Stony Brook University

Numerical Differential Geometry and its Applications

A Dissertation Presented

by

Duo Wang

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

Aug 2012

Stony Brook University

The Graduate School

Duo Wang

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Xiangmin (Jim) Jiao – Dissertation Advisor
Associate Professor, Department of Applied Mathematics and Statistics

Xiaolin Li – Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

James Glimm
Distinguished Professor, Department of Applied Mathematics and Statistics

Xianfeng (David) Gu
Associate Professor, Department of Computer Sciences
Stony Brook University

This dissertation is accepted by the Graduate School.

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Numerical Differential Geometry and its Applications

by

Duo Wang

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2012

We describe a unified computational framework for polynomial fitting method based on weighted least squares approximations. This framework is first motivated by computing normals and curvatures on discrete surface mesh, then it is extended to construct the continuous global surface and calculate high order surface integration. Finally, the idea of generalized finite difference method is extracted by inverting the Vandermonde matrix from local polynomial fitting, it will be used to solve various geometric geometric partial differential equations and an elastic membrane problem.

Different applications require different techniques and impose different challenges including simplicity, accuracy, continuity, robustness and efficiency. Our research focus on high order accuracy, but also give considerations to other requirements. For normal and curvature calculations, surface reconstruction and integration, we demonstrate order of accuracy up to 6. For numerical solution of geometric PDEs and the elastic membrane problem, we introduce an accurate spatial discretization over triangular surface mesh and our semi-implicit schemes

can achieve at least quadratic convergence rate while being much more accurate and stable than using explicit schemes.

Contents

List of Figures

List of Tables

Acknowledgements

First I would like to thank my advisor Prof. Xiangmin Jiao, he is not only an excellent young scholar himself but also a great mentor to students. It was a rewarding and enjoyable five year experience under Prof. Jiao's guidance, I learned a lot from his insights and scientifically rigorous way of thinking, coding and writing. Also I want to thank him for financially supporting my graduate study.

Also I want to thank other members of our research group, Dr. Ying Chen, Dr. Bryan Clark, Dr. Volodymyr Dyedov and Navamita Ray, it was a pleasure working with them for the past several years, thanks for allowing me to use some of their experimental results in this dissertation.

Thank Dr. Rao Garimella for giving me the internship opportunity at Los Alamos National Lab, where I learned a lot about parallel programming and software engineering. I did not include that internship project into my dissertation, but it did help a lot in my job hunting process.

Finally I want to thank my dissertation committee members, Prof. James Glimm, Prof. Xiaolin Li and Prof. David (Xianfeng) Gu, for their support and helpful comments of my research.

Chapter 1

Introduction

Surface arises in various geometric and numerical applications, including meshing, computer graphics, physical and biological simulations. The continuous theory of surface is a well studied mathematical branch called differential geometry, while in numerical simulations, surfaces are represented by discrete models. Discrete mesh is the most commonly used form to represent a surface, where vertices coordinate values and their connectivity are the only information available. In this dissertation, we present a unified computational framework based on polynomial fitting to process discrete surface mesh and demonstrate its applications in surface-related numerical problems.

Many methods have been introduced to compute normals and curvatures over discrete surface meshes, such as [58, 46, 47, 66, 54, 9, 29, 37]. Among them there are two different types of paradigms, namely discrete and continuous. Discrete methods fall shorts on accuracy. A few continuous methods based on polynomial fittings have been proven to deliver converging results [46, 66, 9, 37], however, the numerical behaviors for different parametrization schemes and numerical solvers can differ drastically in practice, and no systematic analysis and comparison have been reported previously for these methods. We show that our polynomial fitting framework based on local orthogonal projection with a safeguard against folding delivers the best combination of simplicity, accuracy, efficiency, and robustness.

Applications such as mesh smoothing and high order integration require a continuous geometric support over the discrete surface mesh. We apply our framework to reconstruct a high order continuous surface. This reconstruction problem also arises in computer graphics [17] and geometric modeling [63], there the focus is on high order continuity. Our method will achieve third- and even higher order accuracy, while guaranteeing global C^0 continuity.

Geometric partial differential equations (PDEs) on moving surfaces occur in various applications, such as surface smoothing in computer-aided design [68] and the

modeling of moving surfaces of materials [8]. Solving these high-order PDEs using explicit methods would require very small time steps to achieve stability, due to their strong nonlinearity and stiffness, whereas using implicit methods would result in complex nonlinear systems of equations that are expensive to solve. We propose semi-implicit schemes for mean curvature flow and surface diffusion over triangulated surfaces. Numerical experiments demonstrate that our method can achieve second-order accuracy for both mean-curvature flow and surface diffusion, while being much more accurate and stable than using explicit schemes.

Finally, we will use our computational framework to solve a physical problem arises in the parachute simulations, the modeling and discretization of the curvature effect of a thin and curved elastic interface, which separates two fluid subdomains. For such an interface, there is often a pressure jump between the two fluid subdomains, which is partially balanced by a normal pressure exerted by the interface due to a curvature effect, in a manner similar to the surface tension in fluid dynamics [41, 39]. Mathematically, this pressure is the normal component of the surface divergence of the stress tensor. We derive an explicit, easy-to-compute formula for the normal pressure, and develop a discretization method for evaluating it from the stress tensor. Also we derive explicit formulas of the surface divergence of the stress tensor, so that we can discretize the problem in a strong form using our generalized finite difference method.

The dissertation is organized as follows. Chapter 2 introduces our computational framework, which serves as the foundation of our research. Chapter 3 describes its application in computing normals and curvatures. Chapter 4 extends the framework from local fitting to globally constructing a continuous surface. Chapter 5 and chapter 6 apply our framework to solve geometric PDEs and the elastic membrane problem.

Chapter 2

The Computational Framework

Given a discrete surface mesh, the first step is to reconstruct its continuous prototype. Due to the difficulty of global parametrization and the fact that differential quantities used in most applications are local to the geometry, a local reconstruction suffices. The idea of local polynomial fitting is to approximate the local surface geometry by a bivariate polynomial. There are several steps to construct such a local polynomial. Section 2.1 introduces the neighbor points selection strategy. Section 2.2 gives several representative methods of local parametrization. Section 2.3 describes the polynomial fitting method based on weighted least squares approximations. Section 2.4 introduces the numerical solvers and section 2.5 analyzes its accuracy and stability.

2.1 Point Selections Strategies

The first step for local polynomial fittings is to choose a collection of nearby points. For discrete meshes, it is typical to select the points based on mesh connectivity, and sometimes coupled with some geometry-based filtration. It is common to use a k-ring neighborhood for some integer k, such as 1, 2, or 3 [66, 9, 23]. The *1-ring neighbor faces* of a vertex v is the faces incident on v, and the *1-ring neighbor vertices* are the vertices of these faces. For an integer $k \geq 1$, the $(k+1)$*-ring neighborhood* of a vertex is the union of the 1-ring neighbors of its k-ring neighbor vertices.

Triangular Mesh When k is constrained to integers, the numbers of vertices in k-ring neighbors grow rapidly as k increases. For finer control, [37] proposed to use half-ring increments by defining the *1.5-ring neighbor faces* to be the faces that share an edge with a 1-ring neighbor face, and the *1.5-ring neighbor vertices* to be

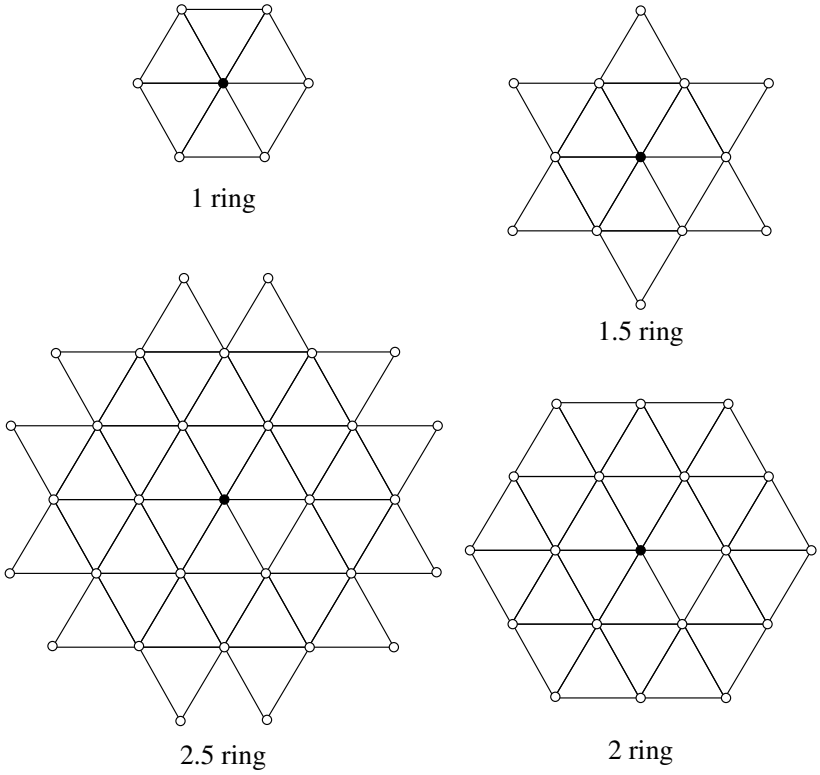


Figure 2.1: Schematics of 1-, 1.5-, 2-, and 2.5-ring neighborhood. Each diagram shows the neighborhood of the center (black) vertex.

the vertices of these faces. For an integer $k \geq 1$, the $(k+1.5)$-*ring neighborhood* is the union of the 1.5-ring neighbors of the k-ring neighbor vertices. Figure 2.1 illustrates the neighborhood definitions up to 2.5 rings. Table 2.1 shows the typical numbers of vertices of a $(d+1)/2$-ring for d up to 6, which have approximately twice as many points as the number of unknowns of degree-d fittings. In our later discussions, we allow k to have half increments when referring to k-ring neighbors. Note that under some pathological situations (such as near the boundary of an open surface), a $(d+1)/2$-ring may have insufficient number of points for degree-d fittings. [37] proposed to adaptively enlarge the neighborhood size by half-ring increments if the number of vertices is fewer than 1.5 times of the required number of points. We follow such an adaptive strategy when appropriate.

Quadrilateral Mesh The above definition of n-ring neighbors for a triangular mesh tends to produce too many points for quadrilateral meshes. We redefine the neighborhood of a vertex as follows:

Table 2.1: Numbers of coefficients in dth degree fittings versus numbers of points in typical $\frac{d+1}{2}$ rings.

degree (d)	1	2	3	4	5	6
#coeffs.	3	6	10	15	21	28
#points in $\frac{d+1}{2}$ ring	7	13	19	31	37	55

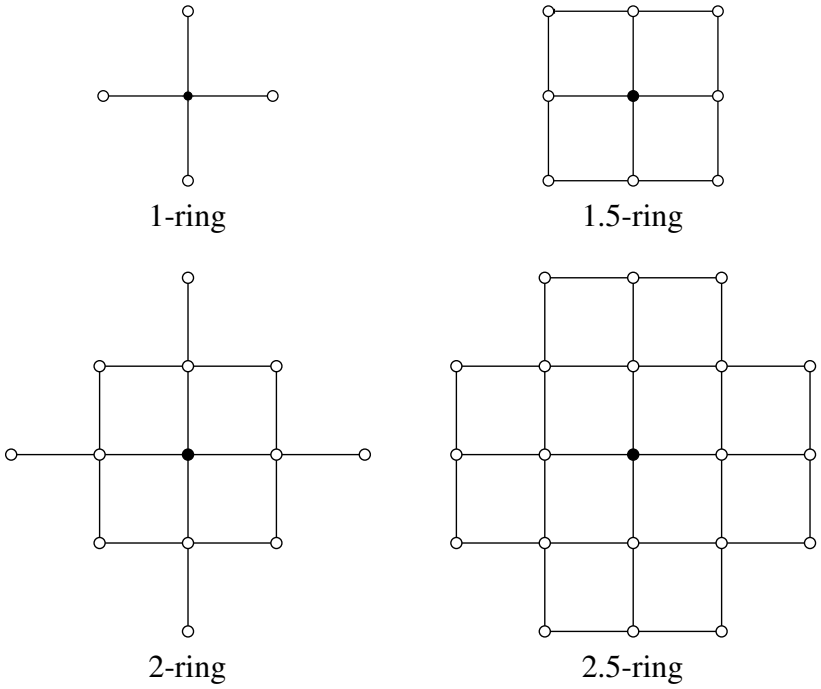


Figure 2.2: Examples of 1-, 1.5-, 2-, and 2.5-rings of typical vertex in quadrilateral mesh. Each image depicts the neighborhood of the center black vertex.

- The *0-ring* of a vertex is the vertex itself ;
- The *k-ring* vertices of a vertex (where $k = 1,2,3,\ldots$) is the set of vertices that share an edge with a vertex in the (k-1)-ring;
- The $(k+0.5)$-ring of a vertex (where $k = 1,2,3,\ldots$) is the union of k-ring vertices and the vertices that share elements with an edge between two vertices in the k-ring.

Figure 2.2 shows the 1-, 1.5-, 2-, and 2.5-rings of a typical vertex in a quadrilateral mesh or a quad-dominant mesh. In general for degree-d fittings, we find it most effective to use a ring of $(d+1)/2$ for a mesh without noise or a ring of $d/2+1$ or larger for meshes with noise.

A k-ring neighborhood depends only on mesh connectivity. For highly irregular meshes it could contain some points that are far away from the point of interest. One could address this problem by choosing the vertices based on distances rather than mesh connectivity (such as using the k nearest neighbors), but it is less efficient and prone to the well-known problem of "short circuiting" (i.e., to choose points that are close in Euclidean distance but are far away in geodesic distance). Under the weighted least squares framework, we can easily filter out any vertex within a k-ring neighbor by simply setting its corresponding weight in the weighting matrix Ω to zero or to a very small number. Typically, the weight can be inversely proportional to some power of the distance from the point to the vertex under consideration and sometime can also depend on the vertex normals for better robustness [37]. Therefore, k-ring neighbors with weighted least squares provides a simple and flexible approach, so we do not consider other point-select strategies here.

2.2 Methods of Local Parametrizations

There are several existing methods of local parametrization, some only applies to 1-ring neighborhood, some requires extra effort by solving a linear system. Parametrization has received significant attention in recent years in geometric modeling and computer graphics [18, 27, 43]. See [19] for a survey of recent work on parametrization. It is important to note that polynomial fittings require only a local parametrization around a vertex, instead of a global parametrization of the whole surface mesh. Each vertex has its own parametrization, this localization can better capture the local geometry information and enforces no restriction on the global properties of the surface as global parametrization does, also it simplifies the problems both theoretically and computationally. Here we compare three representative parametrizations schemes that are commonly used.

Xu's 1-Ring Parametrization To support quadratic fittings, Xu [66, 67] proposed a simple but specialized procedure to parametrize the 1-ring neighborhood of a vertex. At high level, this procedure flattens the neighborhood while preserving both the ratios between the interior angles of the triangles at the vertex and the lengths of the edges incident to the vertex. Such a parametrization is not unique, subject to translation and rotation. Xu eliminated the additional degrees of freedom by making the vertex the origin of the uv plane and choosing one of the edges as the u direction. We refer readers to [67] for details.

Xu's 1-ring parametrization is very simple and efficient. It is approximately isometric at the vertex, so it is likely to be smooth. However, this construction has some limitations, among which the most notable is its limitation to 1-ring neighborhood.

A 1-ring neighborhood sometimes does not even have enough points for quadratic fittings, not to mention cubic or higher-degree fittings. This limitation has serious consequences, as we will demonstrate in numerical experiments. In addition, the algorithm in [66, 67] assumes a 1-ring neighborhood is topologically a disk so that the sum of the interior angles at a vertex is equal to 2π after flattening. The procedure would have to be modified for boundary vertices if it were used for open surfaces (more precisely, for 2-manifolds with boundary).

Local Orthogonal Projection with Safeguard Another simple and efficient procedure for constructing the parametrization is to project the neighborhood orthogonal onto a plane. We refer to this method as *local orthogonal projection* (LOP). To avoid rank deficient (or ill-conditioned) Jacobian matrices, the plane should be approximately tangential to the surface. The LOP is often used in the construction of a local height function, as we describe in Section 3.2.2 and as done in [9] and [37]. Therefore, LOP enjoys the flexibility of utilizing the formulas for either local parametrization or local height function in Table 3.1.

The LOP is simple and efficient, but unlike Xu's parametrization it is more general and can be applied to k-rings for $k \geq 1$. However, if the surface is highly curved and the mesh is too coarse, the projection of the k-ring neighborhood onto the plane may not be one-to-one, which in turn can lead to a violation of the regularity assumption of polynomial fittings.

The proneness to folding is the single most important issue with LOP. In the weighted least squares framework, this problem can be addressed by utilizing the weighting matrix to filter out vertices whose normals form too large an angle with the normal to the uv plane, as suggested in [37]. Specifically, let $\hat{m}_i$ denote a rough estimation of the unit normal at the ith vertex, and let $\hat{m}_0$ denote the normal to the uv plane, typically equal to the approximate vertex normal at the vertex in consideration. Note that the $\hat{m}_i$ are used only for the construction of projection plane and the weights, so some simple averaging of face normals suffices. [37] chose the weight for the ith vertex to be

$$\omega_i = \gamma_i \Big/ \left(\sqrt{\|u_i\|^2 + \varepsilon} \right)^{d/2}, \tag{2.2.1}$$

where ε is a small number to prevent division by zero. The key safeguard in (2.2.1) is γ_i, which is set to $\hat{m}_i^T \hat{m}_0$ if the angle between $\hat{m}_i$ and $\hat{m}_0$ is small but set to 0 if the angle is too large. For a typical mesh, γ_i is approximately equal to 1 and plays no role. For a coarse mesh with rapidly varying normals where a vertex may "wrap around," γ_i would become zero and the vertex is filtered out. This safeguard is simple and efficient, and we employ it for LOP in our tests.

Conformal Parametrizations of k-Ring Besides employing a safeguard, the potential problem of mesh folding of LOP can also be resolved (or alleviated) by using a more sophisticated parametrization strategy. Such an approach was advocated by some authors, such as [23]. A number of methods have been developed for planar parametrization of surfaces in recent years. A few have been implemented in CGAL (www.cgal.org), including barycentric mapping [60], mean value coordinates [18], discrete authalic parametrization [14], and least squares conformal maps (LSCM) [43]; we refer readers to [55] for a complete list.

The planar parametrization methods in general can be categorized as either fixed boundary or free boundary. For fixed-boundary methods, such as barycentric mapping and mean value coordinates, a surface is mapped onto a convex shape (such as a circle or a convex polygon). Such mappings are typically guaranteed to be one-to-one but are not necessarily smooth. Furthermore, because a k-ring neighborhood is not necessarily convex even for $k = 1$, mapping it to a convex shape can lead to arbitrarily large distortions and in turn large errors for polynomial fittings. Among the free-boundary methods, the most appealing types are those based on conformal mappings. For smooth compact surfaces, a conformal mapping is a type of harmonic mapping that preserves angles, so it is smooth by construction. For discrete surfaces, exact conformal mapping in general does not exist, and an approximation is constructed by solving a linear or nonlinear system of equations. Unlike free-boundary methods, fixed-boundary methods cannot guarantee the mapping to be one-to-one and can even lead to flipped triangles [55], resulting in a violation of the regularity assumption for polynomial fittings. Therefore, none of these general parametrization methods is ideal for polynomial fittings.

After extensive experimentation, we selected LSCM of [43] as a representative for the general parametrization methods for our comparisons, because it is a free-boundary method based on conformal mappings, and in our tests it delivered better results than the others available in CGAL. Although LSCM does not guarantee against mesh folding, it is less likely to cause folding than a naive LOP without any safeguard. However, LSCM is more expensive than LOP for its requirement of solving a linear system. Its effectiveness compared to safeguarded LOP is unclear and can only be examined through numerical experimentation, which we report in Section 3.3.

Summary of Different Parametrizations We summarize a high-level comparison of the advantages and disadvantages of different methods in Table 2.2. Note that each of the methods has its own disadvantages. Xu's parametrization and LSCM suffer from inflexibility and inefficiency, respectively. LOP seems to be promising to deliver the best simplicity, efficiency, and flexibility. Its potential problem of mesh folding can be resolved in a simple manner with the weighted least squares

Table 2.2: High-level comparison of different parametrization schemes.

	advantages	disadvantages
Xu	simple and efficient	limited to 1-ring neighborhood
LOP/LHF	simple, efficient, flexible	prone to folding if without safeguard
LSCM	flexible	expensive; no guarantee against folding

formulation. In Section 3.3, we will study the accuracy and efficiency of different methods experimentally and verify this conclusion.

2.3 Weighted Least Squares Polynomial Fitting

In approximation theory, the Taylor series expansion is a powerful tool in deriving numerical approximations. Local polynomial fittings are based on this well-known Taylor polynomial around a point.

We are primarily concerned with surfaces, so the local fitting is basically an interpolation or approximation to a neighborhood of a point P under a local parametrization with parameters u and v, where P corresponds to $u = 0$ and $v = 0$. The polynomial fitting may be defined over the global xyz coordinate system or a local uvw coordinate system. In the former, the neighborhood of the surface is defined by the *coordinate function* $f(u,v) = [x(u,v), y(u,v), z(u,v)]$. In the latter, assuming the uv-plane is approximately parallel with the tangent plane of the surface at P, each point in the neighborhood of the point can be transformed into a point $[u,v,f(u,v)]$ (by a simple translation and rotation), where f is known as the *local height function*.

Let u denote $[u,v]^T$ and $f(u)$ denote a smooth bivariate function, which may be the local height function under orthogonal projection, or the x, y, or z component of the coordinate function for a parametric surface. Let c_{jk} be a shorthand for $\frac{\partial^{j+k}}{\partial u^j \partial v^k} f(0)$. Given a positive integer d, if $f(u)$ has $d+1$ continuous derivatives, it can be approximated to $(d+1)$st order accuracy about the origin $u_0 = [0,0]^T$ by

$$f(u) = \underbrace{\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!}}_{\text{Taylor polynomial}} + \underbrace{\sum_{j,k\geq 0}^{j+k=d+1} \frac{\partial^{j+k}}{\partial u^j \partial v^k} f(\tilde{u},\tilde{v}) \frac{\tilde{u}^j \tilde{v}^k}{j!k!}}_{\text{remainder}}, \tag{2.3.1}$$

where $0 \leq \tilde{u} \leq u$ and $0 \leq \tilde{v} \leq v$. We emphasize that this equality assumes that f is continuously differentiable up to $d+1$, and we refer to this as the *regularity assumption*. The derivatives of the Taylor polynomial are the same as f at u_0 up to degree d, and hence the problem of estimating the derivatives reduces to the

estimation of the coefficients c_{jk} of the Taylor polynomial. Specifically, given a set of data points, say $[u_i, v_i, f_i]^T$ for $i = 1, \ldots, m-1$, sampled from a neighborhood near a point $u_0 = [u_0, v_0, f_0]^T$ on a smooth surface. Plugging in each given point into (2.3.1), we obtain an approximate equation

$$\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u_i^j v_i^k}{j!k!} \approx f_i, \tag{2.3.2}$$

which has $n = (d+1)(d+2)/2$ unknowns (i.e., c_{jk} for $0 \leq j+k \leq d$, $j \geq 0$ and $k \geq 0$), resulting in an $m \times n$ rectangular linear system. We refer to d as the *degree of fitting*. Note that one could enforce the fit to pass through the point u_0 by setting $c_{00} = 0$ and removing the equation corresponding to u_0, reducing to an $(m-1) \times (n-1)$ rectangular linear system, this may be beneficial if the points are known to interpolate a smooth surface.

The above method for estimating the Taylor polynomial is known as *polynomial fitting* or *local polynomial fitting*, because the fitting is in a local neighborhood around point u_0. Let us denote the rectangular linear system obtained from (2.3.2) as

$$Vc \approx f, \tag{2.3.3}$$

where c is an n-vector composed of c_{jk}, and V is a *generalized Vandermonde matrix*. For a local height function, f is an m-vector composed of f_i; for a parametric surface, f is an $m \times 3$ matrix, of which each column corresponds to a component of the coordinate function.

A careful numerical treatment of equation 2.3.3 is necessary because it is the core of our research. By solving this equation we get the coefficient vector c, together with the coordinate values of vertex P and the two tangent vectors, we have the full information of the approximated local surface geometry. In section 5.1, we present a set of formulas to calculate normal, curvature and other surface differential operators. In chapter 5.3, the idea of *Generalized Finite Difference* method is extracted by inverting the *Vandermonde matrix* V, and we propose a semi-implicit scheme for solving various geometric PDEs.

Numerically, (2.3.3) can be solved using the framework of *weighted linear least squares* [26, p. 265], i.e., to minimize a weighted norm (or semi-norm),

$$\min_c \|Vc - f\|_\Omega = \min_c \|\Omega(Vc - f)\|_2, \tag{2.3.4}$$

where Ω is a *weighting matrix*. Typically, Ω is an $m \times m$ diagonal matrix, whose ith diagonal entry ω_i assigns a priority to the ith point $[u_i, v_i]^T$ by scaling the ith row of

V. This formulation is equivalent to the linear least squares problem

$$\tilde{V}c \approx b, \text{ where } \tilde{V} = \Omega V \text{ and } b = \Omega f. \tag{2.3.5}$$

In general, $\tilde{V}$ is $m \times n$ and $m \geq n$. However, this linear system may be rank deficient (i.e., the column vectors of $\tilde{V}$ may not be linear dependent) or ill-conditioned (i.e., the singular values of $\tilde{V}$ may have very different scales) due to a variety of reasons, including poorly scaling, insufficient number of points, or degenerate arrangements of points [38]. The scaling of A can be improved substantially by introducing a *scaling matrix* S and change the problem as

$$\min_d \|Ad - b\|_2, \text{ where } A = \tilde{V}S \text{ and } d = S^{-1}c. \tag{2.3.6}$$

Here S is typically a diagonal matrix. Let $\tilde{v}_i$ denote the ith column of $\tilde{V}$. The ith diagonal entry of S is typically chosen to be either $\|\tilde{v}_i\|_\infty$ [9, 67] or $\|\tilde{v}_i\|_2$ [37], where the latter approximately minimizes the condition number of $\tilde{V}S$ [26, p. 265]. However, the problem may still be ill-conditioned after rescaling, and we will address this issue in the next section.

2.4 Numerical Solvers

In polynomial fittings the most difficult aspect is the solution of the least squares system (2.3.6), because this system can be rank deficient (i.e., under-determined) or even worse be nearly rank deficient (i.e., highly ill-conditioned). Such ill-conditioned problems can occur even if the number of points is greater than the number of unknowns. Although there exist general techniques for solving ill-conditioned least squares problems, such as SVD, an effective solution should take advantage of the special properties of the problem at hand. In this subsection, we compare two techniques, SVD, and a customization of QR factorization.

Singular Value Decomposition In numerical linear algebra, singular value decomposition (SVD) is the standard technique for solving rank-deficient least squares problems; see for example [26, Chapter 5]. Given a linear least squares problem $Ax \approx b$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$, let $A = U\Sigma V^T$ denote the SVD of A, where $U \in \mathbb{R}^{m \times n}$ is composed of left singular vectors u_i of A, $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, and $V \in \mathbb{R}^{m \times n}$ is composed of the right singular vectors v_i of A. A general solution to a rank-deficient problem $Ax \approx b$ is

$$x = \sum_{\sigma_i > \varepsilon} v_i u_i^T b / \sigma_i, \tag{2.4.1}$$

where ε is small number close 0 (such as 10^{-8}). For an ill-conditioned problem, we can set ε to a small factor of σ_1 (such as $10^{-4}\sigma_1$), which effectively would solve a modified problem $\tilde{A}x \approx b$, where $\tilde{A} = \sum_{\sigma_i \geq \varepsilon} \sigma_i u_i v_i^T$. The condition number of $\tilde{A}$, namely the ratio between its largest and smallest singular values, is bounded by $1/\varepsilon$. Given that the noise in the input points is small, limiting the condition number of $\tilde{A}$ effectively limits the sensitivity of x with respect to the noise in b.

In the context of polynomial fittings, SVD has been used to address the potential rank deficiency in [67] and [9, 10]. [67] solved (2.3.6) by constructing the normal equation $A^T Ad = A^T b$ and then solve it using the SVD of $A^T A$, which is equivalent to the eigenvalue decomposition of $A^T A$, as $A^T A$ is symmetric positive semi-definite. Let $\tilde{\sigma}_i$ denote the eigenvalues of $A^T A$ and $\tilde{u}_i$ denote the eigenvectors of $A^T A$. Xu solves the system as $x = \sum_{\tilde{\sigma}_i \geq \varepsilon} \tilde{u}_i \tilde{u}_i^T A^T b / \tilde{\sigma}_i$, where ε was chosen to be 10^{-8}. In contrast, [9] used SVD of A to solve (2.3.6) directly, but no detail was given. In either case, the use of SVD does not take into account the geometric meaning of polynomial fittings, and it does not give higher priorities to lower derivatives. In addition, SVD is far more expensive compared to techniques based on QR factorizations, as we describe next.

QR Factorization with Safeguard Instead of using a standard technique, [37] proposed a customized QR factorization with a safeguard for polynomial fittings. The idea is based on the observation that given the QR factorization $A = QR$, the QR factorization of the first k leading columns of A (i.e., $A_{:,1:k}$ using MATLAB-like notation) are the k leading columns Q (i.e., $Q_{:,1:k}$) and the $k \times k$ leading submatrix of R (i.e., $R_{1:k,1:k}$). If A has a large condition number, one can remove the columns of A from the right to obtain a better conditioned problem. This effectively reduces to a lower-degree fitting, so Proposition 1 is still applicable to bound the errors. Furthermore, because the condition number (in 2-norm) of A is the same as that of R, the condition number can be estimated efficiently. See [37] for more detail. Compared to SVD, this QR factorization based approach is about four to five times faster than using SVD. Note that QR factorization with partial pivoting is another alternative for solving rank deficient least squares problem [26], but like SVD such an approach is less appropriate because it does not give higher priorities to lower derivatives.

2.5 Analysis of Accuracy and Stability

The weighted least-squares formulation is a well-known linear algebra technique. However, in the context of derivative estimation for parametric surfaces, its numerical analysis must be customized and does not seem to exist in the literature. We

present an analysis in this subsection, focusing on two aspects: 1) the accuracy and stability of the least squares problem (2.3.6), and 2) the propagation of the errors from (2.3.6) to differential quantities.

Errors in Least Squares Formulation By the perturbation theory, the error in the solution to Eq. (2.3.6) depends on a number of factors, including the input errors in A and b, the condition number of A, the angle of b with respect to column space of A, as well as the orientation of b within the column space of A [59, Lecture 18]. The entries in A depend only on the parametrization, which we address in subsection 2.5. Here, we focus on the errors in b and the condition number of A.

The error in b depends on both the "noise" in the input points and the residual in (2.3.1). Note that an important assumption behind (2.3.1) is the *regularity assumption*. If this assumption is violated, then the derivatives may be unbounded, and the errors in the remainder may be arbitrarily large. It is therefore very important that the coordinate functions are smooth with respect to the parametrization (i.e., with bounded derivatives).

We now consider the condition number of $Ad \approx b$. Assume the least squares problem is well conditioned and the numerical solver is stable. Let h denote the average edge length of the mesh, and assume the maximum diameter of the neighborhood is $O(h)$. We bound the errors in the approximations to the partial derivatives in terms of h.

Proposition 1: Given a set of points $[u_i, v_i, \tilde{f}_i]$ that interpolate a smooth height function f or approximate f with an error of $O(h^{d+1})$. Assume the point distribution and the weighing matrix are independent of the mesh resolution, and the condition number of the scaled matrix $A = \tilde{A}S$ in (2.3.6) is bounded. The degree-d weighted least squares fitting approximates c_{jk} to $O(h^{d-j-k+1})$.

The proof of the proposition follows that for Theorem 4 in [37].

Error Propagation to Differential Quantities The analysis above considers the errors in the approximations to c_{jk}, i.e. the partial derivatives of the coordinate functions with respect to the parameters u and v. From the coefficients c_{jk}, we compute the normal and curvatures using the formulas in Table 3.1. The errors in c_{jk} therefore can propagate into the computed normals and curvatures. Following the same proof as Theorem 5 in [37], we obtain the following proposition.

Proposition 2: Assume the position, gradient, and Hessian of coordinate functions that approximated to $O(h^{d+1})$, $O(h^d)$ and $O(h^{d-1})$, respectively, and assume the condition number of the Jacobian matrix is bounded. a) The angle between the computed and exact normals is $O(h^d)$; b) the components of the shape operator and

curvature tensor are approximated to $O(h^{d-1})$; c) the Gaussian and mean curvatures are approximated to $O(h^{d-1})$.

In the proposition, it is important that the condition number of the Jacobian matrix is bounded. If the condition number of the Jacobian matrix is unbounded, the errors in c_{jk} can be magnified by an arbitrarily large factor. Note that this proposition makes no claim about the principal directions, as they are inherently unstable if the maximum and minimum curvatures are roughly equal. If the magnitudes of the principal curvatures are well separated, then the principal directions would also have similar convergence rates as the curvatures. When using our symmetric shape operator introduced in Section 3.2, the computed principal directions are guaranteed to be orthonormal.

Summary of Requirements for Accuracy and Stability The analysis above indicates that polynomial fittings produce converging estimations of differential quantities under certain conditions. We summarize the conditions as follows, grouped into three categories. First of all, the input points must satisfy the following:

1. Decreasing neighborhood size: The size of the neighborhood (in terms of the distances between points) should decrease asymptotically for finer meshes (i.e., should be $O(h)$).

2. Accurate input points: For degree-d polynomial fitting, the coordinate functions should be at least $(d+1)$st order accurate (i.e., $O(h^{d+1})$).

These requirements are necessary for asymptotically bounding the errors of the input to polynomial fittings. These conditions are universal for any fitting method, and they may or may not be satisfied by certain applications. Under the above conditions, the parametrization must satisfy the following requirements:

3. Smooth coordinate functions: The partial derivatives of the coordinate functions must be bounded with respect to the parameters in the given neighborhood.

4. Well conditioned Jacobian: The Jacobian matrix must be far from rank deficiency (i.e., must be well conditioned).

These two conditions are related to, but do not necessarily imply, each other. Finally, additional requirements must be imposed on the numerical solver of the linear least squares system (2.3.6):

5. Robust solver: The least squares solver must be stable and at the same time be able to resolve ill-conditioned systems.

Under the above conditions, convergence is guaranteed theoretically for the computed differential quantities, and small errors can be expected in practice for sufficiently fine meshes. On the other hand, if any of the above conditions is violated, convergence is not guaranteed and the errors can be large, although one may still observe convergence and relatively small errors in practice. The above analysis provides us guidelines for choosing the neighborhood, the parametrizations, and numerical solvers. It also provides us a platform for comparing different methods, as we will see in the next chapter.

Chapter 3

Normal and Curvature Computation

Computing normals and curvatures is a fundamental problem for many geometric and numerical computations, including feature detection, shape retrieval, shape registration or matching, surface fairing, surface mesh adaptation or remeshing, front tracking and moving meshes. But their accurate computations on discrete surfaces are challenging.

Two types of methods are widely used in practice, discrete method and continuous method based on polynomial fitting. Discrete method is easy to implement but does not guarantee convergent curvature results for general surfaces; Polynomial fitting based methods are well founded mathematically and can be proven to deliver convergent results under reasonable assumptions, however, the numerical behaviors for different parametrization schemes and numerical solvers can differ drastically in practice, and no systematic analysis and comparison have been reported previously for these methods.

This is a direct application of our framework introduced in Chapter 2. First we give a summary of commonly used methods, then analyzes the classical formulas for continuous parametric surfaces and presents some alternative formulas that are more amenable to numerical computation for their numerical stability. Section 3.3 presents numerical experiments to verify our theoretical analysis and compare the different algorithms in terms of accuracy and runtime efficiency. Our analysis shows that the choice of parametrization and numerical solver for the least squares problem can have significant impact on the accuracy and stability of polynomial fittings. In addition, we show that the methods based on local orthogonal projection with a safeguard against folding delivers the best combination of simplicity, accuracy, efficiency, and robustness.

3.1 Related Works

Many methods have been introduced to compute normals and curvatures over discrete surface meshes, such as [58, 46, 47, 66, 54, 9, 29, 37]. Among them there are two different types of paradigms, namely discrete and continuous. Discrete methods use explicit formulas directly while continuous methods first construct a locally parametrized surface patch, then use the exact formulas for continuous surface to calculate normal and curvature.

For discrete method, normal is usually approximated by the weighted average of neighboring face normals; Gaussian curvature is often approximated using angle deficit method. For mean curvature, [47] proposed a cotangent formula, which is closely related to the formula for Dirichlet energy of [52]. It was shown that the cotangent formula does not produce converging pointwise mean-curvature estimations except for some special cases, as noted in [7, 66, 30, 64]. As another example, [40] proposed a tangent-weighted formula for estimating mean-curvature vectors, whose convergence relies on special symmetric patterns of a mesh. [13] proposed a method for curvature computation based on the theory of normal cycles, but their error analysis is limited to the case of restricted Delaunay triangulations.

Discrete methods fall shorts on accuracy. There is a good reason for this, the discrete approximations only use 1-ring neighborhood information, for a convergent curvature approximation, a minimum of order 2 polynomial is needed, which requires at least 5 adjacent vertices. Another disadvantage of discrete methods is that it requires different formulas for different differential quantities, for example, the approximation schemes for mean and Gaussian curvature are totally different. This is different from polynomial-based continuous methods, as we will show in section 3.2.

A few continuous methods based on polynomial fittings have been proven to deliver converging results [46, 66, 9, 37]. However, these methods can have drastically different numerical behaviors, and sometimes they can deliver very poor results in practice. As we discussed in section 2, accuracy and stability of polynomial fittings are subtle numerical issues, they are complicated due to the interactions among different aspects of the methods, including point selection, parametrization, numerical solvers, as well as their interactions with classical differential geometry formulas. Although our focus is on polynomial fittings only, some of the numerical issues that we address also apply to some other methods, such as those in [25] and [54].

3.2 Formulas for Differential Quantities

Reliable computations of differential quantities on discrete surfaces critically depend on their counterparts for continuous surfaces. The latter is a subject in classical differential geometry and has been studied extensively, but the numerical behaviors of classical formulas have not been carefully scrutinized until recently. In [37], the stability of the classical formulas for height functions was analyzed, and some new formulas were proposed. Here we extend the analysis to the formulas for parametric surfaces and propose some alternative formulas that are more amenable to numerical computations.

3.2.1 Formulas for Parametric Surfaces

Given a smooth surface Γ in the global xyz coordinate system, let $x = f(u)$ be a parametrization of a neighborhood around a vertex on Γ, where $u = [u,v]^T$ (we consider all vectors as column vectors for consistency with the modern convention in numerical analysis.). If the surface is smooth, the coordinate function f defines a smooth surface composed of points $x(u) \in \mathbb{R}^3$. The Jacobian matrix of $x(u)$ with respect to u is then $J = [x_u, x_v]$. The vectors x_u and x_v form a basis of the tangent space of the surface. Let du denote $[du,dv]^T$. The *first fundamental form* of the surface is the quadratic form

$$I(du) = du^T G du, \text{ where } G = J^T J. \tag{3.2.1}$$

G is known as the *first fundamental matrix*. Its determinant is $g = \det(G) = \|x_u \times x_v\|^2$. Let ℓ denote $\sqrt{g}$, i.e., $\ell = \|x_u \times x_v\|$, which we refer to as the "*area element*." The unit normal to the surface is then

$$\hat{n} = \frac{x_u \times x_v}{\ell}. \tag{3.2.2}$$

The *second fundamental form* in the basis $\{x_u, x_v\}$ is given by the quadratic form

$$II(du) = du^T B du \tag{3.2.3}$$

where

$$B = -\begin{bmatrix} \hat{n}_u^T x_u & \hat{n}_u^T x_v \\ \hat{n}_v^T x_u & \hat{n}_v^T x_v \end{bmatrix} = \begin{bmatrix} \hat{n}^T x_{uu} & \hat{n}^T x_{uv} \\ \hat{n}^T x_{uv} & \hat{n}^T x_{vv} \end{bmatrix}, \tag{3.2.4}$$

and is known as the *second fundamental matrix*.

The well-known *Weingarten equations* read $[\hat{n}_u \mid \hat{n}_v] = -[x_u \mid x_v]W$, where W is the *Weingarten matrix* (a.k.a. the *shape operator*) with basis $\{x_u, x_v\}$. By left-

multiplying J^T on both sides, we have $B = GW$, and therefore,

$$W = G^{-1}B. \tag{3.2.5}$$

The *mean curvature* is equal to half of the trace of W. The *Gaussian curvature* is equal to the determinant of W. Let κ_1 and κ_2 denote the eigenvalues of W, which are the *principal curvatures*. Then, $\kappa_H = (\kappa_1 + \kappa_2)/2$ and $\kappa_G = \kappa_1\kappa_2$. Let $\hat{d}_1$ and $\hat{d}_2$ be the eigenvectors of W. Then $\hat{e}_1 = J\hat{d}_1/\|J\hat{d}_1\|$ and $\hat{e}_2 = J\hat{d}_2/\|J\hat{d}_2\|$ are the *principal directions*. The principal curvatures and principal directions are sometimes used to construct a 3×3 matrix

$$C = \kappa_1\hat{e}_1\hat{e}_1^T + \kappa_2\hat{e}_2\hat{e}_2^T. \tag{3.2.6}$$

We refer to C as the *principal curvature tensor* or simply the *curvature tensor* for brevity.

The Weingarten matrix in (3.2.5) is classical in differential geometry, but it is not well-suited for numerical computations. The reason is that the matrix is in general not symmetric, so its eigenvectors are not orthogonal to each other. In addition, the eigenvalues of W are not necessarily real in the presence of round-off errors. For robust numerical computations, we derive a symmetric shape operator as follows. Let $J = QR$ denote the QR factorization of J, where Q is 3×2 with orthonormal column vectors (i.e., $Q^TQ = I$) and R is a 2×2 upper triangular matrix. The QR factorization can be constructed using Gram-Schmidt orthogonalization [26]. Let $\hat{q}_1$ and $\hat{q}_2$ denote the column vectors of Q. The shape operator in the orthonormal basis $\{\hat{q}_1, \hat{q}_2\}$ is the symmetric matrix

$$\tilde{W} = R^{-T}BR^{-1}, \tag{3.2.7}$$

and the curvature tensor is then

$$C = Q\tilde{W}Q^T = J^{+T}BJ^{+}, \tag{3.2.8}$$

where $J^{+} = R^{-1}Q^T$ is the pseudo-inverse of J. Note that the curvature tensor (3.2.8) is equivalent to the *embedded Weingarten map* in [51, p. 20]. Eq. (3.2.8) also has the same form as Eq. (13) in [37], but it is more general in that it applies to parametric surfaces instead of just local height functions (see subsection 3.2.2).

After obtaining the symmetric shape operator $\tilde{W}$, its eigenvalues are guaranteed to be real and its eigenvectors are guaranteed to be orthonormal. More specifically, let w_{ij} denote the entries of $\tilde{W}$. The principal curvatures are then

$$\kappa_{1,2} = \frac{1}{2}\left(w_{11} + w_{22} \pm \sqrt{(w_{11} - w_{22})^2 + 4w_{12}^2}\right). \tag{3.2.9}$$

If $\kappa_1 = \kappa_2$, we choose the principal directions to be $\hat{q}_1$ and $\hat{q}_2$, respectively. Otherwise, the principal direction corresponding to κ_i is

$$\hat{e}_i = \frac{(w_{11} - \kappa_{3-i})\hat{u}_1 + w_{12}\hat{u}_2}{\|(w_{11} - \kappa_{3-i})\hat{u}_1 + w_{12}\hat{u}_2\|} = \frac{w_{12}\hat{u}_1 - (w_{11} - \kappa_i)\hat{u}_2}{\|w_{12}\hat{u}_1 - (w_{11} - \kappa_i)\hat{u}_2\|} \tag{3.2.10}$$

for $i = 1, 2$. For better stability, we use the first equality if $|w_{11} - \kappa_{3-i}| > |w_{11} - \kappa_i|$ and use the second equality otherwise.

3.2.2 Formulas for Height Functions

The preceding formulas apply to any smooth parametrization of a neighborhood of any point on a surface. A special parametrization is associated with the so-called "local height function," obtained by the orthogonal projection of the neighborhood onto a plane that is nearly tangential to the surface. Specifically, given a smooth surface in the global *xyz* coordinate system, it can be transformed into a local *uvw* coordinate system by translation and rotation. Assume both coordinate frames are orthonormal right-hand systems. Let the origin of the local frame be at point $[x_0, y_0, z_0]^T$. Let $\hat{t}_1$ and $\hat{t}_2$ be the unit vectors in the global coordinate system along the positive directions of the u and v axes, respectively. Then, $\hat{m} = \hat{t}_1 \times \hat{t}_2$ is the unit vector along the positive w direction. Let U denote the orthogonal matrix composed of column vectors $\hat{t}_1$, $\hat{t}_2$ and $\hat{m}$, i.e., $U = [\hat{t}_1, \hat{t}_2, \hat{m}]$. Any point x on the surface is then transformed to a point $p(x) = [u, v, f(u)]^T = U^T(x - x_0)$. Conversely, $x = Up + x_0$. The function $f(u) : \mathbb{R}^2 \to \mathbb{R}$ (more precisely, from a subset of $\mathbb{R}^2$ to $\mathbb{R}$) is a *height function* near x_0. In the *uvw* coordinate system, the first two components of the coordinate function are u and v, respectively, so the differential quantities depend on only the derivatives of f with respect to u and v.

Let $\nabla f = [f_u, f_v]^T$ denote the gradient of f with respect to u and $H = \begin{bmatrix} f_{uu} & f_{uv} \\ f_{vu} & f_{vv} \end{bmatrix}$ the Hessian of f, where $f_{uv} = f_{vu}$. For this particular form, the Jacobian matrix and the first and second fundamental matrices can be written explicitly in terms of ∇f and H, resulting in some simple closed-form formulas. Table 3.1 summarizes the formulas of first- and second-order differential quantities of a parametric surface or local height function. We separate the table into three parts by double lines. Most of the first two parts are well known; see e.g. [15]. The formulas for mean and Gaussian curvatures of local height functions were derived in [37]. To the best of our knowledge, the third part of the table has not appeared in the literature previously. Note that in [37] a symmetric shape operator was derived for local height functions using the singular value decomposition (SVD) of the Jacobian matrix, but unfortunately the Jacobian matrix for a parametric surface is too complex to derive an explicit SVD, so we express the symmetric shape operator using QR factorization

Table 3.1: Summary of formulas for continuous parametric surfaces and height functions.

	local parametrization	local height function
description	$x = f(u,v)$	$x = U \begin{bmatrix} u \\ v \\ f(u,v) \end{bmatrix} + x_0$
Jacobian matrix	$J = \begin{bmatrix} x_u & x_v \end{bmatrix} = \underbrace{\begin{bmatrix} \hat{q}_1 & \hat{q}_2 \end{bmatrix}}_{Q} R$	$U \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ f_u & f_v \end{bmatrix}$
1st fundamental matrix	$G = J^T J$	$\begin{bmatrix} 1+f_u^2 & f_u f_v \\ f_u f_v & 1+f_v^2 \end{bmatrix}$
area element	$\ell = \sqrt{\det(G)} = \|x_u \times x_v\|_2$	$1 + f_u^2 + f_v^2$
surface normal	$\hat{n} = \frac{x_u \times x_v}{\ell}$	$\frac{1}{\ell} U \begin{bmatrix} -f_u \\ -f_v \\ 1 \end{bmatrix}$
2nd fundamental matrix	$B = \begin{bmatrix} \hat{n}^T x_{uu} & \hat{n}^T x_{uv} \\ \hat{n}^T x_{uv} & \hat{n}^T x_{vv} \end{bmatrix}$	$\underbrace{\begin{bmatrix} f_{uu} & f_{uv} \\ f_{vu} & f_{vv} \end{bmatrix}}_{H} / \ell$
Weingarten matrix	$W = G^{-1} B$	$\frac{1}{\ell}(J^T J)^{-1} H$
mean curvature	$\kappa_H = \frac{1}{2}\mathrm{tr}(W)$	$\frac{\mathrm{tr}(H)}{2\ell} - \frac{(\nabla f)^T H (\nabla f)}{2\ell^3}$
Gaussian curvature	$\kappa_G = \det(W) = \det(B)/\ell^2$	$\det(H)/\ell^4$
symmetric shape operator	$\tilde{W} = R^{-T} B R^{-1} = [w_{ij}]$ in basis $\{\hat{q}_1, \hat{q}_2\}$	
principal curvature tensor	$C = Q \tilde{W} Q^T = J^{+T} B J^{+}$	
principal curvatures	$\kappa_{1,2} = \frac{1}{2}\left((w_{11}+w_{22}) \pm \sqrt{(w_{11}-w_{22})^2 + 4w_{12}^2}\right)$	
principal directions	$\hat{e}_i = \frac{(w_{11}-\kappa_{3-i})\hat{q}_1 + w_{12}\hat{q}_2}{\|(w_{11}-\kappa_{3-i})\hat{q}_1 + w_{12}\hat{q}_2\|} = \frac{w_{12}\hat{q}_1 - (w_{11}-\kappa_i)\hat{q}_2}{\|w_{12}\hat{q}_1 - (w_{11}-\kappa_i)\hat{q}_2\|}$	

instead.

3.3 Experimental Results

In this section, we report some numerical experiments to verify our preceding analysis in Section 2.5. The experimental results can be affected by a number of factors, such as input errors, surface topology, mesh connectivity, parametrization methods, degrees of fittings, and numerical solvers. For the input meshes, we consider three aspects: closed versus open surfaces, noise-free versus noisy input points, and well-shaped (or regular) versus poor-shaped (or irregular) meshes. For the algorithms, we evaluate four different methods that we described earlier, including (1) Xu's 1-ring parametrization as detailed in [67], (2) local height functions [37],

and parametrizations based on (3) local orthogonal projection or (4) least-squares conformal maps [43]. When appropriate, we use different degrees of polynomial fittings (degrees 2, 3, or 4). We denote these different methods by Xu-d, LHF-d, LOP-d, LSCM-d, respectively, where d is the degrees of polynomial fittings.

The combinations of the above options lead to tens of cases. For each case we compute the normals, and mean and Gaussian curvatures, principal curvatures, and principal directions. For convergence study, we use four meshes of different resolutions for each case and compute the average convergence rate as

$$\text{convergence rate} = \frac{1}{3}\log_2\left(\frac{\text{error of level 1}}{\text{error of level 4}}\right). \tag{3.3.1}$$

The convergence rate is showed at the right end of the curves. Let v denote the total number of vertices, and let $\hat{n}_i$ and $\tilde{n}_i$ denote the exact and computed unit vertex normals at the ith vertex. We measure the relative L_2 errors in normals as $\sqrt{\sum_1^v \|\tilde{n}_i - \hat{n}_i\|_2^2 / v}$. For comprehensiveness, we sometimes also consider the L_∞ error of normals, evaluated as $\max_i \|\tilde{n}_i - \hat{n}_i\|_2$. Let k_i and $\tilde{k}_i$ denote the exact and computed curvatures at the ith vertex. We measure the relative errors of curvatures in L_2 norm as $\|\tilde{\kappa} - \kappa\|_2 / \|\kappa\|_2 = \sqrt{\sum_{i=1}^v (\tilde{\kappa}_i - \kappa_i)^2} \Big/ \sqrt{\sum_{i=1}^v \kappa_i^2}$, and measure the L_∞ error as $\max_i |\tilde{\kappa}_i - \kappa_i| / |\kappa_i|$. Altogether, we obtain thousands of data points. In consideration of dissertation length, we report only a representative subset of results. All of our computations use double-precision floating point arithmetic.

Experiments for Noise-Free Closed Surfaces We first present results on noise-free closed surfaces. We chose a torus with inner radius 0.7 and outer radius 1.3. A torus is representative for smooth surfaces as it contains parabolic, elliptic and hyperbolic points. In practice, a mesh can be well shaped (such as those obtained from CAD software for finite element analysis) or poor shaped (such as those from marching cubes for visualization purposes). To capture both types of meshes, we generated four triangular meshes using a commercial mesh generation software GAMBIT of Fluent Inc. (now part of ANSYS, Inc.) and four others using the isosurface function in MATLAB. Figure 3.1 shows a coarse mesh of either type. To eliminate potential input errors, we projected the vertices onto the torus so that the input points are accurate up to machine precision.

We first compare quadratic fittings using the four methods described earlier, denoted by Xu-2, LHF-2, LOP-2, and LSCM-2, respectively. For Xu-2, 1-ring neighbors were used, and for others 1.5-rings were used. Figure 3.2 shows the L_2 error and L_∞ errors in the computed normals, and Figure 3.3 shows the corresponding results for mean curvature. Note that except for Xu-2, the other methods converged

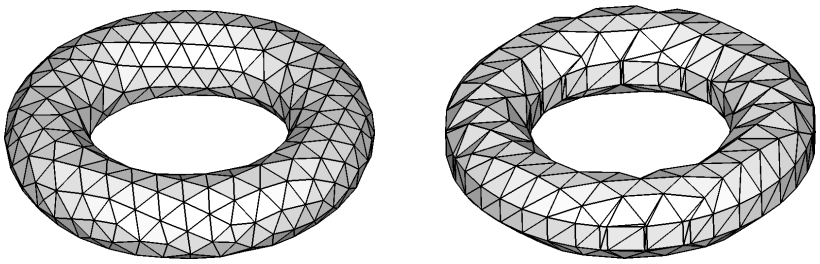


Figure 3.1: Sample meshes of torus generated by commercial mesh generation software (left) and by isosurface algorithm (right).

at about quadratic rates for normals in terms of both L_2 error and L_∞ errors, and converged at about linear rates for curvatures in L_∞ error and nearly quadratic rates in L_2 error. The super-convergence in L_2 error is likely due to statistical cancellation of truncation errors. For Xu-2, the behavior was inconsistent: for well-shaped meshes it performed better than others in L_2 errors but substantially worse in L_∞ errors, and for poor-shaped meshes it failed to converge. This inconsistent behavior is not surprising, because the 1-ring neighbors used in Xu-2 are more compact than 1.5-ring and in turn could sometimes deliver more accurate results, but they sometimes have too few points and lead to rank-deficient or ill-conditioned systems. Even though the SVD employed in [67] could produce numerical solutions in such cases, accuracy is not guaranteed. For completeness, we also report the L_2 errors in the computed Gaussian curvature and principal directions in Figure 3.4, whose behaviors were qualitatively similar to mean curvature.

For noise-free surfaces from CAD models or analytic functions, it is possible and sometimes desirable to obtain higher order estimations. Except for Xu's method, the other three methods are capable of supporting higher-degree fittings. We consider only LHF and LSCM for high-degree fittings and evaluate degree-3 and 4 fittings using 2 and 2.5-rings, respectively, while adaptively increasing the ring size at half increments if the numbers of points are insufficient. Since LHF and LOP deliver nearly identical results (which is evident from the results of quadratic fittings), the primary difference between LHF and LSCM is the parametrization being used.

Figure 3.5 shows the L_2 and L_∞ errors in the computed normals. Figure 3.17 shows the L_2 errors in the computed mean and Gaussian curvatures. Note that LSCM-3 delivered slightly better accuracy than LHF-3 in most cases, but LHF-4 substantially out-performed LSCM-4 in almost all cases. In all cases, the convergence rates of LHF-4 were nearly an order higher than the theoretical prediction due to statistical error cancellations. In contrast, LSCM-4 converged at a rate nearly an order lower than the theoretical prediction for well-shaped meshes and failed to converge for poor-shaped meshes. This behavior of LSCM indicates that parametrizations

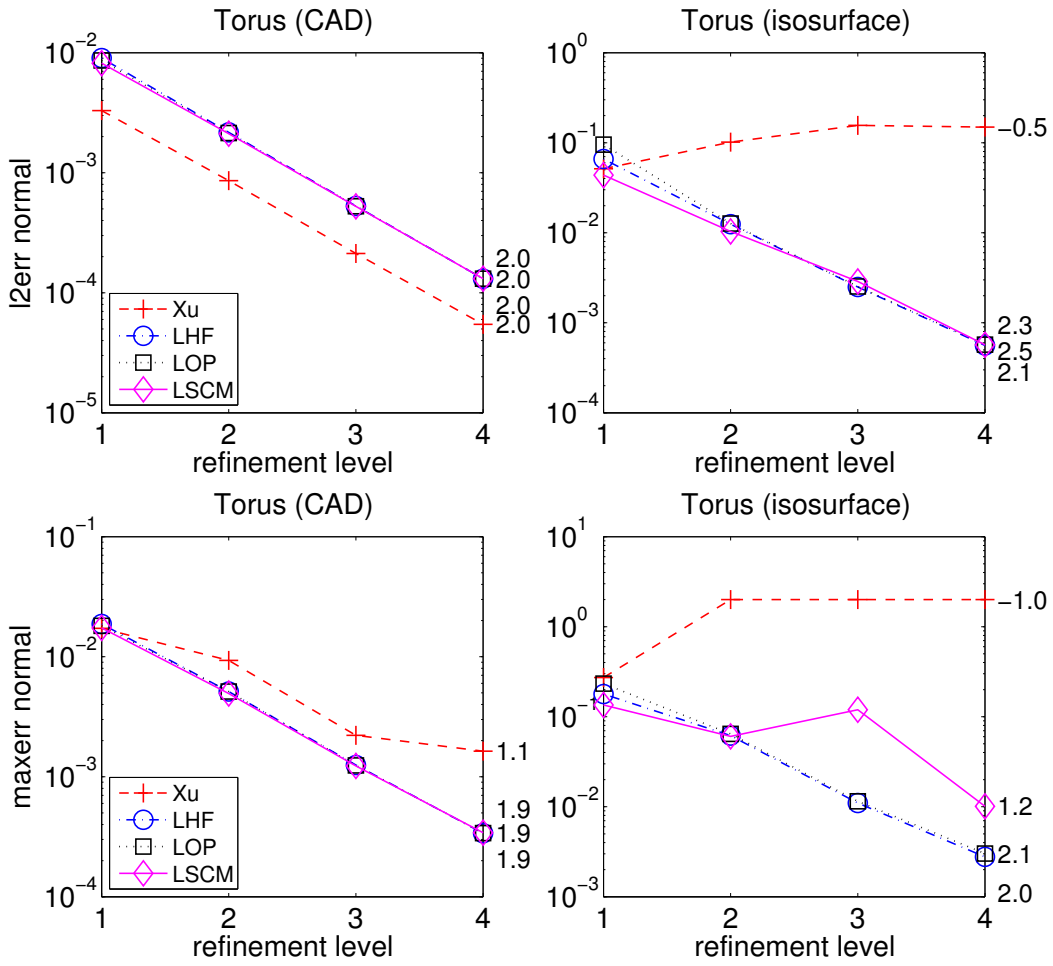
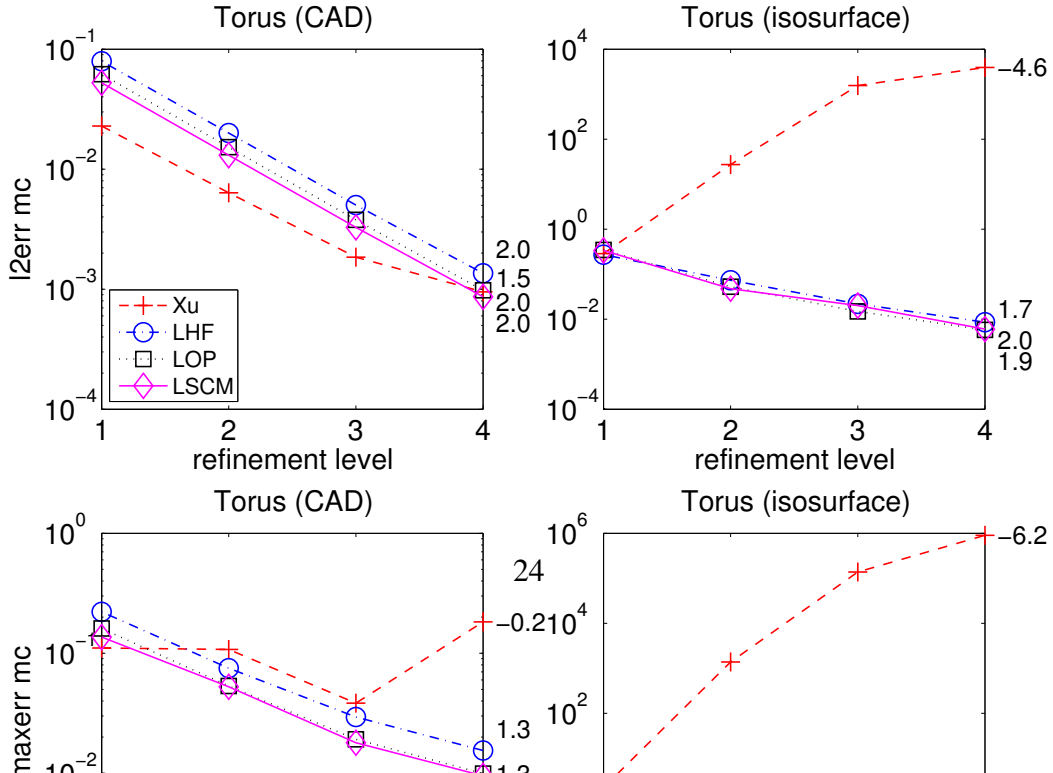


Figure 3.2: Comparison of L_2 (top) and L_∞ (bottom) errors in computed normals using degree-2 fittings for noise-free torus meshes.



can affect the accuracy of high-degree fittings significantly, and LSCM seems to be sufficiently smooth for cubic fitting but insufficient for higher-degree fittings.

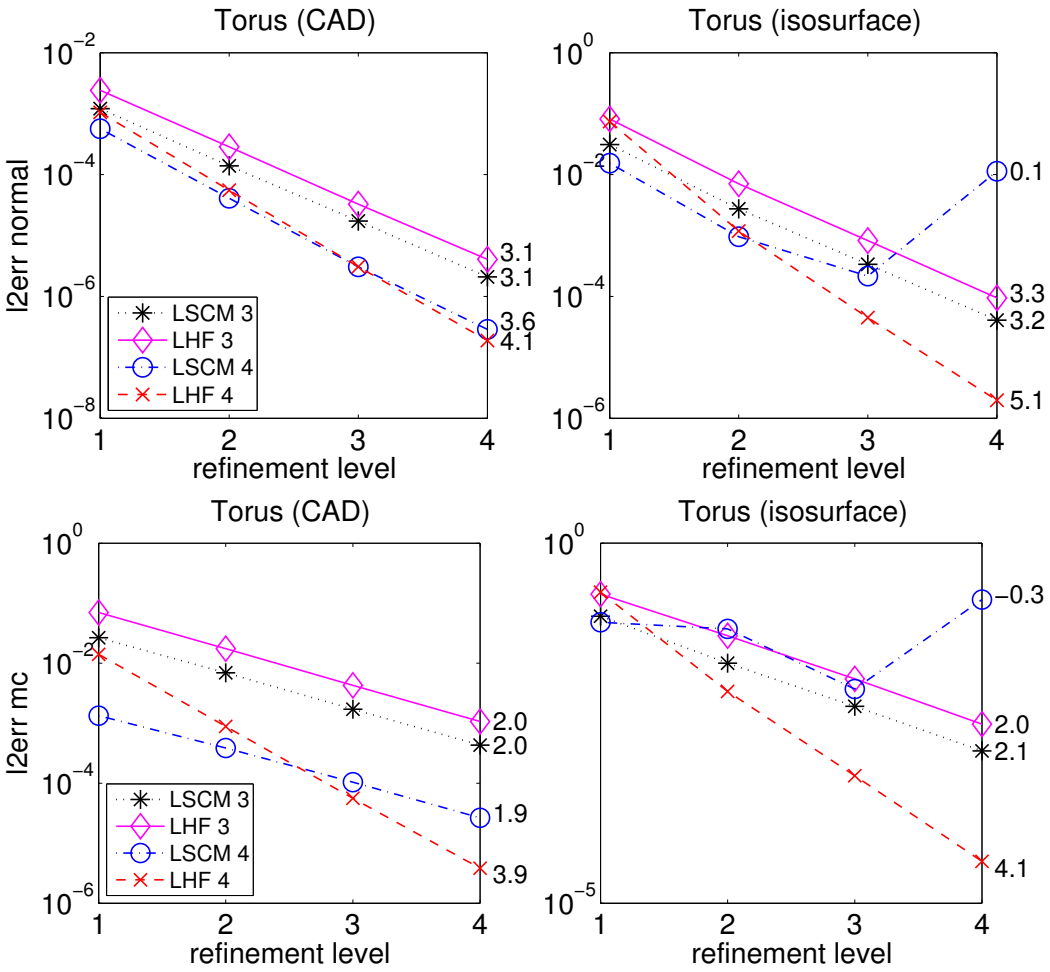


Figure 3.5: Comparison of L_2 errors of in computed normals (top) and mean curvatures (bottom) using degree-3 and 4 fittings for noise-free torus meshes.

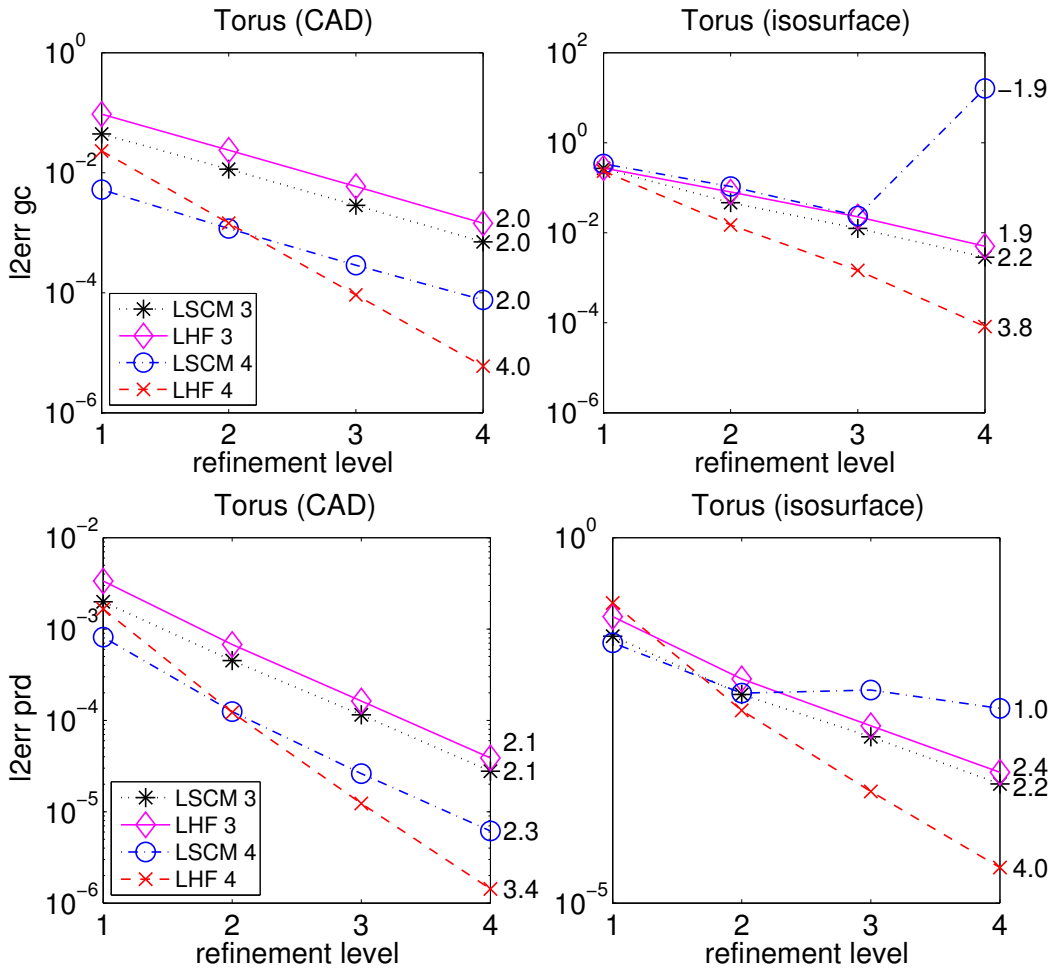


Figure 3.6: Comparison of L_2 errors in computed Gaussian curvatures (top) and principal directions (bottom) using degree-3 and 4 fittings for noise-free torus meshes.

Experiments for Noise-Free Open Surfaces In geometric modeling, surfaces often have boundaries and/or sharp features. It is therefore sometimes necessary to compute differential quantities using one-sided stencils. The boundaries (or sharp features) can adversely affect the computations of the differential quantities for two reasons: first, the neighborhood of a boundary vertex typically has fewer points, which may lead to ill-posed or ill-conditioned equations. Second, the stencil of boundary vertices are asymmetric, preventing statistical error cancellation associated with symmetry. Some existing methods for normal and curvature computations actually require the surface to be closed and rely on symmetry for convergence. Although polynomial fittings in general do not require closed surfaces, the accuracies

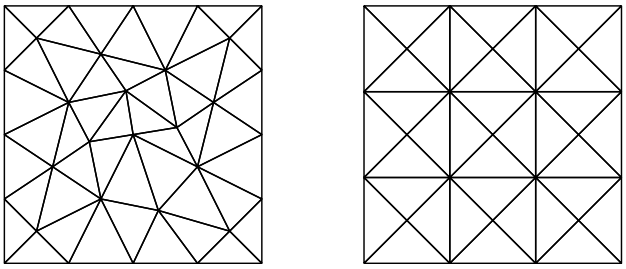


Figure 3.7: Test meshes for open surfaces.

of different methods may vary substantially near boundaries. To assess accuracies, we use a surface defined by the following function adopted from [66]:

$$z = f(x,y) = \exp\left(-\frac{81}{16}\left((x-0.5)^2+(y-0.5)^2\right)\right), \tag{3.3.2}$$

where $(x,y) \in [0,1] \times [0,1]$. We use two types of meshes, as shown in Figure 3.7, which we refer to as irregular and 4-8 meshes, respectively. These meshes are both well-shaped compared to the meshes from isosurfacing algorithms, but the variations of vertex valences can pose some challenges to the algorithms.

Figure 3.8 shows the L_2 and L_∞ errors in the computed normals for Xu-2, LHF-2, LOP-2, and LSCM-2. Because Xu's parametrization is limited to 1-ring neighborhoods that are topological disks, we excluded border vertices when computing errors for Xu-2. However, boundary vertices are included for all the other methods. Figure 3.9 shows the L_2 errors of the computed mean and Gaussian curvatures. Except for Xu-2, the other three methods delivered very similar results, but the convergence rates in L_2 errors of mean curvatures were lower than those for the torus, probably due to the loss of statistical error cancellations along boundaries. For Xu-2, it is worth noting that the curvatures failed to converge for both types of meshes due to insufficient number of points in the stencil.

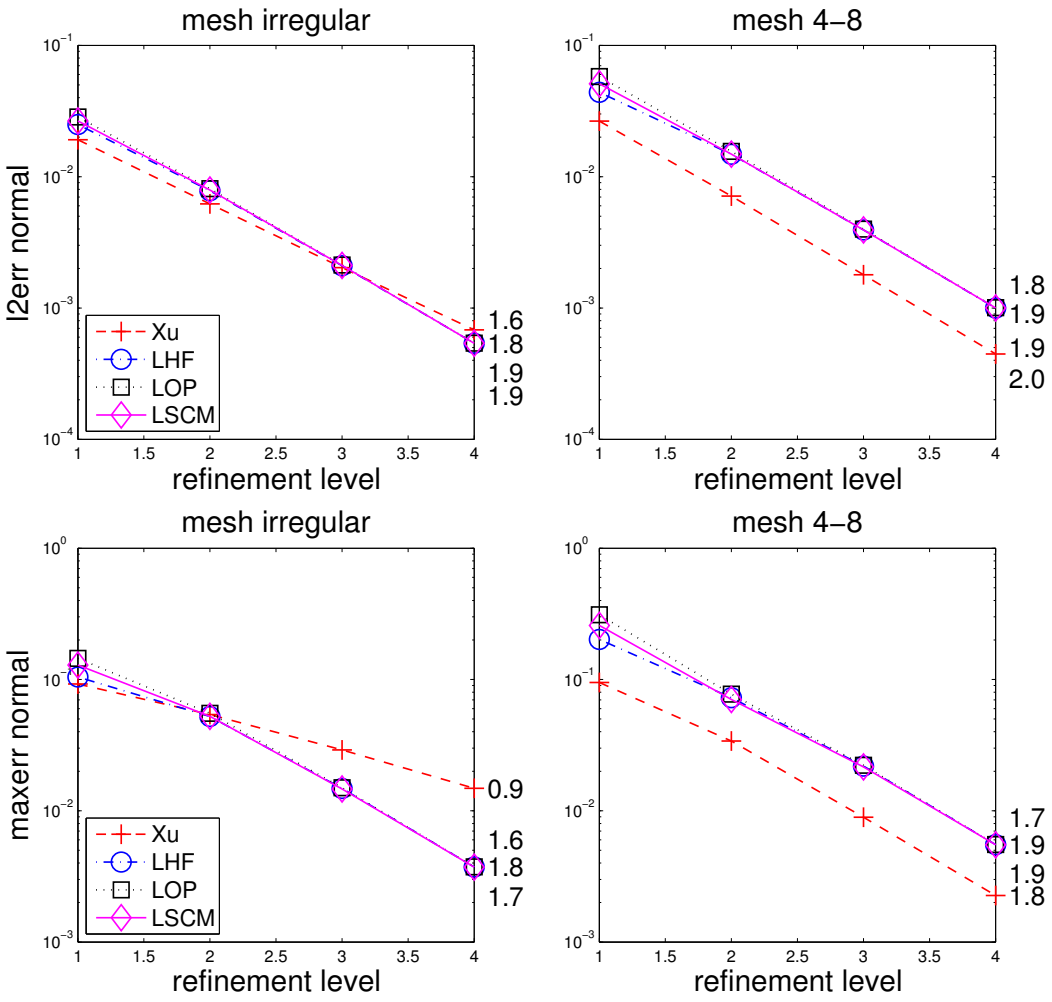


Figure 3.8: Comparison of L_2 (top) and L_∞ (bottom) errors in computed normals using degree-2 fittings for noise-free open surface meshes.

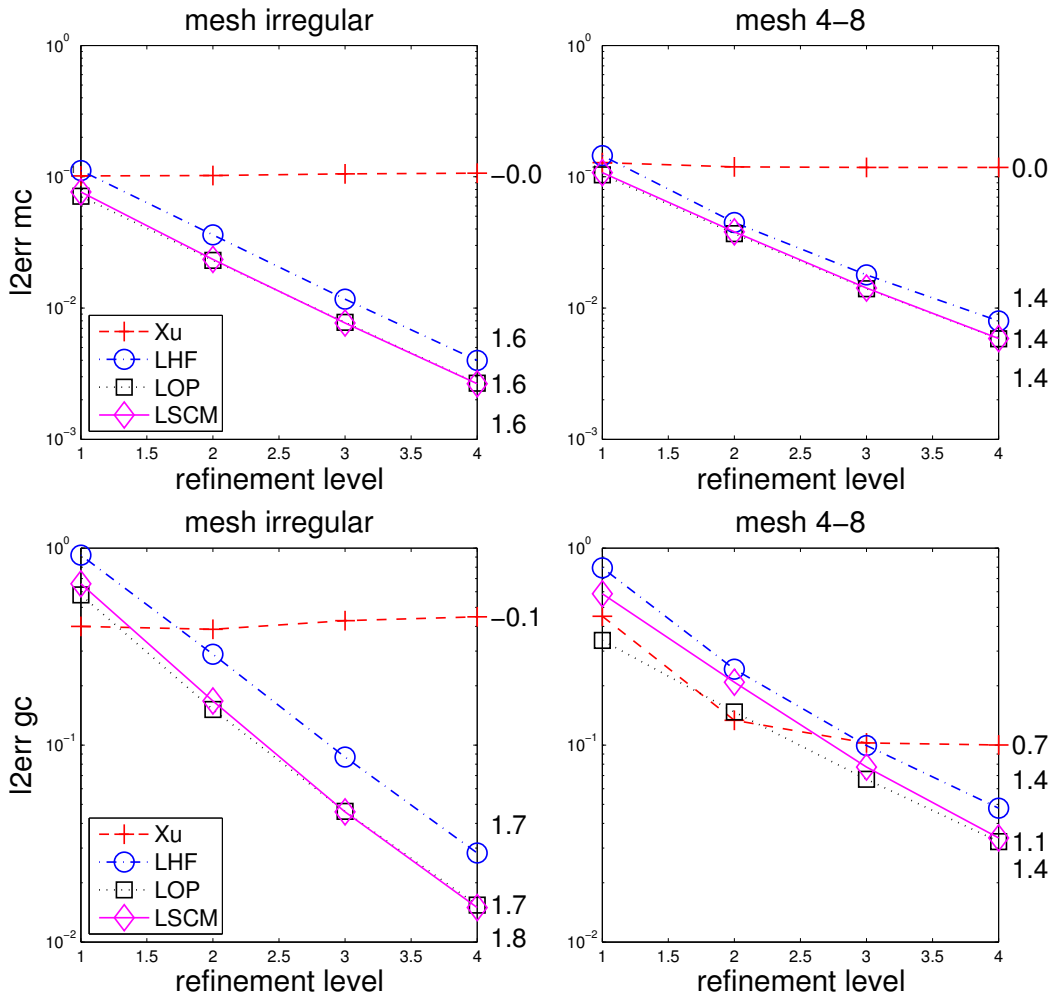


Figure 3.9: Comparison of L_2 errors in computed mean (top) and Gaussian (bottom) curvatures using degree-2 fittings for noise-free open surface meshes.

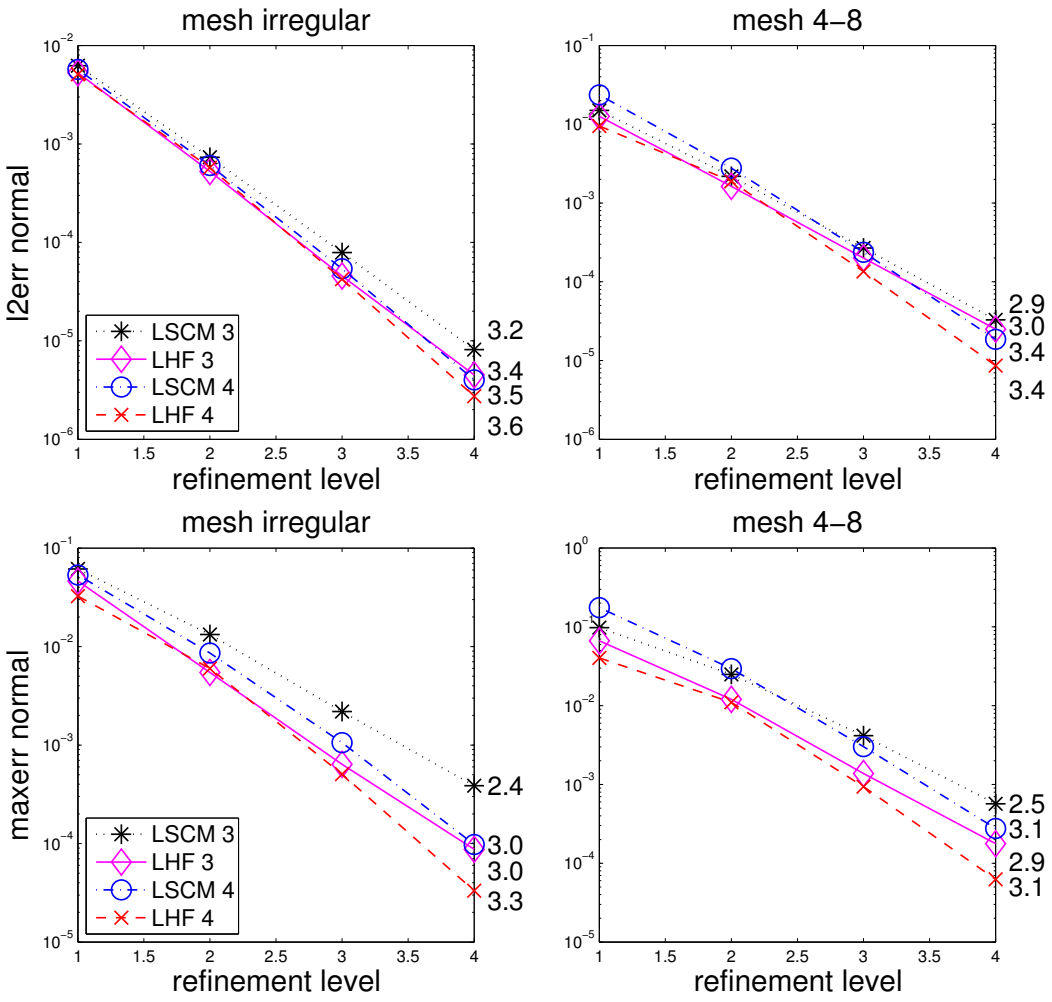
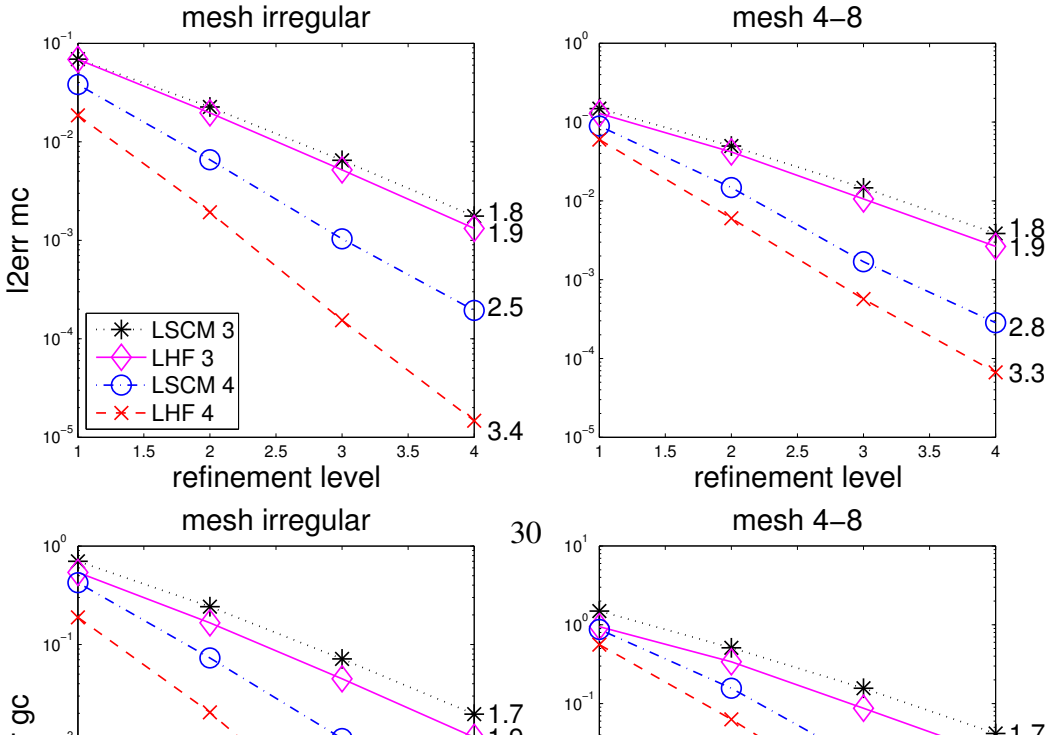


Figure 3.10: Comparison of L_2 (top) and L_∞ (bottom) errors in computed normals using degree-3 and 4 fittings for noise-free open surface meshes.



Figures 3.10 and 3.11 compare the errors for degree-3 and 4 fittings, namely LHF-3, LHF-4, LSCM-3, and LSCM-4. LHF consistently delivered better accuracy than LSCM for both degree-3 and 4 fittings, and LSCM-4 converged substantially slower than LHF-4, which confirms our observation that LSCM is unsuitable for high-degree fittings. From these tests, we conclude that LHF and similarly LOP are more flexible and robust than Xu's method, and they deliver higher accuracy than LSCM for high-degree fittings.

In the proceeding results, we included both the interior and boundary vertices, except for Xu's method. To study the effect of border vertices, Figures 3.12 shows the L_2 errors and their convergence rates in the computed normals using LHF and LSCM for boundary vertices and their 1-ring neighbors Figure 3.13 shows the corresponding results for the mean and Gaussian curvatures. It can be seen that these convergence rates are somewhat lower than those for the interior vertices but are very close to the theoretical analysis. These results indicate that these fitting methods do not rely on symmetry for convergence, although they can benefit from symmetry and statistical error cancellation in practice.

Experiments for Noisy Surface Meshes In many practical situations, the input points are not precisely on an analytical surface but contain noise or input errors. An example is the meshes extracted from isosurfaces, for which the vertex positions contain inherent errors. Another example is meshes obtained from solutions of numerical computations. We now assess the methods for these situations.

To study the effect of numerical errors, we use the meshes obtained from the isosurface function in MATLAB as in Section 3.3, but unlike in Section 3.3 we do not project the vertices onto the true torus. Table 3.2 shows the errors in the vertex positions for the surface mesh extracted using the isosurface function of MATLAB. We compute the position error for each vertex based on its distance to the torus, and report both L_2 and L_∞ errors for five different mesh resolutions. From Table 3.2, it seems that the isosurface function in MATLAB is second-order accurate, so convergence of curvature is not guaranteed theoretically.

For improved noise resistance, we increased the neighborhood size by half a ring for LHF, LOP, and LSCM. Figure 3.14 shows the L_2 and L_∞ errors of the computed normals for degree-2 fittings, and Figure 3.15 shows the L_2 errors in the computed mean and Gaussian curvatures, respectively. It can be seen that Xu-2 had the largest errors, nearly two orders of magnitude larger than the other methods. This again is primarily because Xu's method uses only 1-ring neighbors. For the other three methods, the L_2 errors of the normal directions converged at about second order, while the L_∞ errors converged slower than second order. The L_2 errors of the curvatures converged at about first order. These rates are higher than the theoretical predictions for noisy data, probably due to statistical error cancellation. We note

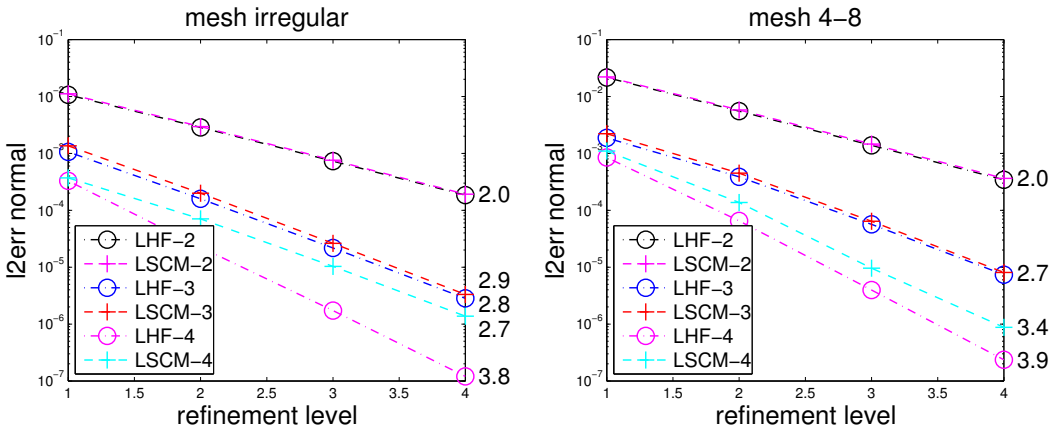


Figure 3.12: L_2 errors in computed normals for boundary vertices for noise-free open surface meshes.

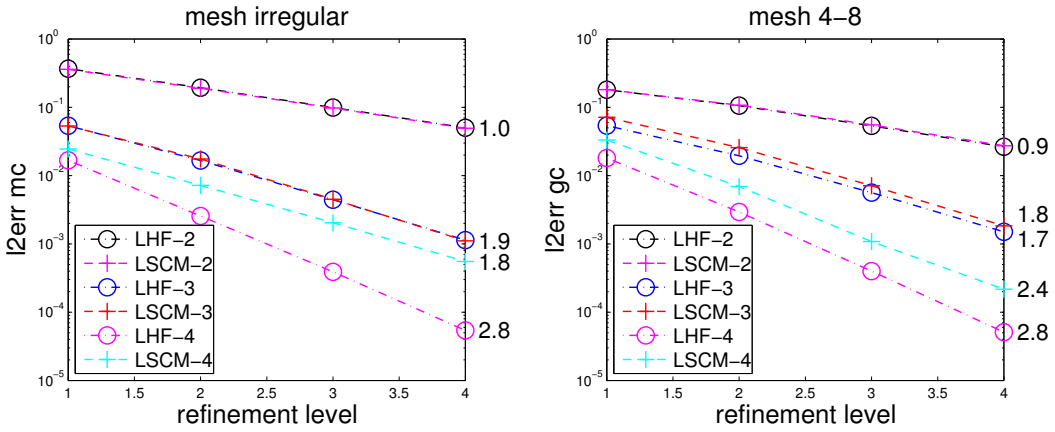


Figure 3.13: L_2 errors in computed mean (top) and Gaussian (bottom) curvatures for boundary vertices for noise-free open surface meshes.

Table 3.2: Input position errors of torus meshes extracted using isosurface function of MATLAB.

#volume cells	#surface verts.	average edge length	error		convergence rate	
			L_2	L_∞	L_2	L_∞
16^3	320	0.11	3.60e-3	9.30e-3	–	–
32^3	1760	0.048	1.12e-3	3.47e-3	1.69	1.42
64^3	7320	0.023	2.59e-4	9.03e-4	2.11	1.94
128^3	31136	0.011	6.71e-5	2.25e-4	1.95	2.00
256^3	122464	0.0057	1.65e-5	5.70e-5	2.03	1.98

that when a first-order isosurface algorithm is used, we observed no convergence and sometimes even divergence for the curvatures.

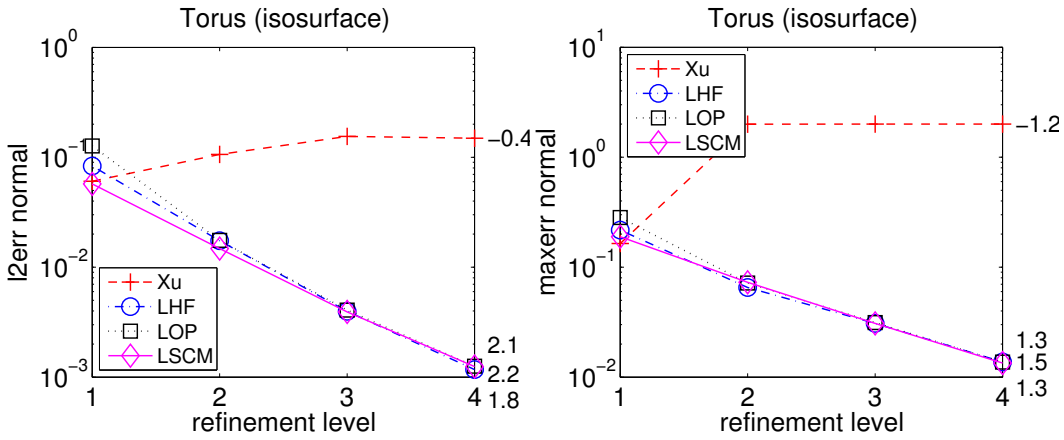


Figure 3.14: Comparison of L_2 (left) and L_∞ (right) errors in computed normals using degree-2 fittings for noisy torus.

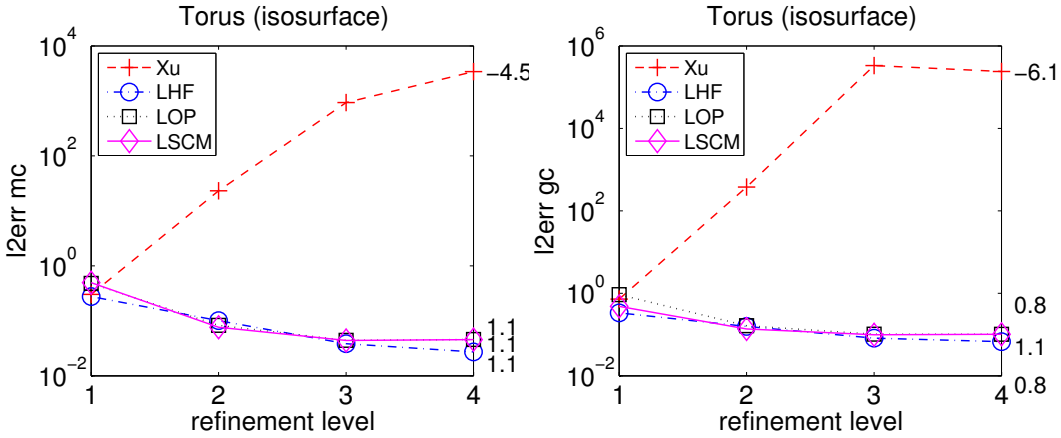


Figure 3.15: Comparison of L_2 errors in computed mean (left) and Gaussian curvatures (right) using degree-2 fittings for noisy torus.

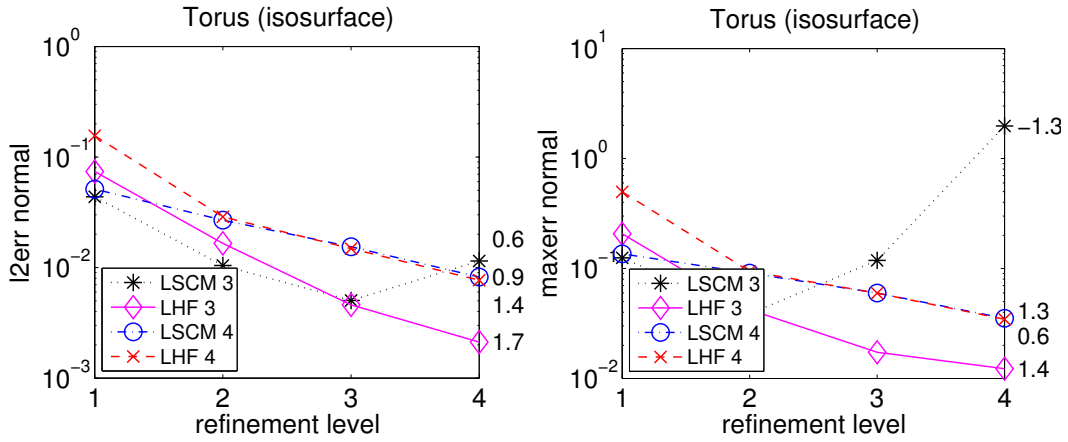


Figure 3.16: Comparison of L_2 (left) and L_∞ (right) errors in computed normals using degree-3 and 4 fittings for noisy torus.

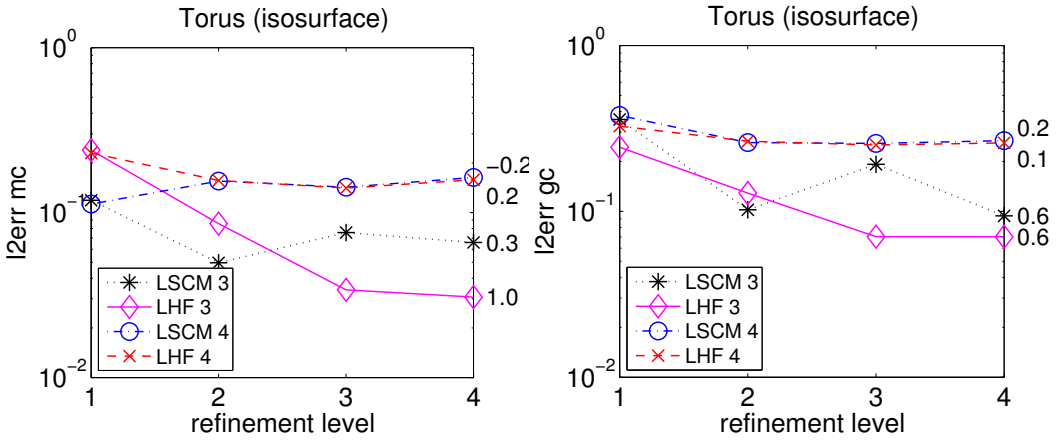


Figure 3.17: Comparison of L_2 errors in computed mean (left) and Gaussian curvatures using degree-3 and 4 fittings for noisy torus.

For completeness, we also present the results of cubic and quartic fittings in Figures 3.16 and 3.17 for the computed normals and curvatures, respectively. The results of these higher-degree fittings were worse than those for quadratic fittings, because a degree-d fitting requires the input errors to be order $d+1$ or higher for optimal convergence.

The input noise in the preceding test was due to numerical errors in isosurfacing, which are beyond our control. To verify our analysis, we report experiments with controlled errors, where the noise is added by perturbing the surface. We use the open surface meshes in this test, and for each vertex we perturb it by adding a noise

of αh^k along the normal direction, where α is a random number between 0 and 0.1 with Gaussian distribution, h is the average edge length, and k is chosen to be 2 or 3. For brevity, we report results only for quadratic fittings.

Figure 3.18 shows the L_2 and L_∞ errors of computed normals for $O(h^3)$ perturbation, and Figure 3.19 shows the corresponding results for $O(h^2)$ perturbations. Figure 3.20 shows the L_2 errors of the computed mean and Gaussian curvatures for $O(h^3)$ perturbation, and Figure 3.21 shows the corresponding results for $O(h^2)$ perturbations. We excluded boundary vertices when computing errors for Xu-2 but included boundary vertices for the other methods. Note that for the normals, each method performed nearly identically for the two different orders of perturbations. Xu-2 again delivered slightly better results than the others for the 4-8 mesh due to its use of a smaller neighborhood. However, for curvatures Xu-2 failed to converge again due to insufficient number of points, and the other methods delivered higher than linear convergence for $O(h^3)$ perturbations and generally lower than linear convergence for $O(h^2)$ perturbations, consistent with the theoretical analysis.

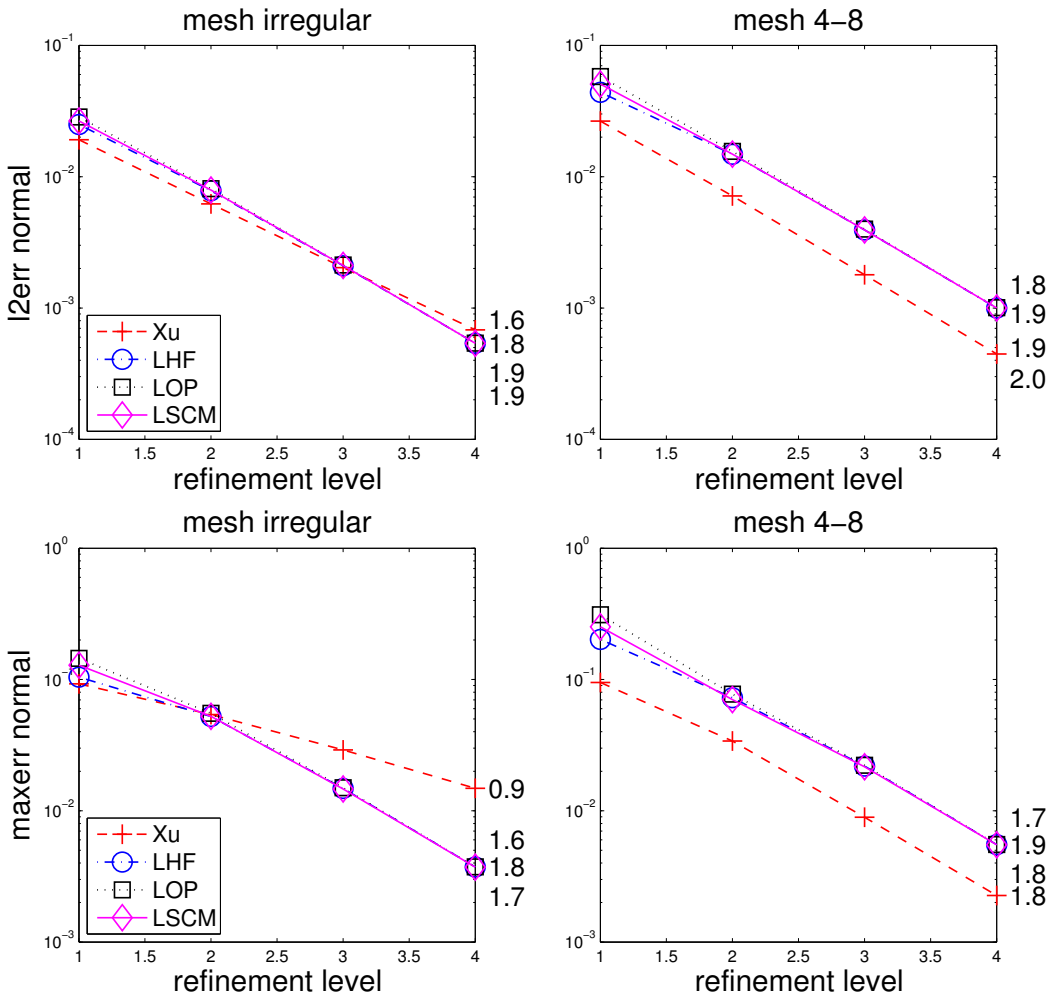


Figure 3.18: Comparison of L_2 (top) and L_∞ (bottom) errors of computed normals using degree-2 fittings for noisy open surface with third-order perturbation.

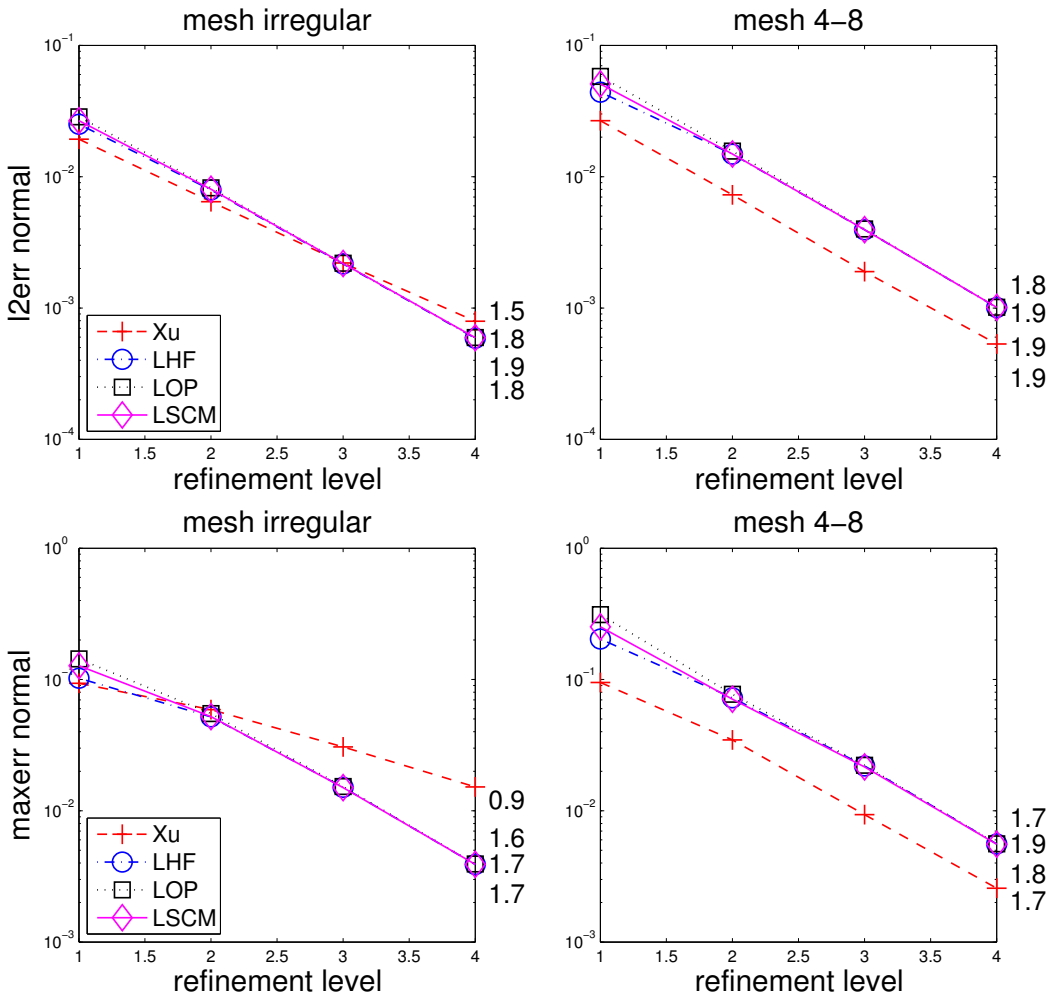


Figure 3.19: Comparison of L_2 (top) and L_∞ (bottom) errors of computed normals using degree-2 fittings for noisy open surface with second-order perturbation.

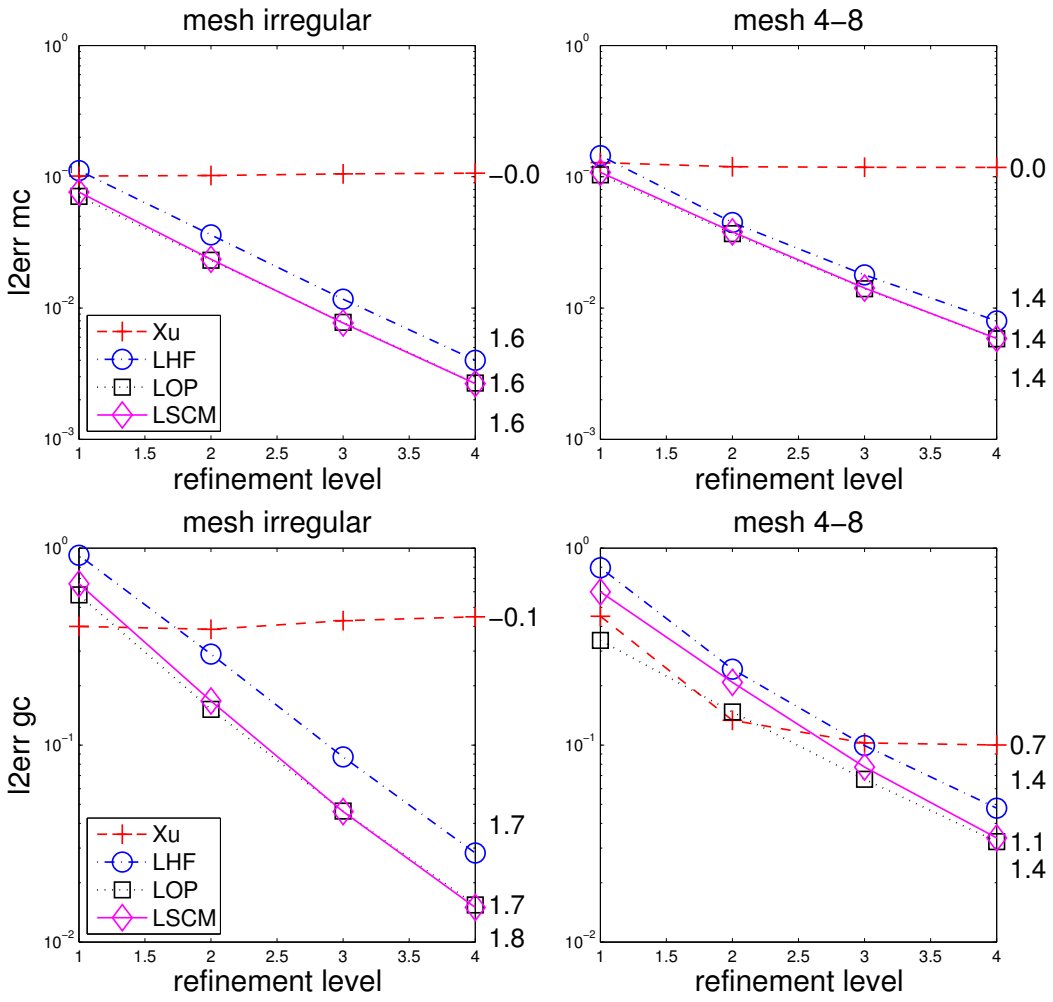


Figure 3.20: Comparison of L_2 errors of computed mean (top) and Gaussian (bottom) curvatures using degree-2 fittings for noisy open surface with third-order perturbation.

Comparison of Efficiency A key practical consideration in choosing a proper method is runtime efficiency. Table 3.3 reports the execution times of the different methods for the well-shaped torus mesh. We implemented the methods of Xu, LOP, and LHF in MATLAB and then converted the MATLAB code into C using Agility MCS (www.agilityds.com). For Xu's method, we used the singular value decomposition procedures in LAPACK [1]. For least-squares conformal mapping, we used the C++ implementation of the parametrization algorithm in CGAL and used the same C code as for LOP for the other numerical computations. All the codes were compiled using gcc 4.2.4, with optimization enabled. We performed

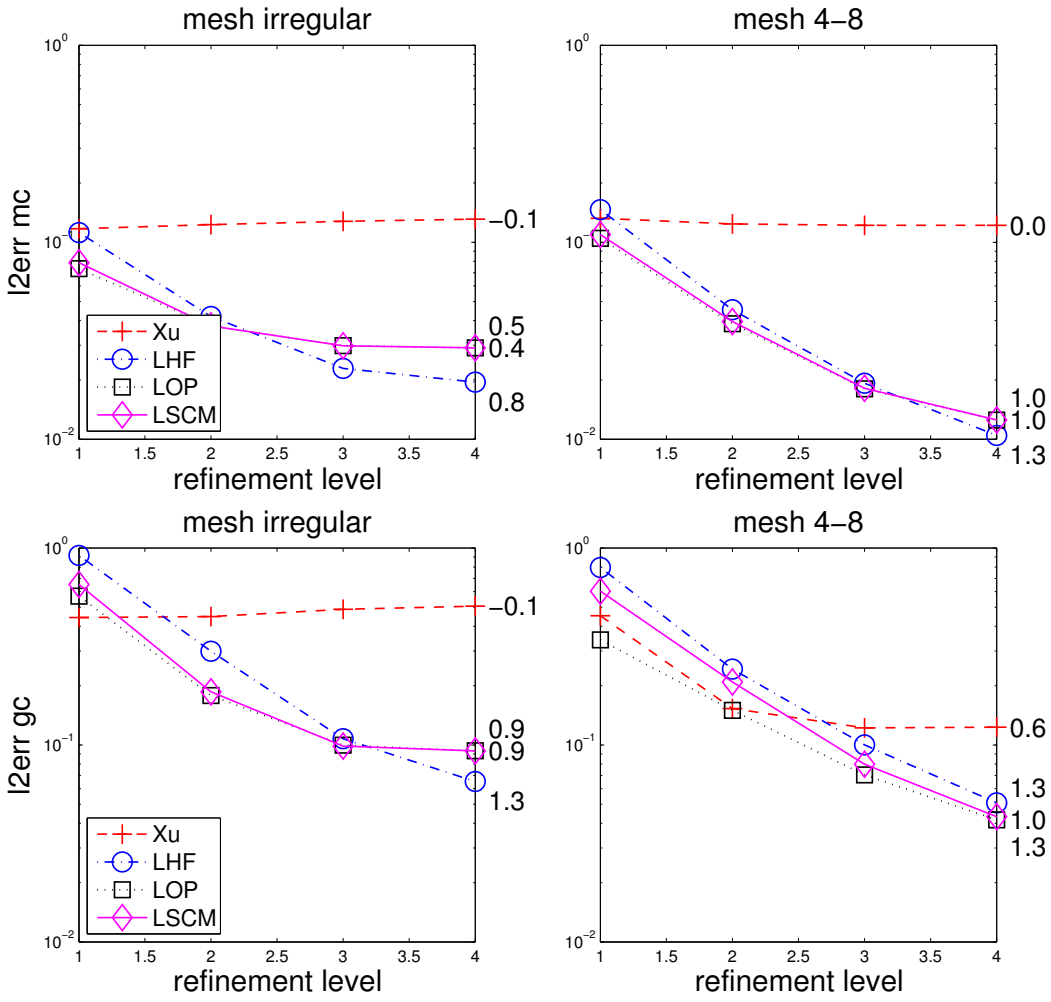


Figure 3.21: Comparison of L_2 errors of computed mean (top) and Gaussian (bottom) curvatures using degree-2 fittings for noisy open surface with second-order perturbation.

Table 3.3: Comparison of execution times in seconds of different methods for torus mesh.

Mesh	#verts.	#tris.	Xu-2	LOP-2	LHF-2	LHF-3	LHF-4	LSCM-2	LSCM-3	LSCM-4
1	1338	2676	0.035	0.011	0.0061	0.013	0.028	0.20	0.36	0.66
2	5246	10492	0.12	0.039	0.023	0.050	0.11	0.81	1.43	2.65
3	21156	42312	0.51	0.16	0.096	0.20	0.44	3.32	5.74	10.58
4	85208	170416	2.06	0.65	0.38	0.83	1.81	13.66	23.21	43.16

the tests on a Linux computer with a 3GHz Intel Duo Core Pentium 4 processor and 2GB of RAM.

As can be seen in the table, LSCM was roughly an order of magnitude more expensive than the others. This is partially due to the overhead of the data structures in CGAL and the high cost for solving linear system for the conformal parametrization. With a better optimized procedure for conformal parametrizations, LSCM may become more competitive in cost. Xu's method was the second most expensive, even though it uses only 1-ring neighbors. This is primarily because of its use of SVD, which is substantially more expensive than QR factorization. The most efficient algorithm was LHF. The effective performance of LHF-2 is about 450 thousand triangles per second, and the cost approximately doubles when the degree of fitting is increased by one. For comparison, we also tested Jet_fitting_3 of [10] on the same computer, and the effective performance of Jet_fitting_3 for second-order fitting was about 28 thousand triangles per second, consistent with their results reported in [10]. Our algorithm and implementation is more than 16 times faster than Jet_fitting_3, probably because of their use of SVD as well as the the overhead of the data structures in CGAL. LOP slightly under-performs LHF, because the linear system for LOP involves three right-hand-side vectors (instead of one in LHF) and also the formulas for parametric surfaces are more complex than those for height functions. From these experimental results, it seems that LHF offers the best combination of efficiency, accuracy, simplicity, and robustness.

Chapter 4

High Order Surface Reconstruction

In chapter 3, we use our local polynomial fitting computational framework to calculate normal and curvature on each vertex over a discrete surface mesh. A natural following question is: can we use this framework to reconstruct a continuous high-order surface from the same input data? This problem is important for many meshing operations, such as generating high-order finite elements, mesh refinement, mesh smoothing and mesh adaptation. In many of these problems, a continuous CAD model may not be available. Instead, only a surface mesh, typically with piecewise linear or bilinear faces, is available.

In this chapter, we consider the problem of reconstructing a highly accurate, continuous geometric support from a given surface mesh. We refer to this problem as *high-order surface reconstruction*. Besides meshing, this reconstruction problem also arises in computer graphics [17] and geometric modeling [63]. In general, the high-order reconstruction should satisfy some (if not all) of the following requirements:

Continuity: The reconstructed surface should be continuous to some degree (e.g., C^0, C^1, or C^2 continuous, depending on applications).

Feature preservation: The reconstruction should preserve sharp features (such as ridges and corners) in the geometry.

Geometric accuracy: The reconstruction should be accurate and asymptotically convergent to the exact surface to certain order under mesh refinement.

Stability: The reconstruction should be numerically stable and must not be oscillatory under noise.

Different applications may have emphasis on different aspects of the problem. For example, in computer graphics and geometric design, the visual effect may be

the ultimate goal, so smoothness and feature preservation may be more important. Therefore, the methods for such applications tend to focus on the first two issues, and the numerical issues of asymptotic convergence and stability are mostly ignored. Our focus is on meshing for finite element or finite volume computations, for which these numerical issues are very important. Indeed, the low order of accuracy of the geometry would necessarily limit the accuracy of the solutions of differential equations, and numerical instabilities and excessive oscillations can have even more devastating effect to numerical simulations. Some efforts, such as isogeometric analysis [32], aim to improve accuracy by using continuous CAD models directly in numerical simulations, but CAD models may sometimes be too complicated to be used directly, such as in moving boundary problems. When a mesh is involved, a high-order reconstruction may sometimes be the best option.

We introduce two methods, called *Weighted Averaging of Local Fittings* (*WALF*) and *Continuous Moving Frames* (*CMF*), for reconstructing a feature-preserving, high-order surface from a given surface mesh. We assume the vertices of the mesh accurately sample the surface and the faces of the mesh correctly specify the topology of the surface. The two methods both utilize the numerical techniques of weighted least squares approximations and piecewise polynomial fittings. They apply to surface meshes composed of triangles and/or quadrilaterals, and also to curves (such as ridge curves on a surface). Unlike existing methods, which are typically only first or second order accurate, our methods can achieve third- and even higher order accuracy, while guaranteeing global C^0 continuity. For its weighted least squares nature, these methods are also tolerant to noise. We describe the theoretical framework of our methods, and also present experimental comparisons of our methods against some others, and its applications in a number of meshing operations. In addition, we show how to calculate high order numerical surface integration based on WALF method.

4.1 Two Methods of Surface Reconstruction

The local polynomial fitting method described in Section 2 only applies locally at each individual vertex of a mesh. There was no coordination among the local fittings at different vertices, so the method does not reconstruct a global continuous surface. To construct a continuous surface, there are at least three different options:

1. compute multiple local fittings at vertices and then compute a weighted averaging of these fittings;
2. enforce continuity of local coordinate frames and weights for local fittings;
3. introduce new control points to define continuous/smooth surface patches.

Most methods in the literature use the latter two options. For example, the moving least squares [42] uses the second option to construct a C^∞ surface from a point cloud. Walton's method [63] adopted by Yams [21, 20] uses the third option. In this section, we describe two methods that belong to the first two categories, respectively. These two methods can be used to construct methods that belong to the third category, such as finding the locations of extra control points of high-order finite elements, as we will describe in section 4.2.1. For simplicity, we will first focus on triangular meshes for smooth surfaces in this section, and will present the extension to quadrilateral meshes and for meshes with sharp features in the next section when describing their applications.

4.1.1 Weighted Averaging of Local Fittings (WALF)

A simple approach to construct a high-order surface is to compute a weighted average of the local fittings at vertices. We refer to this approach as *Weighted Averaging of Local Fittings* (*WALF*). To achieve continuity of the surface, the weights used by the weighted averaging must be continuous over the mesh. One such a choice is the barycentric coordinates of the vertices over each triangle. Consider a triangle composed of vertices x_i, $i = 1,2,3$, and any point p in the triangle. For each vertex x_i, we obtain a point q_i for p from the local fitting in the local uvw coordinate frame at x_i, by projecting p onto the uv-plane. Let ξ_i, $i = 1,2,3$ denote the barycentric coordinates of p within the triangle, with $\xi_i \in [0,1]$ and $\sum_{i=1}^{3} \xi_i = 1$. We define

$$q(u) = \sum_{i=1}^{3} \xi_i q_i(u) \tag{4.1.1}$$

as the approximation to point p. Figure 4.1 shows a 2-D illustration of this approach, where ξ_i are the barycentric coordinates of point p within the edge x_1x_2.

WALF constructs a C^0 continuous surface, as can be shown using the properties of finite-element basis functions: The barycentric coordinates at each vertex of a triangle corresponds to the shape function of the vertex within the triangle, and the shape function of the vertex in all elements forms a C^0 continuous basis function (i.e., the linear pyramid function for surfaces or the hat function for curves). Let ϕ_i denote the basis function associated with the ith vertex of the mesh, and it is zero almost everywhere except within the triangles incident on the ith vertex. Therefore, q can be considered as a weighted average of the polynomials at all the vertices,

$$q(u) = \sum_{i=1}^{n} \phi_i(u) q_i(u), \tag{4.1.2}$$

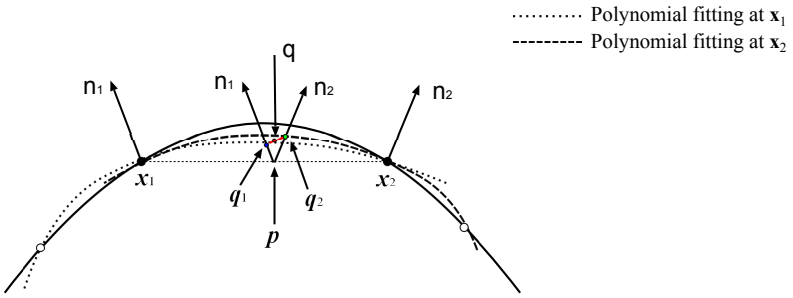


Figure 4.1: 2-D illustration of weighted averaging of local fitting. The black (dashed) curve indicates the exact curve. The blue (darker, solid) and green (lighter, solid) curves indicate the fittings at vertices x_1 and x_2, respectively. q is the WALF approximation of point p and is computed as a weighted average of the points q_1 and q_2 on the blue and green curves, respectively.

and then it is obvious that q is C^∞ within each triangle and C^0 over the whole mesh.

The idea of WALF is intuitive, but the analysis of its order of accuracy is by no means straightforward. If the coordinate systems were the same at all vertices, then the analysis would have been easy, as q would have inherited the accuracy of q_i. In WALF, the local fittings at the three vertices of a triangle are in general in different coordinate systems, and this discrepancy of coordinate systems can lead to additional error terms. Under the same assumptions as Proposition in Section 4.3.2, we obtain the following property of WALF.

Proposition: Given a mesh whose vertices approximate a smooth surface Γ with an error of $O(h^{d+1})$, the distance between each point on the WALF reconstructed surface and its closest point on Γ is $O(h^{d+1}+h^6)$.

Proof: We analyze the accuracy for triangles, but it helps to refer to Figure 4.1 for a 2-D illustration. Let q_i^* denote the intersection of the exact surface with the direction n_i from a point p (i.e., q_i^* is the exact solution for q_i in the fitting at vertex x_i). Let $\bar{q}$ denote the closest point to point $q=\sum_{i=1}^3 \xi_i q_i$ on the exact surface. Let $q^*=\sum_{i=1}^3 \xi_i q_i^*$ and $\bar{q}^*$ be its closest point on the surface. Then,

$$\|q-\bar{q}\| \le \|q-\bar{q}^*\| \le \|q-q^*\| + \|q^*-\bar{q}^*\|. \tag{4.1.3}$$

For $\|q-q^*\|$, note that $\|q-q^*\| \le \sum_{i=1}^3 \xi_i \|q_i-q_i^*\|$. When dth degree fittings are used, $\|q_i-q_i^*\| = O(h^{d+1})$, so

$$\|q-q^*\| = O(h^{d+1}). \tag{4.1.4}$$

For $\|q^*-\bar{q}^*\|$, note that $\|q_1^*-q_2^*\| = |\cos\theta_1|\|q_1^*-p\| + |\cos\theta_2|\|q_2^*-p\|$, where θ_i is the angle between $q_1^*q_2^*$ and $\hat{n}_i$. Note that $\cos\theta_i = O(h)$, since by assumption

$\hat{n}_i$ is at least a first order approximation to the normal at x_i, and also a first order approximation to the normals at q_1^* and q_2^*, whereas the line segment $q_1^* q_2^*$ is at least a first order approximation to a tangent direction at q_1^* and q_2^*. Because p is a point on triangle $x_1x_2x_3$, whose edge length is $O(h)$ by assumption, $\|p - q_i^*\| = O(h^2)$. Therefore,$\|q_1^* - q_2^*\| = O(h^3)$, and similarly for $\|q_1^* - q_3^*\|$ and $\|q_2^* - q_3^*\|$. Because q^* is a point on triangle $q_1^* q_2^* q_3^*$,

$$\|q^* - \bar{q}^*\| \quad = \quad O(h^3)^2 = O(h^6). \tag{4.1.5}$$

Combining (4.1.3-4.1.5), we conclude that $\|q - \bar{q}\| = O(h^{d+1}) + O(h^6) = O(h^{d+1} + h^6)$. □

The above proposition gives an upper bound of the error, and it shows that the error term is high order. The $O(h^6)$ term is due to the discrepancy of local coordinate systems at different vertices. However, in most applications we expect $d < 6$, so the total error would be dominated by the degree of polynomials used in the least squares fitting.

4.1.2 Continuous Moving Frames (CMF)

WALF is a simple and intuitive method, but its order of accuracy has a theoretical limit. We now present a method that can overcome this limitation by using local coordinate frames that move continuously from point to point. We refer to such a scheme as *continuous moving frame* (*CMF*). The basic idea is to use the finite-element basis functions to construct continuous moving frames and weights for local fittings. In particular, assume each vertex has an approximate normal direction at input. Consider a triangle $x_1x_2x_3$ and any point p in the triangle. Let $\hat{n}_i$ denote the unit vertex normal at the ith vertex. We compute a normal at p as

$$\hat{n} = \sum_{i=1}^{3} \xi_i \hat{n}_i \Bigg/ \left\| \sum_{i=1}^{3} \xi_i \hat{n}_i \right\|. \tag{4.1.6}$$

Given $\hat{n}$, we construct a local *uvw* coordinate system along axes $\hat{s}$, $\hat{t}$, and $\hat{n}$, where $\hat{s}$ and $\hat{t}$ form an orthonormal basis of the tangent plane. Within this local coordinate frame, we formulate the weighted least squares as

$$\|\Omega V X - \Omega F\|_2, \tag{4.1.7}$$

where V again is the generalized Vandermonde matrix, and Ω is the weight matrix. In practice, the Vandermonde matrix for a point p should involve a small stencil in the neighborhood of the triangle. We use the union of the stencils of the three

vertices of the triangle. Conceptually, it is helpful to consider the Vandermonde matrix involving all the points of the mesh, but the weight matrix Ω assigns a zero weight for each point that is not in the stencil. For the reconstructed surface to be smooth, it is important that Ω is continuous as the point p moves within the geometric support of the mesh. In addition, it is also important that Ω is invariant of rotation of tangent plane (i.e., be independent of the choice of $\hat{s}$ and $\hat{t}$).

We define the weight as follows: For p within the triangle $x_1x_2x_3$, we first define a weight for each vertex (for example the jth vertex) in the mesh that is in the stencil of vertex i as

$$w_{ij} = \left(\hat{n}_i^T \hat{n}_j\right)^+ / \left(\sqrt{\|x_j - p\|^2 + \varepsilon}\right)^{d/2}, \tag{4.1.8}$$

and set $w_{ij} = 0$ otherwise, where ε is a small number, and

$$\left(\hat{n}_i^T \hat{n}_j\right)^+ = \begin{cases} \hat{n}_i^T \hat{n}_j & \text{if } \hat{n}_i^T \hat{n}_j \geq \eta \\ 0 & \text{otherwise} \end{cases} \tag{4.1.9}$$

for some small $\eta \geq 0$. Then for the weighting matrix Ω, the weight for vertex j is then $\sum_{i=1}^{3} w_{ij}$. Note that the introduction of ε in (4.1.8) is to prevent division by very small numbers when p is very close to a vertex. In practice, we set ε to be 0.01 times the average distance from p to the points in the stencil.

Similar to WALF, CMF constructs a C^0 continuous surface, because Ω, V, and F are all C^0 continuous, as long as the resulting linear system is well-conditioned. The accuracy of CMF follows that for weighted least squares approximation in [37], and we obtain the following property of CMF.

Given a mesh whose vertices approximate a smooth surface Γ with an error of $O(h^{d+1})$, the shortest distance from each point on the CMF reconstructed surface to Γ is $O(h^{d+1})$.

Relationship with Moving Least Squares. The idea of using moving frames is not new, and goes back to Élie Cartan in differential geometry. One of the incarnations of the idea of using moving frames for discrete surfaces is the so-called *moving least squares* (*MLS*) for point clouds [42]. CMF shares some similarities to MLS. In particular, they are both based on weighted least squares approximations within some local frames. However, they also differ in some fundamental ways. First, moving least squares uses global weighting functions that are exponential in the distance, and theoretically, MLS is C^∞. However, because global weighting functions are too expensive to compute, practical implementations typically truncate small weights to zeros, leading to a loss of continuity. In contrast, CMF uses only a local support by construction. Second, MLS does not guarantee the order of accuracy, because its weights are global and purely based on Euclidean distance. Although its convergence was conjectured in [42], we have observed that MLS sometimes fails to

converge even for simple geometries such as a torus. In contrast, CMF uses the mesh connectivity as a clue in selecting the stencils, instead of based on Euclidean distance. Third, CMF can take into account the normals in the weighting function, to filter out points across sharp features. This allows CMF to handle surfaces with sharp features in a natural way, which is important for meshing operations. On the other hand, it is difficult to treat sharp features in the framework of MLS. Because of their local supports, CMF is more easily adapted to treat sharp features, as we describe in the next section.

4.1.3 Safeguards Against Oscillations

High degree Taylor polynomials in general are accurate near the origin of the local coordinate system. However, higher degree polynomials also tend to be more oscillatory than lower degree polynomials. Such oscillations may be particularly problematic for WALF, for which we evaluate the polynomial in a neighborhood of the origin (i.e., within the faces incident on a vertex) instead of at the origin (i.e., the vertex) of the local coordinate system at a vertex. Therefore, large errors may occur if care is not taken. To address this issue, we introduce a safeguard to downgrade the degree of polynomial if necessary. In particular, when solving (2.3.3) to obtain the coefficients for a degree-d fitting, for $2 \leq p < d$ (where $p = j + k$ in (2.3.1)), we solve each pth-order coefficient twice: once by using a degree-p fittings, and once for a degree-d fitting. Let $\hat{c}_p$ denote the solution of the coefficient from the degree-p fitting, and c_p from the degree-d fitting. We consider $\hat{c}_p$ as a more reliable (i.e., less oscillatory) estimate of the coefficient. If $|\hat{c}_p - c_p| > \max\{|\hat{c}_p|, \varepsilon\}$, where ε is a small number such as 0.01, then we consider the degree-d fitting to be overly oscillatory, and decrease the degree of fittings by 1. We repeat the process to check the coefficients and reduce the degree of fittings until all the coefficients pass the test or the degree of polynomial is decreased down to 2. To demonstrate the effectiveness of our safeguards in Section 4.2.2, Table 4.1 shows the number of vertices for which the safeguards were invoked to downgrade the degrees of fittings for the coarsest torus mesh in Figure 4.2. This mesh had 328 vertices. About 26% of vertices were downgraded for degree-six fittings, and about 16% of vertices were downgraded for degree-five fittings.

4.1.4 Treatment of Sharp Features.

Sharp features, such as ridges and corners, are challenging problems in their own right. We have so far implemented a simple treatment for sharp features. First,

Table 4.1: Number of vertices for which safeguards were invoked for coarsest torus mesh.

	degree 2	degree 3	degree 4	degree 5	degree 6
#total downgraded	0	2	6	51	85
downgraded to degree 2	-	2	3	2	2
downgraded to degree 3	-	-	3	3	18
downgraded to degree 4	-	-	-	46	57
downgraded to degree 5	-	-	-	-	8

we identify feature edges and vertices and connect the feature edges to form ridge curves using an algorithm such as that in [33, 34]. We treat the ridge edges as internal boundaries within the mesh and require that the k-ring neighbors of vertices do not go across ridge curves. This is accomplished by virtually splitting the mesh along ridge curves in our mesh data structure. For ridge curves themselves, we separate them into sub-curves that do not have corners in their interior. For each sub-curve, we perform a high-order reconstruction using either WALF or CMF for curves. This treatment is sufficient for most meshing operations.

4.1.5 Experimental Results

We report some experimental results of our two proposed methods, and compare them with some other methods. We first show the mesh convergence study of WALF and CMF. While it is typically unnecessary to use schemes with higher than third or fourth order accuracy, to demonstrate the capabilities and limitations of these two methods, we report results with polynomials of up to degree 6. We performed our experiment using a torus with in-radius of 0.7 and outer-radius of 1.3, with an unstructured triangular mesh. We considered four levels of mesh refinement. The coarsest mesh has 328 vertices and 656 triangles, whereas the finest mesh has 85,276 vertices and 170,552 triangles. Figure 4.2 shows the coarsest mesh used in our experiments. In this test, we randomly generated 10 points on each face of the mesh, then project them onto a high-order surface constructed using WALF or CMF. We compute the error as the shortest distance from each approximate point to the torus.

Accuracy Figure 4.3 shows the L_∞ errors of WALF and CMF for the meshes. In the figure, the horizontal axis corresponds to the level of mesh refinement, and the vertical axis corresponds to the L_∞ errors. In the legends, the "degree" indicates the degree of polynomial fittings used, and "linear" indicates the error for linear

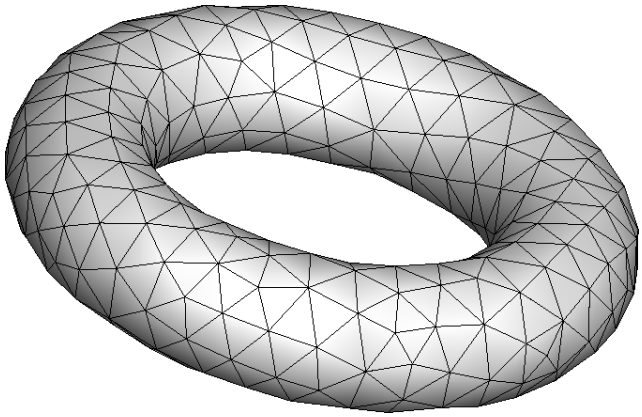


Figure 4.2: Coarse mesh of torus used in our mesh convergence study.

interpolation within triangles. We show the average convergence rates along the right of the plots for each curve, calculated as

$$\text{convergence rate} = \frac{\log\left(\text{error}_4/\text{error}_{\text{base}}\right)}{\log\left(h_4/h_{\text{base}}\right)}, \tag{4.1.10}$$

where error_i denotes the L_∞ error of all the randomly inserted points for the ith coarsest mesh, and h_i is the maximum edge length of the corresponding mesh.

From the figures, it is obvious that quadratic and higher-degree fittings produced far more accurate results than linear interpolation. Both WALF and CMF achieved a convergence rate of $(d+1)$ when d is odd and about $(d+2)$ when d is even. The superconvergence for even-degree fittings is likely due to statistical error cancellations of the leading error terms, which are of odd degrees. However, such error cancellations are not guaranteed when the points are very far from being symmetric, especially near boundaries or sharp features.

Some conclusions can be drawn from our comparisons between WALF with CMF. In terms of accuracy, we note that WALF gave smaller errors (up to 50% smaller for L_∞ errors and 75% smaller for L_1 errors) than CMF for finer meshes, although they delivered very similar convergence rates. The reason for the smaller errors for WALF was probably that WALF uses a smaller stencil for each polynomial fitting. Also notice that for coarse meshes, high-order polynomial fittings do not converge at expected rate, because the safeguards automatically reduce the degree to avoid large oscillations, so their convergence rates are the same as quadratic or cubic polynomials at first.

Approximate Continuity of Normals Both WALF and CMF construct C^0 continuous surfaces, so the surface normal and curvatures can be discontinuous across

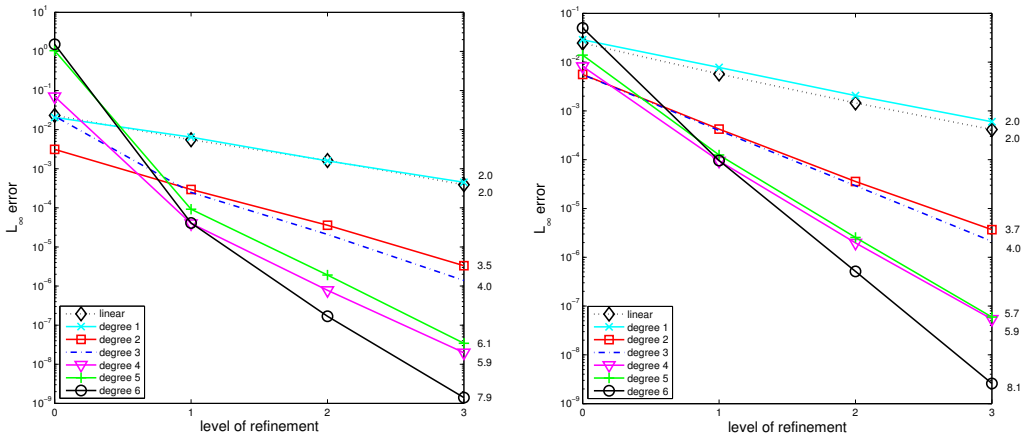


Figure 4.3: L_∞ errors of WALF (left) and CMF (right) under mesh convergence for torus. Both WALF and CMF achieve $(d+1)$st or higher order accuracy for degree-d polynomials.

edges and at vertices. However, this discontinuity (or jump) decreases as the degree of the fittings increases or as the mesh is refined. Specifically, given a mesh whose vertices approximate a smooth surface Γ with an error of $O(h^{d+1})$, where h denotes the maximum edge length and $d \leq 6$, it can be shown that the jump of surface normal across two adjacent face elements is $O(h^d)$ for both WALF and CMF. The proof of this convergence is similar to that of Proposition 2. Instead of proving it formally, we illustrate this convergence experimentally in Figure 4.4. The figure shows the errors in the normal directions at some randomly sampled points on the surface reconstructed by WALF. The error in the normal direction converges at a rate of at least dth order for dth degree fittings. Therefore, the discontinuity of differential quantities along edges is small (specifically, $O(h^d)$) when approximating a smooth surface. This is in contrast with some other methods that can achieve higher-order continuity at the cost of lower-order accuracy.

Comparison with Other Methods Besides WALF and CMF, some other methods have been developed for high-order reconstructions and been used in the meshing community. One method that is worth noting is that proposed by Walton [63] and adopted by Frey for surface meshing [20]. One property of Walton's method is that it achieves C^1 (or G^1) continuity for the reconstructed mesh. However, there does not seem to be any analysis of the accuracy of Walton's method in the literature. Figure 4.5 shows a comparison of the errors of Walton's method as well as WALF using quadratic and cubic fittings, with linear interpolation as the baseline

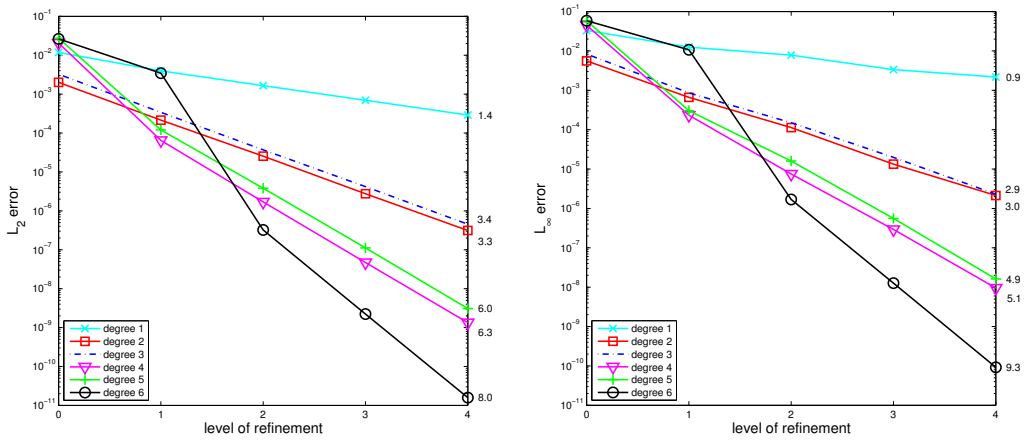


Figure 4.4: L_2 and L_∞ errors of normal directions of the surface reconstructed using WALF method.

of comparison. The two figures show the errors in L_∞ and L_2 norms for the torus. From the result, it is evident that Walton's method converges only linearly and is actually less accurate than linear interpolation for finer meshes. The inaccuracy of Walton's method is probably due to its dependence on the tangent directions of the edges, which are in general unavailable and can be estimated only to low accuracy. Therefore, it is obvious that C^1 (and in fact even C^∞) continuity does not imply accuracy of the reconstruction, although they may produce smooth appearance. On the other hand, high-order methods with C^0 continuity typically produce errors that are too small to cause any noticeable discontinuities.

Efficiency In terms of efficiency, WALF and CMF are comparable when approximating a single point because they both construct and solve a weighted least square problem. However, there is an important difference between the two methods: CMF does not actually reconstruct the whole surface, so whenever we need a point on the surface, a least square problem has to be solved. For WALF, however, once the local polynomials at the mesh vertices have been obtained, we have all the information needed to reconstruct the whole surface. Therefore, the coordinates and differential quantities at any point on the surface can be calculated by polynomial evaluation and weighted averaging. This distinction allows WALF to reuse the polynomial fittings at vertices, and hence it can have smaller amortized cost.

To demonstrate this difference, Tables 4.2 and 4.3 show the timing results of using piecewise linear as well as CMF and WALF methods with degrees 2 to 4, on a Linux desktop with a dual-core 3.16GHz Intel E8500 processor and 4GB of RAM.

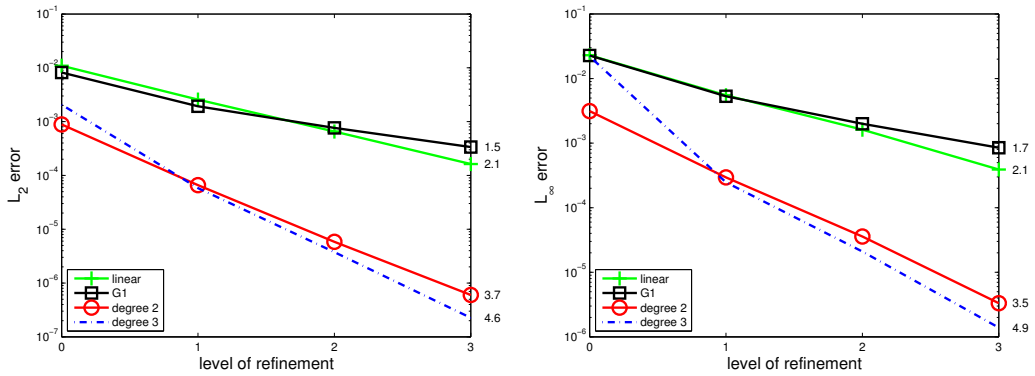


Figure 4.5: Comparison of errors using linear interpolation, Walton's method (labeled as G1), and WALF using quadratic and cubic fittings.

Table 4.2: Timing comparison (in seconds) when approximating one point on each triangle.

mesh	#triangle	linear	degree 2		degree 3		degree 4	
			CMF	WALF	CMF	WALF	CMF	WALF
1	658	0.0066	0.026	0.024	0.031	0.027	0.034	0.031
2	26,76	0.027	0.11	0.10	0.17	0.11	0.15	0.13
3	10,492	0.11	0.43	0.39	0.67	0.44	1.13	0.52
4	42,312	0.43	1.74	1.58	2.67	1.78	4.49	2.09
5	170,416	1.73	7.03	6.40	10.82	7.19	18.15	8.45

We list the times for approximating one point on each face in Table 4.2 and for approximating six points per face in Table 4.3. The results are comparable in the former case, but WALF significantly outperforms CMF in the latter case, as we expected.

4.2 Applications to Meshing and Finite Elements

The targeted applications for high-order surface reconstruction in this dissertation are meshing for finite element analysis. We hereafter further customize our framework for meshing and then apply the resulting techniques to meshing operations.

Table 4.3: Timing comparison (in seconds) when approximating six points on each triangle.

mesh	#triangle	linear	degree 2		degree 3		degree 4	
			CMF	WALF	CMF	WALF	CMF	WALF
1	658	0.039	0.14	0.098	0.16	0.10	0.19	0.11
2	26,76	0.16	0.60	0.40	0.94	0.41	0.83	0.44
3	10,492	0.63	2.38	1.58	3.74	1.63	6.47	1.73
4	42,312	2.51	9.44	6.37	14.81	6.59	25.73	6.95
5	170,416	10.16	38.12	25.64	59.84	26.71	103.89	28.06

4.2.1 High-Order Finite Elements

An application of our method is to construct a high-order (in particular, quadratic or cubic) finite element mesh from a given mesh with only linear elements and accurate vertex coordinates. This problem has practical relevance, because some mesh generators produce only a mesh with linear elements from an accurate CAD model, and it may be desirable to reconstruct high-order elements without having to access the CAD model. In addition, these high-order elements can also be used to define a C^0 continuous surface with high-order accuracy.

We formally stated the problem as follows: Given a mesh with linear elements, assume the vertices are sufficiently accurate (e.g., they are exact or are at least third or fourth-order accurate), construct a finite element mesh with quadratic or cubic elements with third or fourth order accuracy.

When using high-order surface reconstruction, this problem can be solved in the following procedure:

1. For each element σ, loop through its edges. If there is not an element that is abutting σ and has an ID smaller than that of σ, assign a new node ID to each node on the edge; if a node ID has already been assigned in the adjacent element, retrieve the node ID from that adjacent element.

2. Loop through all elements to assign new nodes IDs for new nodes on faces.

3. Expand the array for nodal coordinates, and evaluate the position for each new vertex using high-order surface reconstruction.

We have implemented this procedure for reconstructing quadratic and cubic elements from linear triangles or bilinear quadrilaterals using WALF and CMF. Figure 4.6 shows an example for meshes generated with quadratic and cubic elements for a triangulated torus. The high-order schemes produced notable improvements to

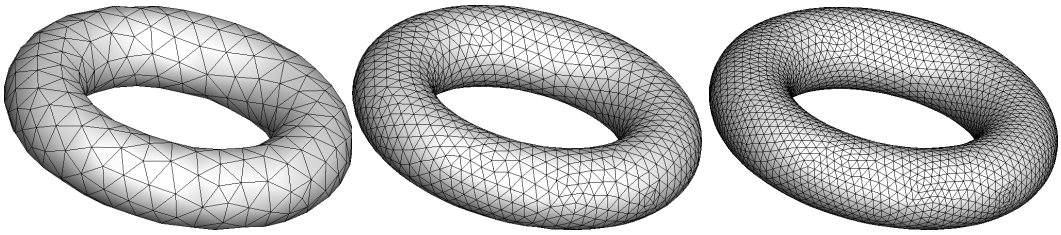


Figure 4.6: Illustration of generating high-order finite elements from given mesh. Left: a coarse torus with linear elements; Middle: same mesh but with quadratic elements, visualized by decomposing each triangle into four triangles. Right: same mesh but with cubic elements, visualized by decomposing each triangle into nine triangles.

the smoothness of the surface to linear approach, and the overall errors are significantly smaller. Note that in actual numerical simulations, not only the geometry but also some field variables defined on the mesh need to be reconstructed to high order. The same high-order reconstruction we presented can be used for that purpose, but it is beyond the scope of this dissertation.

4.2.2 Uniform Mesh Refinement

A problem related to generating a high-order mesh is the uniform refinement of a surface mesh. The problem may be stated as follows: Given a coarse surface mesh with sufficiently accurate vertex coordinates, construct a finer surface mesh by subdividing the elements. Like the previous problem, uniform mesh refinement introduces additional nodes to edges and/or faces, but in addition it also introduces new edges to subdivide the elements. Figure 4.7 shows an example of refining a quadrilateral mesh with sharp features. Note that if the new points are added onto the linear edges and faces, the refined mesh is not only inaccurate but also noticeably nonsmooth, as evident in the left image of the figures. This problem is resolved by using high-order reconstructions. The right image of Figure 4.7 shows the refinement result using WALF with cubic fittings and virtual splitting of the mesh along ridge curves. We show the results both with safeguard and without safeguards describe in Section 4.1.3. It is evident that the surface obtained from WALF is overall much smoother and accurate, and the safeguards significantly improves the accuracy and robustness of the method.

As another example, the bottom-right image of Figure 4.8 show a refinement dragon mesh using WALF and feature treatments. The resulting meshes are much smoother and more accurate. This procedure can be useful for generating high-quality finer resolution meshes from a mesh without requiring access to the CAD model.

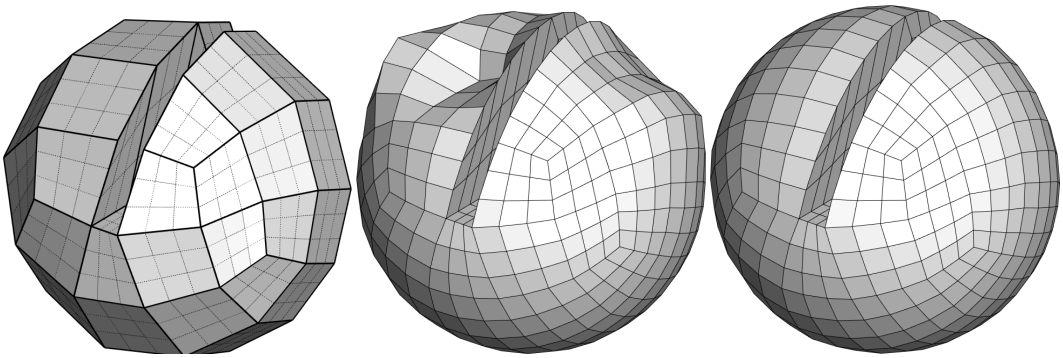


Figure 4.7: Example of refining quadrilateral mesh by subdividing each element into nine quadrilaterals. In the left image, dark lines show the original coarse mesh, and dashed lines show a mesh obtained by bilinear interpolation. The middle image shows the refined mesh using cubic fitting with WALF but without safeguards, and the right image shows the refined mesh with WALF and safeguards.

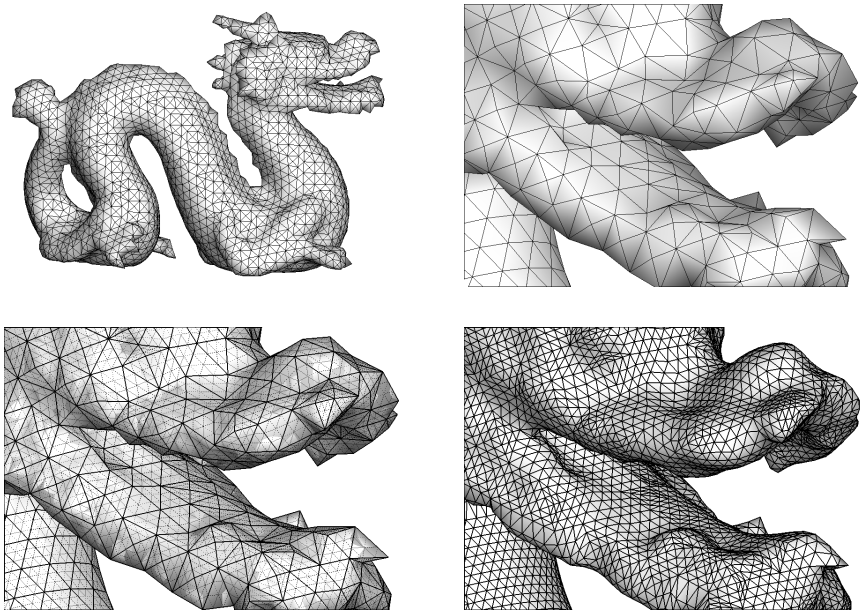


Figure 4.8: Example of refining triangular mesh by subdividing each element into four triangles. Upper row shows a dragon mesh and the zoom in near the head. Lower left image shows a refined mesh using linear interpolation. Lower right image shows refined mesh using WALF with quadratic fitting and feature treatments.

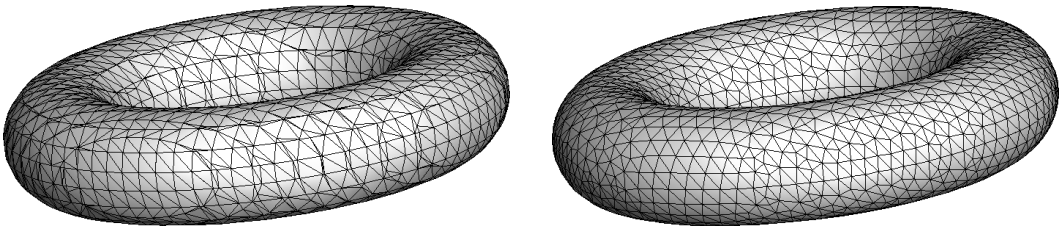


Figure 4.9: Example of applying high-order reconstruction in meshing smoothing. The left image shows the original torus mesh, and the right image shows the smoothed mesh.

4.2.3 Mesh Smoothing and Mesh Adaptivity

More general meshing applications of our techniques are the smoothing and adaptivity of surface meshes. In these settings, existing vertices may also be moved, and new vertices may be added. For these operations, a common approach is to keep the original mesh during mesh smoothing/adaptation, and project new vertices onto the faceted, piecewise linear geometries (see e.g., [22]). Such an approach has only second order accuracy. Another approach taken by Frey [20] was to construct a G^1 continuous surface using Walton's method [63], but our experiments have shown that Walton's method is only about first order accurate despite its G^1 continuity. Other methods have been developed (such as [56]), but none could deliver high-order accuracy.

Instead of using low-order methods, we propose to use high-order surface reconstructions. As an example, we integrate high-order surface reconstruction with the variational mesh smoothing framework described in [36]. The method reduces the discrepancies between the actual elements and ideal reference elements by minimizing two energy functions defined based on conformal and isometric mappings, and it works for both surface and volume meshes. For surface meshes, the method in [36] moves a vertex within an approximate tangent plane at the vertex, and it is only second order accurate at best. To achieve higher accuracy, after computing the motion in the tangent plane, we project the new vertex onto the reconstructed high-order surface, which incurs a high-order correction term. Figure 4.9 shows an example of a smoothed surface mesh using WALF method compared to the original mesh generated by the isosurface function in MATLAB. As it is apparent that the quality of the original mesh is very poor. The maximum and minimum angles were 166.1 and 0.65 degrees before smoothing. After smoothing, they were improved to 128.8 and 23.8 degrees, respectively. This process significantly improved the mesh quality while preserving the geometry to high-order accuracy.

4.3 High Order Surface Integration

Surface integration is a fundamental operation for many scientific and engineering problems. It is a core procedure for a variety of numerical methods such as the boundary integral method, finite element method, integral transforms, finite volume method etc. In geometric processing, computing the surface area and solid volume are fundamental primitives, both of which require surface integration. In computational fluid dynamics, the computation of the flux across a curved interface between different materials requires surface integrals. Another application of surface integration is found in fluid-structure interactions, where the integral of the pressure over the structure is the force applied by the fluid to the structure.

For these applications, the standard method for surface integration is to integrate over the individual triangles, but its accuracy is limited by the piecewise linear approximations to the geometry and the function. The relative error is reduced by either global or adaptive mesh refinement until a set tolerance is reached. A natural but yet fundamental question is whether we can achieve high order of accuracy given a piecewise linear approximation to the surface. In this section we use our computational framework to solve the problem of accurate numerical integration of a function over a triangulated curved surface.

Numerical integration might appear to be an easy problem, as integration in general is a smoothing process and tends to be more stable than differentiation. Although low-order integration schemes are easy, high-order integration schemes are as difficult as high-order differentiation schemes, if not more so. For these problems, numerical integration over discrete surfaces is more than just quadrature rules. The reason is that accurate numerical integration requires high-order approximations (or reconstructions) of the geometry as well as that of the function. Since our research focus on high-order differential quantities computation and surface reconstruction, we can solve the problem high-order numerical integration over surfaces using the same framework.

Related Work We review a few approaches that are generally used to get high-order approximation of surface integrals. In finite element methods, high-order approximations are achieved by using high-order isoparametric elements [71]. Unfortunately, in problems such as fluids simulations involving moving interfaces, high-order elements are rarely used due to difficulties in the generation and adaptation of high-order finite element meshes. Meshless methods, such as moving least squares, offer another set of alternatives for constructing high-order approximations, but accurate numerical integration over such surfaces is subtle [5], and meshless methods are in general expensive in terms of computational cost. In [11], Chien presented a method for high-order surface integration using a high-order in-

terpolated function and geometry. He approximated the function and the surface based on high order piecewise interpolants using Lagrange polynomials. The points chosen for high order interpolation is based on a uniform subdivision of individual triangles. His method requires the function values to be available at any point on the surface, and it becomes inapplicable when only a discrete set of values of the function is given. Also, Lagrange polynomial interpolation tends to be unstable for high degree polynomials.

Another approach is presented in [24], in which adaptive refinement of the mesh is used to minimize relative error along with simplification of trapezoidal rule, which is second order accurate, though the relative error can be minimized to a set tolerance using adaptive mesh refinement. An assumption of that method is that the surface is represented as an implicit function, so that a retraction can be defined from a neighborhood of the triangulated surface to the exact surface to obtain a better approximation of the exact surface.

We propose a novel method for accurate surface integration over discrete surfaces. Our techniques can deliver higher convergence rates, both in theory and in practice, than one would expect from the given piecewise linear approximations. Our method has three key components: the stabilized least-squares fitting to obtain higher order approximation of the geometry and the integrand, a blending procedure based on WALF reconstruction method, and high-degree numerical quadrature rules. Our resulting algorithm is simple and efficient. We demonstrate experimental results of up to sixth order accuracy with our method.

4.3.1 Integration of Continuous Functions over Smooth Surfaces

Let a parametrization of a smooth surface Γ be given as $\Gamma = \{x(\xi) \mid \mathrm{U} \subset \mathbb{R}^2 \to \mathbb{R}^3\}$, with coordinates $\xi \in \mathbb{R}^2$, $x \in \mathbb{R}^3$ and Jacobian $J = \partial x / \partial \xi$, where

$$\xi \equiv \begin{bmatrix} \xi \\ \eta \end{bmatrix}, \quad x \equiv \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad J \equiv \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{bmatrix}. \tag{4.3.1}$$

Let g denote $\sqrt{\det(J^T J)}$, which has the geometric meaning of the "element area" and is equivalent to the Jacobian determinant if the surface were in the xy-plane. The surface integral of a scalar function $\varphi : \Gamma \to \mathbb{R}$ is

$$\int_\Gamma \varphi \, da \equiv \int_U \varphi(\xi) g(\xi) d\xi \, d\eta. \tag{4.3.2}$$

In the following discussion, we will omit the function arguments ξ for conciseness. The simplest application of (4.3.2) is the surface area $A = \int_\Gamma da$, where $\varphi = 1$. Similarly, given a vector-valued function $\varphi : \Gamma \to \mathbb{R}^3$, the surface integral of φ is

$$\int_\Gamma \varphi \cdot da \equiv \int_\Gamma \varphi \cdot \hat{n} da = \int_U \varphi \cdot \hat{n} g d\xi \, d\eta, \tag{4.3.3}$$

where $\hat{n}$ denotes the unit surface normal. Let j_k denote the kth column of J for $k = 1, 2$. Assume j_1, j_2, and $\hat{n}$ form a right-hand system, and let $n \equiv g\hat{n} = j_1 \times j_2$, which we refer to as the *Jacobian-weighted normal*. The surface integral of φ can be written as

$$\int_\Gamma \varphi \cdot da = \int_U \varphi \cdot (j_1 \times j_2) d\xi \, d\eta. \tag{4.3.4}$$

The simplest application of (4.3.4) is the volume $V = \int_\Gamma x \cdot da/3$ for a closed surface Γ, where $\varphi = x/3$. Generally, a global parametrization of the whole surface is computationally difficult. To overcome this difficulty, the surface may be decomposed into non-overlapping regions. Let Γ be decomposed into non-overlapping regions σ_i whose union is Γ, i.e., $\Gamma = \bigcup \sigma_i$. The previous formulas then become

$$\int_\Gamma \varphi da = \sum_i \int_{\sigma_i} \varphi g d\xi \, d\eta \tag{4.3.5}$$

and

$$\int_\Gamma \varphi \cdot da = \sum_i \int_{\sigma_i} \varphi \cdot (j_1 \times j_2) d\xi \, d\eta, \tag{4.3.6}$$

where the integral over σ_i can be computed using local parametrizations of σ_i. Our computations for triangulated surfaces will approximate these formulas based on local parametrizations.

From (4.3.2) and (4.3.4), it is clear that surface integral of a scalar or vector-valued function requires the Jacobian of the surface. The computation of the Jacobian is significantly simplified if we transform the surface from the global *xyz* coordinate system onto a local *uvw* coordinate system. Assume both *xyz* and *uvw* coordinate frames are orthonormal right-hand systems. Let the origin of the local frame be at x_0. Let $\hat{t}_1$ and $\hat{t}_2$ be unit vectors in the *xyz* coordinate system along the positive directions of the *u*- and *v*-axes, respectively, and $\hat{m} = \hat{t}_1 \times \hat{t}_2$ be the unit vector along the positive *w* direction. Let $T = \left[\begin{array}{c|c} t_1 & t_2 \end{array}\right]$ denote the matrix consisting of the unit vectors in the tangent plane, and Q the rotation matrix $\left[\begin{array}{c|c|c} \hat{t}_1 & \hat{t}_2 & \hat{m} \end{array}\right]$. Any

point x on Γ is then transformed to a point $\begin{bmatrix} u \\ v \\ f(u,v) \end{bmatrix} \equiv Q^T(x - x_0)$. We refer to $f(u,v)$ as the *height function* in the *uvw* coordinate frame. In general, f is not a one-to-one mapping over the whole surface, but if the *uv* plane is not too far from the tangent plane at a point $x \in \Gamma$ close to x_0, f would be one-to-one in the neighborhood of x.

Let $p(u,v) = [u,v,f(u,v)]^T$ denote the points on the surface Γ in the *uvw* coordinate frame. Let $\nabla f \equiv [f_u, f_v]^T$ denote the gradient of f with respect to $u \equiv (u,v)$. The Jacobian of p with respect to u is then

$$J = \left[\begin{array}{c|c} & \\ p_u & p_v \\ & \end{array}\right] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ f_u & f_v \end{bmatrix}. \tag{4.3.7}$$

The vectors p_u and p_v form a basis of the tangent space of the surface at p, but they may not be orthogonal to each other. Then, $g = \sqrt{1 + f_u^2 + f_v^2}$. The Jacobian-weighted normal and unit normal in the *xyz* coordinate system are then

$$n = Q \begin{bmatrix} -f_u \\ -f_v \\ 1 \end{bmatrix} = \hat{m} - f_u \hat{t}_1 - f_v \hat{t}_2 \text{ and } \hat{n} = \frac{n}{g}. \tag{4.3.8}$$

These formulas are exact for smooth surfaces.

Quadrature Rules for Integrating Functions Integration in the continuum is defined as a limiting process. However, in many practical applications numerical integration require the use of quadrature rules, which are based on polynomial interpolations and approximate an integral as a weighted sum of function values at some quadrature points. In general, a degree-d quadrature rule is exact for degree-d polynomials, and it is accurate for integrands that are continuously differentiable to dth derivatives.

As a numerical problem, integration is in general well-conditioned because of its smoothing effect. Let integration of $\varphi : \Gamma \subseteq \mathbb{R}^d \to \mathbb{R}$ be given by

$$I\varphi = \int_\Gamma \varphi(x) dx. \tag{4.3.9}$$

The error in $I\varphi$ is bounded by the inequality

$$\|I\widetilde{\varphi} - I\varphi\| \leq K_\varphi \|\widetilde{\varphi} - \varphi\|_\infty, \tag{4.3.10}$$

where $\widetilde{\varphi}$ denotes an approximation of φ, and $K_\varphi = \text{area}(\Gamma)$ is the condition number of $I\varphi$ with respect to changes in φ. Therefore, integrating an approximated function does not change the result drastically. The condition number of the quadrature rules is the sum of the absolute value of its weights. If all the weights are positive, then the condition number of quadrature rule is equal to that of $I\varphi$. However, numerical integration also has an inherent uncertainty due to an infinite number of choices for $\widetilde{\varphi}$ that approximate φ. The distance between $\widetilde{\varphi}$ and φ could be arbitrary, so the error in the numerical integration may be arbitrarily large for an improper choice of $\widetilde{\varphi}$.

In one dimension it suffices to define quadrature rules on a "standard" interval, as any other interval could be transformed to this standard interval. However, this is not the case for two or higher dimensions, since an arbitrary shape may not be mapped into a "standard" shape. For example, an annulus cannot be mapped into a triangle. Therefore, high-dimensional quadrature rules are defined over some primitive shapes, such as triangles or rectangles. In [44], a survey of quadrature rules over triangles can be found. These quadrature rules are then applied to a tessellation of the domain. In this section, we assume the domain of integration is given by a triangulated surface, so we will utilize quadrature rules for triangles.

4.3.2 High-Order Numerical Integration

Now we describe how to evaluate the first-order differential quantities (normals and Jacobians) and the function values at the quadrature points, which are required by the quadrature rules. The main technique here is similar as WALF method for reconstructing discrete surface meshes, here we extend the method to include estimations of high order interpolant functions.

High-Order Piecewise Smooth Geometry The key idea behind our methods is that the polynomial at each vertex gives a high-order approximation to the surface and its differential quantities in the neighborhood of the vertex. Therefore, any weighted average of these polynomials associated with the vertices of a triangle would also give a high-order approximation. However, the matter is complicated by the fact that different local coordinate frames are used at different vertices, so a change of coordinates is necessary.

Our approach is to combine the fittings using the shape functions of a triangle. Let ξ and η denote the natural coordinate of a triangle σ, and let N_i denote the shape

function (or the barycentric coordinate) with respect to the ith vertex of σ, given by

$$\begin{aligned} N_1 &= 1-\xi-\eta \\ N_2 &= \xi \\ N_3 &= \eta. \end{aligned}$$

From the polynomial fitting at each vertex of σ, we first compute a high-order approximation for an arbitrary point $q \in \sigma$ with natural coordinate $\xi = (\xi, \eta)$ within the triangle in the corresponding local *uvw* coordinate at the vertex. This is performed by interpolating the *uv* coordinates at q from the those at the vertices and then evaluating the Taylor polynomial. The resulting point is then transformed into the global *xyz* coordinate system. For each point $q \in \sigma$, let $p_i(\xi)$ denote the reconstructed point associated with the ith vertex for $i = 1, 2, 3$. We then average these points using the shape functions. The geometry of the blended surface within the triangle is then defined by

$$p(\xi) = \sum_{i=1}^{3} N_i(\xi) p_i(\xi). \tag{4.3.11}$$

For conciseness, we will drop out the parameters (ξ, η) in the notation. Following [35], we refer to the blended surface as the *WALF* (*Weighted Average of Least-squares Fittings*) *surface*. It is clear that the blended function is infinitely differentiable within each triangle. In addition, it is C^0 continuous across the boundaries of triangles due to the continuity of the shape functions [35].

To perform integration, we need to compute the differential quantities such as the Jacobian and surface normal at the quadrature points. We compute them using the same WALF method. To compute the surface normal of the polynomial fittings at each vertex, we take their weighted average in the global coordinate system, i.e.,

$$\hat{n} = \sum_{i=1}^{3} N_i \hat{n}_i, \tag{4.3.12}$$

where $\hat{n}_i$ denote the unit normal in the global coordinate system computed from local fitting at the ith vertex of the triangle.

High-Order Piecewise Smooth Function When evaluating the functions at the quadrature points for numerical integration, if an analytical formula is available for the function, one can simply evaluate the analytical formula. However, in most cases the function values are available only at the vertices of the triangulated surface. Similar to surface reconstruction, we blend these local fittings using the shape

function to reconstruct a piecewise smooth function. Let $\tilde{\varphi}_i(\xi)$ denote the function reconstructed from the polynomial at the ith vertex. The blended function within the triangle is then

$$\tilde{\varphi}(\xi) = \sum_{i=1}^{3} N_i(\xi)\tilde{\varphi}_i(\xi). \tag{4.3.13}$$

When the function is a vector function, different components are blended independently. In the next section, we will analyze the accuracy of these reconstructions and also show that both high-order geometry and high-order function reconstructions are necessary for high-order accuracy of numerical integration.

Convergence of Proposed Methods Now we analyze the order of accuracy of our proposed methods. Our results are given in terms of the maximum edge length h in the triangulation. The main result of theoretical analysis is given by the following theorem and corollary:

Theorem: Let Γ be a smooth surface and φ be a scalar function defined over Γ. Let S denote the triangulation of Γ, and $\widetilde{S}$ and $\tilde{\varphi}$ be the surface and function reconstructed from values at vertices on S using weighted average of local fittings of degree d, respectively. Then $\left|\int_{\tilde{S}} \tilde{\varphi}\, da - \int_{\Gamma} \varphi\, da\right| = O(h^d + h^6)$.

As a corollary, we also obtain the following estimation of the errors in surface integral of vector-valued functions.

Corollary: Let Γ be a smooth surface and $\varphi : \Gamma \to \mathbb{R}^3$ be a vector-valued function defined over Γ. Let S denote the triangulation of Γ, and $\widetilde{S}$ and $\tilde{\varphi} : \widetilde{S} \to \mathbb{R}^3$ be the reconstructed surface and the reconstructed function using weighted average of local fittings of degree d, respectively. Then $\left|\int_{\tilde{S}} \tilde{\varphi} \cdot da - \int_{\Gamma} \varphi \cdot da\right| = O(h^d + h^6)$.

4.3.3 Numerical Experiments

We now present some numerical results, focusing on assessing the accuracy and convergence.

Convergence of High-order Integrations The main objectives of our experiments are to verify the high-order convergence predicted by our theoretical analysis, and to assess the effects high-order reconstructions of the geometric and function. For these purposes, it suffices to use simple smooth geometries. We use a unit sphere and a torus (with inner radius 0.7 and outer radius 1.3) as test geometries. For each geometry, we first generated a set of high-quality meshes using the mesh generator Gambit from ANSYS Inc. Note that by construction our method has little requirement on mesh quality. To demonstrate this, for each geometry we also

generated another set of poor-quality meshes using marching cubes. Some example meshes are shown in Figure 4.10 and 4.11, along with the distributions of the angles of the meshes. For mesh convergence study, we generated five meshes of different resolutions for each set of our test meshes, and numbered these meshes from the coarsest (mesh 1) to the finest (mesh 5). The average edge lengths are approximately halved between adjacent mesh resolutions. We use the finest meshes to compute the reference solutions when exact solutions are unknown, and use the other four meshes to estimate convergence. We estimate the error for each mesh as

$$\text{relative error} = \frac{\| \text{ numerical solution} - \text{reference solution } \|}{\| \text{ reference solution } \|}, \tag{4.3.14}$$

and compute the average convergence rate as

$$\text{convergence rate} = \frac{1}{3}\log_2\left(\frac{\text{error of mesh 1}}{\text{error of mesh 4}}\right). \tag{4.3.15}$$

To avoid poor convergence due to inaccurate inputs, in all cases we project the vertices onto the exact surface, so all the vertices are accurate to machine precision of double-precision floating point numbers.

Surface Integral of Scalar Function We first investigate the integration of a scalar function. A simple example is the computation of surface area, for which the integrand is $\varphi = 1$. In this case, we need to reconstruct only the geometry. For sphere and torus, the exact surface areas can be computed analytically, so we use the exact answers as the reference solutions.

Figure 4.12 plots the relative errors of the computed surface areas for the sphere using polynomial fittings of degrees between 1 and 6 under mesh refinement. Figure 4.13 shows the corresponding results for the torus. The average convergence rates are shown on the right end of the curves in the plot. It can be seen that the order of convergence is at least as high as that predicted by the theory. The even-degree polynomials exhibited higher convergence rates than predicted, probably because of statistical error cancellation due to some symmetry of the geometry and the mesh.

For generality, we also test integrating a scalar test function $\varphi(x,y,z) = \sin(x+yz) + e^{xy}$ on both the sphere and the torus. Since the exact integrals are unavailable, we use the solutions form the finest meshes as the reference solutions. The errors for this case are shown in Figure 4.14. These results are qualitatively similar to those of surface areas, and the convergence rates again confirm the theoretical results.

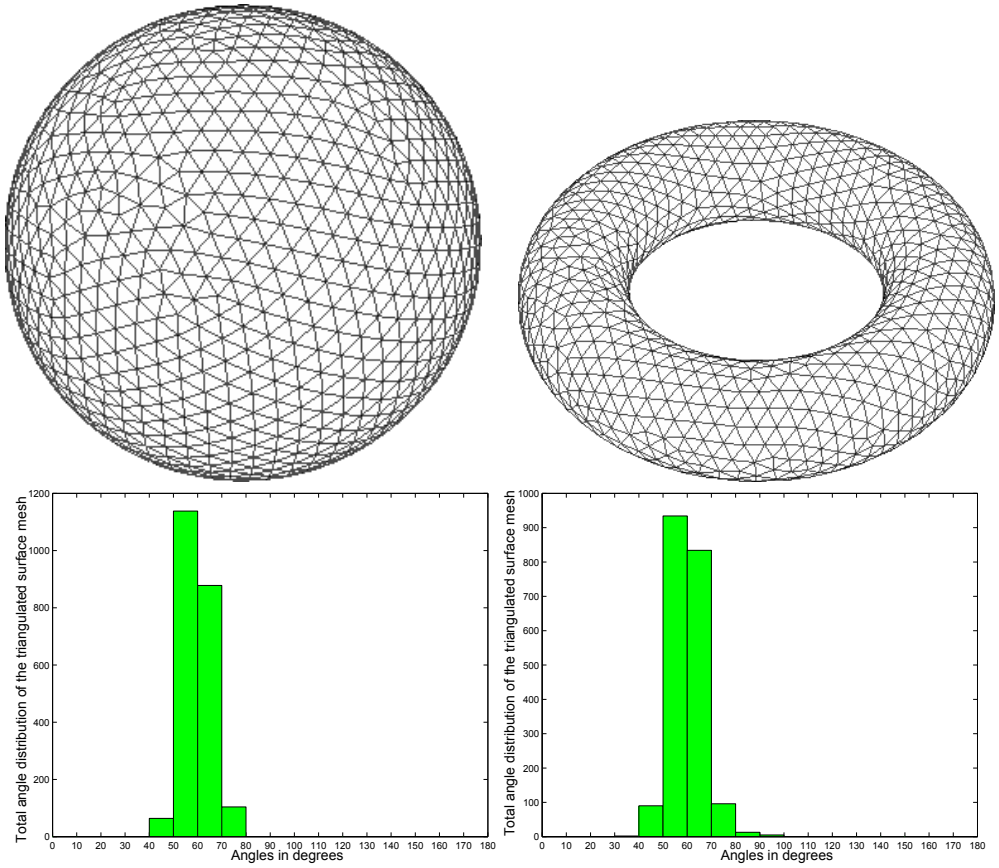


Figure 4.10: Coarsest high quality test meshes for sphere and torus used in our numerical experiments and histograms of the angles of the meshes.

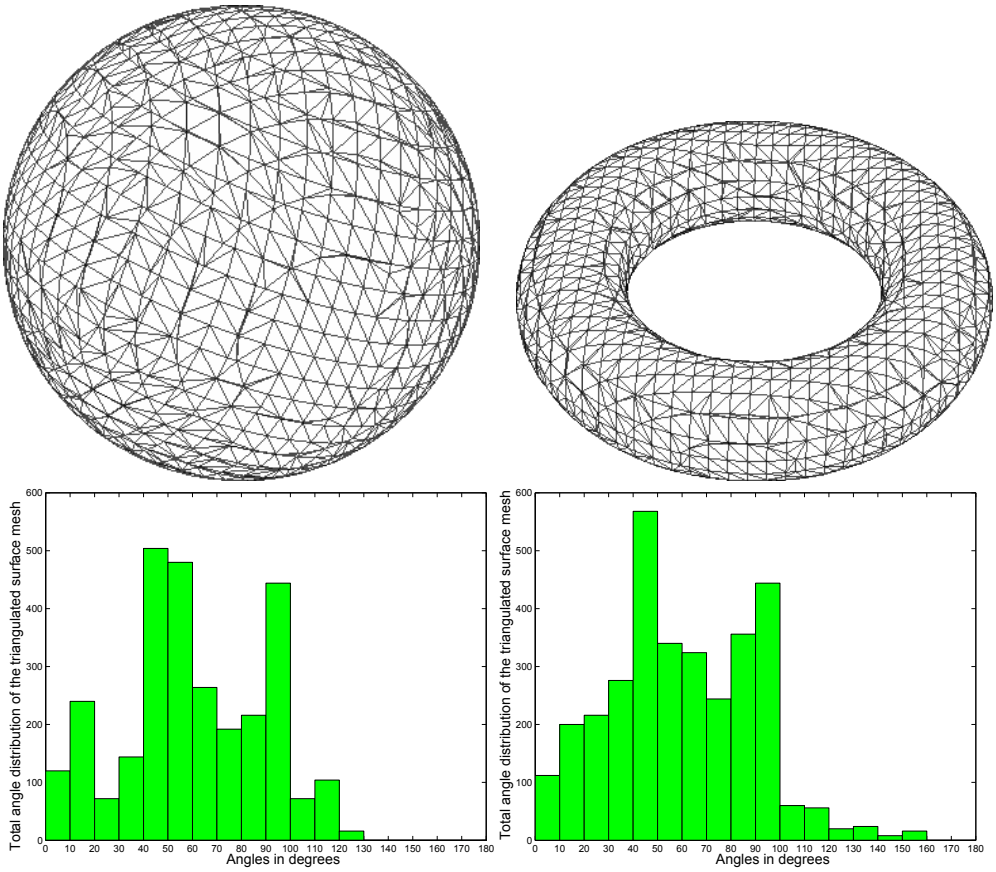


Figure 4.11: Coarsest poor quality test meshes for sphere and torus used in our numerical experiments and histograms of the angles of the meshes.

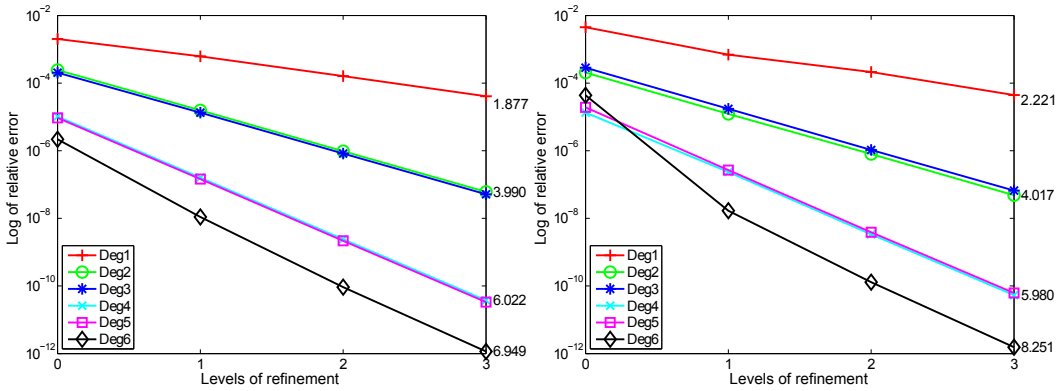


Figure 4.12: Relative errors and average convergence rates of surface areas of sphere under mesh refinement for high-quality (left) and low-quality (right) meshes.

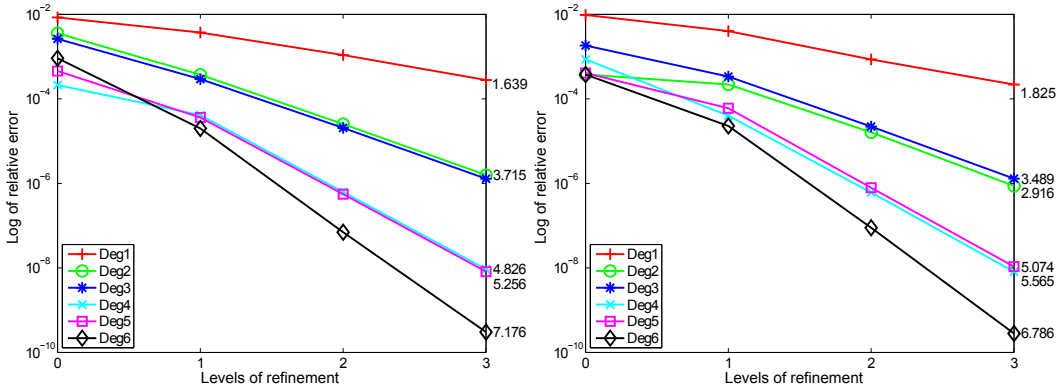


Figure 4.13: Relative errors and average convergence rates of surface areas of torus under mesh refinement for high-quality (left) and low-quality (right) meshes.

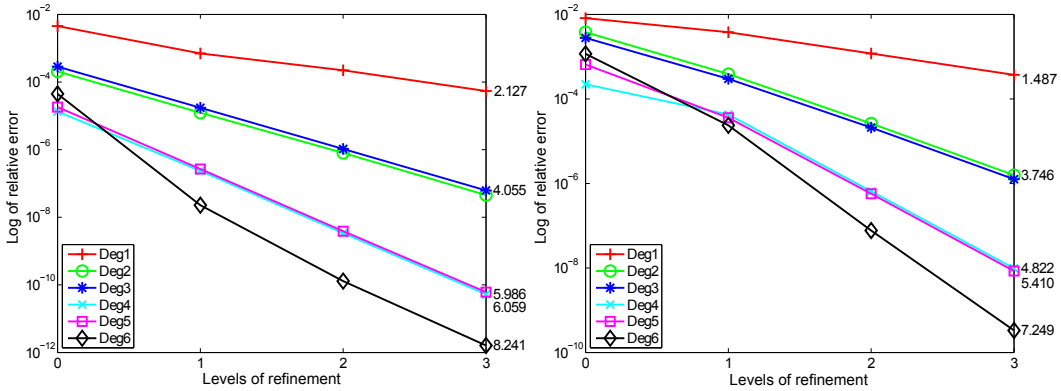


Figure 4.14: Relative errors and average convergence rates for integration of a test scalar function $\varphi(x,y,z) = \sin(x+yz) + e^{xy}$ on the sphere (left) and torus (right).

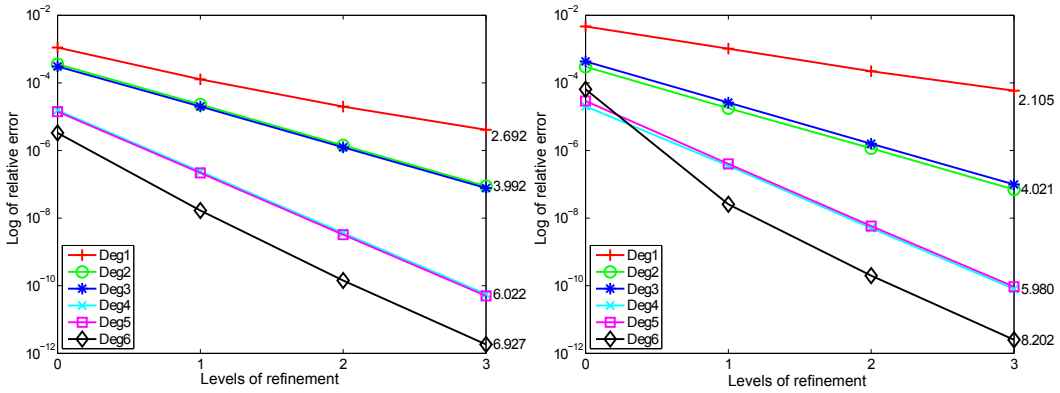


Figure 4.15: Relative errors and average convergence rates of computed volume of sphere under mesh refinement for high-quality (left) and low-quality (right) meshes.

Surface Integral of Vector Functions We now present the results of integrating vector-valued functions (i.e., flux computation) over surfaces. The simplest example is the computation of the volume bounded by a closed surface. By the divergence theorem, the volume is equal to one third of the surface integral of the position vector of the surface, i.e.,

$$V = \int_\Gamma \varphi \cdot da = \int_\Gamma \varphi \cdot \hat{n} da, \tag{4.3.16}$$

where $\varphi = x/3$ and $\hat{n}$ is outward unit normal to the surface Γ. For simple geometries such as sphere and torus, the exact volumes are available analytically and hence we use them as reference solutions in our test.

Figure 4.15 shows the relative errors of the computed volume of the sphere using polynomial fittings of degrees between 1 and 6 under mesh refinement, and Figure 4.16 shows the corresponding results for the torus. The average convergence rates are shown on the right of the plots, which again confirm our theoretical results.

For generality, we also report the result of integrating a vector-valued test function $\varphi(x,y,z) = (x\cos(y), e^y, z+e^z)$ over the sphere and torus. The reference surface is computed using the finest mesh. The relative errors and the average convergence rates are shown in Figure 4.17, which again confirm our theoretical analysis.

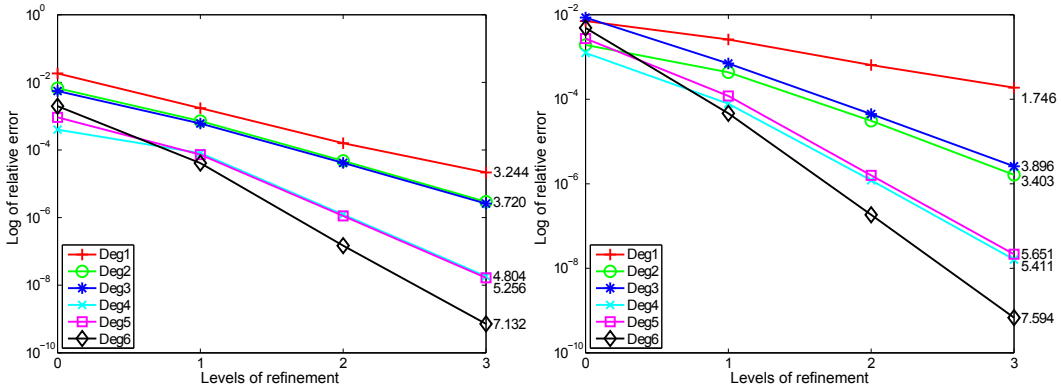


Figure 4.16: Relative errors and average convergence rates of computed volume of torus under mesh refinement for high-quality (left) and low-quality (right) meshes.

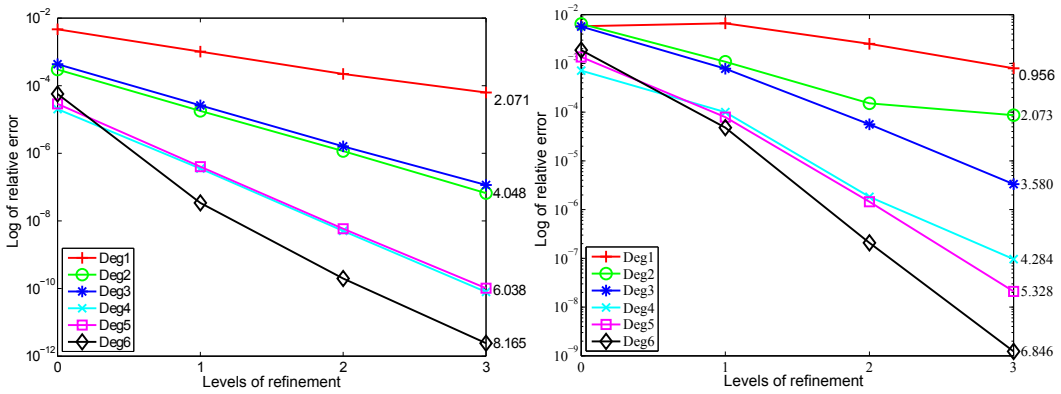


Figure 4.17: Relative errors and average convergence rates for surface integral of test function $\varphi(x,y,z) = (x\cos(y),\, e^y,\, z+e^z)$ on sphere (left) and torus (right).

Chapter 5

Geometric PDEs

Geometric partial differential equations (PDEs) on moving surfaces occur in various applications, such as surface smoothing in computer-aided design [68] and the modeling of moving surfaces of materials [8]. Equations such as mean-curvature flow and surface diffusion are challenging to solve numerically due to their strong nonlinearity and stiffness. Solving these high-order PDEs using explicit methods would require very small time steps to achieve stability, whereas using implicit methods would result in complex nonlinear systems of equations that are expensive to solve. In addition, accurate spatial discretizations of these equations pose challenges in their own rights, especially on triangulated surfaces. We propose new methods for mean curvature flow and surface diffusion using triangulated surfaces. Our method uses a weighted least-squares approximation for improved accuracy and stability, and semi-implicit schemes for time integration for larger time steps and higher efficiency. Numerical experiments demonstrate that our method can achieve second-order accuracy for both mean-curvature flow and surface diffusion, while being much more accurate and stable than using explicit schemes.

In this chapter, we first present an expanded set of surface differential operators in section 5.1, including surface gradient, surface divergence and surface Laplacian, they will be used to solve the surface diffusion equation and an elastic problem on membrane interface in chapter 6. Then we introduce two types of geometric PDEs, namely mean-curvature flow and surface diffusion. The numerical explicit scheme for solving these equations is a direct application of our work in section 3. Then we introduce the *Generalized Finite Difference* method and propose the semi-implicit schemes.

Related Work The numerical solutions of geometric PDEs have attracted substantial attentions in recent years. Several methods have been proposed, including methods using triangulated surfaces, as well as methods using implicit surfaces,

such as the level set method.

[8] reviewed a number of approaches. [2] investigated the numerical solutions of surface diffusion on graphs. [3] proposed a semi-implicit finite element method for solving the diffusion equation governed by the surface Laplacian of the mean curvature, this method converts the fourth-order nonlinear equation into a system of linear elliptic equations with respect to curvature and velocity. The Schur complement approach is then used to solve the resulting linear system. Though the curvature and velocity are evaluated implicitly, the geometric information is approximated at the current time step, i.e. the surface Laplacian operator.

[69] provided a systematic framework to solve a class of geometric partial differential equations. The first-order derivatives, second-order partial derivatives, surface gradient, divergence and Laplacian are expressed as a linear combination of the vertex coordinates, thus a linear system is formed when the overall differential operator is approximated as a linear combination of the p vertex coordinate values in the next time step.

Level set method has also been used to solve geometric PDEs. [50] devised an algorithm for front propagation with curvature-dependent speed, the equation is treated as a Hamilton-Jacobi equation with a viscosity term and it is solved numerically using techniques from hyperbolic conservation laws. [12] solved the surface diffusion equation with the fourth order operator calculated through a hybrid narrow band approach near the interface. Methods based on level set can easily handle singularities during propagation and a non-smooth initial front, but no accuracy analysis is given in the literature. Additionally, the time step for these explicit methods has to be very small to ensure stability.

A semi-implicit scheme is introduced in [57] for mean curvature flow based on the level set framework. The idea is to separate the linear term of the mean curvature term from nonlinear term, then apply the implicit scheme to the linear part while evaluating the nonlinear term at the current time step. The finite difference formula is used on a higher dimensional regular grid to approximate the nonlinear differential operator.

5.1 Extended Surface Operator

In section 3 we present the application of our computational framework to compute normals and curvatures over discrete surface, those differential quantities are geometric properties of the surface itself, in applications such as fluid dynamics, electromagnetics and geometric flows, we need to study functions defined over the surfaces. Physical equations are often written in terms of gradient, divergence, curl, and Laplacian of those functions. The definition and calculation of those operators

on surface are different from when they are in Euclidean space.

In this section, we derive those surface operators using numerical linear algebra. The formula of surface gradient, divergence and Laplacian can be found in [15, 16, 68], but they use the differential geometry terms, which is complicated to understand and difficult to compute numerically. Our alternative derivation provides a intuitive way to understand these operators and make their numerical high order calculations well suited under our polynomial fitting computation framework. First we review those operators in Euclidean space.

5.1.1 Differential operators in Euclidean space

Let $\varphi : \mathbb{R}^3 \to \mathbb{R}$ and $\psi : \mathbb{R}^3 \to \mathbb{R}^3$ denote a scalar and vector function respectively in Euclidean space, the gradient, Hessian, Laplacian of φ and Jacobian, divergence, curl of ψ are classic differential operators in calculus. We list their formulas in the table using terms of linear algebra. Among those operators, the gradient of a scalar function and Jacobian of a vector function are fundamental, all the other operators can be derived using these two operators by composition(Hessian), measure of trace(divergence and Laplacian) and skewness(curl), this observation is the key to our derivation on surface.

scalar function	$\varphi : \mathbb{R}^3 \to \mathbb{R}$	vector function	$\psi : \mathbb{R}^3 \to \mathbb{R}^3$
gradient vector	$\nabla\varphi \equiv \begin{bmatrix} \frac{\partial\varphi}{\partial x} \\ \frac{\partial\varphi}{\partial y} \\ \frac{\partial\varphi}{\partial z} \end{bmatrix}$	Jacobian matrix	$\nabla\psi \equiv \left[\frac{\partial f_j}{\partial x_i}\right]$
Hessian	$H(\varphi) \equiv \nabla(\nabla\varphi)$	divergence	$\nabla \cdot \psi \equiv \mathrm{tr}(\nabla\psi)$
Laplacian	$\Delta\varphi \equiv \mathrm{tr}(H(\varphi))$	curl	$\nabla \times \psi \equiv \mathrm{sk}(\nabla\psi)$

5.1.2 Surface operators in parametric space

Consider a smooth surface $\Gamma : U \subseteq \mathbb{R}^2 \to \mathbb{R}^3$ embedded in $\mathbb{R}^3$. Let $\{u, v\}$ denote a local parametrization of Γ in a neighborhood of a point p on Γ. The Jacobian matrix(tangent space) and normal vector of the surface Γ in the neighborhood are

$$J = [x_u \mid x_v]$$
$$n = x_u \times x_v,$$

respectively. Now we define the two fundamental differential operators on surface, surface gradient of a scalar function and surface Jacobian of a vector function.

Surface gradient Surface gradient is the gradient of function $\varphi : \Gamma \to \mathbb{R}$ defined over surface Γ, We denote it as $\nabla_\Gamma \varphi$, which is 3-vector in principle, generally, surface gradient is not equal to the gradient in Euclidean space, it is the projection of the gradient into the tangent space, $\nabla_\Gamma \varphi = T\nabla\varphi$, this is intuitive to understand in Euclidean space, but the formula in parametric space is a little bit complicated.

In [15] p.102-103, *Gradient on Surface* is defined as a differentiable map $\nabla_\Gamma \varphi : \Gamma \to \mathbb{R}^3$ which assigns to each point $p \in \Gamma$ a vector $\nabla_\Gamma \varphi(p) \in T_p(\Gamma) \subset \mathbb{R}^3$ such that

$$< \nabla_\Gamma \varphi(p),\ v >= d\varphi(v) \text{ for all } v \in T_p(\Gamma)$$

Algebraically, $\nabla_\Gamma \varphi = [x_u, x_v][g^{\alpha\beta}](\varphi_u, \varphi_v)^T$ [68]. Now we derive the computation formula using the term of numerical linear algebra. By the differential-geometry definition above, let v be x_u and x_v respectively, we have following two equations:

$$x_u \cdot \nabla_\Gamma \varphi = \varphi_u \text{ and } x_v \cdot \nabla_\Gamma \varphi = \varphi_v \tag{5.1.1}$$

Let $\nabla\varphi = \begin{bmatrix} \frac{\partial \varphi}{\partial u} \\ \frac{\partial \varphi}{\partial v} \end{bmatrix}$ and $J \equiv [x_u \mid x_v]$. Above equations simplify to (the chain rule)

$$J^T \nabla_\Gamma \varphi = \nabla_u \varphi \tag{5.1.2}$$

If J has full rank (i.e., $J^T J$ is nonsingular), then $\nabla_\Gamma \varphi$ has unique solution in column space of J (i.e. tangent space)

$$\nabla_\Gamma \varphi \equiv J^{+T} \nabla\varphi \tag{5.1.3}$$

where J^{+T} is transpose of pseudo-inverse of J. This is equivalent to the formula in [68]. In a coordinate system aligned with tangent space, $\nabla_\Gamma \varphi = \begin{bmatrix} \nabla_u \varphi \\ 0 \end{bmatrix}$, it is the tangent direction where φ changes most rapidly in the tangent space, and its magnitude is equal to the rate of change.

Surface Jacobian Given a vector field defined on the surface $\psi : U \subseteq \mathbb{R}^3 \to \mathbb{R}^3$, we define surface Jacobian of a tangent vector function ψ as

$$\nabla_\Gamma \psi = J^{+T} (\nabla\psi) J J^{+}, \text{ where } \nabla\psi \equiv \begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_2}{\partial u} & \frac{\partial f_3}{\partial u} \\ \frac{\partial f_1}{\partial v} & \frac{\partial f_2}{\partial v} & \frac{\partial f_3}{\partial v} \end{bmatrix}.$$

For an arbitrary vector field, we define its surface Jacobian as

$$\nabla_\Gamma \psi = J^{+T} \nabla \left(J J^+ \psi \right) J J^+$$

This definition has a geometric meaning: $JJ^+$is projection matrix of tangent space, so $JJ^+\psi$ projects ψ into tangent space, then $J^{+T}\nabla\left(JJ^+\psi\right)$ takes the surface gradient of each coordinate function, finally $J^{+T}\nabla\left(JJ^+\psi\right)JJ^+$ projects every row of $J^{+T}\nabla\left(JJ^+\psi\right)$ into tangent space. Besides its geometric meaning, this definition resembles the Jacobian in Euclidean space in that, together with surface gradient definition, it unifies the surface differential operator theory, all the other operators can be derived from these two operators.

Surface Divergence In Euclidean space, divergence can also be defined as limit of surface integral, i.e.,

$$\nabla \cdot \psi \equiv \lim_{V \to 0} \frac{\iint_{\partial V} \psi \cdot da}{V} \tag{5.1.4}$$

where V, ∂V, and da are defined the same as before, this definition should generalize to surface directly, i.e.,

$$\nabla_\Gamma \cdot \psi \equiv \lim_{A \to 0} \frac{\int_{\partial A} \psi \cdot ds}{A} \tag{5.1.5}$$

In parametric space, for tangential vector field, it is easy to see that

$$\nabla_\Gamma \cdot \psi = \frac{\partial \varphi_1}{\partial u} + \frac{\partial \varphi_2}{\partial v} = \mathrm{tr}\left(J^{+T}\nabla\psi\right) \tag{5.1.6}$$

We show that $\mathrm{tr}(J^{+T}\nabla\psi)$ is rotation invariant. Let P denote rotation matrix and $\tilde{\psi} \equiv P^T\psi$ and $\tilde{J} = P^T J$. Therefore,

$$\tilde{J}^{+T}\nabla\tilde{\psi} = P^T J^{+T}\nabla(P^T\psi) = P^T\left(J^{+T}\nabla\psi\right)P, \tag{5.1.7}$$

so $\tilde{J}^{+T}\nabla\tilde{\psi}$ and $J^{+T}\nabla\psi$ have same eigenvalues and in turn the same trace. If ψ is an arbitrary vector field, decompose ψ into tangential and normal components,

$$\psi = JJ^+\psi + \hat{n}\hat{n}^T\psi. \tag{5.1.8}$$

From Taylor series expansion, $\nabla_\Gamma \cdot \left(\hat{n}\hat{n}^T\psi\right) = 0$, therefore,

$$\nabla_\Gamma \cdot \psi = \nabla_\Gamma \cdot \left(JJ^+\psi\right) = \mathrm{tr}\left(J^{+T}\nabla\left(JJ^+\psi\right)\right) \tag{5.1.9}$$

In the literature, the surface divergence is sometimes defined as [28, page 21]

$$\nabla_\Gamma \cdot f = \left(\nabla_x - nn^T \nabla_x\right) \cdot \left(f - nn^T f\right), \tag{5.1.10}$$

where ∇_x is the gradient operator with respect to x, or as [53, 70]

$$\nabla_\Gamma \cdot f = \frac{1}{\sqrt{g}} \nabla_u \cdot \left(\sqrt{g} J^+ f\right), \tag{5.1.11}$$

where $g = \det(J^T J)$. It can be shown that these definitions are equivalent. Notice that

$$\nabla_\Gamma \cdot \psi = \operatorname{tr}\left(J^{+T} \nabla \left(J J^+ \psi\right)\right) = \operatorname{tr}\left(J^{+T} \nabla \left(J J^+ \psi\right) J J^+\right) \tag{5.1.12}$$

So surface divergence is actually the trace of surface Jacobian matrix.

Note that if f is tangent to Γ, then $f = JJ^+ f$, the formula can be further simplified to

$$\nabla_\Gamma \cdot f = \operatorname{tr}\left(J^+ \left(\nabla_u f\right)\right) = J^+_{1,:} f_u + J^+_{2,:} f_v, \tag{5.1.13}$$

where $J^+_{j,:}$ $(j = 1, 2)$ denotes the jth row of J^+.

Surface Hessian Surface Hessian on surfaces/manifolds was defined in [16]. Let us denote it by $H_\Gamma(\varphi)$, which defines an operator over tangent space, i.e., $x^T H_\Gamma(\varphi) y$ for x and y in tangent space, intuitively, surface Hessian should be$\nabla_\Gamma \left(\nabla_\Gamma \varphi\right)$, where ∇_Γ is the surface gradient operator defined above, in another word it is the Jacobian of the gradient of a scalar function, thus,

$$H_\Gamma(\varphi) = \nabla_\Gamma \left(\nabla_\Gamma \varphi\right) = J^{+T} \nabla \left(J^{+T} \nabla \varphi\right) J J^+$$

We use the fact that $\nabla_\Gamma \varphi$ is a tangent vector field. Surface Hessian such defined is a symmetric matrix, we omit the proof here.

Surface Laplacian Surface Laplacian, sometimes called *Laplace-Beltrami operator*, in [68], surface Laplacian is given as

$$\Delta_\Gamma \varphi = \frac{1}{g}\left(g_{11} f_{22} + g_{22} f_{11} - 2 g_{12} f_{12}\right) = \operatorname{tr}\left(G^{-1} \begin{bmatrix} f_{11} & f_{12} \\ f_{12} & f_{22} \end{bmatrix}\right) \tag{5.1.14}$$

where $f_{ij} = \varphi_{u_i u_j} - \left(\nabla_\Gamma \varphi\right)^T x_{u_i u_j}$

We give an alternative derivation using the trace of surface Hessian,

$$\Delta_\Gamma \varphi = tr(H_\Gamma(\varphi)) = tr\left(J^{+T} \nabla \left(J^{+T} \nabla \varphi\right) J J^+\right)$$

using product rule for differentiation

$$\nabla\left(J^{+T}\nabla\varphi\right) = \underbrace{\left(\nabla(\nabla\varphi)\right)}_{\text{Hessian of } \varphi} J^{+} + \begin{bmatrix} (\nabla\varphi)^T J_u^+ \\ (\nabla\varphi)^T J_v^+ \end{bmatrix} \tag{5.1.15}$$

Because $J^+J = I$ and $(J^+J)_u = 0$, so $J_u^+J = -J^+J_u$. Similarly $J_v^+J = -J^+J_v$. Therefore

$$\begin{bmatrix} (\nabla\varphi)^T J_u^+ \\ (\nabla\varphi)^T J_v^+ \end{bmatrix} J = -\begin{bmatrix} (\nabla\varphi)^T J^+ J_u \\ (\nabla\varphi)^T J^+ J_v \end{bmatrix} = -\begin{bmatrix} (\nabla_\Gamma\varphi)^T J_u \\ (\nabla_\Gamma\varphi)^T J_v \end{bmatrix} \tag{5.1.16}$$

We note that

$$\operatorname{tr}\left(J^{+T}\nabla\left(J^{+T}\nabla\varphi\right)JJ^{+}\right) = \operatorname{tr}\left(J^{+}J^{+T}\nabla\left(J^{+T}\nabla\varphi\right)J\right). \tag{5.1.17}$$

Therefore,

$$\Delta_\Gamma\varphi = \operatorname{tr}\left(G^{-1}\left(\left(\nabla(\nabla\varphi)\right)J^{+}J - \begin{bmatrix} (\nabla_\Gamma\varphi)^T J_u \\ (\nabla_\Gamma\varphi)^T J_v \end{bmatrix}\right)\right) = \operatorname{tr}(G^{-1}M), \tag{5.1.18}$$

where

$$M = \nabla(\nabla\varphi) - \begin{bmatrix} (\nabla_\Gamma\varphi)^T J_u \\ (\nabla_\Gamma\varphi)^T J_v \end{bmatrix}. \tag{5.1.19}$$

This is equivalent to the definition in [68]. Later we will see its simplified form under local coordinate system.

Surface Curl Given vector field $\psi(x)$, the Curl operator$\nabla\times\psi$ measures "rotation per unit area", given a plane Γ at point x_0, Curl is defined as

$$\hat{n}_\Gamma^T\cdot(\nabla\times\psi) = \lim_a \frac{\oint_{\partial a}\psi\cdot t\,ds}{a}, \tag{5.1.20}$$

where a is area of a neighborhood of x_0 within Γ and $\hat{n}_\Gamma$ denotes unit normal to Γ. Algebraically,

$$\nabla\times\psi = \left(\frac{\partial f_3}{\partial y} - \frac{\partial f_2}{\partial z}\right)\hat{e}_1 + \left(\frac{\partial f_1}{\partial z} - \frac{\partial f_3}{\partial x}\right)\hat{e}_2 + \left(\frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y}\right)\hat{e}_3. \tag{5.1.21}$$

This formula does not directly generalize to surfaces. However, the definition based on the skewness of the Jacobian matrix does,

$$\nabla_\Gamma\times\psi \equiv \operatorname{sk}(\nabla_\Gamma\psi) \tag{5.1.22}$$

Summary Here we list the whole set of surface operator definitions.

scalar function	$\varphi : \mathbb{R}^3 \to \mathbb{R}$	vector function	$\psi : \mathbb{R}^3 \to \mathbb{R}^3$
gradient vector	$\nabla_\Gamma \varphi = J^{+T} \nabla \varphi$	Jacobian matrix	$\nabla_\Gamma \psi \equiv J^{+T} \nabla (JJ^+ \psi) JJ^+$
Hessian	$H_\Gamma(\varphi) = \nabla_\Gamma (\nabla_\Gamma \varphi)$	divergence	$\nabla_\Gamma \cdot \psi = tr(\nabla_\Gamma \psi)$
Laplacian	$\Delta_\Gamma \varphi = \mathrm{tr}(H_\Gamma(\varphi))$	curl	$\nabla_\Gamma \times \psi \equiv \mathrm{sk}(\nabla_\Gamma \psi)$

Local height function parametrization Under local height function parametrization, the above formulas have simplified forms. In this local coordinate system, the surface is given as $x \equiv (u, v, f(u,v))$, and the Jacobian of the surface itself is,

$$J = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ f_u & f_v \end{bmatrix} \tag{5.1.23}$$

Similarly,

$$x_{uu} = \begin{bmatrix} 0 \\ 0 \\ f_{uu} \end{bmatrix}, x_{uv} = \begin{bmatrix} 0 \\ 0 \\ f_{uv} \end{bmatrix}, x_{vv} = \begin{bmatrix} 0 \\ 0 \\ f_{vv} \end{bmatrix} \tag{5.1.24}$$

Thus the formulas can be simplified, we list them in the following table.

scalar	$\varphi : \mathbb{R}^3 \to \mathbb{R}$	vector	$\psi : \mathbb{R}^3 \to \mathbb{R}^3$
gradient	$\nabla_\Gamma \varphi = J^{+T} \nabla \varphi$	Jacobian	$n^T \psi J W J^+ + J^{+T} (\nabla \psi) JJ^+$
Hessian	$J^{+T} \left(H_\varphi - \frac{(\nabla \varphi)^T \nabla f}{g} H_f \right) J^+$	divergence	$2Hn^T \psi + tr\left(J^{+T} (\nabla \psi)\right)$
Laplacian	$\mathrm{tr}\left((J^T J)^{-1} \left(H_\varphi - \frac{(\nabla \varphi)^T \nabla f}{g} H_f \right) \right)$	curl	$\mathrm{sk}\left(J^{+T} (\nabla \psi) JJ^+\right)$

where W is the Weingarten matrix $W = \frac{1}{\ell}(J^T J)^{-1} H_f$, H_φ and H_f denote Hessian matrices of the function φ and the surface itself f respective to u and v.

5.2 Mean Curvature Flow and Surface Diffusion

5.2.1 Mean Curvature Flow

The continuum formulation of the mean-curvature flow is as follows. Given a moving surface Γ, the coordinates x of points on Γ are functions of time t as well as some surface parametrization $u = (u, v)$, which can be local instead of global parametrizations. Assume the surface is differentiable. The *mean-curvature flow*

is a second-order nonlinear PDE modeling the motion of the surface driven by the mean curvature, given by

$$\frac{\partial x}{\partial t} = M\hat{n}, \tag{5.2.1}$$

where M denotes the mean curvature and $\hat{n}$ denotes the unit normal vector. The vector $\hat{n}$ involves first-order partial derivatives of x with respect to the parameters u, whereas M involves second-order partial derivatives of x with respect to u.

For a parametric surface, the normal and mean curvature under global coordinate system are(see Table 3.1),

$$n = \frac{1}{\ell}(J_{:,1} \times J_{:,2}) \tag{5.2.2}$$

$$M = \frac{1}{2}\mathrm{tr}(G^{-1}B) \tag{5.2.3}$$

where $\ell = \|x_u \times x_v\|_2 = \sqrt{1 + f_u^2 + f_v^2}$, G is the first fundamental form and B is the second fundamental form. Under local coordinate system, the formulas are simplified as,

$$n = \frac{1}{\ell}\begin{bmatrix} -f_u \\ -f_v \\ 1 \end{bmatrix} \tag{5.2.4}$$

$$M = \frac{\mathrm{tr}(H)}{2\ell} - \frac{(\nabla f)^T H (\nabla f)}{2\ell^3} \tag{5.2.5}$$

where the gradient $\nabla f = [f_u, f_v]^T$ and the Hessian matrix $H = \begin{bmatrix} f_{uu} & f_{uv} \\ f_{vu} & f_{vv} \end{bmatrix}$.

Compared to formula 5.2.3, the mean curvature formula 5.2.5 under local coordinate system has no normal term n(the second fundamental form B has normal term), this separation of normal and curvature is useful when we derive the semi-implicit scheme in section 5.3.

The numerical solutions of geometric PDEs include spatial discretization and temporal discretization. In this dissertation, we consider high order spatial discretization only, temporal discretization will be the future research topic. The explicit scheme is simple and direct given the formulas in section 3, using forward Euler scheme,

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \cdot M_i^{(n)} \mathbf{n}_i^{(n)} \tag{5.2.6}$$

Equation 5.2.1 is analogous to the parabolic equations (such as the heat equation) in terms of its stiff time-step constraints for explicit schemes. If solved explicitly, the time step must be second order to the minimum edge length of a triangulated

surface. That is why we want to introduce a semi-implicit scheme in the next section.

5.2.2 Surface Diffusion

The *surface diffusion* is a fourth-order nonlinear PDE modeling the motion driven by the surface Laplacian of the mean curvature, given by

$$\frac{\partial x}{\partial t} = (\triangle_\Gamma M)\hat{n}, \tag{5.2.7}$$

where $\triangle_\Gamma$ denotes the surface Laplacian operator (i.e., the surface divergence of the surface gradient). Because the term $\triangle_\Gamma M$ involves fourth-order partial derivatives of x with respect to u, the surface diffusion is much more difficult to solve than the mean-curvature flow. If solved explicitly, the time step must be fourth order to the minimum edge length of a triangulated surface, making it extremely inefficient to solve. On the other hand, a fully implicit scheme would be very difficult to derive and to solve due to its strongly nonlinearity.

Similar to mean-curvature flow, we discretize the surface diffusion into a system of ODEs, and obtain an ODE at each vertex of the triangulation, using the forward Euler scheme, we obtain the equation

$$x_i^{(n+1)} = x_i^{(n)} + \triangle t\,(\triangle_\Gamma M)_i^{(n)}\,\hat{n}_i^{(n)}. \tag{5.2.8}$$

To solve (5.2.8), the key step is to discretize the surface Laplacian of the mean curvature. As described in section 5.1, the surface Laplacian of the mean curvature in the local coordinate system is,

$$\triangle_\Gamma M = \mathrm{tr}\left(G^{-1}\left(\nabla^2 M - \frac{(\nabla M)^T \nabla f}{\ell^2} H\right)\right), \tag{5.2.9}$$

where $\nabla^2 M$ and H denote the Hessian matrices of M and f, respectively, i.e.,

$$\nabla^2 M = \begin{bmatrix} M_{uu} & M_{uv} \\ M_{uv} & M_{vv} \end{bmatrix} \qquad \text{and} \qquad H = \begin{bmatrix} f_{uu} & f_{uv} \\ f_{uv} & f_{vv} \end{bmatrix}, \tag{5.2.10}$$

and

$$G^{-1} = \frac{1}{\ell^2}\begin{bmatrix} 1+f_v^2 & -f_u f_v \\ -f_u f_v & 1+f_u^2 \end{bmatrix}. \tag{5.2.11}$$

The mean curvature is given by

$$M = \frac{\mathrm{tr}(H)}{2\ell} - \frac{(\nabla f)^T H \nabla f}{2\ell^3}. \tag{5.2.12}$$

Since M is a second-order differential quantity, ∇M and H_M are third-order and fourth-order differential quantities, respectively. Let ℓ_u and ℓ_v denote the partial derivatives of ℓ with respect to u and v, respectively, i.e.,

$$\ell_u = \frac{f_u f_{uu} + f_v f_{uv}}{\ell} \qquad \text{and} \qquad \ell_v = \frac{f_u f_{uv} + f_v f_{vv}}{\ell}. \tag{5.2.13}$$

It is easy to show that

$$\begin{aligned} M_u &= \frac{\ell \mathrm{tr}(H_u) - \ell_u \mathrm{tr}(H)}{2\ell^2} - \frac{\ell\left(2(\nabla f_u)^T H \nabla f + (\nabla f)^T H_u \nabla f\right) - 3\ell_u (\nabla f)^T H \nabla f}{2\ell^4} \\ &= \underbrace{\frac{\mathrm{tr}(H_u)}{2\ell} - \frac{(\nabla f)^T H_u \nabla f}{2\ell^3}}_{A} - \ell_u \underbrace{\left(\frac{\mathrm{tr}(H)}{2\ell^2} - \frac{3(\nabla f)^T H \nabla f}{2\ell^4}\right)}_{B} - \underbrace{\frac{(\nabla f_u)^T H \nabla f}{\ell^3}}_{C}, \end{aligned} \tag{5.2.14}$$

where H_u involves third-order derivatives with respect to u and v. Similarly,

$$\begin{aligned} M_v &= \frac{\ell \mathrm{tr}(H_v) - \ell_v \mathrm{tr}(H)}{2\ell^2} - \frac{\ell\left(2(\nabla f_v)^T H \nabla f + (\nabla f)^T H_v \nabla f\right) - 3\ell_v (\nabla f)^T H \nabla f}{2\ell^4} \\ &= \frac{\mathrm{tr}(H_v)}{2\ell} - \frac{(\nabla f)^T H_v \nabla f}{2\ell^3} - \ell_v \left(\frac{\mathrm{tr}(H)}{2\ell^2} - \frac{3(\nabla f)^T H \nabla f}{2\ell^4}\right) - \frac{(\nabla f_v)^T H \nabla f}{\ell^3}, \end{aligned} \tag{5.2.15}$$

where H_v involves third-order derivatives with respect to u and v. In summary, we can denote the gradient of mean curvature flow as

$$\nabla M = \frac{\nabla \mathrm{tr}(H)}{2\ell} - \frac{(\nabla f)^T (\nabla H) \nabla f}{2\ell^3} - \left(\frac{\mathrm{tr}(H)}{2\ell^2} - \frac{3(\nabla f)^T H \nabla f}{2\ell^4}\right) \nabla \ell - \frac{H^2}{\ell^3} \nabla f, \tag{5.2.16}$$

where ∇H denotes a third-order tensor. Similarly we can expand the fourth-order term$\nabla^2 M$, we omit the formulas due to the length of this dissertation. Note that for ∇M and $\nabla^2 M$, no term contains the product of third-order and fourth-order differential quantities, this observation is the key when we introduce the semi-implicit scheme for surface diffusion equation in the next section.

5.3 Generalized Finite Difference Method

To overcome the time step restriction imposed by CFL condition for mean-curvature flow and surface diffusion, we prefer to use implicit scheme. From equation 5.2.5, mean curvature is a nonlinear term with product of first and second order differential quantities, a fully implicit scheme would be complicated to solve,

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \cdot M_i^{(n+1)} \mathbf{n}_i^{(n+1)} \tag{5.3.1}$$

Here both normal and mean curvature are evaluated in the next time step. Alternatively, we devise a semi-implicit scheme by evaluating all the first order derivatives in the current time step and all the second order derivatives in the future time step to avoid this nonlinear issue,

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \cdot \widetilde{M}_i^{(n+1)} \mathbf{n}_i^{(n)} \tag{5.3.2}$$

and

$$\widetilde{M}_i^{(n+1)} = \frac{\mathrm{tr}(\widetilde{\mathbf{H}})}{2\ell} - \frac{(\nabla f)^T \widetilde{\mathbf{H}} (\nabla f)}{2\ell^3} \tag{5.3.3}$$

where˜indicates a implicit term that will be evaluated in the next time step. Similar for surface diffusion, we can use explicit first and second order differential quantities, while implicitly evaluating third and fourth order terms. For simplicity, we will use the mean-curvature flow to demonstrate the *generalized finite difference* method.

This semi-implicit scheme is possible if the Hessian term H can be expressed as linear expressions of vertices' coordinate values. Recall the least square fitting equation at a vertex,

$$Ac \approx f, \tag{5.3.4}$$

we observe that the matrix A plays a fundamental role. Since $Ac \approx f$, the polynomial coefficients of function f are given by $c = A^+ f$. We refer to $C = A^+$ as the *coefficient matrix*. Each coefficient of the polynomial can be expressed as a linear combination of f, with the rows of C as the weights. In addition, we can express the derivatives of f at the vertex as linear combinations of the values of f at its neighboring vertices. This procedure can be viewed as a generalization of classic finite difference schemes. For example, the coefficient matrix for a classical 9-point central difference scheme on regular rectangular grid is simply

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2\Delta x} & 0 & -\frac{1}{2\Delta x} & 0 & 0 & 0 \\ 0 & \frac{1}{2\Delta y} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2\Delta y} & 0 \\ 0 & 0 & 0 & \frac{1}{\Delta x^2} & -\frac{2}{\Delta x^2} & 0 & \frac{1}{\Delta x^2} & 0 & 0 \\ \frac{1}{4\Delta x\Delta y} & 0 & -\frac{1}{4\Delta x\Delta y} & 0 & 0 & 0 & -\frac{1}{4\Delta x\Delta y} & 0 & \frac{1}{4\Delta x\Delta y} \\ 0 & \frac{1}{\Delta y^2} & 0 & 0 & -\frac{2}{\Delta y^2} & 0 & 0 & \frac{1}{\Delta y^2} & 0 \end{bmatrix}. \tag{5.3.5}$$

The matrix C is independent of the right hand side vector f. It only depends on the local parametrization, which reflects the local geometric information around the vertex. Let $N(i)$ denote the set of vertices in the neighborhood of vertex i (including itself). The gradient of the displacement vector ϕ and its derivatives are

$$f_u = \sum_{j\in N(i)} C_{2,j} f_j \qquad f_v = \sum_{j\in N(i)} C_{3,j} f_j$$
$$f_{uu} = 2\sum_{j\in N(i)} C_{4,j} f_j \qquad f_{uv} = \sum_{j\in N(i)} C_{5,j} f_j \qquad f_{uu} = 2\sum_{j\in N(i)} C_{6,j} f_j$$

Now the mean curvature term is,

$$\widetilde{M}_i^{(n+1)} = \frac{f_{uu}+f_{vv}}{2\ell} - \frac{f_u^2\widetilde{f_{uu}} + 2f_u f_v \widetilde{f_{uv}} + f_v^2 \widetilde{f_{vv}}}{2\ell^3}$$

which can be further expanded to

$$\frac{1}{2\ell^3}\sum_{j\in N(i)} \left((1+f_v^2)C_{4,j} - 2f_u f_v C_{5,j} + (1+f_u^2)C_{6,j}\right)\cdot(\mathbf{x}_j^{(n+1)} - \mathbf{x}_i^{(n+1)})\cdot\mathbf{n}_i^{(n)} \tag{5.3.6}$$

Equation 5.3.2 now becomes a linear relation with respect to the vertices coordinate values at next time step, we can aggregate these equations over all vertices to form a global linear system,

$$B^{(k)}\phi^{(k+1)} = \phi^{(k)} \tag{5.3.7}$$

where $B^{(k)}$ is a $3n\times 3n$ matrix, and $\phi^{(k+1)}$ and $\phi^{(k)}$ are column vectors of length $3n$, composed of the coordinate values at time step $k+1$ and k.

At the kth step, we solve a linear system

$$M^{(k)}\phi^{(k)} = p^{(k)}, \tag{5.3.8}$$

where $M^{(k)}$ is a $3m\times 3m$ matrix, and $p^{(k)}$ and $\phi^{(k)}$ are column vectors of length $3m$, composed of the traction vectors and displacement vectors at the vertices, respec-

tively. Note that $M^{(k)}$ and $p^{(k)}$ depend on $\phi^{(k-1)}$.

For closed surfaces, the linear system (5.3.8) is underdetermined when no boundary conditions are specified, because there are three extra degrees of freedom for translation, because the solution is invariant of translation. We use the procedure of truncated singular value decomposition (SVD) to solve the linear system. Let the SVD decomposition of $M^{(k)}$ be

$$M^{(k)} = U\Sigma V^T, \tag{5.3.9}$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix composed of the singular values of $M^{(k)}$. The last six diagonal entries of Σ are nearly 0. To solve for $\phi^{(k)}$, we discarded the last six singular values and their corresponding singular vectors and compute $\phi^{(k)}$ as

$$\phi^{(k)} = \sum_{j=1}^{3n-3} \frac{1}{s_j} v_j u_j^T p^{(k)}, \tag{5.3.10}$$

where s_j denotes the jth entry of Σ, and u_j and v_j denote the jth column of U and V, respectively. This numerical discretization also applies to open surfaces with Dirichlet or other boundary condition.

5.4 Numerical Experiments and Comparisons

In this section, we perform numerical experiments to verify our preceding analysis. The mean curvature flow test will focus on our semi-implicit method and its rate of convergence. The test will measure the error in the total surface area and the error in the encapsulated volume of a spherical mesh. These values will then produce a rate of convergence for each measure of error. We will compare our semi-implicit methods to their explicit counterparts in terms of accuracy, convergence, and stability. Overall, we want to see our semi-implicit method produce better accuracy, and maintain higher stability than its explicit counterpart, while achieving second-order convergence in surface area errors and encapsulated volume errors.

Mean Curvature Flow - Semi-Implicit Method Convergence Results

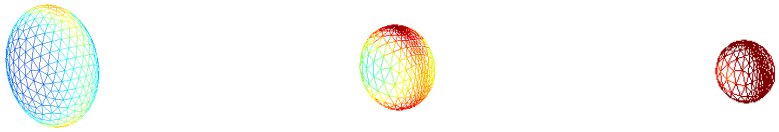


Figure 5.1: The evolution of a closed ellipsoid mesh (axes = 4, 6, 8) under mean curvature flow using our semi-implicit method. The colormap reflects the mean curvature. (left) surface before evolution. (center) surface after 3.00 seconds of evolution. (right) surface after 3.75 seconds of evolution.

This test will examine the accuracy and stability of our semi-implicit method. We will work with a spherical mesh of radius = 1, an ellipsoid mesh with axes of length 4, 6, and 8, and a torus mesh with major radius = 1, and minor radius = 0.25. We will use three levels of refinement for our initial mesh, with total number of vertices being 368, 1450, and 5804. For all refinement levels, we will use 50 time steps at $\triangle t = 0.001$. For both tests, we will measure the surface area error and the encapsulated volume error. For spherical mesh test, we will also measure L^{∞} error. For surface area calculation, we simply sum the surface area of all triangular elements. For volume calculation, we connect all mesh vertices to the origin, forming tetrahedrons, then sum the volume of all tetrahedrons to obtain the total volume encapsulated by the surface mesh.

For a sphere, the exact calculation for the radius using mean curvature flow is the following:

$$R_{exact} = \sqrt{R_{initial}^2 - 2t}$$

R_{exact} is the exact radius of the sphere we expect to obtain after time t, from an initial sphere with radius of $R_{initial}$. Additionally, we will measure convergence using the following equation:

$$O_{conv} = Log_2(\frac{error_i}{error_{i+1}}) \tag{5.4.1}$$

For our sphere test, we obtain the following results, shown in Table 5.1

Table 5.1: L^{∞}errors of vertex placement, surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on a spherical mesh of radius = 1, after 50 time steps for $\triangle t = 0.001$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O	L^{∞}_{err}
1	368	9.17e-2	n/a	5.26e-2	n/a	1.68e-4
2	1450	2.19e-2	2.07	1.27e-2	2.05	8.76e-5
3	5804	3.98e-3	2.46	2.47e-3	2.36	8.33e-5

As can be seen in Table 5.1 both the surface area error and the volume error exhibit greater than quadratic convergence rate in all cases.

Since we do not have an exact value for comparison in the ellipsoid test, we use a super-refined mesh of 23188 vertices to get an approximation of our errors. As before, we will use three levels of refinement for our initial mesh, with 368, 1450, and 5804 vertices. For each refinement level, we will use 50 time steps at $\triangle t = 0.001$. For ellipsoid test, we obtain the results shown in Table 5.2

Table 5.2: Surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on an ellipsoid mesh with axes 4, 6 and 8, after 50 time steps at $\triangle t = 0.001$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O
1	368	4.78e-1	n/a	4.84e-1	n/a
2	1450	1.18e-1	2.02	1.16e-1	2.07
3	5804	2.52e-2	2.22	2.32e-2	2.32

Surface Diffusion - Semi-Implicit Method Convergence Results This test will examine the accuracy and stability of our semi-implicit method for surface diffusion. We will work with the same ellipsoid from before, with axes of length 4, 6, and 8. We will use three levels of refinement for our initial mesh, with our number of vertices being 368, 1450, and 5804. For all refinement levels, we will use 50 time steps at $\triangle t = 1.e-6$. Because we do not have an exact value for comparison in the ellipsoid test, we will again use a super-refined mesh of 23188 vertices to get an approximation of our errors, to test for convergence. With our ellipsoid test, we obtain the results seen in Table 5.3

Table 5.3: Surface area errors, encapsulated volume errors and convergence rates for our semi-implicit surface diffusion method on an ellipsoid mesh with axes 4, 6 and 8 after 22 time steps at $\triangle t = 5e-6$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O
1	368	8.97e-1	n/a	1.50e-0	n/a
2	1450	2.20e-1	2.03	3.60e-1	2.06
3	5804	4.41e-2	2.32	7.20e-2	2.32

As can be seen, our obtained errors lead to a greater than quadratic convergence rate in all cases.

Comparison of Explicit and Semi-Implicit Methods Here we will compare our semi-implicit method with its explicit counterpart on a spherical surface with a radius of 1. We will use three levels of refinement for our initial mesh, with 368, 1450, and 5804 vertices. Overall, we want to see our semi-implicit method produce better accuracy, and maintain higher stability than its explicit counterpart, while achieving second-order convergence in surface area errors and encapsulated volume errors. We obtain the results seen in Figure 5.2.

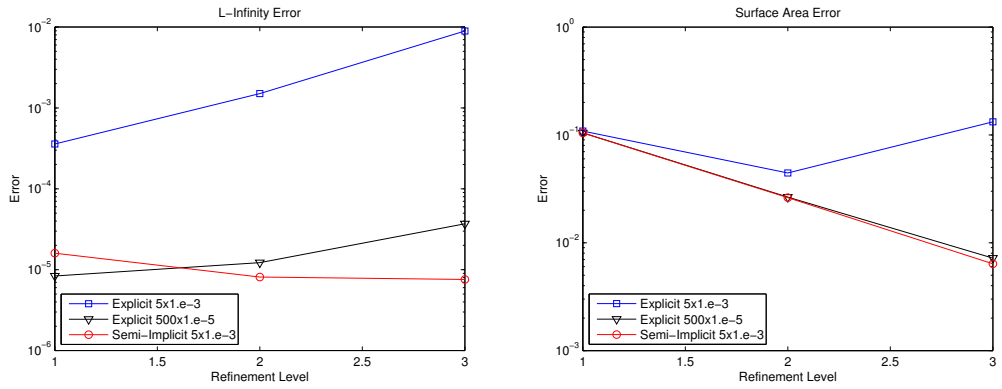


Figure 5.2: A comparison of our semi-implicit mean curvature flow method and its explicit counterpart on a spherical mesh of radius = 1 after 0.005 seconds. Compared are the explicit method after 5 time steps at $\triangle t = 1.e-3$ and 500 time steps at $\triangle t = 1.e-5$, and our semi-implicit method for 5 time steps at $\triangle t = 0.001$, with refinement levels of 368, 1450, and 5804 vertices. (left) Displays the L^{∞}error of both methods, with the explicit method diverging in both cases and our semi-implicit method converging. (right) Displays the surface area error of both methods with the explicit method ($\triangle t = 1.e-3$) diverging, the explicit method ($\triangle t = 1.e-5$) converging with a rate of 1.88 and our semi-implicit method ($\triangle t = 1.e-3$) converging with a rate of 2.04.

Because of their differences in stability, the explicit method cannot be compared effectively with our semi-implicit method using the same size time step. As can be seen in Figure 5.2, the explicit method after 5 time steps at $\triangle t = 1.e-3$ diverges in both L^{∞} errors and surface area errors, opposed to our semi-implicit method which converges in both cases. The results for the encapsulated volume errors were excluded because of their similarity to the surface area errors. Since the explicit method is less stable than our semi-implicit method, it requires a much smaller time step of $\triangle t = 1.e-5$ in order to produce convergent results in the surface area error and volume error. To create an equivalent realtime measure of 0.005 seconds, we must use 500 time steps with $\triangle t = 1.e-5$. By shrinking the time step, the explicit method's surface area error converges with a rate of 1.88, with similar results in its volume error. However, we still see divergence in its L^{∞} errors. In order to produce convergent results in its L^{∞} errors, we would have to shrink the time step to $\triangle t = 1.e-8$. While the explicit method can produce convergent results with a small enough time step, the results are still slightly worse than our semi-implicit method, and more importantly, even with comparable results in accuracy the major issue of requiring such small time steps for explicit method ultimately results in

very long runtimes, as will be explored soon. Overall, we see our semi-implicit method all of our goals: it produces better accuracy, and maintains higher stability than its explicit counterpart, while achieving second-order convergence.

Chapter 6

An Elastic Membrane Problem

In this chapter, we will use our computational framework to solve a physical problem, the modeling and discretization of the curvature effect of a thin and curved elastic interface, which separates two fluid subdomains. For such an interface, there is often a pressure jump between the two fluid subdomains, which is partially balanced by a normal pressure exerted by the interface due to a curvature effect, in a manner similar to the surface tension in fluid dynamics [41, 39]. In 3-D space, such an interface is a two-dimensional object, commonly referred to as a *membrane* or *membrane shell* [62, Chapter 13]. Examples of a membrane include the canopy of a parachute, the biological membrane of a cell, airbags, balloons, etc. We refer to the normal pressure exerted by a membrane interface as the *interface pressure*, and refer to its corresponding stress vector as *interface stress*.

For a membrane interface, the interface pressure is an important part of the physics as it corresponds to the actual load distribution on a membrane. Mathematically, this pressure is the normal component of the surface divergence of the stress tensor. However, since the surface divergence of the stress tensor is typically not computed explicitly in most computations for membranes (or shells), as a derived quantity, this normal pressure is not readily available in the simulation results. One primary goal here is to derive an explicit, easy-to-compute formula for the normal pressure, and develop a discretization method for evaluating it from the stress tensor, so that it can be visualized and verified directly. Another goal is to derive explicit formulas of the surface divergence of the stress tensor, so that we can discretize the problem in a strong form with a generalized finite difference method or some meshless method, which are less demanding in mesh quality than finite element methods and are sometimes advantageous in dynamic simulations where the mesh quality may be difficult to maintain [5].

Background and Related Work An elastic membrane is a special case of a shell, where the ratio between the thickness and other dimensions is very small (typically less than 0.01 [62]). The modeling of shells is an important subject in structural mechanics. Because of its complexity and its practical relevance, there is vast literature on the modeling of shells. For an excellent comprehensive review on shell modeling, see [6].

For membranes, the interface pressure is in many ways similar to the effect of surface tension on thin films or interfaces in fluid mechanics. The effect of surface tension is given by the well-known Young-Laplace equation in fluid mechanics [41, 39], which states that when all the forces are balanced, the pressure difference between two sides of an interface (a.k.a. the Laplace pressure) is equal to twice the mean curvature times surface tension. In 1993, Povstenko [**?**] performed a theoretical investigation of the generalization of the Young-Laplace equation to heterogeneous surface tension in solids, and showed that the jump in interface stress across an interface is equal to the normal component of the surface divergence of interface stress tensor. However, the equation in [**?**] did not explicitly refer to curvatures, unlike the Young-Laplace equation. In [**?**], a connection between the interface stress and principal curvature was stated, but no derivation was given, and the equation appeared to have an inconsistency in terms of rotation invariance (more in Section 6.2.2). In this chapter, we derive a generalization of the Young-Laplace equation, which can be expressed in terms of either the interface stress tensor and the shape operator, or the Cauchy stress tensors and the curvature tensor.

The modeling of shells and membranes involve many different aspects, including the kinematic hypotheses, constitutive models, boundary conditions, dynamics, and discretizations. The focus here is only on the computation and the discretizations of the curvature effect and of the surface divergence of the stress tensor. We do not propose any new constitutive laws or boundary conditions. However, we will investigate the coupling of our formulas with a number of constitutive laws (including Kirchhoff-Love and Mindlin-Reissner models [6]) and some simple boundary conditions to verify our computations of the interface pressure.

Some discrete models have been developed for membranes in the literature. The most common approach in engineering is the finite element method using shell elements or membrane elements; see survey articles such as [6]. Among these models, the membrane elements are designed for relatively thin structures, including fabric-like objects such as tents or cots. These models are very accurate in practice. However, since the finite element methods are formulated in a weak form, the divergence of the stress tensor is not computed explicitly, so the interface pressure is not readily available from the simulation results.

Another type of models is the spring mesh models (or mass-spring models). A spring mesh is a system of vertices and edges, in which each edge is a spring, and the

springs are connected by "pin-joints" at the vertices. These models are conceptually simple and computationally efficient, so they are widely used in computer graphics (see e.g., [4, 48]). However, the accuracy of these models is questionable. In [61], Van Gelder showed that the common practice of assigning the same stiffness to all springs causes significant distortions, even for uniform elastic membranes. New formulas were proposed in [61] to assign spring stiffness based on the angles and areas of the triangles, but this model needs to assume Poisson's ratio to be zero for general triangular meshes. Like finite element methods, the spring-mesh models do not compute the interface pressure, which is the subject of our research.

Contributions and Organization In this chapter, we derive the equations for the interface pressure of elastic membranes due to the curvature effect. Our main result is Theorem 6.2.2, or the *interface-pressure theorem*, which states that the interface pressure is equal to the trace of the matrix product of the curvature tensor and the Cauchy stress tensor in the tangent plane. This theorem is applicable to stress tensors computed from a broad range of constitutive models. We also describe how to discretize the equation on triangulated surfaces to high-order accuracy. From a theoretical point of view, this theorem can be viewed as a generalization of the well-known Young-Laplace equation for surface tension in fluid mechanics, as it includes the this equation as a special case (Corollary **??**). This theorem is also useful from a practical point of view, as its discretization allows coupling with various membrane models and finite-element methods to compute the interface pressure for visualization or further processing. In addition, we explore a strong-form discretization of the surface divergence of the stress tensor, so that an elastic model may be solved with our generalized finite difference method, which are advantageous in some dynamic simulations. We present the theoretical derivations and numerical verifications of our theorems and discretizations.

6.1 Background of Elasticity Theory

In this section, we review some fundamental concepts in elasticity theory for solids using linear algebra notation (i.e., matrices and vectors), instead of the indicial notation that is common in the literature of mechanics and differential geometry. Note that order-one and order-two tensors are similar to vectors and matrices, so our notation is not a large departure from the standard convention. We will in general treat vectors as column vectors, and whenever possible we will make matrices consistent with the convention for order-two tensors.

6.1.1 Elasticity of Solid

We first review elasticity for solid bodies. This material can be found in standard textbooks in structural mechanics (such as [31] and [45]). We will present its extension to membranes in Section 6.1.2.

Elasticity Equation of a Solid For a solid body, the elasticity equation is given as

$$\rho \frac{\partial^2 \phi}{\partial t^2} = f + \nabla \cdot \sigma, \tag{6.1.1}$$

where ϕ denotes the displacement vector, f is the body force (such as gravity),

$$\sigma = \left[\begin{array}{c|c|c} \sigma_1 & \sigma_2 & \sigma_3 \end{array}\right] = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} \tag{6.1.2}$$

is the Cauchy stress tensor with $\sigma_{ij} = \sigma_{ji}$, and $\nabla \cdot \sigma$ is the divergence of σ, i.e.,

$$\nabla \cdot \sigma = \begin{bmatrix} \nabla \cdot \sigma_1 \\ \nabla \cdot \sigma_2 \\ \nabla \cdot \sigma_3 \end{bmatrix} = \begin{bmatrix} \frac{\partial \sigma_{11}}{\partial x_1} + \frac{\partial \sigma_{12}}{\partial x_2} + \frac{\partial \sigma_{13}}{\partial x_3} \\ \frac{\partial \sigma_{21}}{\partial x_1} + \frac{\partial \sigma_{22}}{\partial x_2} + \frac{\partial \sigma_{23}}{\partial x_3} \\ \frac{\partial \sigma_{31}}{\partial x_1} + \frac{\partial \sigma_{32}}{\partial x_2} + \frac{\partial \sigma_{33}}{\partial x_3} \end{bmatrix}. \tag{6.1.3}$$

Strain and Stress Tensors The key terms in the elasticity equation are the Cauchy stress tensor and its divergence. For a linear elastic material, the stress tensor is determined by the Green-Lagrangian strain tensor ε and a constitutive equation that relates the stress and strain.

To define the strain, let $x = (x_1, x_2, x_3)$ denote a point in the undeformed (unstressed) configuration of a solid body $\mathscr{B}$. For a vector field $f(x) : \mathscr{B} \subset \mathbb{R}^3 \to \mathbb{R}^3$ in the undeformed configuration, let ∇f denote the gradient operator of f, i.e.,

$$\nabla f = \left[\frac{\partial f_i}{\partial x_j}\right]_{ij} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}. \tag{6.1.4}$$

This convention is consistent with the tensor-based definition of the gradient of a vector in [65]. Let $y = (y_1, y_2, y_3)$ denote the point in the deformed configuration corresponding to x, and then the deformation vector is $\phi = y - x$. For finite-strain

theory, the Lagrangian finite strain tensor is

$$\varepsilon = \frac{1}{2}\left(\nabla\phi + (\nabla\phi)^T + (\nabla\phi)^T \nabla\phi\right), \tag{6.1.5}$$

which is nonlinear and is invariant under rigid-body motion. For very small deformations, the strain tensor may be linearized to be

$$\varepsilon_{\text{linear}} = \frac{1}{2}\left(\nabla\phi + (\nabla\phi)^T\right), \tag{6.1.6}$$

which is not invariant under rigid-body motion.

The Cauchy stress tensor, which we denoted by σ, is a measure of internal forces on the deformed configuration. As stated by Cauchy's stress theorem (see e.g. [31, Chapter 2]), this stress tensor has the physical meaning that, the stress vector at a point on the plane with unit normal vector d is equal to σd. Because of its physical meaning, the Cauchy stress tensor will be used, unless otherwise noted. In computational mechanics, the 2nd Piola–Kirchhoff stress tensor, denoted by S, is also often used. S is related to the Cauchy stress tensor through the transformation

$$\sigma = \frac{1}{\det(F)} FSF^T, \tag{6.1.7}$$

where $F = \nabla y$ is known as the *deformation gradient*. In words, S relates force in the reference configuration to volume in the reference configuration, so some computational models often use it instead of σ.

The strain and stress tensors are related through the constitutive equation, which depends on the material properties. The theoretical results in this chapter are independent of the constitutive models. In most of our examples, we assumes small strain but allow large deformation and large rotation (as in the St. Vernant-Kirchhoff model [31, Chapter 4]). Under this assumption, the Cauchy stress tensor is obtained from the *generalized Hooke's law*

$$\sigma = \mathscr{C} : \varepsilon, \tag{6.1.8}$$

where $\mathscr{C}$ is a fourth-order tensor known as the *elasticity tensor*. In matrix notation, we can write the relationship as

$$\begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} (1-\nu) & \nu & \nu \\ \nu & (1-\nu) & \nu \\ \nu & \nu & (1-\nu) \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \end{bmatrix} \tag{6.1.9}$$

and

$$\sigma_{ij} = \frac{E}{1+\nu}\varepsilon_{ij} \text{ for } i \neq j, \tag{6.1.10}$$

where E is *Young's modulus* and ν is *Poisson's ratio*. More concisely, let $\lambda = E\nu/((1+\nu)(1-2\nu))$ be *Lamé's first parameter*, and $\mu = E/(2+2\nu)$ *be Lamé's second parameter* or *shear modulus*. We then have

$$\sigma = \lambda \text{tr}(\varepsilon) I + 2\mu\varepsilon, \tag{6.1.11}$$

where I is the identity matrix and $\text{tr}(\varepsilon) = \sum_{i=1}^{3} \varepsilon_{ii}$ is the trace of ε.

6.1.2 Elastic Models of Membranes

The preceding elasticity concepts and equations can be degenerated to two dimensions to model the mid-surface of a membrane (or more generally, of a thin shell). The most important change is the surface divergence of the stress tensor $\nabla_\Gamma \cdot \sigma$ in the elasticity equation, replacing the divergence in Euclidean space,

$$\rho \frac{\partial^2 \phi}{\partial t^2} = f + \nabla_\Gamma \cdot \sigma. \tag{6.1.12}$$

where ϕ is the displacement on S, ρ is the mass per unit area, F is the body force, and σ is the Cauchy stress tensor. The use of $\nabla_\Gamma \cdot \sigma$ is valid under the assumption that the variation of σ is negligible in the normal direction to the membrane.

The Cauchy stress tensor σ is related to the strain tensor ε through the constitutive equation under some kinematic hypothesis about the membrane (or shell), such as the Kirchhoff-Love assumptions or Mindlin-Reissner assumptions [6]. In particular, one common assumption for these models is the following:

> *The thickness of the membrane (or shell) does not change during deformation.*

As a consequence, ε_{33} (i.e., the strain in the normal direction) is assumed to be zero. Let ε^* denote the modified strain tensor for the membrane, which we will discuss further in Section 6.3. Let $J = [y_u \,|y_v]$ denote the Jacobian matrix of the deformed configuration, where $y = x + \phi$. Let $T = JJ^+$ denote the orthogonal projection matrix onto the tangent space. For linear elastic models, the Cauchy stress tensor is then

$$\sigma = \lambda^* \text{tr}(\varepsilon^*) T + 2\mu\varepsilon^*, \tag{6.1.13}$$

where

$$\lambda^* = \frac{2\mu\lambda}{\lambda + 2\mu} = \frac{\nu E}{1-\nu^2} \tag{6.1.14}$$

is the Lamé's first parameter for plates and shells. Compared with (6.1.11), λ, ε, and I are replaced by λ^*, ε^*, and T, respectively.[1] Note that although based on linear elasticity, (6.1.13) is in fact nonlinear in the displacements, since T is on the deformed configuration. Given the modified strain tensor, we note the following property of the Cauchy stress tensor:

Proposition: In a membrane model, $n^T \sigma n = 0$, where σ is the Cauchy stress tensor and n denotes the unit normal to the *deformed* surface.

This property is satisfied by virtually all models for thin shells, because $n^T \sigma n$ is the internal pressure in the direction normal to the mid-surface, which was observed to be close to zero in practice. From Property 6.1.2, the interface pressure is not due to the normal component of σ; instead, it is an effect from the surface divergence of σ, which we analyze in the next section.

6.2 Interface Pressure of Membranes

We now analyze the surface divergence of the stress tensor and the interface pressure of membranes. We note that all of our results are described in terms of the Cauchy stress tensor, so nearly all the geometric differential operators in this section are computed on the deformed configuration unless otherwise noted.

6.2.1 Surface Divergence of Stress Tensor

The surface divergence of the stress tensor σ over the deformed configuration Γ is defined as

$$\nabla_\Gamma \cdot \sigma = \sum_{i=1}^{3} (\nabla_\Gamma \cdot \sigma_{i,:})\, e_i = \begin{bmatrix} \nabla_\Gamma \cdot \sigma_{1,:} \\ \nabla_\Gamma \cdot \sigma_{2,:} \\ \nabla_\Gamma \cdot \sigma_{3,:} \end{bmatrix}, \tag{6.2.1}$$

where $\sigma_{i,:}$ denotes the ith row of σ, $\nabla_\Gamma \cdot \sigma_{i,:}$ is the surface divergence given in Section 5.1 and e_i denotes the ith standard unit vector. We decompose $\nabla_\Gamma \cdot \sigma$ into its tangential component $\nabla_T \cdot \sigma$ and normal component $\nabla_N \cdot \sigma$, i.e.,

$$\nabla_\Gamma \cdot \sigma = \nabla_T \cdot \sigma + \nabla_N \cdot \sigma, \tag{6.2.2}$$

where $\nabla_T \cdot \sigma = JJ^+ \nabla_\Gamma \cdot \sigma$ and $\nabla_N \cdot \sigma = nn^T \nabla_\Gamma \cdot \sigma$, n denotes the unit outward normal to Γ. We obtain the following lemma regarding the surface divergence and its tangential component.

[1] Replacing I by T is justified from (6.1.7), $\det(J) \approx 1$ for small strains, and $S_{33} = 0$ in the 2nd Piola-Kirchhoff stress tensor. Replacing λ by λ^* is needed since $\varepsilon_{33} = 0$ in ε^*.

Lemma:The surface divergence of the stress tensor on the membrane, denoted by $\nabla_\Gamma \cdot \sigma$, is

$$\nabla_\Gamma \cdot \sigma = (T\sigma)_u J^{+T}_{:,1} + (T\sigma)_v J^{+T}_{:,2}, \tag{6.2.3}$$

where $T = JJ^+$ and $J^{+T}_{:,j}$ denotes the jth column of J^{+T}. The tangent component of $\nabla_\Gamma \cdot \sigma$ with base vectors x_u and x_v is

$$J^+ \nabla_T \cdot \sigma = J^+ \left((T\sigma)_u J^{+T}_{:,1} + (T\sigma)_v J^{+T}_{:,2} \right). \tag{6.2.4}$$

Proof. From (5.1.10), we have

$$\nabla_\Gamma \cdot \sigma_i = \mathrm{tr}\left((\nabla_u (T\sigma_i)) J^{+T} \right) \tag{6.2.5}$$

and

$$\nabla_\Gamma \cdot \sigma = (T\sigma)_u J^{+T}_{:,1} + (T\sigma)_v J^{+T}_{:,2}. \tag{6.2.6}$$

Then, $J^+ \nabla_T \cdot \sigma = JJ^+ \nabla_\Gamma \cdot \sigma = J^+ \left((T\sigma)_u J^{+T}_{:,1} + (T\sigma)_v J^{+T}_{:,2} \right)$.

If there is no shear stress on Γ, then $T\sigma = \sigma$, and $\nabla_\Gamma \cdot \sigma$ simplifies to

$$\nabla_\Gamma \cdot \sigma = \sigma_u J^{+T}_{:,1} + \sigma_v J^{+T}_{:,2}. \tag{6.2.7}$$

6.2.2 Interface Pressure

Both $\nabla_\Gamma \cdot \sigma$ and its tangent component involve partial derivatives of σ. For nonlinear constitutive models, it may appear daunting to compute the interface pressure (namely $n^T \nabla_\Gamma \cdot \sigma$), directly from (6.2.3). In the following, we show that the interface pressure can be computed without differentiating σ. We will give our result in the global coordinate system, which is easier to understand and is independent of local parametrizations. However, computationally it is more convenient to use an equation in a local coordinate system. For the latter, we introduce a new symbol $\hat{\tau}$ to denote a 2×2 stress tensor in a local uv coordinate system with basis vectors y_u and y_v on the deformed membrane. In particular,

$$\widehat{\tau} = \lambda^* \mathrm{tr}(\varepsilon^*) I + 2\mu J^+ \varepsilon^* J, \tag{6.2.8}$$

where I is the 2×2 identity matrix. Let $\sigma_T = T\sigma T = J\hat{\tau}J^+$, and it is equal to σ if there is no shear stress. Note that $\hat{\tau}$ is asymmetric unless $J^T J = gI$ with $g = \det\left(J^T J\right)$. The symmetry can recovered by right-multiplying $\hat{\tau}$ by G^{-1}, i.e., $\bar{\tau} = \hat{\tau} G^{-1}$. We now give our main result, which we refer to as the *interface-pressure theorem*.

Theorem: The interface pressure is equal to

$$n^T \nabla_N \cdot \boldsymbol{\sigma} = \mathrm{tr}(C\boldsymbol{\sigma}_T) = \mathrm{tr}(C\boldsymbol{\sigma} T), \tag{6.2.9}$$

in the global xyz coordinate system, where n is the unit normal to the deformed surface, and C is the curvature tensor of the deformed surface, and $\boldsymbol{\sigma}_T = T\boldsymbol{\sigma} T$. In the local uv coordinate system with y_u and y_v as the base vectors,

$$n^T \nabla_N \cdot \boldsymbol{\sigma} = \mathrm{tr}(W\hat{\boldsymbol{\tau}}), \tag{6.2.10}$$

where W is the *Weingarten matrix* of the deformed surface.

Proof: From (6.2.3) and $J^{+T}_{:,j} = J\left(J^T J\right)^{-1}_{:,j}$, the normal component of $\nabla \cdot \boldsymbol{\sigma}$ is

$$\begin{aligned} n^T \nabla_N \cdot \boldsymbol{\sigma} &= n^T (T\boldsymbol{\sigma})_u J^{+T}_{:,1} + n^T (T\boldsymbol{\sigma})_v J^{+T}_{:,2} \\ &= n^T (T\boldsymbol{\sigma})_u J \left(J^T J\right)^{-1}_{:,1} + n^T (T\boldsymbol{\sigma})_v J \left(J^T J\right)^{-1}_{:,2}. \end{aligned} \tag{6.2.11}$$

Since $T\boldsymbol{\sigma} = T\boldsymbol{\sigma} T + T\boldsymbol{\sigma} nn^T = J\hat{\boldsymbol{\tau}} J^+ + T\boldsymbol{\sigma} nn^T = J\left(\hat{\boldsymbol{\tau}} J^+ + J^+ \boldsymbol{\sigma} nn^T\right)$, by the chain rule

$$\begin{aligned} n^T (T\boldsymbol{\sigma})_u J &= n^T \left(J_u \left(\hat{\boldsymbol{\tau}} J^+ + J^+ \boldsymbol{\sigma} nn^T\right) + J\left(\hat{\boldsymbol{\tau}} J^+ + J^+ \boldsymbol{\sigma} nn^T\right)_u\right) J \\ &= n^T J_u \hat{\boldsymbol{\tau}}, \end{aligned}$$

where the second equality uses the facts that $n^T J = 0$ and $J^+ J = I$. Similarly, $n^T \boldsymbol{\sigma}_v J = n^T J_v \hat{\boldsymbol{\tau}}_T$. Substituting these into (6.2.11), we have

$$\begin{aligned} n^T \nabla_N \cdot \boldsymbol{\sigma} &= n^T J_u \hat{\boldsymbol{\tau}} \left(J^T J\right)^{-1}_{:,1} + n^T J_v \hat{\boldsymbol{\tau}} \left(J^T J\right)^{-1}_{:,2} \\ &= \mathrm{tr}\left(B\hat{\boldsymbol{\tau}} (J^T J)^{-1}\right) \\ &= \mathrm{tr}\left((J^T J)^{-1} B\hat{\boldsymbol{\tau}}\right) \\ &= \mathrm{tr}\left(W\hat{\boldsymbol{\tau}}\right), \end{aligned}$$

where $B = [J_u n \,|J_v n]$ is the second fundamental matrix, and $W = (J^T J)^{-1} B$ is the Weingarten matrix. Because the curvature tensor is $W = J^+ CJ$ and $\hat{\boldsymbol{\tau}} = J^+ \boldsymbol{\sigma} J$, we obtain $n^T \nabla_N \cdot \boldsymbol{\sigma} = \mathrm{tr}\left(W\hat{\boldsymbol{\tau}}\right) = \mathrm{tr}\left(J\left(J^+ CJ\right)\left(J^+ \boldsymbol{\sigma} J\right) J^+\right) = \mathrm{tr}(C\boldsymbol{\sigma}_T) = \mathrm{tr}(C\boldsymbol{\sigma} T)$.

Note that this theorem does not require $\boldsymbol{\sigma}$ to be free of shear stress or $n^T \boldsymbol{\sigma} n = 0$, so it can be coupled with linear or nonlinear models for membranes. Our formulas do not require differentiating the stress tensor, but requires only computing the curvature tensor on the deformed geometry.

Note that our result is analogous to the Young-Laplace equation [41, 39] for the surface tension effect in fluid dynamics. In effect, the Young-Laplace equation is a

special case.

For a smooth membrane, if the force density is equal to f in all directions at a point x, then the interface pressure at any point due to the tangential stress around x is $p = 2Hf$, where $H = (\kappa_1 + \kappa_2)/2$ is the mean curvature.

Because $\hat{\tau} = fI$ and $\mathrm{tr}(W) = 2H$, from equation6.2.9 we have $n^T \nabla_N \sigma = \mathrm{tr}(WfI) = f\mathrm{tr}(W) = 2Hf$.

6.3 Coupling with Constitutive Models

In this section, we investigate the coupling of the interface-pressure theorem with constitutive models. We first propose a continuum model for thin membranes without shear deformation, which we solve by projecting the strain and stress tensors onto the tangent space, and verify it through two examples. For completeness, we also show an example of coupling the theorem with Mindlin-Reissner model, which considers shear deformation.

Linear Elastic Model Without Shear Deformations For thin membranes, it is often reasonable to assume that there is no shear deformations, as in the Kirchhoff-Love shell theory. In this situation, to allow easier matrix manipulations we write the modified strain tensor as

$$\varepsilon^* = T\varepsilon T, \tag{6.3.1}$$

and then substitute it into (6.1.13) to compute the stress tensor. This expression is equivalent to setting $\gamma_{xz} = \gamma_{yz} = \varepsilon_z = 0$ (as required by the Kirchhoff-Love assumptions) in

$$\varepsilon = \begin{bmatrix} \varepsilon_x & \gamma_{xy} & \gamma_{xz} \\ \gamma_{xy} & \varepsilon_y & \gamma_{yz} \\ \gamma_{xz} & \gamma_{yz} & \varepsilon_z \end{bmatrix}, \tag{6.3.2}$$

at a point where the tangent plane is parallel to the xy-plane. In the following, we apply our theorem to two examples with simple geometries that have other reference solutions.

Elastic Spherical Balloon We first consider an example of an inflated spherical balloon, which was analyzed in [49]. In this situation, the normal displacement is uniform. Assume large displacements but small strain, and suppose the radii of the balloon without stress and under stress are r_0 and r_1, respectively. Let $I_{3\times3}$ and $I_{2\times2}$ denote the 3×3 and 2×2 identity matrix, respectively. Since $\phi = (r_1 - r_0)/r_0 x$ and

$\nabla_x \phi = (r_1 - r_0)/r_0 I_{3\times 3}$, we have

$$\varepsilon^* = \frac{1}{2}\left(2\frac{r_1 - r_0}{r_0} + \left(\frac{r_1 - r_0}{r_0}\right)^2\right) JJ^+ = \frac{1}{2}\frac{r_1^2 - r_0^2}{r_0^2} JJ^+, \tag{6.3.3}$$

and $\mathrm{tr}(\varepsilon^*) = (r_1^2 - r_0^2)/r_0^2$. The Weingarten matrix W for the inflated sphere is $W = \frac{1}{r_1} I_{2\times 2}$. From (6.2.8), the stress tensor in the tangent space is

$$\begin{aligned}
\hat{\tau} &= \lambda^* \mathrm{tr}(\varepsilon^*) I_{2\times 2} + 2\mu J^+ \varepsilon^* J \\
&= \lambda^* \left(\frac{r_1^2 - r_0^2}{r_0^2}\right) I_{2\times 2} + \mu \left(\frac{r_1^2 - r_0^2}{r_0^2}\right) I_{2\times 2} \\
&= (\lambda^* + \mu) \frac{r_1^2 - r_0^2}{r_0^2} I_{2\times 2}.
\end{aligned}$$

From (6.2.10), the interface pressure is then

$$p = \mathrm{tr}(W\hat{\tau}) = \mathrm{tr}\left(\frac{1}{r_1}\hat{\tau}\right) = 2(\lambda^* + \mu)\frac{r_1^2 - r_0^2}{r_0^2 r_1} = \frac{E}{1-\nu}\frac{r_1^2 - r_0^2}{r_0^2 r_1}. \tag{6.3.4}$$

If the deformation is small, then $(r_1 + r_0)/r_0 \approx 2$, and we obtain

$$p \approx \frac{2E}{1-\nu}\frac{r_1 - r_0}{r_0 r_1} = \frac{2E}{1-\nu}\left(\frac{1}{r_0} - \frac{1}{r_1}\right). \tag{6.3.5}$$

This is similar to the result $p \approx 2K(1/r_0 - 1/r_1)$ in [49], where $K = E$ if $\nu = 0$. From (6.3.5), we can see that interface pressure is approximately inversely proportional to the radius, so it is harder to inflate a spherical balloon at the initial stage.

Uniformly Expanding Cylinder Next, consider a cylinder that expands along the radial direction. Unlike the example of spheres, for cylinders the 2×2 Weingarten matrix is no longer a multiple of identity matrix, so this case tests our theorem for an anisotropic geometry. Suppose the axis of the cylinder passes the origin and is parallel to the z-axis, and the radii of the cylinder without stress and under stress are r_0 and r_1, respectively. Consider the point $(r_0, 0, 0)$, and suppose the u direction is tangent to the cross section curve and the v direction is parallel z-axis, so $J = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $W = \frac{1}{r_1}\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$. Since $\phi = [(r_1 - r_0)/r_0 x, (r_1 - r_0)/r_0 y, 0]^T$

and $\nabla_x \phi = \text{diag}\left(\frac{r_1 - r_0}{r_0}, \frac{r_1 - r_0}{r_0}, 0\right)$, we have

$$\varepsilon^* = \frac{1}{2}\text{diag}\left(\frac{r_1^2 - r_0^2}{r_0^2}, \frac{r_1^2 - r_0^2}{r_0^2}, 0\right) JJ^+ = \frac{1}{2}\text{diag}\left(0, \frac{r_1^2 - r_0^2}{r_0^2}, 0\right), \tag{6.3.6}$$

and $\text{tr}(\varepsilon^*) = \frac{1}{2}(r_1^2 - r_0^2)/r_0^2$. Then,

$$\begin{aligned}
\hat{\tau} &= \lambda^* \text{tr}(\varepsilon^*) \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + 2\mu J^+ \varepsilon^* J \\
&= \frac{\lambda^*}{2} \frac{r_1^2 - r_0^2}{r_0^2} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \mu \frac{r_1^2 - r_0^2}{r_0^2} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \\
&= \left(\frac{\lambda^*}{2} + \mu\right) \frac{r_1^2 - r_0^2}{r_0^2} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},
\end{aligned}$$

and

$$p = \text{tr}(W\hat{\tau}) = \left(\frac{\lambda^*}{2} + \mu\right) \frac{r_1^2 - r_0^2}{r_0^2 r_1} = \frac{E}{2(1-\nu^2)} \frac{r_1^2 - r_0^2}{r_0^2 r_1}. \tag{6.3.7}$$

To obtain a reference solution, we apply a model for curves for the case of $\nu = 0$ to a circular cross section passing through the origin. Under small deformation, we can use an analysis analogous to [49]: The tension at any point on the circle is $K(r_1 - r_0)/r_0$, where K is a spring constant. The balance of force requires $p \approx K(r_1 - r_0)/(r_0 r_1) = K(1/r_0 - 1/r_1)$ along any arc, which approximates (6.3.7) when $K = E$, $\nu = 0$, and $(r_1 + r_0)/r_0 \approx 2$.

6.4 Numerical Experimentation

We present a series of examples and numerical experiments to verify our formulation and numerical discretization. In equation 6.2.10, the local stress tensor $\hat{\tau}$ is first-order quantities of the displacements and the Weingarten matrix W is second-order quantities of the surface itself, which can be calculated into high-order accuracy under our polynomial fitting computational framework

6.4.1 Numerical Experiments Under Nonuniform Expansion

First, we present some numerical experiments to demonstrate the accuracy and convergence of our discretizations of the tangential stress and interface pressure on a surface triangulation. For this purpose, we artificially expand a torus with inner

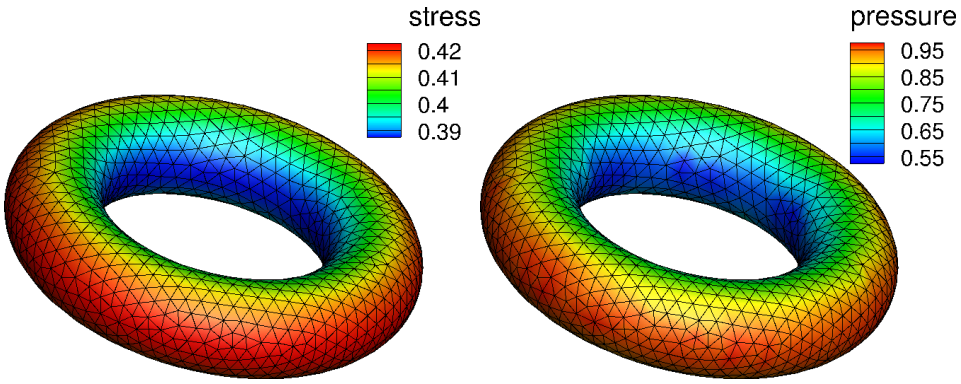


Figure 6.1: A torus with inner and outer radii 0.3 and 1 m was artificially expanded into one with inner and outer radii 0.32 and 1.05 m. Colors of left and right images indicate the tangential stress and interface pressure, respectively.

radius 0.3 m and outer radius 1 m into a torus with inner radius 0.32 m and outer radius 1.05 m, and compute the tangential stress and interface pressure using linear elastic model described in Section 6.1.2 with the scaled Young's modulus $E = 100$ kPa and Poisson ratio $\nu = 0.2$. Figure 6.1 shows the deformed surface color-coded by the "mean" tangential stress and also by the interface pressure. It can be seen that the stress (and also pressure) is larger at the outside than at the inside. The reason is that those areas have larger relative displacements in the tangent space.

To verify the accuracy of our numerical computations, we performed a grid convergence study. We used a series of five successively refined triangular meshes for the torus, where the average edge lengths are approximately halved at each refinement. We computed the reference solution analytically, and computed the numerical solutions using quadratic, cubic, and quartic least squares fittings. Figure 6.2 shows the L_2 and L_∞ errors of the computed interface pressure for the meshes. The average convergence rates are shown on the right-end of each curve, computed as $\log_2 |\varepsilon_5/\varepsilon_1|/4$, where ε_i denotes the error on the ith mesh. The numbers of vertices of the meshes are 328, 1,348, 5,269, 21,103 and 85,276, the numbers of faces are 565, 2,696, 10,538, 42,206 and 170,552 respectively.

The computation of interface pressure involves second-order derivatives. Based on the analysis in [37], when using dth-order polynomial fitting, the curvature tensor and hence the interface pressure are $(d-1)$st order accurate, given that the vertex positions are at least $(d+1)$st order accurate. The numerical result for cubic fitting agrees with this prediction. For quadratic and quartic fittings, the numerical results exhibited better convergence rates than predicted, most likely due to statistical error cancellation.

6.4.2 Deformation Under Pressure Differences

To demonstrate how our theory can be useful for more complex problems, we show some examples of computing the displacements of a membrane interface under given normal pressure differences. The equation we need to solve is

$$\nabla_\Gamma \cdot \sigma = [p]n, \tag{6.4.1}$$

given $[p]$ is a given pressure jump, and n is the normal to the deformed surface. This is the inverse problem of the one in the previous subsection and is much more difficult. It is relevant to fluid-structure interaction problems, where some pressure jump may be posed on the two sides of a membrane interface. For simplicity, we assume small strain and no shear deformation.

Here it is similar to the semi-implicit scheme for solving mean-curvature flow in Section 5. The gradient of the displacement vector ϕ and its derivatives are

$$\nabla_u \phi = \left[\sum_{j\in N(i)} C_{2,j}\phi_j \,\middle|\, \sum_{j\in N(i)} C_{3j}\phi_j \right],$$
$$\left[\frac{\partial^2\phi}{\partial u^2} \,\middle|\, \frac{\partial^2\phi}{\partial u\partial v} \,\middle|\, \frac{\partial^2\phi}{\partial v^2} \right] = \left[\sum_{j\in N(i)} 2C_{4j}\phi_j \,\middle|\, \sum_{j\in N(i)} C_{5j}\phi_j \,\middle|\, \sum_{j\in N(i)} 2C_{6j}\phi_j \right],$$

where ϕ_j denotes the displacement vector at vertex j, and C_{ij} denotes the (i,j) entry of the coefficient matrix C.

To discretize (6.4.1), we need to discretize the derivatives of ϕ, the Jacobian matrices $\bar{J} = \nabla x$ and $J = \nabla(x+\phi)$ for the undeformed and deformed configurations, respectively, as well as $\bar{J}_u$, $\bar{J}_v$, J_u, and J_v. These derivatives can all be discretized as above. However, because of the presence of ϕ in J, (6.4.2) results in a nonlinear system of equations after discretization. To solve this nonlinear system, we devise an iterative process to compute J, its derivatives, and n explicitly using the displacement vector from the previous iteration, starting with $\phi^{(0)} = 0$ in the first step. Let m denote the number of vertices of the surface mesh. At the kth step, we solve a linear system

$$M^{(k)}\phi^{(k)} = p^{(k)}, \tag{6.4.2}$$

where $M^{(k)}$ is a $3m \times 3m$ matrix, and $p^{(k)}$ and $\phi^{(k)}$ are column vectors of length $3m$, composed of the traction vectors and displacement vectors at the vertices, respectively. Note that $M^{(k)}$ and $p^{(k)}$ depend on $\phi^{(k-1)}$.

For closed surfaces, the linear system (6.4.2) is underdetermined when no boundary conditions are specified, because there are six extra degrees of freedom for translation and rotation, because the solution is invariant of translation and rotation. We

use the procedure of truncated singular value decomposition (SVD) to solve the linear system. Let the SVD decomposition of $M^{(k)}$ be

$$M^{(k)} = U\Sigma V^T, \tag{6.4.3}$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix composed of the singular values of $M^{(k)}$. The last six diagonal entries of Σ are nearly 0. To solve for $\phi^{(k)}$, we discarded the last six singular values and their corresponding singular vectors and compute $\phi^{(k)}$ as

$$\phi^{(k)} = \sum_{j=1}^{3n-6} \frac{1}{s_j} v_j u_j^T p^{(k)}, \tag{6.4.4}$$

where s_j denotes the jth entry of Σ, and u_j and v_j denote the jth column of U and V, respectively. This numerical discretization also applies to open surfaces with Dirichlet or other boundary condition.

For demonstration purpose, we report some experimental results for the deformations of an ellipsoid under some uniform pressure difference. We use the scaled Young's modulus $E = 100$ kPa and Poisson ratio $\nu = 0.2$. Figure 6.3(a) shows the result for an ellipsoid with semi-axes 1.2, 1, and 1 under a pressure load difference $[p] = 2$ kN/m. Figure 6.3(b) shows the result for a unit half sphere under the same configuration with the boundary fixed fixed. Displacement vectors are shown in the figures.

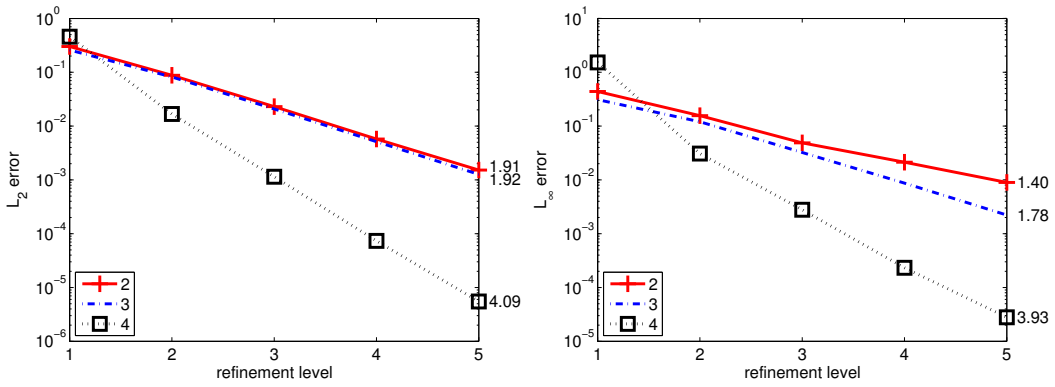
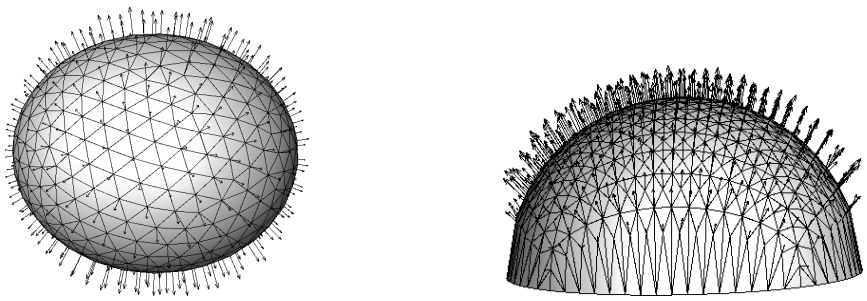


Figure 6.2: Convergence results for computed interface pressure under grid refinement for the torus. Average convergence rates are shown on the right-end of each curve.



(a) (b)

Figure 6.3: Expansion of an ellipsoid (a) and a half sphere (b) under a small pressure load under a pressure difference $[p] = 2$ kPa.

Chapter 7

Conclusions and Future Work

In this dissertation, we propose a computational framework to process discrete surface mesh based on weighted least square polynomial fitting. Several applications have been presented with rigorous theoretical analysis and extensive numerical experimental results, including calculation of differential quantities, surface reconstruction and integration, numerical solutions to geometric PDEs and an elastic membrane problem. In all those applications, our approach delivers high order accuracy results and is tolerant to noise due to its least square nature.

For the future work, there are several directions we could further explore. For the local parametrization, we adopt the local orthogonal projection method, mesh folding is avoided by imposing different weights to different neighbor points. When there are few points around a highly curved area, we could use the parametrization based on cylindrical or spherical mappings. For surface reconstruction, we assume vertices coordinate values are the only information available, but for some models, normal directions are also given, we can modify our framework to incorporate this additional information, thus improve the accuracy of the reconstructed surface. For mean curvature flow and surface diffusion, a high order temporal discretization is needed to achieve overall high order accuracy. The generalized finite difference and finite volume methods could be unified under the same theoretical framework.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. SIAM, 3rd edition, 1999.

[2] E. Bänsch, P. Morin, and R. H. Nochetto. Surface Diffusion of Graphs: Variational Formulation, Error Analysis, and Simulation. *SIAM Journal on Numerical Analysis*, 42(2):773, 2004.

[3] E. Bänsch, P. Morin, and R. H. Nochetto. A finite element method for surface diffusion: the parametric case. *J. Comput. Phys.*, 203:321–343, 2005.

[4] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on computer graphics and interactive techniques*, SIGGRAPH '98, pages 43–54, New York, NY, USA, 1998. ACM.

[5] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Comput. Methods Appl. Mech. Engrg.*, 139:3–47, 1996.

[6] M. Bischoff, W. A. Wall, K.-U. Bletzinger, and E. Ramm. Models and finite elements for thin-walled structures. In E. Stein, R. de Borst, and T. J. R. Hughes, editors, *Encyclopedia of Computational Mechanics*, volume 2: Solids, Structures and Coupled Problems, chapter 3. Wiley, 2004.

[7] V. Borrelli, F. Cazals, and J.-M. Morvan. On the angular defect of triangulations and the pointwise approximation of curvatures. *Comput. Aided Geom. Des.*, 20(6):319–341, 2003.

[8] J. W. Cahn and J. E. Taylor. Surface motion by surface diffusion. *Acta Metallurgica et Materialia*, 42(4):1045–1063, 1994.

[9] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Comput. Aid. Geom. Des.*, 22:121–146, 2005.

[10] F. Cazals and M. Pouget. Jet_fitting_3: A generic C++ package for estimating the differential properties on sampled surfaces via polynomial fitting. *ACM Trans. Math. Softw.*, 35(3):1–20, 2008.

[11] D. Chien. Numerical evaluation of surface integrals in three dimensions. *Mathematics of Comp*, 64:727–743, 1993.

[12] D. L. Chopp and J. A. Sethian. Motion by intrinsic Laplacian of curvature. *Interfaces and Free Boundaries*, 1:107–123, 1999.

[13] D. Cohen-Steiner and J.-M. Morvan. Restricted Delaunay triangulations and normal cycle. In *Proc. of 19th Annual Symposium on Computational Geometry*, pages 312–321, 2003.

[14] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *Proceedings of Eurographics 2002 Conference*, pages 209–218, 2002.

[15] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[16] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser Boston, 1992.

[17] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Comput. Graph. (TOG)*, 24(3), 2005.

[18] M. S. Floater. Mean value coordinates. *Comput. Aid. Geom. Des.*, 20:19–27, 2003.

[19] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 157–186. Springer Verlag, 2005.

[20] P. J. Frey. About surface remeshing. In *Proceedings of 9th International Meshing Roundtable*, pages 123–136, Oct. 2000.

[21] P. J. Frey. Yams: A fully automatic adaptive isotropic surface remeshing procedure. Technical report, INRIA, 2001. RT-0252.

[22] R. Garimella. Triangular and quadrilateral surface mesh quality optimization using local parametrization. *Comput. Meth. Appl. Mech. Engrg.*, 193(9-11):913–928, 2004.

[23] T. Gatzke and C. Grimm. Estimating curvature on triangular meshes. *Int. J. Shape Modeling*, 12:1–29, 2006.

[24] K. Georg. Approximation of integrals for boundary element methods. *SIAM J. Sci. Stat. Comput*, 12:443–453, 1991.

[25] J. Goldfeather and V. Interrante. A novel cubic-order algorithm for approximating principal direction vectors. *ACM Trans. Graph.*, 23:45–63, 2004.

[26] G. H. Golub and C. F. Van Loan. *Matrix Computation*. Johns Hopkins, 3rd edition, 1996.

[27] C. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization for 3d meshes. *SIGGRAPH*, 22:358–363, 2003.

[28] W. G. Gray, A. Leijnse, R. L. Kolar, and C. A. Blain. *Mathematical Tools for Changing Spatial Scales in the Analysis of Physical Systems*. CRC Press, Boca Raton, FL, 1993.

[29] E. Grinspun, Y. Gingold, J. Reisman, and D. Zorin. Computing discrete shape operators on general meshes. *Eurographics (Computer Graphics Forum)*, 25:547–556, 2006.

[30] K. Hildebrandt, K. Polthier, and M. Wardetzky. On the convergence of metric and geometric properties of polyhedral surfaces. *Geometria Dedicata*, 123:89–112, 2006.

[31] K. D. Hjelmstad. *Fundamentals of Structural Mechanics*. Springer, New York, 2 edition, 2005.

[32] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. *Comput. Meth. Appl. Mech. Engrg.*, 194:4135–4195, 2005.

[33] X. Jiao. Volume and feature preservation in surface mesh optimization. In *Proceedings of 15th International Meshing Roundtable*, 2006.

[34] X. Jiao and N. Bayyana. Identification of C^1 and C^2 discontinuities for surface meshes in CAD. *Comput. Aid. Des.*, 40:160–175, 2008.

[35] X. Jiao and D. Wang. Reconstructing high-order surfaces for meshing. *Engineering with Computers*, 2011. Available online. DOI: 10.1007/s00366-011-0244-8.

[36] X. Jiao, D. Wang, and H. Zha. Simple and effective variational optimization of surface and volume triangulations. In *Proceedings of 17th International Meshing Roundtable*, pages 315–332, 2008.

[37] X. Jiao and H. Zha. Consistent computation of first- and second-order differential quantities for surface meshes. In *ACM Solid and Physical Modeling Symposium*, pages 159–170. ACM, 2008.

[38] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting: An Introduction*. Academic Press, 1986.

[39] L. D. Landau and E. M. Lifshitz. *Fluid Mechanics*. Butterworth-Heinemann, Oxford, 1987.

[40] T. Langer, A. G. Belyaev, and H.-P. Seidel. Exact and interpolatory quadratures for curvature tensor estimation. *Comput. Aid. Geom. Des.*, 24:443–463, 2007.

[41] P. S. Laplace. *Méchanique Céleste*, volume suppl. 10th vol. 1806.

[42] D. Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67:1517–1531, 1998.

[43] B. Lévy, S. Petitjean, N. Rry, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21:362–371, 2002.

[44] J. N. Lyness and R. Cools. A survey of numerical cubature over triangles. In *Proc. of Symposia in Applied Mathematics*, volume 48, pages 127–150, 1994.

[45] J. E. Marsden and T. J. Hughes. *Mathematical Foundations of Elasticity*. Prentice-Hall, 1983.

[46] D. S. Meek and D. J. Walton. On surface normal and Gaussian curvature approximations given data sampled from a smooth surface. *Comput. Aid. Geom. Des.*, 17:521–543, 2000.

[47] M. Meyer, M. Desbrun, P. Schröder, and A. Barr. Discrete differential geometry operators for triangulated 2-manifolds. In H.-C. Hege and K. Poltheir, editors, *Visualization and Mathematics*, volume 3, pages 34–57. Springer, 2002.

[48] H. N. Ng and R. L. Grimsdale. Computer graphics techniques for modeling cloth. *IEEE Computer Graphics and Applications*, 16:28–41, 1996.

[49] W. A. Osborne and W. Sutherland. The elasticity of rubber balloons and hollow viscera. *Proc. R. Soc. Lond.*, 81:485–499, 1909.

[50] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: algorithms Based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79:12–49, 1988.

[51] J. Peters and U. Reif. *Subdivision Surfaces*. Springer, 2008.

[52] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Exper. Math.*, 2:15–36, 1993.

[53] S. Rosenberg. *The Laplacian on a Riemannian Manifold: An Introduction to Analysis on Manifolds*. Cambridge University Press, Cambridge, 1998.

[54] S. Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *Proc. of 2nd International Symposium on 3D Data Processing, Visualization and Transmission*, pages 486–493, 2004.

[55] L. Saboret, P. Alliez, and B. Lévy. Planar parameterization of triangulated surface meshes. In C. E. Board, editor, *CGAL User and Reference Manual*. 3.3 edition, 2007. Online at http://www.cgal.org.

[56] I. B. Semenova, V. V. Savchenko, and I. Hagiwara. Two techniques to improve mesh quality and preserve surface characteristics. In *Proceedings of 13th International Meshing Roundtable*, pages 277–288, 2004.

[57] P. Smereka. Semi-implicit level set methods for curvature and surface diffusion motion. *Journal of Scientific Computing*, 19:439–456, 2003.

[58] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proc. of Int. Conf. on Computer Vision*, pages 902–907, 1995.

[59] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[60] W. T. Tutte. How to draw a graph. *Proceedings London Mathematical Society*, 13:743–768, 1963.

[61] A. Van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *J. Graphics Tools*, 3:21–42, February 1998.

[62] E. Ventsel and T. Krauthammer. *Thin Plates and Shells: Theory, Analysis, and Applications*. Marcel Dekker, Inc., 2001.

[63] D. Walton. A triangular G1 patch from boundary curves. *Comput. Aid. Des.*, 28(2):113–123, 1996.

[64] M. Wardetzky. Convergence of the cotan formula - an overview. In A. I. Bobenko, J. M. Sullivan, P. Schröder, and G. Ziegler, editors, *Discrete Differential Geometry*, pages 275–286. Birkhäuser Basel, 2007.

[65] Z. U. A. Warsi. *Fluid Dynamics: Theoretical and Computational Approaches*. CRC, New York, 2006.

[66] G. Xu. Convergence of discrete Laplace-Beltrami operators over surfaces. *Comput. Math. Appl.*, 48:347–360, 2004.

[67] G. Xu. Consistent approximation of some geometric differential operators. Technical report, Institute of Computational Mathematics, Chinese Academy of Sciences, 2007. Research Report No. ICM-07-02.

[68] G. Xu and Q. Zhang. A general framework for surface modeling using geometric partial differential equations. *Computer Aided Geometric Design*, 25(3):21, 2008.

[69] G. Xu and Q. Zhang. A general framework for surface modeling using geometric partial differential equations. *Comput. Aid. Geom. Des.*, 25:181–202, 2008.

[70] G. Xu and Q. Zhang. A general framework for surface modeling using geometric partial differential equations. *Comput. Aid. Geom. Des.*, 25:181–202, 2008.

[71] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis & Fundamentals*. Elsevier, 6th edition, 2005.