Francisco, California.

Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001*, 67–76.

Clough, R., and Tocher, J. 1965. Finite element stiffness matrices for analysis of plates in blending. In *Proceedings of Conference on Matrix Methods in Structural Analysis*.

Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Jr., F. P. B., and Wright, W. 1996. Simplification envelopes. In *Proceedings of ACM SIGGRAPH 1996*, 119–128.

Cohen-Or, D., Solomovici, A., and Levin, D. 1998. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics 17*, 2 (Apr.), 116–141.

Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 1999*, 317–324.

Duarte, C., and J.T.Oden. 1996. H-p clouds – an h-p meshless methods. *Numerical Methods for Partial Differential Equations.*, 1–34.

Duchon, J. 1975. Spline minimizing rotation-invariant semi-norms in sobolev spaces. In *Multivariate Approximation Theory*, L. S. C. Chui and J. Ward, Eds., 85–100.

Dyn, N., Levin, D., and Rippa, S. 1986. Numerical procedures for surface fitting of scattered data by radial functions. *SIAM Journal on Scientific and Statistical Computing 7*, 2 (Apr.), 639–659.

Farwig, R. 1986. Multivariate interpolation of arbitrarily spaced data by moving least squares methods. *J. Comput. Appl. Math. 16*, 79–93.

Fleishman, S., Alexa, M., Cohen-Or, D., and Silva, C. T. 2003. Progressive point set surfaces. *ACM Transactions on Graphics 22*, 4 (Oct.), 97–1011.

Franke, R., and Nielson, G. 1980. Smooth interpolation of large sets of scattered data. *International Journal Numerical Methods Engineering 15*, 1691–1704.

Grimson, W. E. L. 1983. Surface consistency constraints in vision. *Computer Vision, Graphics, and Image Processing 24*, 1 (Oct.), 28–51.

Hardy, R. L. 1971. Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research 76*, 1906–1915.

Jones, T. R., Durand, F., and Desbrun, M. 2003. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics 22*, 3 (July), 943–949.

Keren, D., and Gotsman, C. 1998. Tight fitting of convex polyhedral shapes. *International Journal of Shape Modeling*, 111–126.

Lancaster, P., and Salkauskas, K. 1981. Surfaces generated by moving least squares methods. *Mathematics of Computation 37*, 155 (July), 141–158.

Lawson, C. 1977. Software for C^1 surface interpolation. In *Mathematical Software III*, J. Rice, Ed. Academic Press, 161–164.

Litwinowicz, P., and Williams, L. 1994. Animating images with drawings. *Computer Graphics 28*, Annual Conference Series (July), 409–412.

Lucy, L. 1977. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal 82*, 12, 1013–1024.

McLain, D. H. 1974. Drawing contours from arbitrary data points. *Comput. J 17*, 4, 318–324.

McLain, D. H. 1976. Two dimensional interpolation from random data. *Comput. J 19*, 2, 178–181.

Morse, B., Yoo, T. S., Rheingans, P., Chen, D. T., and Subramanian, K. 2001. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. *Shape Modelling International* (May).

Muraki, S. 1991. Volumetric shape description of range data using "blobby model". In *Proceedings of ACM SIGGRAPH 1991*, 227–235.

Museth, K., Breen, D. E., Whitaker, R. T., and Barr, A. H. 2002. Level set surface editing operators. *ACM Transactions on Graphics 21*, 3 (July), 330–338.

Nayroles, B., G. T., and Villon, P. 1992. Generalizing the finite element methods: diffuse approximation and diffuse elements. *Computational Mechanics 10*, 307–318.

Nielson, G. M. 1993. Scattered data modeling. *IEEE Computer Graphics and Applications 13*, 1 (Jan.), 60–70.

Nishimura, H., Ohno, H., Kawata, T., Shirakawa, I., and Omura, K. 1983. Links-1: A parallel pipelined multimicrocomputer system for image creation. In *Conference Proceedings of the 10th Annual International Symposium on Computer Architecture, SIGARCH*, 387–94. also in Tutorial: Computer Graphics Hardware: Image Generation and Display, Computer Society Press, Washington, 1988, p. 320-327.

Nooruddin, F. S., and Turk, G. 2000. Interior/exterior classification of polygonal models. In *IEEE Visualization 2000*, 415–422.

Nooruddin, F. S., and Turk, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (Apr.), 191–205.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. 2003. Multi-level partition of unity implicits. *ACM Transactions on Graphics 22*, 3 (July), 463–470.

Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8, 429–446. ISSN 0178-2789.

Savchenko, V. V., Pasko, A. A., Okunev, O. G., and Kunni, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14*, 4 (Oct.), 181–188.

Shepard, D. 1968. A two-dimensional interpolation function for irregularly-spaced data. In *Proc. ACM National Conference*, 517–524.

Shewchuk, J. R. 1997. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania. Available as Technical Report CMU-CS-97-137.

Stam, J. 1998. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. *Computer Graphics 32*, Annual Conference Series, 395–404.

Taubin, G. 1993. An improved algorithm for algebraic curve and surface fitting. In *International Conference on Computer Vision*, 658–665.

Taubin, G. 1995. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 1995*, 351–358.

Terzopoulos, D. 1988. The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-10*, 4 (July), 417–438.

Turk, G., and O'Brien, J. F. 1999. Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 1999*, 335–342.

Turk, G., and O'Brien, J. F. 2002. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4 (Oct.), 855–873.

Wyvill, G., McPheeters, C., and Wyvill, B. 1986. Soft objects. In *Advanced Computer Graphics (Proceedings of Computer Graphics Tokyo '86)*, Springer-Verlag, T. L. Kunii, Ed., 113–128.

Yngve, G., and Turk, G. 2002. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics 8*, 4 (Oct.), 346–359.

Zhao, H., and Osher, S. 2002. Visualization, analysis and shape reconstruction of unorganized data sets. In *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer.