High-Order Surface Reconstruction and its Applications to Surface Integrals and Surface Remeshing

A Dissertation Presented

by

Navamita Ray

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

December 2013

Stony Brook University

The Graduate School

Navamita Ray

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

James Glimm – Chairperson of Defense
Distinguished Professor, Department of Applied Mathematics and Statistics

Xiaolin Li
Professor, Department of Applied Mathematics and Statistics

Timothy Tautges
Computational Scientist, Math and Computer Science Division
Argonne National Labratory

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

High-Order Surface Reconstruction and its Applications to Surface Integrals and Surface Remeshing

by

Navamita Ray

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2013

High-order surface reconstruction is a numerical technique to obtain high-order approximations of both geometry and its differential quantities such as normals, curvatures, etc., over a discrete surface mesh. Its computational framework is based on local polynomial fittings using a weighted least squares approach. In this dissertation, we complete the scope of this framework to compute high-order approximations of surface integrals and demonstrate the application of the complete framework to various mesh-based numerical computations for high-order numerical methods. The computational framework relies on an efficient underlying mesh data structure for various traversal queries. For this purpose, an array-based mesh data structure was developed to represent the mesh for efficient mesh query and modification operations. Our methods are mainly developed for applications with high-order methods in mind.

Surface integration is a fundamental operation in many scientific and engineering applications. The standard methods for numerical computation are generally limited to second-order of accuracy due to lower-order approximations to geometry and integrand. This limitation is overcome by extending the computational framework for high-order surface reconstruction to a function defined over the surface and coupling it with high-order quadrature rules. We theoretically analyze the accuracy of our method and prove that it can achieve high-order of accuracy and verify it with numerical experiments as well. A widely used operation by many applications is the modification of the surface mesh by vertex redistribution, edge flipping, refinement or coarsening, such that the resulting mesh improves certain properties such as mesh quality, error distribution, etc. It is vital to preserve the geometric accuracy of the mesh as it undergoes the modification operations. Our computational framework provides an efficient high-order point projection strategy that can be easily coupled with various mesh quality improving techniques. We develop remeshing strategies coupling existing mesh quality improving techniques with high-order surface reconstruction, to produce high-quality and high-order accurate surface meshes. The developed algorithms are made robust to allow untangling mildly folded triangles and also take into account the approximation issues related to high-order approximations in under-resolved regions. All of our algorithms are based on an array-based half-facet mesh data structure called AHF, for efficient mesh query and modification operations. It was developed for 2D/3D non-manifold meshes with mixed-dimensional submeshes for increased applicability.

We present the theoretical framework of our methods, show experimental comparisons against other methods, and demonstrate their utilization to geometric PDE's, high-order finite elements, biomedical image-based surface meshes, and complex interface meshes in fluid simulations.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Xiangmin Jiao for his valuable guidance and support throughout my study. It has been a great pleasure and valuable experience to work with him. I have learned a lot from his rigorous and scientific way of understanding and solving a problem, as well as gain from his vast experience in coding and scientific writing. I express my gratitude for all his help. I also thank the Department of Applied Mathematics and Statistics and my advisor for the financial support provided during my graduate study.

I would like to thank collaborators Prof. James Glimm of Stony Brook University, Dr. Daniel Einstein of Pacific National Laboratory and Dr. Timothy Tautges of Argonne National Laboratory for providing the oppurtunity to work on interesting problems, guidance as well as financial support.

I would like to thank past and current members of our research group for all their friendship, help and encouragement. In particular, Rebecca Conley, Xinglin Zhao, Cao Lu, Hong Xu, Tristan Delaney, Dr. Bryan Clark, Dr. Duo Wang, Dr. Vladimir Dyedov and Dr. Ying Chen. I would also like to thank Dr. Duo Wang, Dr. Bryan Clark, Dr. Vladimir Dyedov and Tristan Delanay for allowing me to use some of their experimental results in this disseration. It was a great pleasure to work with Yijie Zhou and Hyunkyun Lim.

I thank my brother Tathagata Ray for his love and support. Finally, I dedicate my dissertation to the memory of my dear parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Surface meshes are widely used in various scientific and engineering problems such as computer graphics, image analysis, physical and biological simulations. They not only represent computational grids for various discretization methods, but are also numerical objects in themselves. The accuracy of numerical methods especially high-order methods, is highly dependent on the geometrical accuracy of the mesh as well as on that of differential or integral quantities defined over it. The situation is further complicated if the surface mesh does not have an underlying CAD model representation or is evolving, in which case it is extremely difficult to maintain a CAD representation. As a result one generally has access to a discrete mesh for numerical computations. We consider the problem of computing high-order approximations over discrete surface meshes and their use in various mesh-based computations.

In [31], a weighted least squares based local polynomial fitting approach was proposed to compute high-order accurate[1] approximations to both geometry and differential quantities such as normals, curvatures, etc. Since the local polynomial fittings at nodes of the surface mesh do not result in a continuous geometrical support for the mesh, two methods were proposed in [29, 51] to obtain such continuous support for the geometry. This numerical technique is known as *high-order surface reconstruction* over discrete meshes. So far the proposed methods were used to compute high-order approximations to the geometry and differential quantities. In this dissertation, we complete the scope of this framework to compute high-order

[1]By high-order accuracy, we mean that the errors converge at a rate of third or higher order with respect to some measure of edge lengths under mesh refinement. Note that high-order accuracy does not imply high degree of continuity, and conversely high degree of continuity does not imply high-order accuracy either.

approximation of surface integrals and demonstrate the application of the complete framework to various mesh-based numerical computations for high-order numerical methods.

The motivation is the fundamental importance of surface integration in many scientific and engineering applications. It is widely used by numerical methods such as finite element, boundary integral, meshless methods, etc., and forms the basis of geometric primitives such as surface area and volume, etc. As an analytic antiderivative is rarely available in practice, it is mostly computed numerically using approximations to both geometry and the integrand along with quadrature rules. The standard methods are generally limited to second-order of accuracy due to lower-order approximations to geometry and integrand. However, for many numerical methods, especially high-order methods, it is important to compute the integrals to high-order of accuracy. This limitation can be overcome by extending the computational framework for high-order surface reconstruction to do the same for a function defined over the surface and couple it with high-order quadrature rules. We theoretically analyze the accuracy of the method, prove that it can indeed achieve high-order of accuracy and verify it with numerical experiments. This completes the scope of the underlying framework based on local polynomial fittings to compute high-order approximations over a discrete surface to the geometry, its differential quantities, functions defined over it and surface integration with provable accuracy.

We next focus on a mesh-based operation known as surface remeshing that is widely used by many applications. It modifies the surface mesh by vertex redistribution, edge flipping, refinement or coarsening such that the resulting mesh improves certain properties such as mesh quality, error distribution, etc. Apart from the improvement in the targeted property, it is also vital to preserve the geometrical accuracy of the mesh after these operations have taken place, especially for high-order numerical methods. The problem is particularly challenging if no CAD model is available for the underlying geometry, and is even more so if the surface meshes contain some inverted elements. We investigate this problem of optimizing, adapting, and untangling a surface triangulation with high-order accuracy. We develop remeshing strategies that couple various mesh quality improving techniques with high-order surface reconstruction producing high-quality triangular meshes, while untangling mildly folded triangles and preserving the geometry to high-order accuracy.

We observe that high-order point projection works well for sufficiently resolved meshes. However, for under-resolved regions, such as those representing highly curved regions, it might become problematic and result in artifical spikes. This makes it important to introduce safeguards called limiters to properly reduce to a lower-order point projection in such problematic regions. We demonstrate the effectiveness of our methods by experimental comparisons against other methods, as well as show its utilization to geometric PDE's, high-order finite elements, biomedical image-based surface meshes, and complex interface meshes in fluid simulations.

Any meshing algorithm or mesh-based numerical method relies on its mesh data structures to perform a range of operations such as traversal, query and modification. Thus the underlying data structure strongly influences the overall performance and storage requirement of the algorithm. For our algorithms, we developed an array-based half-facet mesh data structure, denoted as AHF, which can perform efficient mesh query and modification operations. To increase its applicability, AHF was developed to support 2D/3D non-manifold meshes with mixed-dimensional submeshes. We present some comparisons of the memory requirements and computational costs, and also demonstrate its effectiveness with a few sample applications.

There are four main contributions of this work. First, we extend the concept of high-order surface reconstruction to do the same for a function defined on the surface. This allows us to compute high-order approximations of surface integrals over discrete surface. Second, we integrate various mesh operations with high-order surface reconstruction to preserve geometrical accuracy, and develop algorithms for mesh smoothing, optimization, and adaptation. We improve the robustness of high-order reconstructions to allow remeshing of under-resolved meshes by introducing geometric limiters and untangling mildly folded triangles. Third, we develop a general, efficient and simple mesh data structure that is used as the underlying mesh representation for all of our algorithms. Finally, we apply our methods to a variety of applications with complex meshes, including those from biomedical images and interface meshes in fluid simulations.

1.2 Organization

The dissertation is organized as follows. In chapter 2, we describe briefly the main concepts and results of high-order surface reconstruction [29, 31, 51]. These concepts will be generalized and coupled with various mesh-based operations in later chapters.

In chapter 3, we develop a compact, efficient and general array-based mesh data structure AHF that forms the underlying mesh representation for our algorithms. We present some comparisons of the memory requirements and computational costs, and also demonstrate its effectiveness with a few sample applications [16].

In chapter 4, we solve the problem of computing high-order accurate numerical integration of a function over a discrete surface. This involves generalizing the concept of high-order surface reconstruction to that of high-order function reconstruction and coupling them together with high-degree quadrature rules. We present theoretical analysis of the accuracy of our method as well as experimental results of up to sixth order accuracy [44].

In chapter 5, we investigate the problem of optimizing, adapting, and untangling a surface triangulation with high-order accuracy in a robust manner, so that the resulting mesh has sufficient accuracy for high-order numerical methods [12, 43]. We present the theoretical framework of our methods as well as experimental comparisons against other methods.

In chapter 6, we demonstrate the utilization of high-order surface remeshing to a number of applications such as solving geometric PDE's, high-order finite elements, biomedical image-based surface meshes and complex interface meshes in fluid simulations.

Chapter 2

High-Order Surface Reconstruction

We begin with a brief description of the concepts involved in high-order surface reconstruction over a discrete mesh. We first describe the local polynomial fitting using a weighted least squares approach. The main assumption of the local polynomial fittings using a weighted least squares approach is that the input discrete mesh is representative of 2-manifold regular surface. Using the powerful Taylor series expansion to approximate the neighborhood of a point on the surface to high-order not only allows better geometrical approximation but also computation of high-order and convergent differential quantities. We also discuss neighborhood selection and weights for linear systems as well as robust solution of the linear system. Finally, we describe two methods to obtain C^0 continuous geometric support for the discrete mesh namely, *Weighted Averaging of Local Fittings* (*WALF*) *surface* and *continuous moving frame* (*CMF*) *surface*. This section is mainly based on the work done in [29, 31, 51].

2.1 Local Polynomial Fittings

2.1.1 Local Polynomial Fitting at a Point

Taylor series expansion is perhaps the most powerful tool in numerical analysis that is widely used for numerical approximations. They form the basic computational unit for local polynomial fittings. We assume that the given discrete mesh is representative of a manifold and regular surface. Therefore, a small neighborhood of a point on the surface can be approximated using a bivariate polynomial which can be obtained by fitting a Taylor series truncated to some order. It is important to note that such polynomial fittings only require a local parametrization around the point, instead of a global parametrization of the surface mesh. Each point 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, simplifying the problems both theoretically and computationally.

The neighborhood a point P under a local parameterization with parameters u and v, where P corresponds to $u = 0$ and $v = 0$ can be approximated or interpolated by fitting a truncated Taylor polynomial. 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* $\mathbf{f}(u,v) = [x(u,v), y(u,v), z(u,v)]$. 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 $\mathbf{u}$ denote $[u,v]^T$ and $f(\mathbf{u})$ 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 $\frac{\partial^{j+k}}{\partial u^j \partial v^k} f(\mathbf{0})$. Given a positive integer d, if $f(\mathbf{u})$ has $d+1$ continuous derivatives, it can be approximated to $(d+1)$th order accuracy about the origin $\mathbf{u}_0 = [0,0]^T$ by

$$f(\mathbf{u}) = \underbrace{\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!}}_{\text{Taylor polynomial}} + \underbrace{\sum_{j,k\geq 0}^{j+k=d+1} \frac{\partial^{j+k}}{\partial u^j \partial v^k} f(\tilde{u},\tilde{v}) \frac{\tilde{u}^j \tilde{v}^k}{j!k!}}_{\text{remainder}}, \tag{2.1.1}$$

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 $d+1$, which is generally known as the *regularity assumption*. The derivatives of the Taylor polynomial are the same as f at $\mathbf{u}_0$ 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. Thus, 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 $\mathbf{u}_0 = [u_0, v_0, f_0]^T$ plugging in each given point into (2.1.1) gives us an approximate equation

$$\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u_i^j v_i^k}{j!k!} \approx f_i, \tag{2.1.2}$$

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 $\mathbf{u}_0$ by setting $c_{00} = 0$ and removing the equation corresponding to $\mathbf{u}_0$, 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* as the fitting is obtained for a local neighborhood around point $\mathbf{u}_0$.

Let us denote the rectangular linear system obtained from (2.1.2) as

$$\mathbf{V}\mathbf{c} \approx \mathbf{f}, \tag{2.1.3}$$

where $\mathbf{c}$ is an n-vector composed of c_{jk}, and $\mathbf{V}$ is a *generalized Vandermonde matrix*. For a local height function, $\mathbf{f}$ is an m-vector composed of f_i; for a parametric surface, $\mathbf{f}$ is an $m \times 3$ matrix, of which each column corresponds to a component of the coordinate function. This idea is generalized to obtain fittings for functions defined on the surface. We discuss more about it in section 4.3.2.

By solving this equation we get the coefficient vector $\mathbf{c}$. Together with the coordinate values of vertex P and the two tangent vectors, we have the full information of the approximated local surface geometry. Numerically, (2.1.3) can be solved using the framework of *weighted linear least squares* [23, p. 265], i.e., to minimize a weighted norm (or semi-norm),

$$\min_{\mathbf{c}} \|\mathbf{V}\mathbf{c} - \mathbf{f}\|_{\mathbf{W}} = \min_{\mathbf{c}} \|\mathbf{W}(\mathbf{V}\mathbf{c} - \mathbf{f})\|_2, \tag{2.1.4}$$

where $\mathbf{W}$ is a *weighting matrix*. Typically, $\mathbf{W}$ 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 $\mathbf{V}$. This formulation is equivalent to the linear least squares problem

$$\tilde{\mathbf{V}}\mathbf{c} \approx \mathbf{b}, \text{ where } \tilde{\mathbf{V}} = \mathbf{W}\mathbf{V} \text{ and } \mathbf{b} = \mathbf{W}\mathbf{f}. \tag{2.1.5}$$

In general, $\tilde{\mathbf{V}}$ is $m \times n$ and $m \geq n$. However, this linear system may be rank deficient (i.e., the column vectors of $\tilde{\mathbf{V}}$ may not be linearly independent) or ill-conditioned (i.e., the singular values of $\tilde{\mathbf{V}}$ 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 $\tilde{\mathbf{V}}$ can be improved substantially by introducing a *scaling matrix* $\mathbf{S}$ changing the problem to

$$\min_{\mathbf{d}} \|\mathbf{A}\mathbf{d} - \mathbf{b}\|_2, \text{ where } \mathbf{A} = \tilde{\mathbf{V}}\mathbf{S} \text{ and } \mathbf{d} = \mathbf{S}^{-1}\mathbf{c}. \tag{2.1.6}$$

Here $\mathbf{S}$ is typically a diagonal matrix. Let $\tilde{\mathbf{v}}_i$ denote the ith column of $\tilde{\mathbf{V}}$. The ith diagonal entry of $\mathbf{S}$ is typically chosen to be either $\|\tilde{\mathbf{v}}_i\|_\infty$ [10, 53] or $\|\tilde{\mathbf{v}}_i\|_2$ [31], where the latter approximately minimizes the condition number of $\tilde{\mathbf{V}}\mathbf{S}$ [23, p. 265]. However, the problem may still be ill-conditioned after rescaling, and we will address this issue in the next section.

2.1.2 Robust Solution of Least Squares Fits

The weighted least squares problem described in the previous section is applicable to any scenario where a set of points approximates a surface. For us, we have a piecewise linear triangulation of the surface as the input and thus the local fittings are generally constructed at vertices of the 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.

Before we discuss the solution of the linear system, we note that the weighting matrix **W** assigns priorities to different rows of the linear system corresponding to different points that are being fit. Thus **W** has no effect on the solution if **V** is a non-singular square matrix, but different **W** would lead to different solutions for rectangular matrices. Let $\hat{\mathbf{m}}_\mathbf{i}$ denote a first-order approximation to the unit normal at the *i*th vertex (e.g., obtained by averaging face normals). In general, for the *i*th row corresponding to the *i*th point $\mathbf{x}_i$, it is desirable to assign its weight w_i to some larger value if $\mathbf{x}_i$ is close to the origin of the local coordinate system $\mathbf{x}_0$, or a smaller value (or even zero) if $\mathbf{x}_i$ is far away from $\mathbf{x}_0$ or its normal $\hat{\mathbf{m}}_i$ is too wide an angle from the normal $\hat{\mathbf{m}}_0$ at $\mathbf{x}_0$. In particular, we choose the weight at the *i*th vertex as

$$w_i = \frac{\gamma_i^+}{\left(\|\mathbf{u}_i\|^2/h+\varepsilon\right)^{d/2}}, \tag{2.1.7}$$

where $\gamma_i^+ \equiv \max(0, \hat{\mathbf{m}}_i^T \hat{\mathbf{m}}_0)$, $h \equiv \sum_{i=1}^m \|\mathbf{u}_i\|^2/m$, and $\varepsilon \approx 0.01$. The factor γ_i^+ serves as a safeguard against drastically changing normals for coarse meshes or non-smooth areas. The denominator $\left(\|\mathbf{u}_i\|^2/h+\varepsilon\right)^{d/2}$ prevents the weights from becoming too large at points that are too close to $\mathbf{u}_0$ and makes the weights approximately equal to $w_i \approx \left(\|\mathbf{u}_i\|^2/h\right)^{-d/2}$.

Because **W** allows the flexibility to under-weigh (and even filter out) undesirable points, we use a simple procedure to select points based on mesh connectivity when constructing the linear system. In particular, we use a $k-$ring neighborhood with half-ring increments:

- The *1-ring neighbor faces* of a vertex v are the faces incident on v, and the *1-ring neighbor vertices* are the vertices of these faces.
- The *1.5-ring neighbor faces* are the faces that share an edge with a 1-ring neighbor face, and the *1.5-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, and the $(k+1.5)$*-ring*

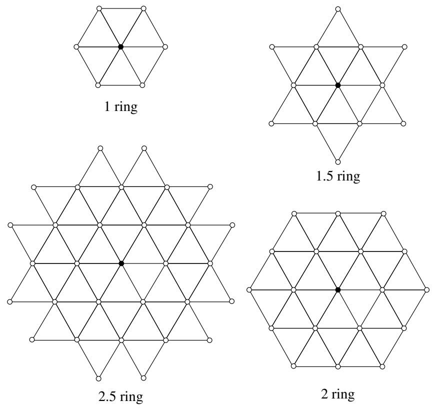


Figure 2.1: Stencil for point selection

neighborhood is the union of the 1.5-ring neighbors of the k-ring neighbor vertices.

Figure 2.1 illustrates the definitions up to 2.5 rings. In general for dth degree fitting, we use the $(d+1)/2$-ring for accurate input or $(d/2+1)$-ring for relatively noisy input.

Unlike $\mathbf{W}$, the scaling matrix $\mathbf{S}$ does not change the exact solution of $\mathbf{c}$. However, $\mathbf{S}$ can significantly improve the conditioning of the linear system and in turn improve the accuracy in the presence of rounding errors. Note that the scaling matrix $\mathbf{S}$ cannot improve the condition number of $\mathbf{V}$ if the ill-conditioning is caused by the lack of points or some unfortunate selection of points, which is a well-known issue [38]. It can be alleviated by including additional points in the fitting if possible. However, if the number of points is fixed, solving this ill-conditioned system is similar to solving an under-constrained linear system. 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.

Instead of using the truncated SVD to solve this linear system, we use a variant of the reduced QR factorization to address the problem. Let the reduced QR factorization of $\mathbf{WVS}$ be

$$\mathbf{WVS} = \mathbf{QR}, \tag{2.1.8}$$

where $\mathbf{Q}$ is $m \times n$ with orthonormal column vectors and $\mathbf{R}$ is a $n \times n$ upper-triangular matrix. The condition number of $\mathbf{WVS}$ is the same as that of $\mathbf{R}$, which can be estimated accurately and efficiently using a variant of back substitution. If the condition number of $\mathbf{R}$ is too large (e.g., $\geq 10^6$), we then reduce the degree of the fitting by removing the last few columns that correspond to the highest derivatives. Let $\tilde{\mathbf{Q}}$ and

$\tilde{\mathbf{R}}$ denote the reduced matrices. The final solution of $\mathbf{x}$ is given by

$$\mathbf{x} = \mathbf{T}\tilde{\mathbf{R}}^{-1}\tilde{\mathbf{Q}}^T\mathbf{S}\mathbf{b}, \tag{2.1.9}$$

where $\tilde{\mathbf{R}}^{-1}$ denotes a back substitution step. Compared to the solution based on SVD, this procedure is more accurate asymptotically as it gives highest priority to the lower-order coefficients of the polynomial while maintaining good scaling of the matrix, and at the same time it is more efficient than SVD. If the degree of fitting is reduced due to either an ill-conditioned $\tilde{\mathbf{R}}$ or insufficient number of points in the stencil, a cure could be increasing the size of the stencil so that the system is more stable. The weighted local least squares polynomial fitting provides us the theoretical foundation for high-order reconstruction of surfaces, established by the following proposition [31]:

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 weighting matrix are independent of the mesh resolution, and the condition number of the linear system is bounded by some constant. The degree-d weighted least squares fitting approximates c_{jk} to $O(h^{d-j-k+1})$.*

2.2 Surface Reconstruction: WALF and CMF

The local polynomial fitting method described in Section 2.1.1 is computed locally at each vertex of the mesh. Since there is no coordination among the local fittings at different vertices, the method does not reconstruct a global continuous surface. In this section we describe two methods that reconstructs a global C^0 continuous surface from the local polynomial fittings. The first method blends the local fittings of all vertices of an element by a weighted averaging with the linear finite element basis functions as weights. This method is referred to as *Weighted Averaging of Local Fittings* (*WALF*). The second method enforces continuity of local coordinate frames and weights for local fittings. This method is referred as *Continuous Moving Frame* (*CMF*).

2.2.1 Weighted Averaging of Local Fittings (WALF)

The key idea is that the polynomial at each vertex gives a high-order approximation to the surface over the stencil 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

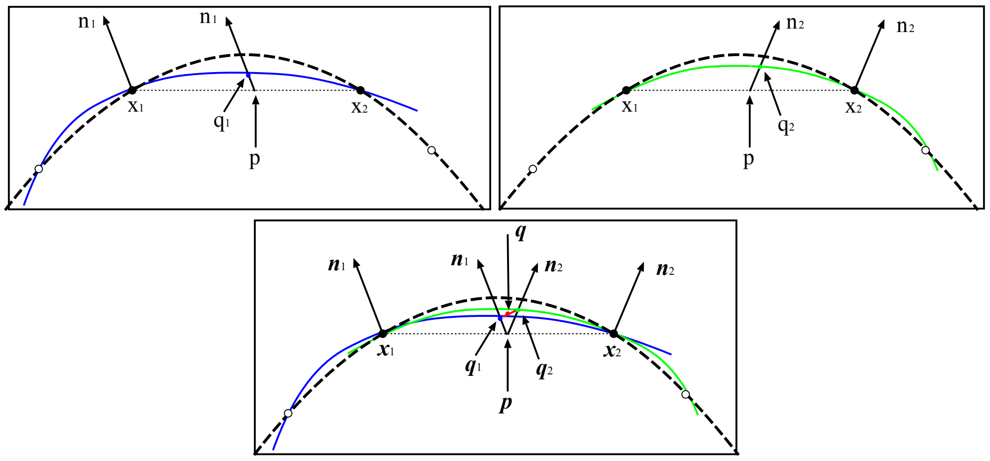


Figure 2.2: 2-D illustration of weighted averaging of local fitting. The black (dashed) curve indicates the exact curve. The blue (darker, solid) and green (lighter, solid) curves indicate the fittings at vertices $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. $\mathbf{q}$ is the WALF approximation of point $\mathbf{p}$ and is computed as a weighted average of the points $\mathbf{q}_1$ and $\mathbf{q}_2$ on the blue and green curves, respectively.

is necessary. Our approach to obtain a piecewise smooth geometry is to blend the local fittings to obtain *WALF (Weighted Average of Least-squares Fittings) Surface.* Continuity of the surface is achieved by using weights that are continuous over the mesh. One such choice is the linear shape functions (or barycentric coordinates) of the vertices over each triangle. Consider a triangle composed of vertices $\mathbf{x}_i$, $i = 1,2,3$, and any point $\mathbf{p}$ in the triangle. For each vertex $\mathbf{x}_i$, we obtain a point $\mathbf{q}_i$ for $\mathbf{p}$ from the local fitting in the local *uvw* coordinate frame at $\mathbf{x}_i$, by projecting $\mathbf{p}$ onto the *uv*-plane. Let N_i, $i = 1,2,3$ denote the linear shape functions (or barycentric coordinates) of $\mathbf{p}$ within the triangle, with $N_i \in [0,1]$ and $\sum_{i=1}^{3} N_i = 1$. We define

$$\mathbf{q}(\mathbf{u}) = \sum_{i=1}^{3} N_i \mathbf{q}_i(\mathbf{u}) \tag{2.2.1}$$

as the approximation to point $\mathbf{p}$. Figure 2.2 shows a 2-D illustration of this approach, where N_i are the barycentric coordinates of point $\mathbf{p}$ within the edge $\mathbf{x}_1\mathbf{x}_2$.

WALF constructs a C^0 continuous surface, as can be shown using the properties of finite-element basis functions. 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, $\mathbf{q}$ can be considered as a weighted average of the polynomials at all the vertices,

$$\mathbf{q}(\mathbf{u}) = \sum_{i=1}^{n} \phi_i(\mathbf{u})\mathbf{q}_i(\mathbf{u}), \tag{2.2.2}$$

and then it is obvious that $\mathbf{q}$ is C^∞ within each triangle and C^0 over the whole mesh. 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 leads to additional error terms.

Proposition 2. *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 $O(h^{d+1} + h^6)$.*

The proof of above proposition can be found at [28]. It gives an upper bound of the error, and 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.

2.2.2 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 $\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ and any point $\mathbf{p}$ in the triangle. Let $\hat{\mathbf{n}}_i$ denote the unit vertex normal at the ith vertex. We compute a normal at $\mathbf{p}$ as

$$\hat{\mathbf{n}} = \sum_{i=1}^{3} N_i \hat{\mathbf{n}}_i \bigg/ \left\| \sum_{i=1}^{3} N_i \hat{\mathbf{n}}_i \right\| . \tag{2.2.3}$$

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

$$\|\mathbf{WVX} - \mathbf{WF}\|_2, \tag{2.2.4}$$

where $\mathbf{V}$ again is the generalized Vandermonde matrix, $\mathbf{W}$ is the weight matrix and $\mathbf{F}$ is either the local height function or coordinate functions.

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

We define the weight as follows: For $\mathbf{p}$ within the triangle $\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$, we first define a weight for each vertex (for example the jth vertex) in the mesh that is in the stencil of vertex $\mathbf{x}_i$ as

$$w_{ij} = N_i \left(\hat{\mathbf{n}}_i^T \hat{\mathbf{n}}_j\right)^+ / \left(\sqrt{\|\mathbf{x}_j - \mathbf{p}\|^2 + \varepsilon}\right)^{d/2}, \tag{2.2.5}$$

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

$$\left(\hat{\mathbf{n}}_i^T \hat{\mathbf{n}}_j\right)^+ = \begin{cases} \hat{\mathbf{n}}_i^T \hat{\mathbf{n}}_j & \text{if } \hat{\mathbf{n}}_i^T \hat{\mathbf{n}}_j \geq \eta \\ 0 & \text{otherwise} \end{cases} \tag{2.2.6}$$

for some small $\eta \geq 0$. Then for the weighting matrix $\mathbf{W}$, weight for vertex j is then $\sum_{i=1}^{3} w_{ij}$. Note that the introduction of ε in (2.2.5) is to prevent division by very small numbers when $\mathbf{p}$ is very close to a vertex. In practice, we set ε to be 0.01 times the average distance from $\mathbf{p}$ to the points in the stencil.

Similar to WALF, CMF constructs a C^0 continuous surface, because $\mathbf{W}$, $\mathbf{V}$, and $\mathbf{F}$ 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 [31], and the following proposition reports the accuracy of CMF.

Proposition 3. *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 $O(h^{d+1})$.*

2.2.3 Comparison of WALF and CMF

WALF and CMF both can be used for high-order surface reconstruction, and they deliver similar accuracies. However, they may be preferable under different situations for efficiency and/or stability considerations. In terms of efficiency, WALF and CMF are comparable when approximating a single point because they both construct and solve a weighted least squares 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 squares 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. Therefore, WALF is more efficient for a global, uniform refinement of a mesh, as the local fittings at the vertices can be reused repeatedly, whereas CMF is more efficient for local adaptations.

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. In terms of stability, CMF tends to perform better near the boundary or sharp singularities, whereas WALF may not have sufficient number of points in the stencil. However, both WALF and CMF may introduce large variations if the mesh is under-resolved at regions with high curvatures, especially when cubic or higher-order polynomials are used. In such cases, it is necessary to introduce additional safeguards to downgrade the degree of polynomial.

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 [50] and adopted by Frey for surface meshing [18]. 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.

Chapter 3

Mesh Data Structures

We next present the mesh data structures on which all algorithms developed in this dissertation depend. Mesh data structures are the foundation of meshing algorithms (such as mesh generation and modification) and mesh-based numerical methods (such as finite element and finite volume methods). The underlying data structure strongly influences the overall performance of the algorithms or simulations since it is used to perform all the mesh-based combinatorial operations. As a result, they have been widely investigated since the inception of mesh generation and computational geometry [1, 2, 20, 34, 35, 36, 45, 48].

Mesh-data structures are highly application specific resulting in tailor made data structures satisfying varying levels of mesh representation, adjacencies and other requirements. The increasing complexities and the ever-changing demands of the application codes often pose new requirements on mesh data structures. Examples of such new demanding applications include coupled multiphysics simulations and multi-component systems, which may pose diverse requirements within each code as well as requirements on interoperability across different codes. In light of these new challenges of meshing applications, we summarize a few requirements that should be satisfied by a mesh data structure:

1. *Generality*: Support mixed-dimensional, non-manifold (oriented or non-oriented) meshes in 1-D, 2-D, and 3-D, and be easily generalizable to higher dimensions.

2. *Efficiency*: Support all local adjacency queries in constant time, assuming the valence of each mesh entity is bounded by a small constant.

3. *Simplicity*: Be simple and intuitive, and be easy to implement.

4. *Extensibility*: Allow extensibility for performance and parallelization.

5. *Interoperability*: Facilitate interoperability with application codes, such as simulation codes, multigrid solvers, etc.

6. *Compactness of memory footprint*: Require minimal storage in addition to element connectivity.

In this chapter, we develop a mesh data structure that meets all the above requirements. We refer to our data structure as the *Array-based Half-Facet* data structure, or *AHF*. The AHF unifies the array-based half-edge and half-face data structures for surface and volume meshes [1], and further generalizes them to support mixed-dimensional and non-manifold meshes by introducing the concept of *sibling half-facets*.

Our data structure is easy to implement and is efficient in both memory and computational cost. In addition, some of its fields can be created dynamically for fine-grain operations and be removed afterwards. It can be used to perform efficient mesh queries and modification. As an array-based data structure, AHF facilitates better interoperability across different application codes, different programming languages (such as MATLAB, C, C++, FORTRAN, etc.), and different hardware platforms. We implement AHF on top of MOAB[48] which further improves its interoperability through the iMesh interface (`http://www.itaps.org/`). In addition, our data structure is unique in its comprehensive implementation in MATLAB, which allows rapid prototyping and deployment of meshing algorithms and other mesh-based numerical methods in a productive fashion. The AHF is used as the underlying mesh data structure for all algorithms described in this dissertation.

3.1 Background

We develop data structures for representing discrete geometric and topological objects in 1-D, 2-D, or 3-D, arising from numerical computations in engineering and scientific applications. These objects correspond to *curves*, *surfaces*, and *volumes*, respectively, typically embedded in two- or three-dimensional Euclidean spaces. Topologically, a d-dimensional object is a *manifold with boundary* if every point in it has a neighborhood homeomorphic to either a d-dimensional ball or half-ball, where the points whose neighborhood is homeomorphic to a half-ball are *boundary* (or *border*) *points*. In practice, it is quite common to have topological objects that are *non-manifold*, especially for curves and surfaces. These non-manifolds are typically composed of a union of a finite number of manifolds with boundaries, and sometimes embedded in a higher-dimensional manifold structure. In 3-D space, a surface is *oriented* if it is possible to make a consistent choice of surface normal vector at every point; otherwise it is *non-oriented*.

In our setting, a *mesh* is a simplicial complex representing discretely a geometric or topological object. We say a mesh is 1-D, 2-D, or 3-D if the object that it represents is topologically 1-D, 2-D, or 3-D, respectively. We say a mesh is a *manifold* or *non-manifold* if its geometric realization is a manifold or non-manifold, respectively. A *mesh* is composed of 0-D, 1-D, 2-D, and 3-D entities, which we refer to as *vertices*, *edges*,*faces*, and *cells*, respectively. Typically, a face is either a triangle or quadrilateral, and a cell is a tetrahedron, prism, pyramid, or hexahedron, especially for finite element methods, although general polygons and polyhedra are also often used in finite volume meshes. For now our focus is on finite-element meshes.

In a d-dimensional mesh, we refer to the d-dimensional entities as *elements*, and refer to the $(d-1)$-dimensional sub-entities as its *facets*. More specifically, the facets of a cell are its faces, the facets of a face are its edges, and a facet of an edge are its vertices. Each facet has an orientation with respect to the containing element. For example, each edge of a triangle has a direction, and all the edges form an oriented loop. Thus it makes sense to call the facets as *half-facets.* Each facet may have multiple incident elements, especially for non-manifold entities. We refer to all such half-facets as *sibling half-facets*. A half-facet without any sibling is a *border half-facet*, and we refer to any vertex incident on a border half-facet as a *border vertex*. A mesh is said to be *conformal* if the pairwise intersection of any two entities is either another entity or is empty. We consider only conformal meshes, which may be manifold or non-manifold. In the case of surface meshes, the mesh may be oriented or non-oriented.

In some engineering applications, especially in coupled or multi-component systems, the domain of interest may be composed of a union of topologically 1-D, 2-D, and 3-D objects, such as a mixture of cables, thin-shells, and solids. We refer to such a domain and its mesh as *mixed-dimensional*. These meshes are sometimes referred to as *mixed* or *hybrid meshes*, which we avoid here since they may also refer to meshes with mixed-types of elements of the same dimension. We refer to a subset of the mesh corresponding to a 1-D, 2-D, and 3-D object in the domain as a *sub-mesh*. It is common, although not required, for the sub-meshes to share some mesh entities, especially shared vertices. Our goal is to design data structures for consistent representations for mixed-dimensional meshes with shared entities.

3.2 Related Work

Mesh data structures have been studied for a long time due to their importance in mesh-based applications. There are a number of mesh representations such as entity-based, boundary representations, corner table, radial-edge, winged, half-edge/face, incidence graphs, etc. These representations support a range of properties such as topological relations (manifold, non-manifold), different entities types (triangles, tetrahedrons, polygonal meshes), depending on the what needs to be supported, information that is stored and what can be queried using it by the application. A complete review is beyond the scope of this work. We mention a few data structures that are relevant in our context.

The *half-edge data structure*, a.k.a. the *doubly-connected edge list* (*DCEL*), is a popular data structure for 2-D and surface meshes (see e.g. [14]), especially oriented, manifold, polygonal surface meshes with or without boundary. The DCEL uses edges as the core object. The edge within each face is called a *directed edge* or *half-edge*. In an oriented manifold surface mesh, suppose the edges within each face can be ordered in counter-clockwise direction with respect to outward normal (or upward normal for 2-D meshes). Each edge has two incident faces, and the two half-edges have opposite orientations and hence are said to be *opposite* or *twin* of each other. An edge on the boundary does not have a twin half-edge. There are various implementations of DCEL. A typical implementation, such as that in CGAL [34, 17],OpenMesh [5] and Surface_Mesh [47], stores the mappings from each half-edge to its opposite half-edge, its previous and next half-edge within its face, its vertices, its incident face, as well as the mapping from each vertex and each face to an incident half-edge. More compact representations, such as [1], can be obtained by storing only the mapping between opposite half-edge, optionally the mapping from each vertex to an incident half-edge, along with the element connectivity. The above implementations do not support non-manifold or non-oriented meshes.

A generalization of the concept of DCEL to volume meshes is the so-called the *half-face data structure* [1, 36]. Within each cell, suppose the edges of each face are oriented in counter-clockwise order with respect to the outward normal of the cell. We refer to the oriented faces as *half-faces*. For typical meshes in engineering applications, each face in the interior of a volume mesh has two corresponding half-faces with opposite orientations, which are said to be *opposite* or *twin* of each other. The data structure in [1] was designed for 3-manifold with boundary composed of the standard elements. Although this is the typical case in applications, a volume mesh may also be non-manifold. For example, this can happen if the domain contains two objects that intersect at a single vertex or along an edge. For generality, we consider volume meshes that may be manifold or non-manifold, and allow them to be oriented or non-oriented. OpenVolumeMesh [36] was developed for general polytopal meshes, which may be non-manifold. However, the data structure in OpenVolumeMesh is quite complicated, because unlike the edges in a cell, the faces within a cell do not have a natural topological order.

In [6], a compact representation of simplicial meshes in two and three dimensions was proposed. The representations are based on storing the link for a set of $(d-2)-$simplices. In 2D, this representation is similar to the half-edge data structure. The main difference is that there is no storage of direct cross-connections between half-edges representing the same edge that are present in a purely half-edge representation. In 3D, it becomes an edge-based representation using a cycle of vertex labels for compact representation. This representation only supports simplicial, pseudomanifold and orientable meshes. In [8], a generalized indexed data structure with adjacency (IA^*data structure) is proposed for non-manifold, non-regular simplicial meshes in arbitrary dimensions. This data structure is a generalization of a similar adjacency-based IA data structure [40], which is applicable for manifold meshes. The non-regularity feature supports representation of dangling edges and triangles in the mesh. This data structure encodes all the top simplices along with a set of boundary, a partial set of co-boundary (incident) and adjacency (same-dimension neighbor) relations. Though it achieves compactness in memory storage by storing only a partial set of adjacency relations, it cannot support mixed-dimensional meshes due to its inherent design.

Though mesh data structures are very application specific, there are certain requirements that can be assumed to be common to all applications. A few such requirements are the ability to represent a mesh, query the topology and geometry of the mesh, modify it as well as assign and operate on mesh based data. Over the past few years, people have been working on abstracting these ideas to develop mesh frameworks that can be accessed through a well defined interface such that the application codes are more maintable and extensible. There are a number of mesh frameworks such as FMDB [45], MOAB [48], MSTK [20], GRUMMP [24], etc. These frameworks differ in terms of underlying mesh representations as well as specific implementations such as array-based or pointer-based representations.

Pointer-based Versus Array-based Implementations

A mesh data structure may be implemented using either pointers or arrays. The pointer-based implementations are more common, since they are relatively easy to manipulate. For example, the DCEL implementation in CGAL is pointer-based. Other examples, which are not based on half-edges or half-faces, include FMDB [45], MSTK [20], libMesh [35], etc. In such an implementation, the entities are represented as “objects” explicitly, and pointers (or handles) are used to refer to these explicit objects.

In contrast, in an array-based implementation, we do not represent the entities in the mesh as objects. Instead, an attribute of all the entities of the same type is stored in one or a few arrays, and the attributes for a single entity may be distributed in different arrays. An entity may be referenced through an ID or “handle”, which can be mapped easily to array indices. The half-edge and half-face data structures in

[1], MOAB [48], and OpenVolumeMesh [36] are array-based.

We choose to use array-based, pointer-free implementations for a number of reasons. First, in an array-based implementation, we can treat intermediate dimensional entities (such as half-facets) as implicit entities, and reference them without forming explicit objects. This can lead to significant savings in storage, especially on computers with 64-bit pointers. Second, using arrays can also lead to faster memory access and hence better efficiency. In addition, array-based implementations also offer better interoperability across application codes, different programming languages, and different hardware platforms (such as between GPUs and CPUs).

3.3 AHF: An Array-Based Half-Facet Datastructure

The basic half-edge and half-face data structures described in the previous section are simple and are restricted to oriented, manifold meshes (with or without boundary) in 2-D and 3-D, respectively. However they can be unified and generalized in order to support mixed-dimensional meshes, which may be non-manifold and/or non-oriented. In this section, we develop the concept of half-facet to unify different dimensions and extend the idea of an opposite half-facet to sibling half-facets to treat non-manifold situations.

3.3.1 Unification of Half-Edges and Half-Faces into Half-Facets

We unify the concept of half-edge and half-face data structures in order to make it dimension independent. The key abstractions for a d-dimensional mesh in this unified data structure are:

1. *vertices*: 0-dimensional entities (a.k.a. nodes);
2. *elements*: d-dimensional entities (faces and cells in 2-D and 3-D, respectively);
3. *half-facets*: $(d-1)$-dimensional sub-entities of a d-dimensional entity (half-edges and half-faces of a 2-D and 3-D element).

We do not require explicit representation of intermediate dimensional entities between 1 and $d-1$. This is made possible by the fact that every element has a standard numbering convention for its vertices and its facets. As a result, we can treat the half-facet as an *implicit entity*, and refer to a half-facet using the element ID and its local ID within the element. We refer to this data model as the *half-facet*

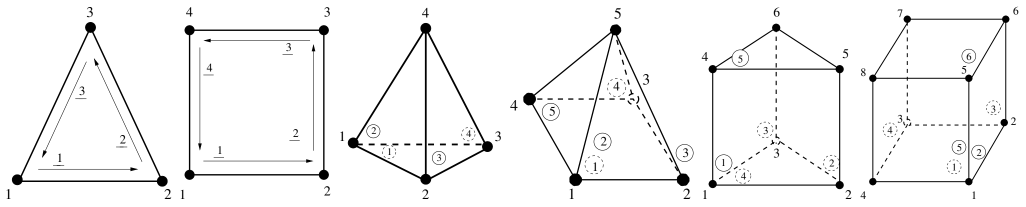
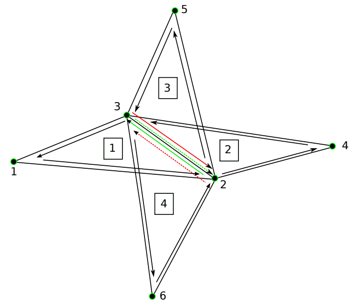


Figure 3.1: CGNS numbering conventions for 2-D and 3-D elements. Underscored numbers correspond to local edge IDs, and circled ones correspond to local face IDs.

data structure. For standard elements, we follow the convention of the CGNS (CFD General Notation System) [41, 49], as illustrated in Fig. 3.1.

The above unified model is not limited to $d = 2$ and 3. In fact, it can be applied to any d, as long as the numbering convention is predefined for the vertices and the facets. In particular, for $d = 1$, we refer to this representation for curves as the *half-vertex data structure*, which is a specialization of the half-facet data structure to curves. For $d \geq 2$, it provides a compact representation, since the intermediate dimensional entities are not stored but referenced implicitly instead. In addition, this idea can also be used for meshes with high-order elements (such as six-node triangles or 10-node tetrahedra), where the mid-edge, mid-face or mid-cell nodes do not affect the definition and identification of the half-facets.

We store the data structure similar to that in [1]. The element connectivity is stored in arrays in a manner similar to a typical finite element code. The mappings between sibling half-facets are stored in a 2-D array or in separate arrays, one per unique element type. Let $|E|$ and f denote the number of elements and the maximum number of facets in an element, respectively. We denote each half-facet by a two tuple $\langle eid, lfid \rangle$, where eid denotes the element ID, which starts from 1, and $lfid$ denotes the local facet ID, which starts from 0. For typical meshes, the two tuple can be encoded into a single 32-bit unsigned integer, by using first d bits to storing the local facet ID of a d-dimensional element, and using the remaining bits to store the element ID. This allows up to about 500 million elements for volume meshes. For very large meshes, we assign a 32-bit unsigned integer to the element ID and an unsigned 8-bit integer to the local ID, and store them in separate arrays. This allows up to 4 billion elements with minimal extra storage overhead. Depending on whether the half-facet IDs are encoded in a single integer or in two integers, we can store the mappings in either a single array or two arrays, respectively, where each array is of size $|E| \times f$. In addition, the vertex to half-facet mapping is stored in a single 1-D array. For most meshes, we need to store only one incident half-facet. Some complications may arise for non-manifold vertices, which we address next.



element connectivity

element	vertices		
1	1	2	3
2	2	4	3
3	2	5	3
4	2	3	6

sibling half-edges

element	sibhes		
1	nil	$\langle 2,2\rangle$	nil
2	nil	nil	$\langle 3,2\rangle$
3	nil	nil	$\langle 4,0\rangle$
4	$\langle 1,1\rangle$	nil	nil

vertex to half-edges

vertices	v2he
1	$\langle 1,0\rangle$
2	$\langle 3,0\rangle$
3	$\langle 4,1\rangle$
4	$\langle 2,1\rangle$
5	$\langle 3,1\rangle$
6	$\langle 4,2\rangle$

Figure 3.2: Example of a non-manifold mesh, along with its element connectivity, sibling half-edges, and mapping from each vertex to an incident half-edge.

3.3.2 Generalization to Non-manifold Meshes

In a non-manifold mesh, there can be more than two elements abutting the same facet, unlike a manifold mesh where there can be only up to two. We still refer to an oriented facet within an element as a *half-facet*, but it no longer has a "twin" or "opposite" half-facet in general. We refer to the half-facets corresponding to the same facet as *sibling half-facets*. The orientations of two sibling half-facets are not required to be opposite to each other, and therefore this generalization also allows representing non-oriented meshes, such as the Möbius strip.

An important issue is the storage for mapping between the sibling half-facets. Instead of *doubly-connected linked list* for twin half-facets, we use a cyclic linked list, which allows us to preserve the storage structure and also to traverse all the elements incident on a half-facet. Figure 3.2 shows an example of a non-manifold edge (edge joining vertex 2 and 3) present in the given triangulated mesh, as well as the element connectivity, sibling half-edge map and vertex to half-edge maps. Note that we do not necessarily need to sort the half-facets in any particular order. In fact, an ordering may not be well-defined in some cases. However, the data structure does not exclude the user from ordering the half-facets.

In a d-dimensional non-manifold mesh, when $d > 1$, there may exist a vertex whose neighborhood is not a d-dimensional ball or half-ball, but instead the union of two or more d-dimensional ball or half-balls that intersect at an entity of lower than

$(d-1)$ dimension (such as at a vertex in a surface mesh, or at a vertex or edge in a volume mesh). We refer to such a vertex as a *non-manifold vertex*. Some minor modification to the data structure is necessary in this setting. In particular, at a non-manifold vertex, we need to store a 1-to-n mapping instead of a 1-to-1 mapping to its incident half-facets. Note that this also covers the case where there exists an edge in a volume mesh whose neighborhood is the union of two or more d-dimensional ball or half-balls that intersect only at the vertex.

3.3.3 Generalization to Mixed-Dimensional Meshes

The concept of *sibling half-facets* unifies the half-vertex, half-edge, and half-face data structures for 1-D, 2-D, and 3-D meshes, which may be manifold or non-manifold with boundary. In a mixed-dimensional mesh, the sub-meshes of different dimensions can share entities. In particular, it is most common for the meshes to share vertices. However, these entities may have different representations.

This unification allows an easy extension to support mixed-dimensional meshes, which may be composed of sub-meshes of 1-D, 2-D, and 3-D. Figure 3.3 shows a diagram of a typical half-facet data structure, where the half-vertices and half-edges are only required for explicit edges and faces in the mesh, respectively. We refer to this data structure for mixed-dimensional meshes as *Array-based Half-Facet data structure*, or *AHF*. This data structure is very simple and modular, as the individual sub-meshes of different dimensions are self-contained, and they can be maintained separately. They also allow us to traverse between multiple dimensions efficiently. The interactions of the different dimensions are all performed through the shared vertices.

3.4 Construction and Query of AHF

We next describe some detailed algorithms for the construction of AHF, as well as some queries.

3.4.1 Construction of AHF

In the half-facet data structure, there are two components: *sibhfs* (sibling half-facets) and *v2hf* (vertex to half-facet). The former is central to AHF, as nearly all adjacency queries require it. These sibling half-facets should map to each other and form a cycle. The latter array, v2hf, is optional for many operations, it can be created only when needed. In general, we construct the AHF in two steps, which construct these two mappings, respectively. In a mixed-dimensional mesh, the AHF

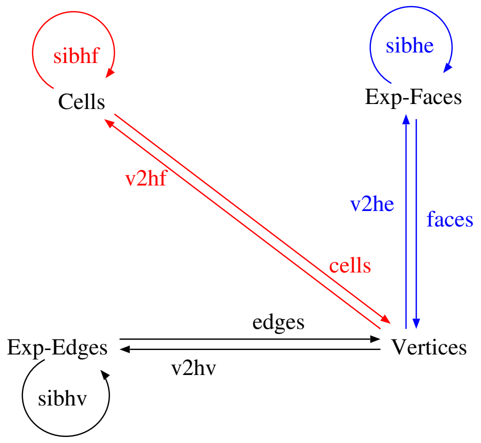


Figure 3.3: Typical AHF for mixed-dimensional meshes is composed half-vertex (black, for explicit edges only), half-edge (blue, for explicit faces only), and half-face (red) data structures.

for the submesh of each dimension is constructed independent of each other. In the following, we describe the two steps in a manner independently of the dimension of the mesh.

Identification of Sibling Half-Facets

During the first step, we determine the sibling half-facets and construct a cyclic mapping between them. The key components of this step are two intermediate mappings:

***v2hfs*:** a mapping from each vertex to its incident half-facets in which the vertex has largest ID;

***v2adj*:** a mapping from each vertex to its adjacent vertices in each of the above incident half-facets.

Algorithm 1 outlines the procedure for the first stage, which is applicable to half-facets in arbitrary dimensions, and it is particularly efficient in 1 to 3 dimensions.

The computational cost of Algorithm 1 is linear, assuming that the number of facets incident on a vertex is bounded by a small constant, say c. To analyze the storage requirement, let $|H|$ denote the number of half-facets in the mesh. The output requires approximately $|H|$ integers. The algorithm requires two intermediate maps v2hf and v2adj, which require $c|H|$ integers total. This intermediate storage requirement can be further reduced by equally distributing the vertices into b buckets

Algorithm 1 Determination of sibling half-facets.

```
Input: elems: element connectivity
Output: sibhfs: cyclic mappings of sibling half-facets
 1: for each elements $e$ in elems do
 2:     for each facet $f$ in $e$ do
 3:         $v \leftarrow$ vertex with largest ID within $f$;
 4:         $us \leftarrow$ set of adjacent vertices of $v$ in $f$;
 5:         Append $f$ into v2hfs($v$), and append $us$ into v2adj($v$,$f$);
 6:     end for
 7: end for
 8: for each elements $e$ in elems do
 9:     for each facet $f$ in $e$ do
10:         if sibhfs($f$) is not set then
11:             $v \leftarrow$ vertex with largest ID within $f$, and $us = v2adj$(v,f);
12:             Find half-facets in v2hfs($v$) s.t. v2adj($v$,$\cdot$)=$us$;
13:             Form a cyclic mapping for these half-facets in sibhfs;
14:         end if
15:     end for
16: end for
```

and process each bucket in a separate pass, which then would require only $|H|/b$ integers.

Construction of Incident Half-Facet of Vertex

During the second step, we construct a mapping from each vertex to an incident half-facet. This is done by utilizing the sibling half-facets obtained from the first step, as outlined in Algorithm 2.

When determining the incident half-facets, we give higher-priorities to border half-facets, so that from its output v2hf, we can determine whether a vertex v is a border vertex by simply checking whether v2hf(v) is a border half-facet. In addition, a minor variant of Algorithm 2 can be used to construct a bitmap of vertices to determine all the vertices which are border vertices without forming v2hf. These functions can be useful, for example when extracting the boundary of a mesh or when imposing boundary conditions in numerical computations. The computational cost of Algorithm 2 is also linear in the number of vertices plus the number of half-facets. It does not require any intermediate storage. In addition, the AHF can be used to extract the internal boundaries between different materials in a mesh.

Algorithm 2 Construction of mapping from vertex to an incident half-facet.

```
Input:  elems: element connectivity
        sibhfs: cyclic mappings of sibling half-facets
Output: v2hf: vertex to an incident half-facet
 1: for each elements $e$ in elems do
 2:     for each vertex $v$ of $e$ do
 3:         if v2hf($v$)==0 then
 4:             v2hf($v$)$\leftarrow$ a facet incident on $v$ in $e$
 5:         end if
 6:     end for
        {Give border facets higher priorities}
 7:     for each facet $f$ in $e$ do
 8:         if sibhfs($e$,$f$)==0 then
 9:             for each vertex of $f$ do
10:                 Set v2hf($v$) to $\langle e, f \rangle$;
11:             end for
12:         end if
13:     end for
14: end for
```

3.4.2 Algorithms for Adjacency Queries

In this section, we discuss the algorithms for adjacency queries. We mainly focus on the following two classes of queries:

1. Upward incidence query: Given an explicit d-dimensional entity, obtain the (d+1)-dimensional entities incident on it.

2. Neighborhood/Same dimension query: Given an explicit d-dimensional entity, obtain neighbor d-dimensional entities that share a $(d-1)$-dimensional sub-entity with it.

We consider these operations for mixed-dimensional meshes with 1-D, 2-D, and 3-D explicit entities, which results in six operations total:

1a. for each vertex, obtain incident edges;

1b. for each edge, obtain incident faces;

1c. for each face, obtain incident cells;

2a. for each edge, obtain vertex-connected neighbor edges;

2b. for each face, obtain edge-connected neighbor faces;

2c. for each cell, obtain face-connected neighbor cells;

We describe the procedures in a dimension-independent fashion. The first class of operations are across the sub-meshes of different dimensions. The algorithm proceeds in two steps:

1. **Half-facet identification**: Given an explicit d-dimensional entity, find a corresponding half-facet h in the $(d+1)$-dimensional sub-mesh through the shared vertices;

2. **Obtain sibling half-facets**: Find the sibling half-facets of this half-facet h, and decode the half-facet IDs to obtain all the adjacent $(d+1)$-dimensional entities.

The first step involves a local one-ring search of a vertex of the facet whereas the second step is simply gathering all the sibling half-facets that are already stored in the map sibhfs. In terms of computational cost, the first step takes time proportional to the number of incident entities of a vertex, and the second step takes time proportional to the size of the output. Both steps in general require constant time.

In the second class of operations, given a d-dimensional entity, we simply need to loop through its (d-1)-dimensional facets, for each of its facets obtain the sibling half-facets. Then by decoding the ID of the sibling half-facets, we obtain the neighbor d-dimensional entities. The algorithm takes time proportional to the output size, which is a constant.

Figures 3.4 and 3.5 shows examples of these operations. Figure 3.4 shows all the incident triangles of a given non-manifold edge (blue). In figure 3.5, the left panel shows the set of neighbor edges (in red) of a given (blue) edge and the right panel shows the neighborhood faces (in red) of a given (blue) triangle.

3.5 Implementations

Implementation in MATLAB

Since our data structure is array-based and pointer-free, we can implement it conveniently in MATLAB. The MATLAB provides a user-friendly programming environment, so that our MATLAB implementation allows rapid prototyping and testing of sophisticated meshing algorithms and mesh-based numerical methods. In our MATLAB implementation, we support two ways to store the half-facet:

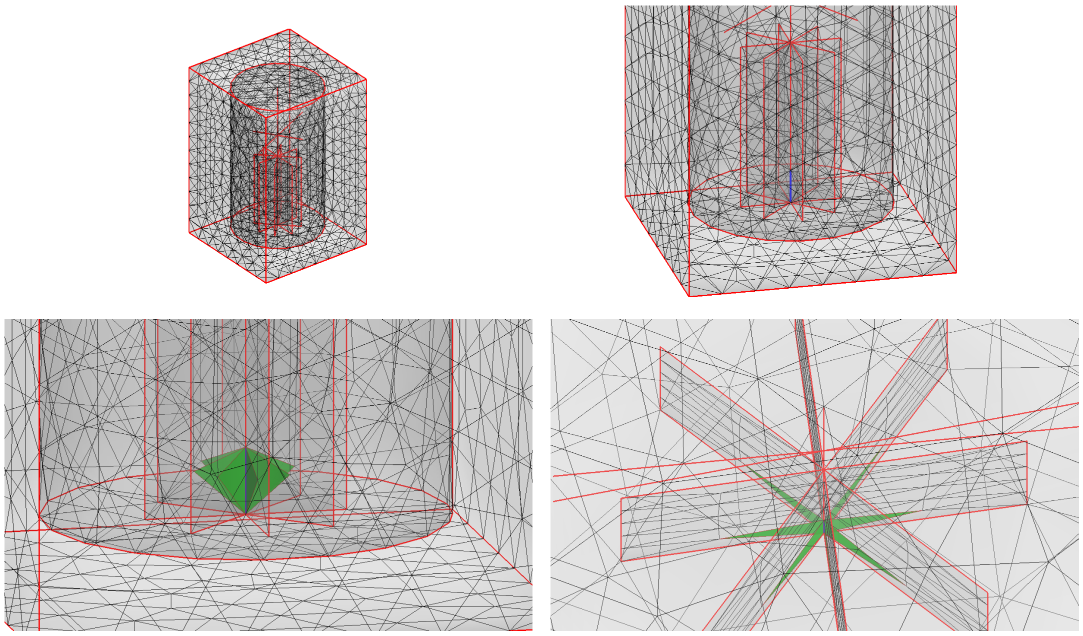


Figure 3.4: An example of upward incidence query. This shows all the incident triangles on a given explicit edge (blue).

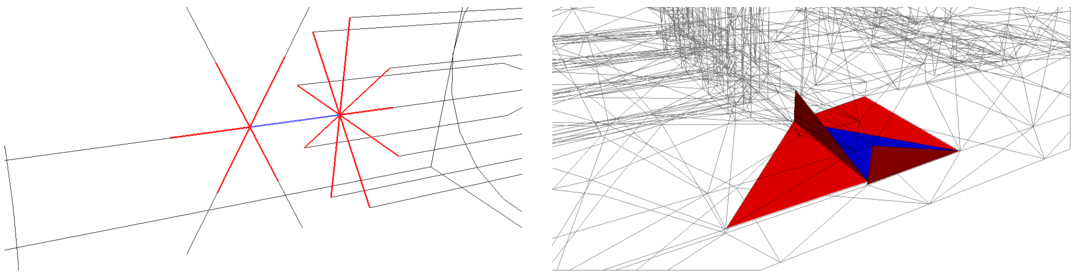


Figure 3.5: Example adjacency queries for non-manifold meshes. Left panel shows neighbor edges (red) of a given blue edge. Right panel shows the neighbor faces of a given blue triangle.

1. Encode into an integer: We encode the two-tuple ID in a single integer, where d bits are reserved the local facet ID for a d-dimensional mesh.

2. Use MATLAB struct: We store the element ID and local facet ID into a 32-bit integer array and an 8-bit integer array, respectively, and they pack these two arrays into a single MATLAB struct.

For a mixed dimensional mesh, the complete mesh is packed into a single struct composed of the arrays of different dimensions. Both of our implementations are compatible with MATLAB Coder, which allows generation of efficient and portable ANSI C code from our MATLAB implementation. The generated C code can be used as stand-alone libraries, or be compiled into MEX functions and be called in MATLAB or GNU Octave.

Integration of AHF into MOAB

The Mesh Oriented datABase (MOAB [48]) is a mesh framework designed to support a range of mesh related operations, such as memory efficient mesh representation, mesh querying and representation of application specific data. The internal storage of MOAB is array-based and supports querying of adjacent entities by target dimension. MOAB is designed to take advantage of temporal and spatial locality offered by large collections of entities.

The non-vertex entity-to-entity adjacencies are created and stored only upon application request. Since most of the applications do require some kind of auxiliary entity adjacency information, MOAB may require significant storage in such situations . This imposes extra storage requirements on the application. The AHF comes handy precisely in such situations. It adds the flexibility of intermediate entity adjacency querying without explicitly storing them. In addition, MOAB does not support implicit entities, and does not store the neighboring information of entities of the same dimension. Therefore, it is desirable to incorporate AHF into MOAB. We refer to this implementation as *MOAB_AHF*, which can be created dynamically for some queries and be deallocated afterwards.

There are some important differences between a standalone AHF implementation and MOAB_AHF. In terms of storage, MOAB uses "tags" to store application specific data defined on mesh entities or entity sets. In MOAB_AHF, we store sibhfs and v2hf as tags, instead of standalone arrays. In addition, since MOAB references elements and other entities through handles, we store the sibhfs as two tags: one for the handles to the elements, and the other for local facet IDs. Another difference is that MOAB uses a different numbering convention of the local facet IDs than that of CGNS. To be self-consistent, we use the MOAB's own numbering convention in MOAB_AHF. Finally, in MOAB_AHF, we construct the sibhfs without forming

Table 3.1: Times taken to construct data structures by standalone AHF and MOAB_AHF.

mesh	#verts	#edges	#tris	#tets	AHF	MOAB_AHF
1	345	121	378	1357	0.002231	0.00969
2	447	137	678	1503	0.002376	0.01433
3	1443	225	1824	6794	0.008991	0.04659
4	1724	2081	3688	8177	0.01218	0.05989
5	2151	282	2556	9746	0.0126219	0.06112
6	119960	27215	42476	711014	0.935569	4.31953

the v2hfs and v2adj in Algorithm 1, and instead using MOAB's built-in function "get_adjacency" to identify the elements adjacent to an entity.

3.6 Experimental Comparisons

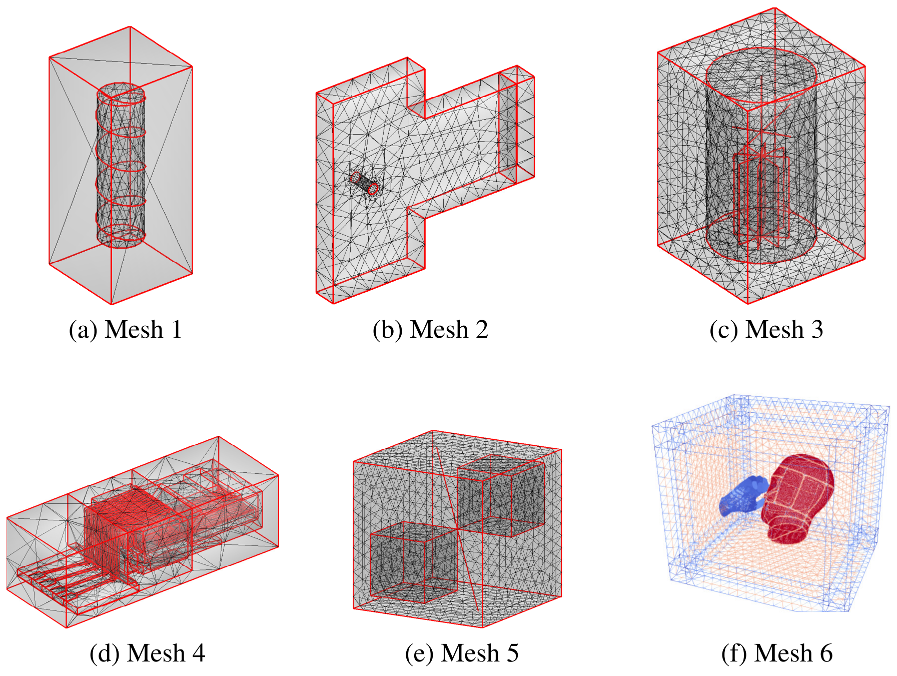
We now present some experimental studies of AHF, MOAB and MOAB_AHF in terms of storage requirements and computational cost.

3.6.1 Cost in Construction of Data Structure

We first compare the computational times in constructing the data structures. For AHF, we used the C code generated from our MATLAB implementation with MATLAB Coder 2.4 released with MATLAB R2013a. We compiled AHF, MOAB, and MOAB_AHF using gcc 4.4.3 with optimization enabled. All the tests were performed on a Linux computer with a 3.16GHz Intel Core 2 Duo processor and 4GB of RAM.

We use a set of six meshes (shown in figure 3.6), which are all mixed-dimensional tetrahedral meshes, containing explicit triangles and edges, courtesy of CST Computer Simulation Technology AG.

Table 3.1 shows the sizes of these meshes as well as the run times taken by the standalone AHF and MOAB_AHF for constructing the mesh data structure. The overall cost is approximately linear in the number of vertices for both AHF and MOAB_AHF. However, our current implementation of MOAB_AHF takes about 6–7 times longer than AHF, because Algorithm 1 builds the intermediate arrays for v2hes and v2adjs, which are more efficient than using MOAB's built-in function "get_adjacency" to identify sibling half-facets. The cost of constructing MOAB_AHF can be optimized by using Algorithm 1.


(a) Mesh 1 (b) Mesh 2 (c) Mesh 3

(d) Mesh 4 (e) Mesh 5 (f) Mesh 6

Figure 3.6: The meshes used for comparison.

3.6.2 Storage Costs

We compare the storage requirements of AHF and MOAB. Let C, F_{exp}, E_{exp} and V represent the set of cells, explicit faces, explicit edges and vertices of the given mesh, and let $|\cdot|$ denote the number of items in a set. AHF stores three maps, which require the following number of entities:

element connectivity: $n_c = 2\left|E_{exp}\right| + v_f\left|F_{exp}\right| + v_c\left|C\right|$

sibling half-facet map: $n_s = 2\left|E_{exp}\right| + s_f\left|F_{exp}\right| + f_c\left|C\right|$

vertex to half-facet map: $n_v = 3\left|V\right|$

where v_f , v_c , s_f and f_c are the numbers of vertices per face, vertices per cell, edges (sides) per face, and faces per cell, respectively.

Note that for the half-facet ID $\langle eid, lfid\rangle$, we can encode it in a 32-bit integer or store eid and lfid in a 32-bit and a 8-bit integer, respectively. The storage required by the former in bytes is

$$S_{\text{AHF1}} = 4(n_c+n_s+n_v) = 16\left|E_{exp}\right| + 4(v_f+s_f)\left|F_{exp}\right| + 4(v_c+f_c)|C| + 12|V|, \tag{3.6.1}$$

Table 3.2: Storage requirements in kilobytes of AHF, MOAB, and OpenVolumeMesh for three largest meshes in Table 3.1.

mesh	AHF		MOAB	MOAB_AHF	OpenVolumeMesh
	integer	struct			
4	435.094	486.955	1039.31	897.78	897.176
5	444.496	496.907	1041.69	904.60	1055.26
6	27857.3	31163.7	64514.35	56889.90	71074.8

whereas the latter requires

$$S_{\text{AHF2}} = 4n_c + 5n_s + 5n_v, \tag{3.6.2}$$

which is about 12% larger than S_{AHF1}. If the element connectivity is required, the extra storage required by AHF is only about 60% of these.

The storage requirement of MOAB is higher than AHF, as it stores both the connectivity and upward adjacencies. The upward adjacencies are created and stored the first time a query requiring the adjacency is performed. MOAB_AHF may reduce the storage requirement.

To put this into perspective, we also compare its storage against OpenVolumeMesh [36]. In OpenVolumeMesh, all the top-down and bottom-up incidence relations are stored explicitly using integer handles. Let E and F denote the set of all edges and faces (including the implicit edges and faces) of the given mesh. The number of required handles to encode top-down and bottom-up incidences are

$$n_{\text{OVM}} = f_c\,|C| + (v_f+2)\,|F| + (s_f+2)\,|E| + s_e\,|V|, \tag{3.6.3}$$

where v_f , v_c , s_c and f_c are the average numbers of vertices per face, vertices per cell, edges (sides) per cells, and faces per cell, respectively. Assuming each integer handle is 32-bit, then the storage will be about $S_{\text{OVM}} = 4n_{\text{OVM}}$.

Table 3.2 shows the storage requirements for the last three meshes in Table 3.1. As it can be seen from the table, AHF requires about half the amount storage required by OpenVolumeMesh and MOAB. MOAB_AHF has reduced the storage of MOAB slightly, but further reduction is still possible.

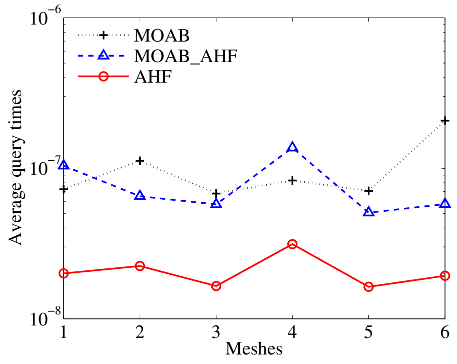
3.6.3 Computational Costs of Adjacency Queries

In this subsection, we report the times for the six queries described in Section 3.4.2. We report the performance results of AHF, MOAB, and MOAB_AHF. We omit OpenVolumeMesh, as it could not load the mixed-dimensional meshes. These queries are performed over explicit entities, and they return explicit entities only. Figure 3.7 shows the average time taken to perform the incident and neighborhood queries for 1-D, 2-D, and 3-D entities, respectively. The average time is measured by the total elapsed time of the algorithm for all the entities divided by the number of the entities. The results confirm that all the mesh query operations take approximately constant time, regardless of mesh sizes. Both AHF and MOAB_AHF improve the performance of MOAB, and AHF outperforms MOAB by an order of magnitude. The computation costs of MOAB_AHF are comparable to MOAB's own adjacency functionalities for upward incident queries, whereas same dimensional neighborhood queries are at least an order of magnitude better. This is reasonable since MOAB computes the adjacencies by performing boolean operations on vertex-entity adjacencies. This result indicates that the AHF data structure can significantly improve the MOAB data model for adjacency queries. Further code optimization of MOAB_AHF can lead to even better performance.

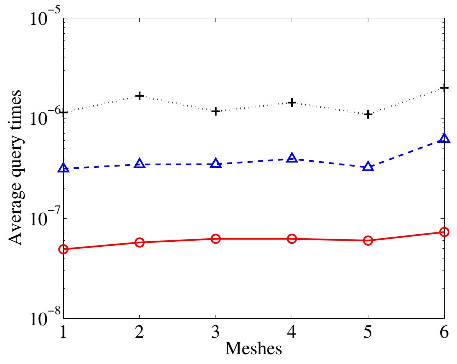
3.7 Mesh Modification Operations

Mesh-modification can be implemented relatively easily in AHF. For mixed dimensional meshes, the AHF is particularly attractive, because the adaptivity for different dimensions can be done nearly independently, and they only need to be synchronized at the shared vertices. This leads to very modular adaptivity strategies.

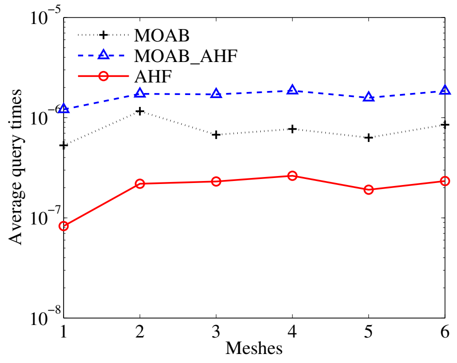
Within each dimension, the AHF can be modified either locally or globally. The local modification is performed through the mesh-modification primitives, such as edge flipping, edge splitting, and edge collapse, especially for triangular and tetrahedral meshes. As an example, Figure 3.8 illustrates the standard flipping operations for tetrahedra between a current n-complex of elements and a resulting m-complex, where the complex is the neighborhood of elements considered. We consider the standard operations, where the $n-m$ combinations are 3-2, 2-3, 4-4 and 2-2. For the 4-4 and 2-2 flipping operations, both the numbers of vertices and of elements remain the same, and therefore the operation involves only updating the elems (element connectivity), sibhfs (sibling half-faces), and v2hf (vertex-to-half-face) mappings locally within the complex. For 3-2 flipping, and similarly for edge collapse, the number of elements decreases. This may result in a hole in the arrays elems and sibhfs. We fill the hole by swapping the element with the highest ID into the hole and updating the half-facets in sibhfs and v2hf, so that the element IDs remain consecutive. For 2-3 flipping, and similarly for edge splitting, the number of elements increases. This may require reallocating and copying the arrays. To avoid excessive memory copying, we expand the array by a small percentage (e.g. by 20%) each reallocation, so that the amortized cost for the local modifications is



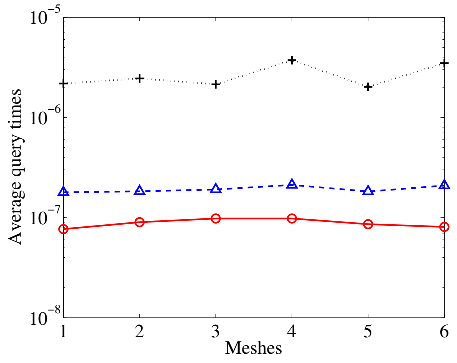
(a) Vertex to incident edges



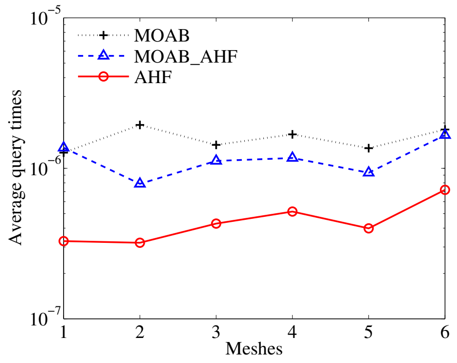
(b) Edge to neighbor edges



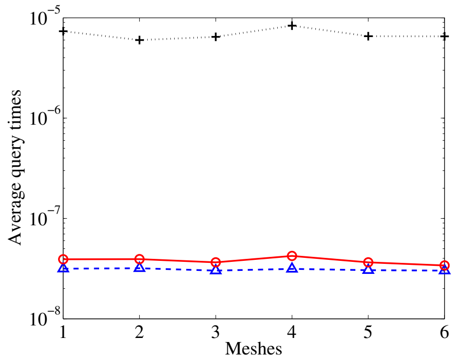
(c) Edge to incident faces



(d) Face to neighbor faces



(e) Face to incident cells



(f) Cell to neighbor cells

Figure 3.7: Average times (in seconds and logarithmic scale) to perform queries in 1-D (a, b), 2-D (c, d) and 3-D (e, f).

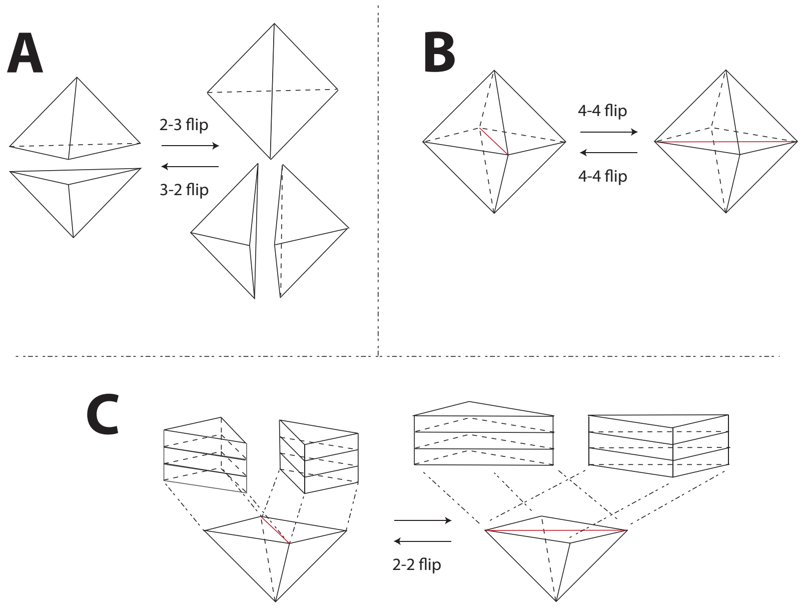


Figure 3.8: Standard edge-flipping operations on a tetrahedral mesh. The *2-3* and *3-2 flips* (**A**) increase or decrease the number of tetrahedra, respectively. In a manifold mesh, the *4-4 flip* (**B**) can be applied in the interior. The *2-2 flip* (**C**) is defined only on the boundary surface, or a tetrahedral mesh with prismatic boundary layers [15].

constant, assuming the number of flipping and splitting operations are proportional to the size of the mesh.

As an example, Figure 3.9 shows a tetrahedral mesh for a heart model, which was first generated by using TetGen [46], followed by applying flipping and smoothing implemented using AHF. The figure also shows the quality measures in terms of the orthogonality, skewness and uniformity [33], which are important measures for finite volume methods.

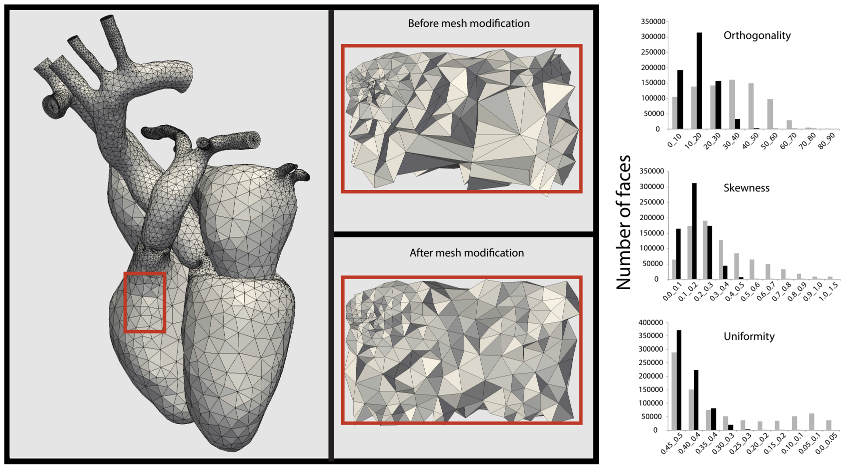


Figure 3.9: An example tetrahedral mesh of a heart model optimized using mesh flipping and smoothing implemented using AHF. The histograms show the quality measures in terms of orthogonality, skewness and uniformity of the mesh before (gray) and after (black) optimization, where smaller values correspond to better qualities.

Chapter 4

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, surface finite elements, integral transforms, finite volume method etc. For example, in the boundary integral method the solution is obtained by solving an integral equation, which in turn is solved by forming a linear system using collocation methods. The entries in the linear system are surface integrals. 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. Surface finite element method, which is a special case of the finite element method for solving partial differential equations on surfaces, also requires surface integrals. The application of surface integration is also found in fluid-structure interactions, where the integral of the pressure over the structure is the force applied by the fluid to the structure. Surface integrals also appear in surface, interface and colloidal sciences as well as in the semiconductor industry and for pharmaceutical manufacturing. They are helpful in understanding various processes such as adhesion and fracture processes as well as in the manipulation of nanoscale objects.

For these applications, the standard method for surface integration is to integrate over the individual triangles. Most of the methods have difficulty approximating the derivatives of the geometry that appear in the integral with sufficient accuracy. Because of this, numerical integration involves replacing and sometimes totally ignoring these derivatives with simpler approximations. This limits its accuracy to second-order as piecewise linear approximations to the geometry and the function are used during the approximation process. Subsequent reduction of the relative error is obtained by performing either global or adaptive mesh refinement until the error becomes less than a set tolerance. A natural but yet fundamental question is whether we can achieve high-order accuracy given a piecewise linear approximation to the surface. In this chapter, we address this question and prove that it is possible to compute the numerical integration of a function over a surface mesh to high-order of accuracy.

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 (or cubature) rules. The reason is that accurate numerical integration requires high-order approximations (or reconstructions) of the geometry as well as that of the function. Therefore, high-order numerical integration over surfaces is not straightforward.

In this chapter, 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: a stabilized least-squares fitting to obtain higher order approximation of the geometry and the integrand, a blending procedure based on linear finite-element shape functions, and high-degree numerical quadrature rules. The first two components are based on the computational framework to reconstruct a surface to high-order of accuracy by the WALF method (section 2.2.1) and its extension to reconstruct a function over a surface. Our resulting algorithm is relatively simple and efficient.

4.1 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 [56]. Unfortunately, in problems such as fluid 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 [3], 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 interpolated function and geometry. He approximates the function and the surface based on high order piecewise interpolants using Lagrange polynomials. The points chosen for high-order interpolation are 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.

In [22], another approach is presented to compute surface integrals, especially encountered in boundary element methods. It is based on the assumption that the input surface is given implicitly which allows defining a retraction from a neighborhood of a triangulated surface approximation to the exact surface to obtain a better approximation of the exact surface. However, the accuracy of geometry obtained is in a way lost by using a simplified trapezoidal rule resulting in a second-order accurate method. A subdivision of the triangulation by an adaptive refinement is used to minimize the error locally below a given tolerance. This approach is effective especially when the integrand contains a singularity.

4.2 Background

We first begin with a brief review of concepts for smooth surfaces in differential calculus and differential geometry, which will serve as the foundation of our method for discrete surfaces.

4.2.1 Integration of Continuous Functions over Smooth Surfaces

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

$$\xi \equiv \begin{bmatrix} \xi \\ \eta \end{bmatrix}, \quad \mathbf{x} \equiv \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \mathbf{J} \equiv \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{bmatrix}. \tag{4.2.1}$$

Let g denote $\sqrt{\det(\mathbf{J}^T\mathbf{J})}$, which is analogous to the absolute value of the Jacobian determinant if the surface were in the xy-plane. The surface integral of a scalar function $\varphi : \Gamma \to \mathbb{R}$ is

$$\int_{\Gamma} \varphi \, da \equiv \int_{U} \varphi(\xi) g(\xi) d\xi \, d\eta. \tag{4.2.2}$$

In the following discussion, we will omit the function arguments ξ for conciseness. The simplest application of (4.2.2) 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

$$\int_\Gamma \varphi \cdot d\mathbf{a} \equiv \int_\Gamma \varphi \cdot \hat{\mathbf{n}} da = \int_U \varphi \cdot \hat{\mathbf{n}} g d\xi \, d\eta, \tag{4.2.3}$$

where $\hat{\mathbf{n}}$ denotes the unit surface normal. Let $\mathbf{j}_k$ denote the kth column of $\mathbf{J}$ for $k = 1, 2$. Assume $\mathbf{j}_1$, $\mathbf{j}_2$, and $\hat{\mathbf{n}}$ form a right-hand system, and let $\mathbf{n} \equiv g\hat{\mathbf{n}} = \mathbf{j}_1 \times \mathbf{j}_2$, which we refer to as the *Jacobian-weighted normal*. The surface integral of φ can be written as

$$\int_\Gamma \varphi \cdot d\mathbf{a} = \int_U \varphi \cdot (\mathbf{j}_1 \times \mathbf{j}_2) d\xi \, d\eta. \tag{4.2.4}$$

The simplest application of (4.2.4) is the volume $V = \int_\Gamma \mathbf{x} \cdot d\mathbf{a}/3$ for a closed surface Γ, where $\varphi = \mathbf{x}/3$. Generally, a global parameterization 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

$$\int_\Gamma \varphi da = \sum_i \int_{\sigma_i} \varphi g d\xi \, d\eta \tag{4.2.5}$$

and

$$\int_\Gamma \varphi \cdot d\mathbf{a} = \sum_i \int_{\sigma_i} \varphi \cdot (\mathbf{j}_1 \times \mathbf{j}_2) d\xi \, d\eta, \tag{4.2.6}$$

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

From (4.2.2) and (4.2.4), it is clear that the 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 $\mathbf{x}_0$. Let $\hat{\mathbf{t}}_1$ and $\hat{\mathbf{t}}_2$ be unit vectors in the xyz coordinate system along the positive directions of the u- and v-axes, respectively, and $\hat{\mathbf{m}} = \hat{\mathbf{t}}_1 \times \hat{\mathbf{t}}_2$ be the unit vector along the positive w direction. Let $\mathbf{T} = \left[\begin{array}{c|c} \mathbf{t}_1 & \mathbf{t}_2 \end{array} \right]$ denote the matrix consisting of the unit vectors in the tangent plane, and $\mathbf{Q}$ the rotation matrix $\left[\begin{array}{c|c|c} \hat{\mathbf{t}}_1 & \hat{\mathbf{t}}_2 & \hat{\mathbf{m}} \end{array} \right]$.

Any point $\mathbf{x}$ on Γ is then transformed to a point $\begin{bmatrix} u \\ v \\ f(u,v) \end{bmatrix} \equiv \mathbf{Q}^T(\mathbf{x}-\mathbf{x}_0)$. 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 $\mathbf{x} \in \Gamma$ close to $\mathbf{x}_0$, f would be one-to-one in the neighborhood of $\mathbf{x}$.

Let $\mathbf{p}(u,v) = [u, v, f(u,v)]^T$ 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 $\mathbf{u} \equiv (u,v)$. The Jacobian of $\mathbf{p}$ with respect to $\mathbf{u}$ is then

$$\mathbf{J} = \left[\begin{array}{c|c} \mathbf{p}_u & \mathbf{p}_v \end{array}\right] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ f_u & f_v \end{bmatrix}. \tag{4.2.7}$$

The vectors $\mathbf{p}_u$ and $\mathbf{p}_v$ form a basis of the tangent space of the surface at $\mathbf{p}$, 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

$$\mathbf{n} = \mathbf{Q}\begin{bmatrix} -f_u \\ -f_v \\ 1 \end{bmatrix} = \hat{\mathbf{m}} - f_u\hat{\mathbf{t}}_1 - f_v\hat{\mathbf{t}}_2 \text{ and } \hat{\mathbf{n}} = \frac{\mathbf{n}}{g}. \tag{4.2.8}$$

These formulas are exact for smooth surfaces. In the next section, we will present their high-order approximations and their use in surface integration for triangulated surfaces.

4.2.2 Numerical Integration over surfaces

In practice, an analytic anti-derivate is rarely available for the integral. As a result, we need to numerically approximate the integral. Integration in the continuum is defined as a limiting process. This fact allows us to approximate integrands by simpler functional forms such as polynomials and subsequently integrate these approximate functions over the domain to obtain a numerical value of the integral. That is, numerical integration requires 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 polynomial, and is accurate for integrands that are continuously differentiable to dth derivatives. In mathematics, the generalization of quadrature rules to two or higher dimensions is called *cubature rules*, but it is also often referred to as *high-dimensional quadrature rules*. We use the term *quadrature rules* as the distinction between numerical integrations in one-dimension and higher-dimensions is immaterial algorithmically.

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

$$I\varphi = \int_\Gamma \varphi(\mathbf{x}) d\mathbf{x}. \tag{4.2.9}$$

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

$$\|I\widetilde{\varphi} - I\varphi\| \leq K_\varphi \|\widetilde{\varphi} - \varphi\|_\infty, \tag{4.2.10}$$

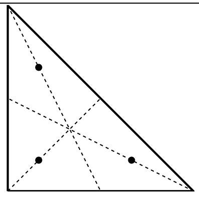
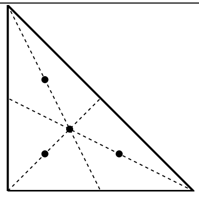
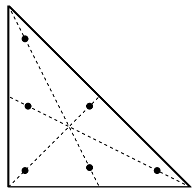
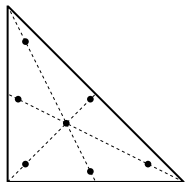
where $\widetilde{\varphi}$ denotes an approximation of φ, and $K_\varphi = \text{area}(\Gamma)$ is the condition number of $I\varphi$ with respect to changes in φ. Therefore, integrating an approximated function does not change the result drastically. The condition number of a quadrature rule is the sum of the absolute value of its weights. If all the weights are positive, then the condition number of the 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 [39], a survey of quadrature rules over triangles can be found. These quadrature rules are then applied to a tessellation of the domain. Since we assume the domain of integration is given by a triangulated surface, we utilize quadrature rules (4.1) for triangles.

4.3 High-Order Numerical Integration

The main limitation of standard methods is the use of low-order approximations to both the geometry and the integrand, especially computation of convergent and accurate derivatives of the geometry. This limitation can be overcome by using high-order approximations to both the geometry (i.e, its differential quantities) and the integrand along with high-order quadrature rules. High-order surface reconstruction using the weighted averaging of local fittings as described in section 2.2.1 can be used to approximate the geometry and its derivatives. We extend its compu-

Table 4.1: Numerical integration formulas for triangles.

Order	Figure	Error	#points	Coordinates	Weights
Quadratic		$O(h^3)$	3	$1/6, \quad 1/6$	$1/3$
				$2/3, \quad 1/6$	$1/3$
				$1/6, \quad 2/3$	$1/3$
Cubic		$O(h^4)$	4	$1/3, \quad 1/3$	-27/48
				$0.6, \quad 0.2$	25/48
				$0.2, \quad 0.6$	25/48
				$0.2, \quad 0.2$	25/48
Quartic		$O(h^5)$	6	$\alpha_1, \quad \alpha_1$	0.1099517436553
				$\alpha_1, \quad \beta_1$	0.1099517436553
				$\beta_1, \quad \alpha_1$	0.1099517436553
				$\alpha_2, \quad \beta_2$	0.223381589678
				$\beta_2, \quad \alpha_2$	0.223381589678
				$\beta_2, \quad \beta_2$	0.223381589678
			with	$\alpha_1 = 0.0915762135098$	$\beta_1 = 0.8168475729805$
				$\alpha_2 = 0.1081030181681$	$\beta_2 = 0.4459484909160$
Quintic		$O(h^6)$	7	$1/3, \quad 1/3$	9/25
				$\alpha_1, \quad \alpha_1$	0.125939180544827
				$\alpha_1, \quad \beta_1$	0.125939180544827
				$\beta_1, \quad \alpha_1$	0.125939180544827
				$\alpha_2, \quad \beta_2$	0.132394152788506
				$\beta_2, \quad \alpha_2$	0.132394152788506
				$\beta_2, \quad \beta_2$	0.132394152788506
			with	$\alpha_1 = 0.101286507323456$	$\beta_1 = 0.797426985353087$
				$\alpha_2 = 0.05971587178977$	$\beta_2 = 0.470142064105115$

tational framework to reconstruct a function defined over the surface to high-order of accuracy by using similar concepts. Finally, we compute the first order differential quantities (normals and Jacobians) as well as the integrand at the quadrature points from the reconstructed surface and integrand, which are then used by the quadrature rules.

4.3.1 High-Order Piecewise Smooth Geometry

We use *WALF (Weighted Average of Least-squares Fittings) Surface* (section 2.2.1) to obtain a piecewise smooth geometrical approximation of the input geometry. To perform integration, we need to compute the differential quantities such as the Jacobian and surface normal at the quadrature points. We now derive the analytical expression of the exact Jacobian or surface normal of the WALF surface.

Assume the polynomial fitting is constructed at each vertex using a local orthonormal *uvw* coordinate frame, where the *uv* plane is nearly tangential to the surface, and the *w* coordinate is orthogonal to the *uv* plane. 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 $\mathbf{q} \in \sigma$ with natural coordinates $\boldsymbol{\xi} = (\xi, \eta)$ within the triangle in the corresponding local *uvw* coordinate system at the vertex. This is performed by interpolating the *uv* coordinates at $\mathbf{q}$ 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 $\mathbf{q} \in \sigma$, let $\mathbf{p}_i(\boldsymbol{\xi})$ denote the reconstructed point associated with the ith vertex for $i = 1, 2, 3$. Then the blended WALF surface within the triangle is defined by

$$\mathbf{p}(\boldsymbol{\xi}) = \sum_{i=1}^{3} N_i(\boldsymbol{\xi})\mathbf{p}_i(\boldsymbol{\xi}). \tag{4.3.1}$$

For conciseness, we will drop out the parameters (ξ, η) in the notation. It is clear that the WALF surface 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.

Let $\mathbf{Q}_i$ denote the rotation matrix $\left[\begin{array}{c|c|c} \hat{\mathbf{t}}_{1i} & \hat{\mathbf{t}}_{2i} & \hat{\mathbf{m}}_i \end{array}\right]$ for the ith vertex, where $\hat{\mathbf{m}}_i$ denotes the first order approximation to the unit normal at the ith vertex, and let $\mathbf{T}_i$ denote the matrix composed of the first two columns of $\mathbf{Q}_i$. Let $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ denote the vertices of the triangle in the xyz coordinate system, and $\mathbf{J} = [\mathbf{x}_2 - \mathbf{x}_1 \mid \mathbf{x}_3 - \mathbf{x}_1]$ be the Jacobian of the linear triangle with respect to ξ. Let $\mathbf{p}_i(\xi)$ denote the reconstructed point in the xyz coordinate system by the polynomial associated with the ith vertex. The Jacobian of the WALF-reconstructed triangle with respect to $\xi = (\xi, \eta)$, denoted by $\nabla_\xi \mathbf{p}$, is then

$$\begin{aligned}\nabla_\xi \mathbf{p}(\xi) &= \nabla_\xi \sum_{i=1}^{3} (N_i(\xi)\mathbf{p}_i(\xi)) \\ &= \sum_{i=1}^{3} \left(\left(\nabla_\xi N_i(\xi)\right) \mathbf{p}_i(\xi) \right) + \sum_{i=1}^{3} \left(N_i(\xi) \nabla_\xi \mathbf{p}_i(\xi) \right).\end{aligned}$$

Noting that

$$\sum_{i=1}^{3} \left(\left(\nabla_\xi N_i(\xi)\right) \mathbf{p}_i(\xi) \right) = \left[\begin{array}{c|c} \mathbf{p}_2(\xi) - \mathbf{p}_1(\xi) & \mathbf{p}_3(\xi) - \mathbf{p}_1(\xi) \end{array}\right]. \qquad (4.3.2)$$

Then, $\nabla_\xi \mathbf{u} = \begin{bmatrix} \frac{\partial u}{\partial \xi} & \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} & \frac{\partial v}{\partial \eta} \end{bmatrix} = \mathbf{T}_i^T \mathbf{J}_f$, and

$$\begin{aligned}\nabla_\xi \mathbf{p}_i(\xi) &= \nabla_\mathbf{u} \left(\mathbf{Q}_i \mathbf{p}_i(\mathbf{u}) + \mathbf{x}_i\right) \left(\nabla_\xi \mathbf{u}\right) \\ &= \left(\mathbf{Q}_i \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \frac{\partial f_i}{\partial u} & \frac{\partial f_i}{\partial v} \end{bmatrix} \right) \mathbf{T}_i^T \mathbf{J}_f \\ &= \left(\mathbf{T}_i + \hat{\mathbf{m}}_i \nabla_\mathbf{u} f_i(\mathbf{u})\right) \mathbf{T}_i^T \mathbf{J}_f,\end{aligned}$$

where $\mathbf{u}$ is also functions of ξ. Therefore,

$$\sum_{i=1}^{3} N_i(\xi) \nabla_\xi \mathbf{p}_i(\xi) = \sum_{i=1}^{3} \left(N_i(\xi) \left(\mathbf{T}_i + \hat{\mathbf{m}}_i \nabla_\mathbf{u} f_i(\mathbf{u}(\xi))\right) \mathbf{T}_i^T \mathbf{J}_f \right). \qquad (4.3.3)$$

The Jacobian of the WALF-reconstructed triangle with respect to $\boldsymbol{\xi} = (\xi, \eta)$, is then

$$\nabla_{\xi}\mathbf{p} = \sum_{i=1}^{3} \left(N_i \left(\mathbf{T}_i + \hat{\mathbf{m}}_i \nabla_{\mathbf{u}} f_i \right) \mathbf{T}_i^T \mathbf{J} \right) + \left[\begin{array}{c|c} \mathbf{p}_2 - \mathbf{p}_1 & \mathbf{p}_3 - \mathbf{p}_1 \end{array} \right]. \tag{4.3.4}$$

The normal of the blended surface can be obtained by taking the cross product of the two column vectors of $\nabla_{\xi}\mathbf{p}$.

4.3.2 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. We note that using a similar approach as WALF, one can reconstruct a smooth function φ or $\boldsymbol{\varphi}$ defined over the surface. For a vector-valued function $\boldsymbol{\varphi}$, if it does not have specific physical or geometric meaning, its individual components can be approximated independently of each other.

For a function defined at each vertex of the triangulated surface, a high-order approximation at any point inside a triangle can be obtained using local polynomial fitting at its vertices as is done for the height function. For a function φ smooth over the surface, we can consider it as a function of the parameters $\mathbf{u} = (u, v)$. We expand φ into a Taylor (or Maclaurin) series about $\mathbf{u}_0 = (0,0)$ similar to equation (2.1.1) as

$$\varphi(\mathbf{u}) = \sum_{p=0}^{d} \sum_{j,k \geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!} + O(\|\mathbf{u}\|^{d+1}). \tag{4.3.5}$$

Given a set of points in the stencil of $\mathbf{u}_0$ with known values for φ, we get

$$\sum_{p=0}^{d} \sum_{j,k \geq 0}^{j+k=p} c_{jk} \frac{u_i^j v_i^k}{j!k!} \approx \varphi(u_i, v_i). \tag{4.3.6}$$

Clearly, the only difference between high-order reconstruction of the height function f and the integrand φ is the right hand side of the resulting linear system. The system of equation given by (4.3.6) can be solved in the same manner as for the height function. In the case of a vector function, we perform high order reconstruction for each component independently.

Similar to surface reconstruction, we blend these local fittings using the shape function to reconstruct a piecewise continuous support of the function. Let $\tilde{\varphi}_i(\xi)$ denote the function reconstructed from the polynomial at the *i*th vertex. The blended function within the triangle is then

$$\tilde{\varphi}(\xi) = \sum_{i=1}^{3} N_i(\xi)\tilde{\varphi}_i(\xi). \tag{4.3.7}$$

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.

4.3.3 Overall Algorithm

After obtaining the high-order reconstructions of the surface and the integrand, the remainder of our algorithm is to apply numerical quadrature over each triangle. The general form of the quadrature over a triangle σ applied to scalar surface integrals is

$$\int_\sigma \varphi \, da = \int_\sigma \varphi(\xi)g(\xi)d\xi \, d\eta = \frac{1}{2}\sum_k w_k \varphi(\xi_k,\eta_k)g(\xi_k,\eta_k), \tag{4.3.8}$$

where (ξ_k,η_k) are the natural coordinates of the quadrature points within the triangle, and w_k is the associated weights of the quadrature points. For a comprehensive list of quadrature rules, readers are referred to [13]. The overall flow of our algorithm is shown in Figure 4.1.

The above method has many advantages. First of all, it is easy to implement, efficient and robust. Second, it does not suffer from oscillations due to high order polynomial interpolation. The least squares formulation mitigates any such oscillation that might have been present due to high order polynomial interpolation. Another advantage of the method is that it is largely independent of mesh quality in the sense that it does not depend on the angles or Jacobian or other general quality attributes of a mesh. As a result we can still get high order for very poor quality meshes (Subsection 4.5.1). There are only two requirements imposed on the input mesh. First, the vertices approximate the exact surface up to machine precision. Second, there is a sufficient number of points in the stencils. The connectivity of the mesh is used extensively to form the stencils at each vertex about which the local polynomial fitting is to take place.

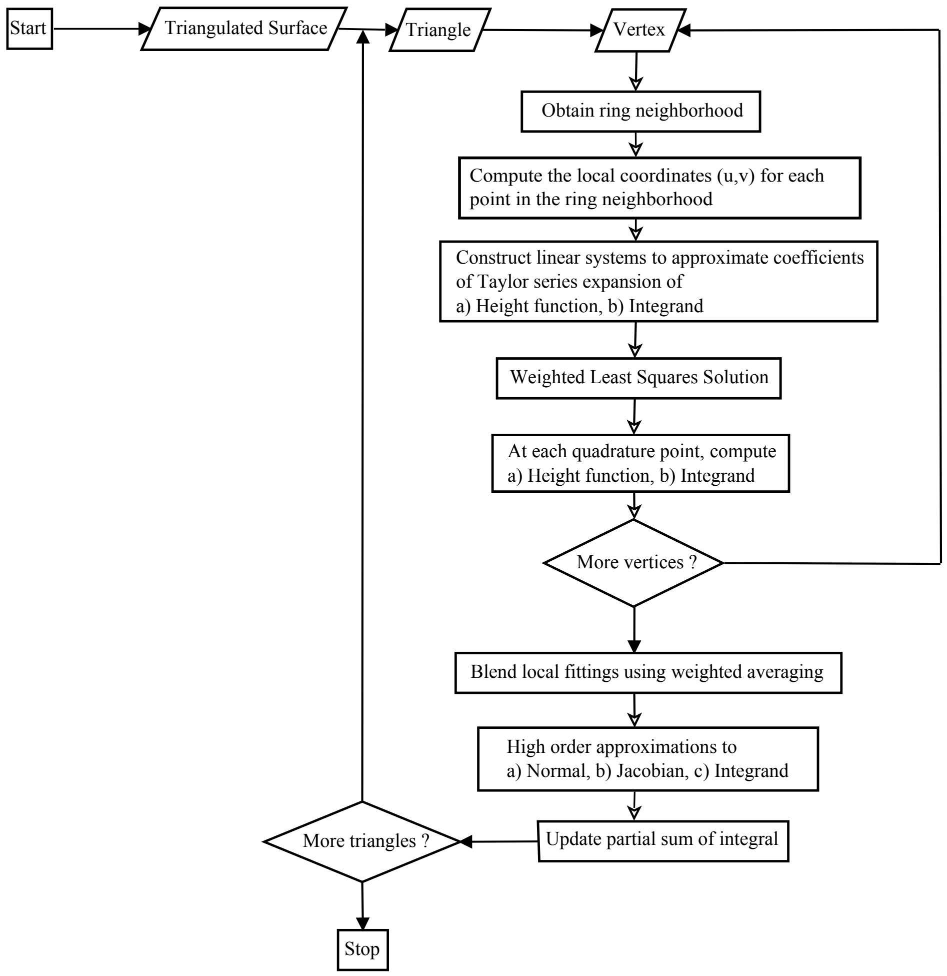

Start
Triangulated Surface
Triangle
Vertex
Obtain ring neighborhood
Compute the local coordinates (u,v) for each point in the ring neighborhood
Construct linear systems to approximate coefficients of Taylor series expansion of a) Height function, b) Integrand
Weighted Least Squares Solution
At each quadrature point, compute a) Height function, b) Integrand
More vertices ?
Blend local fittings using weighted averaging
High order approximations to a) Normal, b) Jacobian, c) Integrand
Update partial sum of integral
More triangles ?
Stop

Figure 4.1: Flowchart of our algorithm for high-order surface integration where w, φ and g denote the weight, integrand and Jacobian of the surface at a quadrature point. In case of vector functions, $\varphi = \boldsymbol{\varphi} \cdot \hat{\boldsymbol{n}}$.

4.4 Theoretical Convergence

In this section, we analyze the order of accuracy of our method. The results are given in terms of a local measure of the mesh resolution such as the average edge length h in the triangulation. The main result of theoretical analysis is given by the following theorem and corollary:

Theorem 4. *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 $\left|\int_{\Gamma} \varphi \, da - \int_{\tilde{S}} \tilde{\varphi} \, da\right| = O(h^d + h^5)$.*

As a corollary of Theorem 4, we also obtain the following estimation of the errors in surface integrals of vector-valued functions.

Corollary 5. *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 $\left|\int_{\Gamma} \varphi \cdot d\mathbf{a} - \int_{\tilde{S}} \tilde{\varphi} \cdot d\mathbf{a}\right| = O(h^d + h^5)$.*

When quadrature rules are used for the integrals numerically, in general it suffices to use degree-d quadrature rules over the triangles to preserve the accuracy. The proof of Theorem 4 is somewhat involved, as it involves the analysis of the accuracies of the WALF-reconstructed surface, its approximate normal and Jacobian, and the reconstructed function. We will analyze these individual terms in the following sections and then use the results to prove the accuracy of the integration. Note that the barrier of $O(h^5)$ in the theorem above is somewhat pessimistic, due to technical difficulties in the analysis of the approximation to the Jacobian in Section 4.4.2. In numerical experiments, we observe up to eighth-order convergence rate in our numerical experiments as shown in Section 4.5.

4.4.1 Accuracy of Weighted Averaging of Fittings

Let Γ denote a smooth surface and S be a triangulation of Γ. Suppose the triangulation S is composed of triangles σ_i, i.e., $S = \cup_{i=1}^{K} \sigma_i$, where K is the number of triangles. Let $\tilde{\sigma}_i$ denote the high-order reconstruction of σ_i by WALF, then $\tilde{\Gamma} = \sum_{i=1}^{K} \tilde{\sigma}_i$. For each point $\mathbf{q}_0 \in \sigma_i$ in S, let $\mathbf{v}_i$ be the ith vertex of σ_i, and let $\mathbf{m}_i$ denote a first-order approximation to the unit normal at $\mathbf{v}_i$ used in constructing the local uvw coordinate frame at $\mathbf{v}_i$. Let $\mathbf{q}_i$ denote the reconstructed point for $\mathbf{q}_0$ based on local fittings at the vertices $\mathbf{v}_i$. Let $\mathbf{q}$ be the reconstructed point of $\mathbf{q}_0$ on $\tilde{\Gamma}$, i.e., $\mathbf{q} = \sum_{i=1}^{3} N_i \mathbf{q}_i$. Let $\mathbf{r}_i$ denote the intersection of lines $\mathbf{q}_0\mathbf{q}_i$ with the exact surface

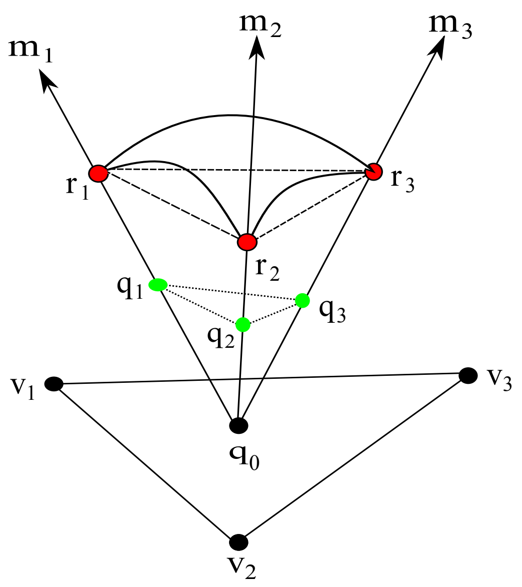


Figure 4.2: Illustration of notation used in proofs. Point $\mathbf{q}_0$ is a point inside a linear triangle $\triangle\mathbf{v_1 v_2 v_3}$. The reconstructed points corresponding to local fittings at vertices $\mathbf{v}_i$ are denoted by green points $\mathbf{q}_i$. The intersection of the first order normals $\mathbf{m}_i$ with the exact surface is denoted by $\mathbf{r}_i$.

Γ; see Figure 4.2 for an illustration of these points. We first have the following lemma regarding the distances between $\mathbf{r}_i$ and $\mathbf{r}_j$, which impose a lower bound for the minimum error that can be obtained by the weighted averaging scheme.

Lemma 6. *Assume that the local coordinate systems are aligned with approximate surface normals that are at least first-order accurate. The distance between points* $\|\mathbf{r}_i - \mathbf{r}_j\| = O(h^3)$*,* $i, j = 1,2,3$.

This lemma was a side product in the analysis of the order of accuracy of WALF surface in [29]. For completeness, we give a separate proof as follows.

Proof. It suffices to prove for only $\|\mathbf{r}_1 - \mathbf{r}_2\|$. Let θ_i denote the angle at $\mathbf{r}_i$ in the triangle $\mathbf{q}_0\mathbf{r}_1\mathbf{r}_2$. We have

$$\begin{aligned}\|\mathbf{r}_1 - \mathbf{r}_2\| &= \left|\cos\theta_1 \|\mathbf{r}_1 - \mathbf{q}_0\| + \cos\theta_2 \|\mathbf{r}_2 - \mathbf{q}_0\|\right| \\ &\le (|\cos\theta_1| + |\cos\theta_2|) \max_{i=1,2} \{\|\mathbf{r}_i - \mathbf{q}_0\|\}\,.\end{aligned}$$

Note that $|\cos\theta_1| = O(h)$, because $\mathbf{r}_1\mathbf{r}_2$ is at least a first-order approximation to the tangent direction at $\mathbf{r}_1$, and $\mathbf{m}_1$ is by assumption a first-order approximation to the normal direction at $\mathbf{v}_1$ and in turn also at $\mathbf{r}_1$. Similarly, $|\cos\theta_2| = O(h)$. Furthermore, $\|\mathbf{r}_i - \mathbf{q}_0\| = O(h^2)$, because $\mathbf{q}_0$ is a linear approximation to $\mathbf{r}_i$ in the local coordinate frame aligned with $\mathbf{m}_i$. Therefore, $\|\mathbf{r}_1 - \mathbf{r}_2\| \leq O(h)O(h^2) = O(h^3)$. $\square$

For error analysis, we need to define a mapping from $\tilde{\Gamma}$ to Γ. Let $\mathbf{r}$ be the projection of $\mathbf{q}$ onto Γ along the direction $\sum_{j=1}^{3} N_j \mathbf{m}_j$, and let Π denote the mapping from $\tilde{\Gamma}$ onto Γ, i.e., $\mathbf{r} = \Pi(\mathbf{q})$. Assume the triangulation S is a dense enough triangulation of Γ, so that the projection from $\tilde{\Gamma}$ to Γ is one-to-one and onto. Let γ_i denote the range of this projection from points in σ_i. Therefore, the triangulation S defines local parameterizations for both Γ and $\tilde{\Gamma}$ within each triangle. With this mapping, we obtain the following theorems regarding the reconstructed surface and function.

Theorem 7. *Given a mesh whose vertices approximate a smooth surface Γ with an error of at most $O(h^{d+1})$, for each point $\mathbf{q}$ on the WALF surface obtained from degree-d fittings, $\|\mathbf{q} - \Pi(\mathbf{q})\| = O(h^{d+1} + h^6)$.*

Proof. Let $\mathbf{r} = \Pi(\mathbf{q})$. We have

$$\begin{aligned}
\|\mathbf{q} - \mathbf{r}\| &= \left\| \sum_{i=1}^{3} N_i \mathbf{q}_i - \mathbf{r} \right\| \\
&= \left\| \sum_{i=1}^{3} N_i \mathbf{q}_i - \sum_{i=1}^{3} N_i \mathbf{r}_i + \sum_{i=1}^{3} N_i \mathbf{r}_i - \mathbf{r} \right\| \\
&\leq \left(\sum_{i=1}^{3} N_i \|\mathbf{q}_i - \mathbf{r}_i\| \right) + \left\| \mathbf{r} - \sum_{i=1}^{3} N_i \mathbf{r}_i \right\|.
\end{aligned}$$

Because $\mathbf{q}_i$ is a local fitting with dth degree polynomial at the local coordinate frame at the ith vertex of σ, $\|\mathbf{q}_i - \mathbf{r}_i\| = O(h^{d+1})$. For the second term, note that $\sum_{i=1}^{3} N_i \mathbf{r}_i$ is a linear approximation to $\mathbf{r}$ over triangle $\mathbf{r}_1\mathbf{r}_2\mathbf{r}_3$ of length $O(h^3)$, and hence $\left\|\mathbf{r} - \sum_{i=1}^{3} N_i \mathbf{r}_i\right\| = O(h^6)$. Overall, $\|\mathbf{q} - \mathbf{r}\| = O(h^{d+1} + h^6)$. $\square$

In the following, we generalize this result to functions reconstructed by weighted averaging of least-squares fitting (WALF). We introduce some additional notation here. Let $\varphi : \Gamma \to \mathbb{R}$ be a scalar function defined on a smooth surface Γ and let $\tilde{\varphi} : \tilde{\Gamma} \to \mathbb{R}$ denote the WALF reconstructed scalar function over $\tilde{\Gamma}$. More specifically, $\tilde{\varphi}$ is defined as follows. Let $\bar{\varphi}_i(\mathbf{x}) : S \to \mathbb{R}$ denote the local fitting of a point $\mathbf{x}$ on S at the ith vertex of a triangle σ containing $\mathbf{x}$. Let $\xi = (\xi, \eta)$ denote the local parameters (i.e., the natural coordinates) of the points within σ and $\tilde{\mathbf{x}}$ denote the reconstructed point on $\tilde{\Gamma}$. Then,

$$\tilde{\varphi}(\tilde{\mathbf{x}}(\xi)) = \sum_{i=1}^{3} N_i(\xi) \bar{\varphi}_i(\mathbf{x}(\xi)), \tag{4.4.1}$$

where N_i denotes the linear finite-element shape function associated with the ith point. Using this notation, we have the following theorem.

Theorem 8. *Given a mesh whose vertices approximate a smooth surface Γ with an error of at most $O(h^{d+1})$, let $\mathbf{r}$ be the projection of the point $\tilde{\mathbf{x}}$ on the reconstructed WALF surface with degree-d fittings onto the exact surface Γ. Then for the degree-d WALF reconstructed function $\tilde{\varphi}(\tilde{\mathbf{x}})$, $|\tilde{\varphi}(\tilde{\mathbf{x}}) - \varphi(\mathbf{r})| = O(h^{d+1} + h^6)$.*

The proof for Theorem 8 shares some similarities with that for Theorem 7, but it is slightly more complicated because the additional function φ. We give the proof as follows.

Proof. Note that ξ parameterizes $\tilde{x}(\xi)$ within a triangle σ. Then

$$\begin{aligned}
|\tilde{\varphi}(\tilde{\mathbf{x}}(\xi)) - \varphi(\mathbf{r})| &= \left|\sum_{i=1}^{3} N_i \bar{\varphi}_i(\mathbf{x}(\xi)) - \varphi(\mathbf{r})\right| \\
&= \left|\sum_{i=1}^{3} N_i \bar{\varphi}_i(\mathbf{x}(\xi)) - \sum_{i=1}^{3} N_i \varphi(\mathbf{r}_i) + \sum_{i=1}^{3} N_i \varphi(\mathbf{r}_i) - \varphi(\mathbf{r})\right| \\
&\le \left(\sum_{i=1}^{3} N_i \left|\bar{\varphi}_i(\mathbf{x}(\xi)) - \varphi(\mathbf{r}_i)\right|\right) + \left|\varphi(\mathbf{r}) - \sum_{i=1}^{3} N_i \varphi(\mathbf{r}_i)\right| .
\end{aligned}$$

Because $\bar{\varphi}_i$ is a local fitting with dth degree polynomial at the local coordinate frame at the ith vertex of σ, $|\bar{\varphi}_i(\mathbf{x}(\xi)) - \varphi(\mathbf{r}_i)| = O(h^{d+1})$. For the second term, note that $\sum_{i=1}^{3} N_i \varphi(\mathbf{r}_i)$ is a linear approximation to $\varphi(\mathbf{r})$ over triangle $\mathbf{r}_1\mathbf{r}_2\mathbf{r}_3$ of length $O(h^3)$, and hence $\left|\varphi(\mathbf{r}) - \sum_{i=1}^{3} N_i \varphi(\mathbf{r}_i)\right| = O(h^6)$. $\square$

4.4.2 Accuracy of Jacobian

The mapping Π from $\tilde{\Gamma}$ to Γ allows us to obtain the reference solution for the Jacobian as well as for the normals. For triangle $\tilde{\sigma}_i \in \tilde{\Gamma}$, let $\tilde{\mathbf{J}}_i$ denote the Jacobian matrix for the mapping from natural coordinates ξ of σ_i, i.e., $\tilde{\mathbf{J}}_i(\mathbf{q}(\xi)) = \left[\frac{\partial \mathbf{q}}{\partial \xi} \middle| \frac{\partial \mathbf{q}}{\partial \eta}\right]$. Similarly, for triangle $\gamma_i \in \Gamma$, let $\mathbf{J}_i$ denote the Jacobian matrix for the mapping from ξ to γ_i, i.e., $\mathbf{J}_i(\mathbf{r}(\xi)) = \left[\frac{\partial \mathbf{r}}{\partial \xi} \middle| \frac{\partial \mathbf{r}}{\partial \eta}\right]$. In numerical integration, these Jacobian matrices are used to compute the surface area as well as the surface normal. We first prove the following lemma regarding the Jacobian matrices.

Lemma 9. *Let a_{ij} denote an entry of matrix $\tilde{\mathbf{J}}_\mathbf{i}(\mathbf{q}(\xi)) - \mathbf{J}_\mathbf{i}(\mathbf{r}(\xi))$. Then*

$$|a_{ij}| = O(h^{d+1} + h^6). \tag{4.4.2}$$

Proof. Note that

$$\tilde{\mathbf{J}}_{\mathbf{i}}(\mathbf{q}(\xi)) - \mathbf{J}_{\mathbf{i}}(\mathbf{r}(\xi)) = \left[\frac{\partial(\mathbf{q}-\mathbf{r})}{\partial \xi} \middle| \frac{\partial(\mathbf{q}-\mathbf{r})}{\partial \eta} \right]. \tag{4.4.3}$$

Due to Theorem 8, $\|\mathbf{q}-\mathbf{r}\| = O(h^{d+1}+h^6)$. The parameters ξ and η are always between 0 and 1 independent of h, therefore $\|\partial(\mathbf{q}-\mathbf{r})/\partial\xi\| = O(h^{d+1}+h^6)$. Similarly $\|\partial(\mathbf{q}-\mathbf{r})/\partial\eta\| = O(h^{d+1}+h^6)$. Hence for each entry a_{ij} in $\tilde{\mathbf{J}}_{\mathbf{i}} - \mathbf{J}_{\mathbf{i}}$, $|a_{ij}| = O(h^{d+1}+h^6)$. □

Let $\tilde{g}_i = \sqrt{\det(\tilde{\mathbf{J}}_i^T \tilde{\mathbf{J}}_{\mathbf{i}})}$ and $g_i = \sqrt{\det(\mathbf{J}_i^T \mathbf{J}_i)}$. Let $\tilde{\mathbf{n}}(\mathbf{q})$ denote the unit normal to $\tilde{\Gamma}$ at point $\mathbf{q}$, obtained by normalizing $\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}$, and similarly let $\mathbf{n}(\mathbf{r})$ denote the unit normal to Γ at $\mathbf{r}$, obtained by normalizing $\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}$. We consider $\tilde{g}_i$ as an approximation g_i, and $\tilde{\mathbf{n}}(\mathbf{q})$ as an approximation to $\mathbf{n}(\mathbf{r})$. Because these quantities are obtained from simple arithmetic operations from $\tilde{\mathbf{J}}_i$ and $\mathbf{J}_i$, respectively, a direct consequence of Lemma 9 is the following theorem.

Theorem 10. *$|\tilde{g}_i - g_i| = O(h^{d+2}+h^7)$ and $\|\tilde{\mathbf{n}}(\mathbf{q}) - \mathbf{n}(\mathbf{r})\| = O(h^d + h^5)$.*

Proof. From Lemma 9 $|a_{ij}| = O(h^{d+1}+h^6)$, we get $\left|\frac{\partial q_i}{\partial \xi} - \frac{\partial r_i}{\partial \xi}\right| = O(h^{d+1}+h^6)$ and $\left|\frac{\partial q_i}{\partial \eta} - \frac{\partial r_i}{\partial \eta}\right| = O(h^{d+1}+h^6)$ where $\mathbf{q} = (q_1,q_2,q_3)$ and $\mathbf{r} = (r_1,r_2,r_3)$. Now, $\tilde{g}_i = \sqrt{\det(\tilde{\mathbf{J}}_i^T \tilde{\mathbf{J}}_{\mathbf{i}})} = \left\|\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}\right\|$ and $g_i = \sqrt{\det(\mathbf{J}_i^T \mathbf{J}_i)} = \left\|\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}\right\|$. We note that each term of $\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}$ is of the form $\frac{\partial r_k}{\partial \xi}\frac{\partial r_j}{\partial \eta} - \frac{\partial r_j}{\partial \xi}\frac{\partial r_k}{\partial \eta}$ for appropriate indices j and k. We also note that $\left\|\frac{\partial q_i}{\partial \xi}\right\| = O(h)$ as it should be at least a first order approximation of a tangent direction at $\mathbf{q}$. Therefore,

$$\begin{aligned}
\left\|\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} - \frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}\right\| &\approx O(h^{d+2}+h^7) \\
\left\|\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}\right\| &\approx \left\|\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}\right\| + O(h^{d+2}+h^7),
\end{aligned} \tag{4.4.4}$$

i.e, $|\tilde{g}_i - g_i| = O(h^{d+2} + h^7)$. Similarly,

$$\begin{aligned}
\|\tilde{\mathbf{n}}(\mathbf{q}) - \mathbf{n}(\mathbf{r})\| &= \left\| \frac{1}{\tilde{g}_i}\left(\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}\right) - \frac{1}{g_i}\left(\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}\right)\right\| \\
&= \left\| \frac{g_i - \tilde{g}_i}{g_i\tilde{g}_i}\left(\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}\right) + \frac{1}{g_i}\left(\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta} - \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}\right)\right\| \\
&\leq \left| \frac{g_i - \tilde{g}_i}{g_i\tilde{g}_i}\right| \left\|\left(\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta}\right)\right\| + \frac{1}{g_i}\left\|\frac{\partial \mathbf{q}}{\partial \xi} \times \frac{\partial \mathbf{q}}{\partial \eta} - \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}\right\| \\
&= \frac{1}{|g_i|}\left(|\tilde{g}_i - g_i| + O(h^{d+2} + h^7)\right) \\
&= O(h^d + h^5),
\end{aligned}$$

where the last equality is because $|g_i| = O(h^2)$. □

Note that this result gives an upper bound of the errors in the approximation to the surface areas and the normal, and in turn they can bound the errors of the numerical integration. However, this approximation largely depends on our definition of the mapping Π from $\tilde{\Gamma}$ and Γ. There may be an “optimal” mapping for which the error may be bounded by $O(h^6)$ instead of $O(h^5)$. However, we are content with this result because $O(h^5)$ is a small enough error bound for almost all practical purposes.

4.4.3 Accuracy of Integration

Using the above results, we can now prove our main theorem about the accuracy of integration, i.e., Theorem 4.

Proof. Suppose the triangulation S is composed of triangles σ_i, i.e., $S = \cup_{i=1}^{K}\sigma_i$, where K is the number of triangles. Let $\tilde{\sigma}_i$ denote the high-order reconstruction of σ_i by WALF, then $\tilde{S} = \sum_{i=1}^{K}\tilde{\sigma}_i$, and

$$\int_{\tilde{S}} \tilde{\varphi}\, da = \sum_{i=1}^{K}\int_{\tilde{\sigma}_i} \tilde{\varphi}\, da. \tag{4.4.5}$$

Using the parameterization in subsection 4.4.2, we have $\Gamma = \sum_{i=1}^{K}\gamma_i$, and

$$\int_{\Gamma} \varphi\, da = \sum_{i=1}^{K}\int_{\gamma_i} \varphi\, da. \tag{4.4.6}$$

Then

$$\int_{\Gamma} \varphi\, da = \sum_{i=1}^{K} \int_{\sigma_i} \varphi(\xi) g_i(\xi)\, d\xi\, d\eta,$$

$$\int_{\tilde{S}} \tilde{\varphi}\, da = \sum_{i=1}^{K} \int_{\sigma_i} \tilde{\varphi}(\xi) \tilde{g}_i(\xi)\, d\xi\, d\eta.$$

For conciseness, we omit the parameters ξ in the following. Therefore,

$$\begin{aligned}
\left| \int_{\Gamma} \varphi\, da - \int_{\tilde{S}} \tilde{\varphi}\, da \right| &= \left| \sum_{i=1}^{K} \int_{\sigma} (\varphi g_i - \tilde{\varphi} \tilde{g}_i)\, d\xi\, d\eta \right| \\
&\leq \sum_{i=1}^{K} \left| \int_{\sigma} ((\varphi - \tilde{\varphi})\, g_i + \tilde{\varphi}\, (g_i - \tilde{g}_i))\, d\xi\, d\eta \right| \\
&\leq \sum_{i=1}^{K} \left(\int_{\sigma} |(\varphi - \tilde{\varphi})\, g_i|\, d\xi\, d\eta + \int_{\sigma} |\tilde{\varphi}\, (g_i - \tilde{g}_i)|\, d\xi\, d\eta \right) \\
&\leq \underbrace{|\varphi - \tilde{\varphi}|_{max} \sum_{i=1}^{K} \int_{\sigma} g_i\, d\xi\, d\eta}_{1} + \underbrace{|\tilde{\varphi}|_{max} \sum_{i=1}^{K} \int_{\sigma} |g_i - \tilde{g}_i|\, d\xi\, d\eta}_{2}
\end{aligned}$$

The first term is the condition number of the problem with a perturbation to the integrand and is bounded by $O(h^{d+1} + h^6)$ times the surface area. The error of the integral depends on the accuracy of the last integral in the second term. Before we determine its accuracy, we note that the number of triangles in the surface mesh is inversely proportional to average area of triangle i.e, $K = O(1/h^2)$. Using this result along with Lemma 9, $|\tilde{g}_i - g_i| = O(h^{d+2} + h^7)$, we get

$$\begin{aligned}
\sum_{i=1}^{K} \int_{\sigma} |(g_i - \tilde{g}_i)|\, d\xi\, d\eta &= \sum_{i=1}^{K} \int_{\sigma} O(h^{d+2} + h^7)\, d\xi\, d\eta \\
&= O(h^d + h^5)
\end{aligned}$$

Hence, $\left| \int_{\Gamma} \varphi\, da - \int_{\tilde{S}} \tilde{\varphi}\, da \right| = O(h^d + h^5)$. $\square$

Given Theorem 4, then Corollary 5 follows naturally, because the surface integrals $\int_{\Gamma} \varphi \cdot d\mathbf{a}$ can be reduced to the scalar integral of $\varphi \cdot \hat{\mathbf{n}}$, and from Theorem 10, $\varphi \cdot \hat{\mathbf{n}}$

can be approximated to $O(h^d + h^5)$, just as the scalar function φ in Theorem 4.

4.5 Numerical Experiments

We now present the numerical results with our methods, focusing on assessing the accuracy and convergence. We also demonstrate the usefulness of our method with an application and report the performance.

4.5.1 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 of high-order reconstructions of the geometry and function. For these purposes, it suffices to use simple smooth geometries. We used a torus (with inner radius 0.7 and outer radius 1.3) as the test geometry, and 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, we also generated another set of poor-quality meshes using marching cubes. Some example meshes are shown in Figure 4.3. For studying mesh convergence, 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

$$\text{relative error} = \frac{\|\ \text{numerical solution} - \text{reference solution}\ \|}{\|\ \text{reference solution}\ \|}, \tag{4.5.1}$$

and compute the average convergence rate as

$$\text{convergence rate} = \frac{1}{3}\log_2\left(\frac{\text{error of mesh 1}}{\text{error of mesh 4}}\right). \tag{4.5.2}$$

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.

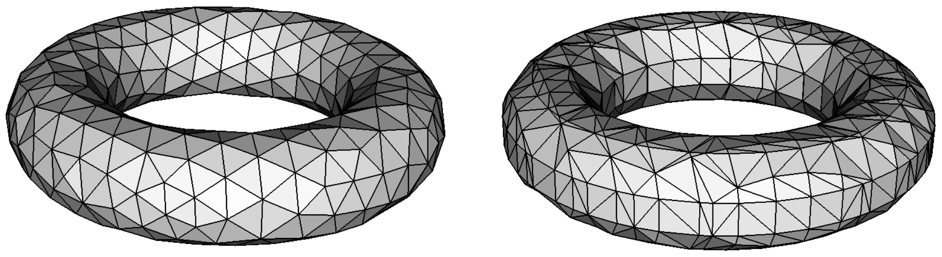


Figure 4.3: Coarsest high-quality (left) and poor-quality (right) test meshes for torus used in our numerical experiments.

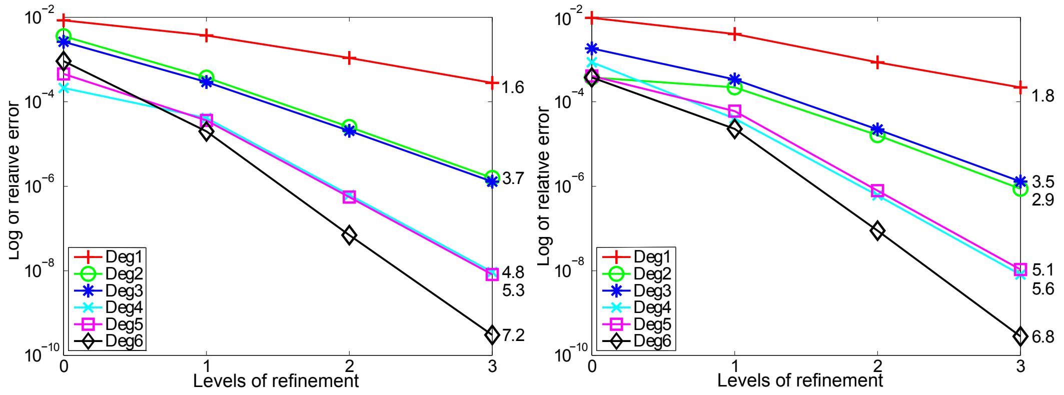


Figure 4.4: Relative errors and average convergence rates of surface area of a torus under mesh refinement for high-quality (left) and low-quality (right) meshes.

Surface Integral of Scalar and Vector Functions

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 torus, the exact surface area can be computed analytically, so we use the exact answer as the reference solution.

Figure 4.4 plots the relative errors of the computed surface areas for the torus using polynomial fittings of degrees between 1 and 6 under mesh refinement. 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 integrating vector-valued functions, the simplest example is the computation of the volume bounded by a closed surface. By the divergence theorem, the volume is

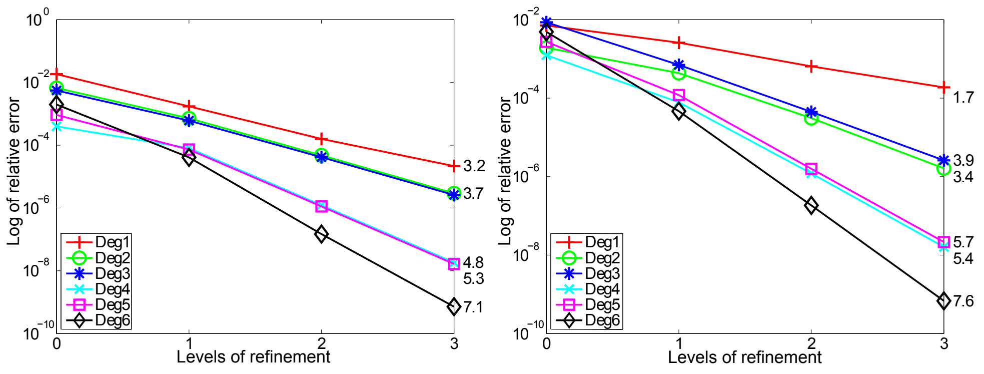


Figure 4.5: Relative errors and average convergence rates of computed volume of a torus under mesh refinement for high-quality (left) and low-quality (right) meshes.

equal to one third of the surface integral of the position vector of the surface, i.e.,

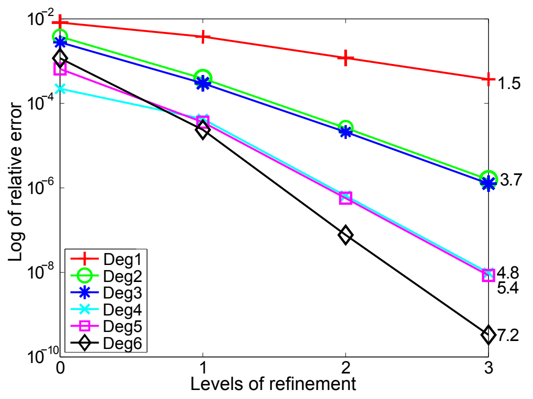
$$V = \int_{\Gamma} \varphi \cdot d\mathbf{a} = \int_{\Gamma} \varphi \cdot \hat{\mathbf{n}} da, \tag{4.5.3}$$

where $\varphi = \mathbf{x}/3$ and $\hat{\mathbf{n}}$ is the outward unit normal to the surface Γ. For simple geometries such as a torus, the exact volumes are available analytically and hence we use them as reference solutions in our test. Figure 4.5 shows the relative errors of the computed volume of the torus using polynomial fittings of degrees between 1 and 6 under mesh refinement. The average convergence rates are shown on the right of the plots, which again confirm our theoretical results.

For generality, we also test integrating a scalar test function $\varphi(x,y,z) = \sin(x+yz) + e^{xy}$ and a vector-valued test function $\varphi(x,y,z) = (x\cos(y),\, e^{y},\, z+e^{z})$ on the torus. Since the exact integrals are unavailable, we use the results for the finest meshes as reference solutions. The errors for this case are shown in Figure 4.6. These results are qualitatively similar to those of surface areas, and the convergence rates again confirm the theoretical results.

Necessity of High-Order Reconstructions of Surface

In the previous subsection we showed that our method can deliver high-order convergence rates. It is important to note that the high-order reconstructions of both the surface and the function are necessary in assuring high convergence rates. To demonstrate this, Figure 4.7 (left) shows the computed surface area of the torus using high-order reconstruction of the function but using piecewise linear geometry defined by the linear triangles of the meshes, and Figure 4.7 (right), which is the same as Figure 4.4 (left), shows the corresponding results using high-order



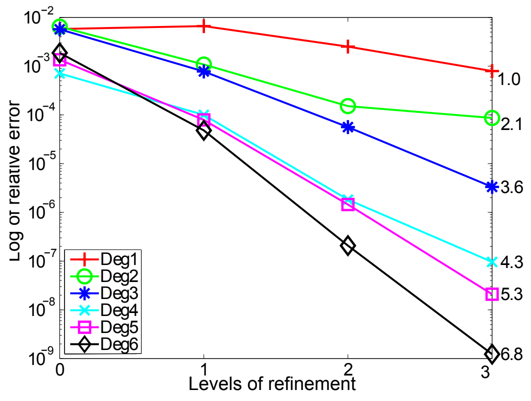
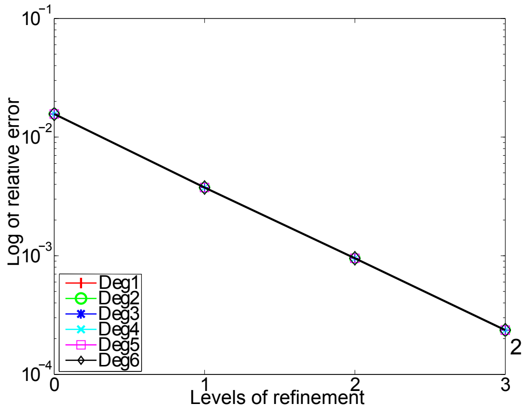


Figure 4.6: Relative errors and average convergence rates for integration of a test scalar function (left) and a test vector-valued function (right) on the torus.



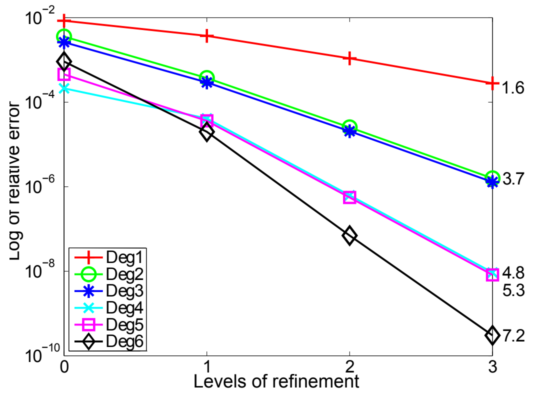
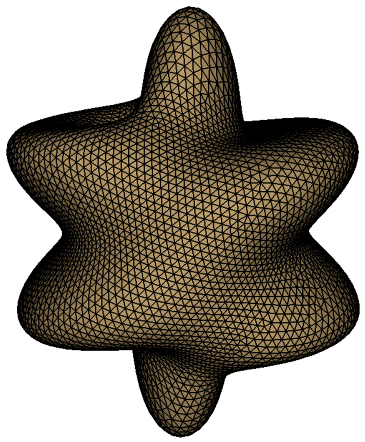


Figure 4.7: Comparison of convergence rates of computed surface area of the torus using high-order reconstructions of (left) only the function vs. (right) both the geometry and function.

reconstructions of both the geometry and the function. Using only high-order reconstruction of the geometry but not the function would lead to a result similar to that in Figure 4.7 (left). From this comparison, it is evident that both the geometry and the function need to be reconstructed to high-order accuracy in order to achieve high-order surface integration.

Complicated Geometries

We show some examples of our method applied to more complicated geometries as well as meshes with sharp features. For testing a complicated smooth domain of integration, we used a spherical surface harmonic of degree 6 and order 1 (Figure 4.8).



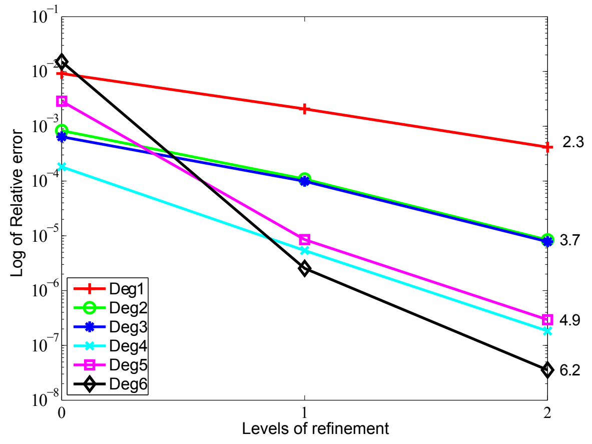


Figure 4.8: Convergence rates of computed surface area of a spherical harmonic of degree 6 (right). The domain of integration is shown on the left.

The convergence of errors under mesh refinement for computing the surface area is shown in Figure 4.8 (right). It can be seen that high order of convergence is achieved as predicted.

To demonstrate the applicability of our method to meshes with sharp features, we apply our algorithm to a slotted sphere (Figure 4.9). The presence of sharp features in the input geometry requires some preprocessing of the mesh. In particular, we first identify all the sharp ridges and corners of the input geometry using a simple algorithm. Thereafter, we virtually split the mesh by duplicating the vertices along ridges, and then apply our algorithm to this new mesh. The new connectivity ensures that stencils for points on the ridge edges are one sided. Figure 4.9 shows the convergence results for computing the volume of the slotted sphere. We observe that for fittings of degree 4, 5 and 6, the order of convergence is about 4. This is due to the reduction of degree of fittings for points on the sharp features, which have insufficient number points in their stencils to achieve higher order. As a result, the overall degree of the algorithm is reduced.

4.5.2 Performance Results

We present some performance results for our algorithms. We implemented our algorithm in MATLAB and then converted the code into C using MATLAB Coder. We performed the tests on a Linux machine with a 3.16GHz Intel Core 2 Duo processor and 4GB of RAM. Tables 4.2 shows the execution times for the computations of the volume of the torus, using polynomials of degrees between 1 and 6. It can be seen that the runtimes scale linearly with the size of the meshes. In addition,

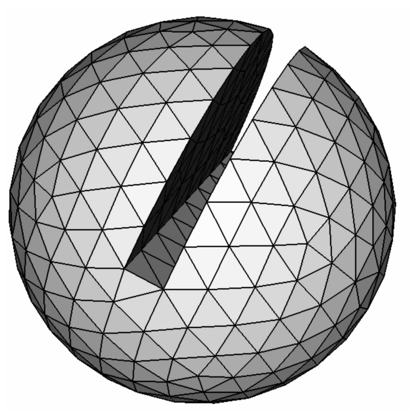
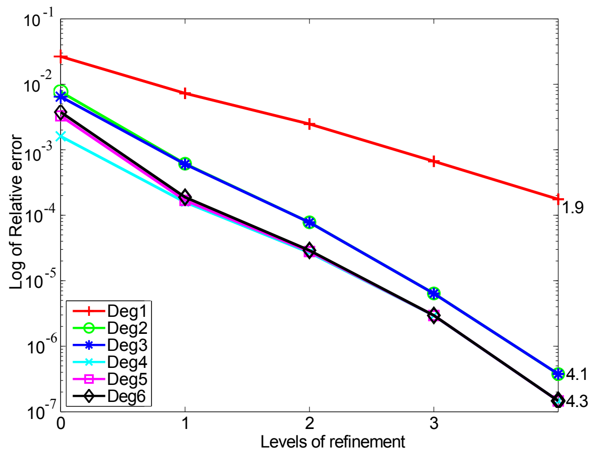



Figure 4.9: Convergence rates of computed volume of a slotted sphere (right). The domain of integration is shown on the left.

Table 4.2: Runtime in seconds for computation of volume of triangulated torus.

mesh	#vertices	degree 1	2	3	4	5	6
1	544	0.0927	0.172	0.34	0.785	1.45	2.44
2	1,896	0.322	0.612	1.26	2.73	5.57	11.2
3	7,528	1.27	2.35	4.87	11.0	21.0	44.3
4	31,392	5.34	9.84	20.18	47.2	89.1	168.4

the higher the degree of the polynomials, the more expensive the algorithms are. Compared to integration with linear triangles, it is expected that polynomial fittings would be much more expensive, because the stencil for the fitting is large and least squares problems need to be solved. The computational complexity of solving the linear system is dominated by that of finding the reduced QR factorization. For our current implementation, polynomial fittings are two to three orders of magnitude more expensive than using linear triangles, but its runtime can be improved with further performance optimization.

Chapter 5

High-Order Surface Remeshing

Surface meshing and remeshing are widely used mesh-based operations in geometry processing, mesh generation, and numerical solutions of partial differential equations with complex geometry. In the past, researchers have mostly focused on generating piecewise linear (or bilinear) meshes (such as [18]), which have only up to second-order accuracy. In recent years, high-order surface meshing and remeshing (using piecewise quadratic or higher-degree elements or polynomial patches) have become increasingly important. The fundamental reason behind this trend is simple: high-order numerical methods require the same or higher order accuracy for the geometry to achieve the desired accuracy. With only piecewise linear approximations, the discretizations of differential quantities (such as normals or curvatures) or the solutions of the differential equations may only achieve low-order accuracy, and sometimes may not converge at all. The problem is even more demanding in the problems with evolving geometry, where the surface mesh must be adapted, and sometimes needs to be untangled to remove mildly (or nearly) folded triangles.

We investigate the problem of smoothing, optimizing and adapting a surface mesh with high-order accurate nodal positions, so that the new mesh can preserve the order of accuracy of numerical methods, such as high-order finite element methods and generalized finite difference methods. The focus is to improve the accuracy, stability and robustness of mesh-based geometry processing and numerical computations. We couple various mesh quality improving techniques with high-order surface reconstruction to develop remeshing strategies that produce high-quality triangular meshes, while untangling mildly folded triangles and preserving the geometry to high-order accuracy. We further improve the robustness of our algorithms for high-order point projections in under-resolved regions by introducing geometric limiters to reduce the projection to lower order. We present the theoretical framework of our methods, show experimental comparisons against other methods, and demonstrate its utilization to geometric PDE's, high-order finite elements, biomedical image-based surface meshes, and complex interface meshes in fluid simulations.

Our methods can achieve high order of accuracy for sufficiently resolved regions of the geometry. For high-order surface reconstruction we use either WALF or CMF (2.2). The error for point positions using WALF and CMF are bounded by $O(h^d + h^6)$ and $O(h^d)$ respectively, where h is average edge length and d is the degree of fitting. Though WALF limits the accuracy to $O(h^6)$ for degree of fittings $d > 6$, in practice degrees of fitting greater than 6 are rarely used. The introduction of a geometric limiter allows us to systematically use a lower-order point positioning for under-resolved regions where high-order fittings tend to be problematic. As a result, it improves the robustness of our algorithms. We use smooth geometries without any sharp features to test our algorithms, though the method can easily be modified to treat sharp features, by the null space checking at a vertex to restrict movement of points on sharp features to ridges or to restrict movement entirely.

5.1 Related Work

We start by reviewing some background for high-order surface remeshing. We first motivate the problem by giving a brief overview of two types of numerical methods over unstructured meshes: finite element methods and generalized finite difference methods.

High-Order Finite Element Methods

The most well-known numerical methods on unstructured meshes are probably finite element methods. When using isoparametric elements [25], these methods use quadratic or cubic interpolation within each element for both representing the geometry and approximating the solutions. Additional nodes are required within each element to construct these interpolations. For example in two dimensions, a quadratic interpolation over a triangle requires six points and a cubic interpolation requires ten points. Higher than cubic interpolation is possible but is rarely used in practice.

Generally speaking, a quadratic interpolation *can* approximate a function up to third-order accuracy, and a cubic interpolation *can* approximate a function up to fourth-order accuracy. However, this order of accuracy may not be attainable if any part of the algorithm is low order, including the geometry. In particular, if linear elements are used for approximating the surface, or if the nodal coordinates are only second-order (or third-order) accurate, then the overall numerical method will be limited to at most second-order (or third-order) accuracy, defeating the purpose of using quadratic (or cubic) elements. For this reason, the order of accuracy of any optimization strategy must be at least the same or higher than that of the numerical methods to be used.

In [29], we described a refinement scheme to generate quadratic or cubic surface meshes for high-order finite element methods from a given surface mesh with piecewise linear elements but high-order accurate nodal positions. Here we use our methods to improve the quality of a high-order finite element mesh (subsection 6.3).

Generalized Finite Difference Methods

Besides finite-element methods, another class of high-order method is the generalized finite difference methods, which have been gaining popularity in recent years. Unlike the finite element methods, the generalized finite difference methods are based on weighted least-squares approximations rather than interpolation, so they have more flexibility in defining the stencils for numerical differentiation and in constructing local patches for numerical integration, which are the core computations in most numerical discretizations of differential equations.

The generalized finite difference schemes are often used in meshless methods [4, 42]. However, they also apply to unstructured meshes, where the mesh connectivity can be used as an aid for efficient construction of the stencils for computing least squares approximations. For example, the work done by Jiao and co-authors in [31, 52] for computing normals and curvatures of triangulated surfaces to high-order accuracy is essentially a generalized finite difference scheme. Given a surface mesh, they construct the stencils of a center vertex as its k-ring and $k.5$-ring neighborhood, where $k = 1, 2, 3, \ldots$ Let us define the 0-ring of a vertex as the vertex itself. Then, the k-ring vertices are those share an edge with a vertex in the $(k-1)$-ring, and the $k.5$-ring vertices are those share a face with two vertices in the k-ring. The definition of half-rings for triangular meshes was introduced in [31]. Our definition here is simpler than the original definition in [31] and is same as that in [29], and as a result it is better suited for quadrilateral meshes. Typically, it is advisory to use the $(d+1)/2$-ring for noise-free surfaces or $(d/2+1)$-ring for noisy surfaces, so that the number of points in the stencil is about 1.5 to two times the number of coefficients for least-squares fittings. These stencils are based on mesh connectivity alone. In addition, each point has an associated weight in the stencil in the weighted least-squares formulation, which can be used to reduce the influence of points that are far away from the center vertex or to filter out vertices that are on the opposite side of a sharp feature. Compared to meshless methods, this approach allows much more efficient construction of stencils and also avoids the issues of short circuiting of stencils. We refer readers to [31, 52] for further details of the stencils.

We emphasize that the generalized finite difference methods use the mesh connectivity for only defining the topology of the surface and for constructing the stencils for least squares approximations. This is a major departure from the finite-element methods, which use the elements of the mesh to define the interpolation of the geometry. We focus on remeshing for generalized finite difference methods. Our method produces high-quality, high-order accurate surface meshes, so they can be refined using high-order surface reconstruction to obtain valid, high-order surface meshes for high-order finite element methods.

5.2 High-Order Surface Smoothing and Untangling

Vertex redistribution a.k.a mesh smoothing is widely used to improve the quality of the mesh while preserving accuracy of nodal positions so that it can be used in numerical computations. We begin with a global mesh smoothing algorithm which combines variational mesh optimization [30] and weighted Laplacian smoothing with high-order surface reconstruction.

The variational mesh optimization technique improves the quality of a mesh by minimizing the difference between an ideal and an actual element via minimization of energy functions based on conformal and isometric mapping. In general, we apply variational smoothing of the mesh for most iterations except when there is a significant number of folded triangles in the current mesh or it has regions with very sharp angles in which case weighted Laplacian smoothing is performed. This is because the energy function for variational optimization becomes infinite at a degenerate element, which forms a barrier to prevent inverted elements from being untangled. However, it is not uncommon for the input mesh to be mildly or nearly folded. Therefore for robustness, we must be able to handle mildly folded meshes.

We consider a triangle to be *inverted* (i.e, *folded*) if its normal direction is more than 90 degrees off from its neighborhood. The folding is considered "mild" if the area of folded triangle is small compared to the area of its one-ring neighborhood. By untangling (or unfolding) we mean resolving these folded triangles so that there are no folded triangles in the resulting mesh. An example of a folded triangle is shown in 5.1. We resolve mesh folding by using a weighted Laplacian smoothing (5.2.1) because it tends to move a vertex to a weighted average of its one-ring neighborhood, so repeatedly applying the procedure tends to avoid mesh folding and can even unfold a mesh. For clarity, we emphasize that our untangling process attempts to resolve folded triangles of a smooth surface in a local fashion. It does not attempt to resolve global self-intersections of a surface.

In order to preserve the accuracy of the nodal positions, we couple these mesh quality improving techniques with high-order surface reconstruction for high-order point projection. High-order point projection works well for sufficiently resolved meshes. However, for under-resolved regions, such as those representing highly

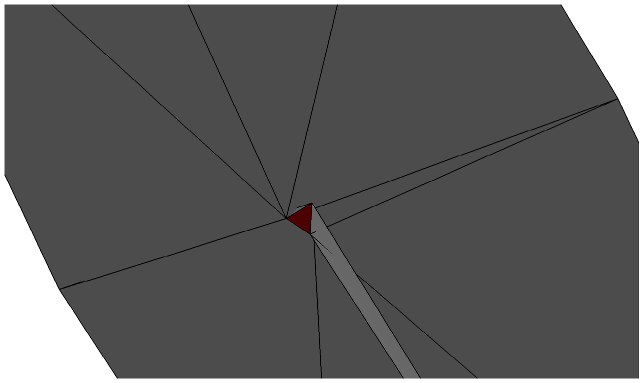


Figure 5.1: A mildly folded triangle

curved regions, it might tend to be problematic and result in artifical spikes. As a result, we further improve the robustness of our algorithms for under-resolved regions by introducing a geometric limiter based on an *a posterior* error indicator to reduce to lower order by using a point-wise volume-conserving smoothing approach [37]. The introduction of a geometric limiter allows us to systematically use a lower-order point positioning for under-resolved regions where high-order fittings tend to be problematic. Figure 5.2 shows the overall flow of the smoothing algorithm. The displacements of the vertices after the smoothing step are first scaled to lie within the projection of the one ring neighborhood with further asynchronous scaling to prevent the introduction of additional folded triangles due to concurrent motion of the vertices. Finally, these scaled displacements are projected to a surface reconstructed locally using either WALF or CMF, subject to a geometric constraint. We next discuss each of these steps in more detail.

5.2.1 Mesh Smoothing Algorithms

Variational Mesh Optimization by Vertex Movement

We use the variational mesh optimization proposed in [30] to improve mesh quality. The main idea of the variation optimization is to minimize the total energy of the mesh by defining energy functions based on discrete conformal and isometric mappings from an ideal reference triangle to an actual triangle. For simplicity, we consider only isotropic triangular surface meshes whose angles are optimized with respect to an equilateral triangle as illustrated in figure 5.3a.

Given a triangle $\tau \equiv \mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ in $\mathbb{R}^3$, let θ_i denote the angle at the ith vertex of the ideal triangle, $\mathbf{l}_i$ denote the opposite edge of the ith vertex of the actual triangle, and

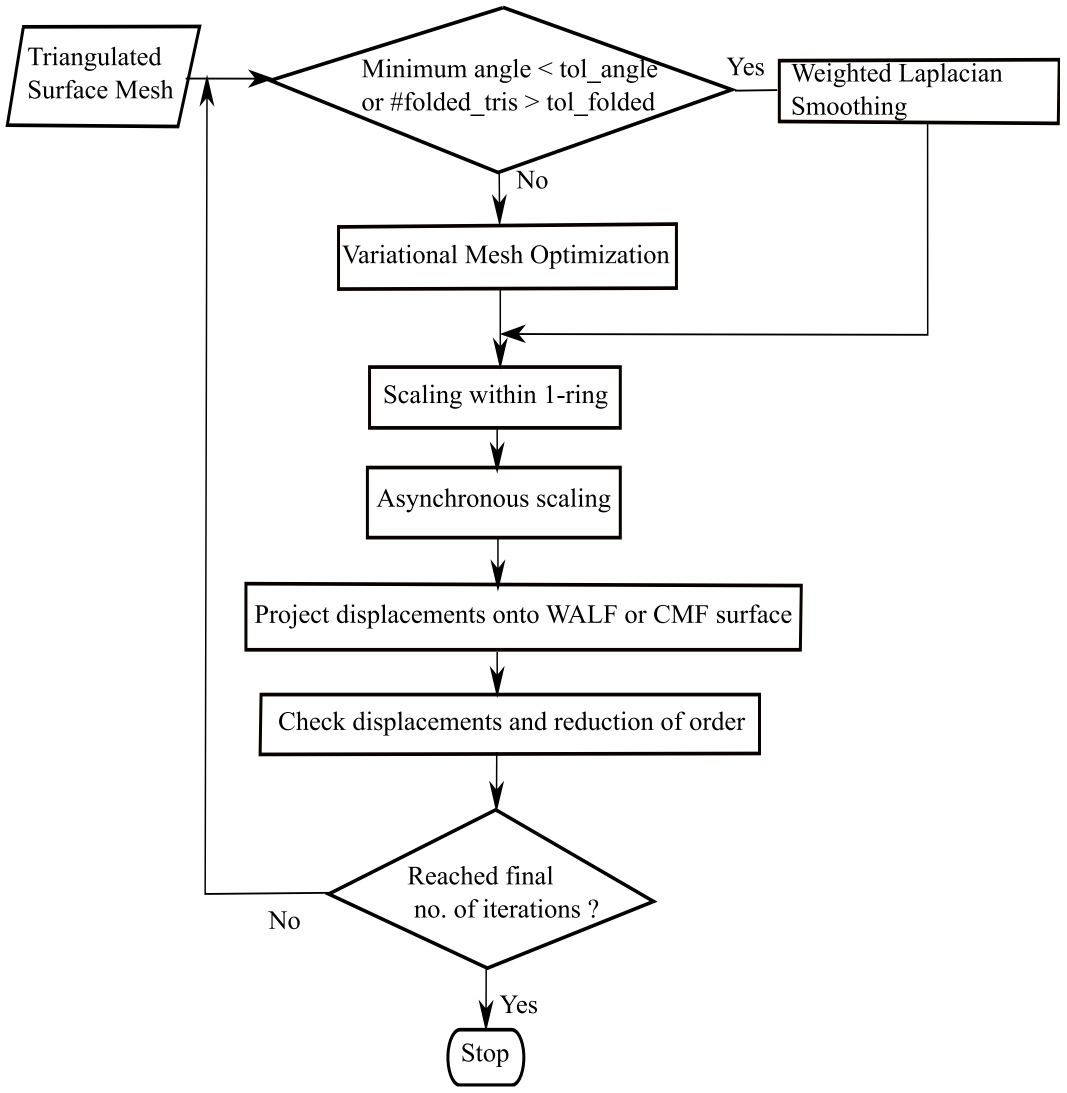
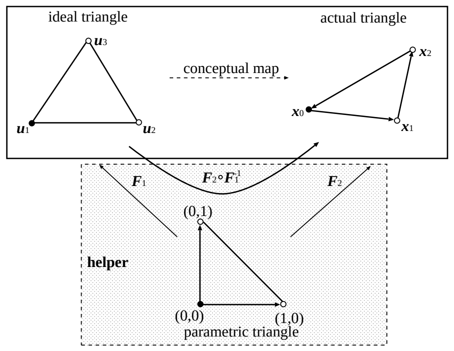
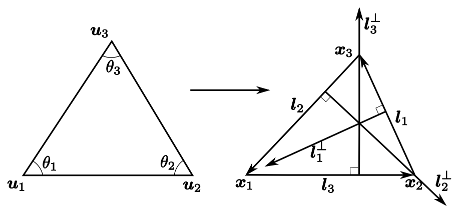

Triangulated Surface Mesh
Minimum angle < tol_angle or #folded_tris > tol_folded
Yes
Weighted Laplacian Smoothing
No
Variational Mesh Optimization
Scaling within 1-ring
Asynchronous scaling
Project displacements onto WALF or CMF surface
Check displacements and reduction of order
Reached final no. of iterations ?
No
Yes
Stop

Figure 5.2: Major steps of the smoothing algorithm.



(a) Discrete mapping between ideal and actual triangle



(b) Naming convention for triangles

Figure 5.3: Illustration of the discrete mapping between an ideal triangle and an actual triangle via a parametric triangle as well as the naming convention followed for triangles.

A denote twice the area of triangle. The energy for this triangle is

$$E_\theta(\tau) = \frac{\omega}{A}\sum_{i=1}^{3} \|\mathbf{l}_i\|^2, \tag{5.2.1}$$

where $\omega = 1/\sqrt{3}$. To improve the quality of a mesh M, we minimize the total energy $\sum_{\tau\in M} E_\theta(\tau)$. We achieve this using an iterative procedure similar to the block-Jacobi solver for Newton's method. In particular, we compute the gradient and Hessian of E_θ over all the triangles, and obtain the gradient and Hessian at each vertex by adding their corresponding values at its incident triangles. After obtaining the gradient and Hessian, one could apply one step of Newton's method to determine a displacement $\mathbf{d}_v$ for the vertex, i.e., by solving $\left(\nabla^2_{\mathbf{x}_v} E_\theta\right)\mathbf{d}_v = -\nabla_{\mathbf{x}_v} E_\theta$. To preserve the geometry, we must constrain the displacement to be nearly tangential to the surface. We compute the tangent vectors using an eigenvalue analysis as described in [26]. Specifically, given a vertex, let $\hat{\mathbf{n}}_j$ and ω_j denote the face normal and face area of its jth incident face, respectively. Let

$$\mathbf{N} = \sum_j \omega_j \hat{\mathbf{n}}_j \hat{\mathbf{n}}_j^T, \tag{5.2.2}$$

which is symmetric and positive semi-definite. Let

$$\mathbf{N} = \sum_{i=1}^{3} \lambda_i \hat{\mathbf{e}}_i \hat{\mathbf{e}}_i^T, \tag{5.2.3}$$

where the λ_i are the eigenvalues (and also the singular values) with $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$, and the $\hat{\mathbf{e}}_i$ are the corresponding eigenvectors (and also the singular vectors) of $\mathbf{N}$. Assuming there are no acute sharp features, the eigenvalues provide an indication of singularity at the vertex: If the surface is smooth at v, then $\lambda_2 \ll \lambda_1$ and $\lambda_3 \ll \lambda_1$; at a ridge vertex with two distinct normal directions, $\lambda_3 \ll \lambda_2$; at a sharp corner, all three eigenvalues have similar sizes. Therefore, the null space of $\mathbf{N}$ (more precisely, its *numerical null space* spanned by the eigenvectors corresponding to relatively small eigenvalues) gives the local tangent at the vertex. We consider the eigenvalue λ_i to be small if $\lambda_i \leq \varepsilon\lambda_1$ for some small ε (such as 0.003 as suggested in [26]).

For a vertex on a smooth surface, let $\hat{\mathbf{t}}_1$ and $\hat{\mathbf{t}}_2$ denote two orthonormal tangent vectors at v, and let $\mathbf{T} = [\hat{\mathbf{t}}_1 \mid \hat{\mathbf{t}}_2]$. Instead of solving for $\mathbf{d}_v$ in $\mathbb{R}^3$, we reformulate the problem to solve for $\mathbf{d}_v = \mathbf{T}\mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^2$. The gradient and Hessian of E_θ with respect to $\mathbf{u}$ are then $\nabla_{\mathbf{u}} E_\theta = \mathbf{T}^T \nabla_{\mathbf{x}_v} E_\theta$ and $\nabla^2_{\mathbf{u}} E_\theta = \mathbf{T}^T \left(\nabla^2_{\mathbf{x}_v} E_\theta\right) \mathbf{T}$, respectively. Therefore, Newton's equation for minimizing E_θ with respect to $\mathbf{u}$ becomes

$$\left(\mathbf{T}^T \left(\nabla^2_{\mathbf{x}_v} E_\theta\right) \mathbf{T}\right) \mathbf{u} = -\mathbf{T}^T \nabla_{\mathbf{x}_v} E_\theta, \tag{5.2.4}$$

and therefore

$$\mathbf{d}_v = -\mathbf{T}\left(\mathbf{T}^T \left(\nabla^2_{\mathbf{x}_v} E_\theta\right) \mathbf{T}\right)^{-1} \mathbf{T}^T \nabla_{\mathbf{x}_v} E_\theta. \tag{5.2.5}$$

The eigenvalue analysis allows us to treat sharp features if necessary by replacing $\mathbf{T}$ in (5.2.5) by the unit tangent vector to the ridge curve for a point on a ridge curve or by fixing a vertex if it is a corner. However, we do not treat sharp features during any of the smoothing algorithms though they can be easily integrated.

Weighted Laplacian Smoothing

In weighted Laplacian smoothing, we move each vertex toward a weighted average of the centroids of its incident triangles, where the weight for each centroid is equal to distance from the vertex to the centroid. This is equivalent to a weighted averaging of neighboring vertices, where the weight for each vertex is equal to one third of the sum of the distances from the vertex to the centroids of adjacent triangles. We constrain the displacements for each vertex onto the tangent space by computing tangent vectors using an eigenvalue analysis as described in [26].

5.2.2 High-Order Point Projection

In both variational mesh optimization and weighted-Laplacian smoothing, the vertices are moved within the tangent space, which are in general only second-order accurate. To achieve high-order accuracy, we must project the points onto a high-order surface reconstruction. We reconstruct the high-order surface using either

Weighted Averaging of Local Fittings (*WALF*) or *Continuous moving frames (CMF)* scheme as described in section 2.2. In order to properly project the displacements, we first scale them with respect to their one-ring neighborhood such that a unique triangle can be computed over which the high-order surface can be reconstructed. Since, this scaling does not take into account the concurrent movement of vertices, we further rescale them to prevent mesh folding.

Displacement Scaling

Importantly, the higher-order surface is reconstructed locally over an appropriate triangle. To identify that triangle, a search is performed over the local neighborhood of the vertex until a triangle is found that contains the point $\mathbf{v}+\mathbf{d}$ inside one of its offsets, where $\mathbf{v}$ and $\mathbf{d}$ represent any vertex and its displacement, respectively. An offset of a triangle with vertices $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ is the triangle

$$(\mathbf{v}_1+\beta\mathbf{n}_1, \mathbf{v}_2+\beta\mathbf{n}_2, \mathbf{v}_3+\beta\mathbf{n}_3) \tag{5.2.6}$$

where $\mathbf{n}_i$ is the normal at the vertex $\mathbf{v}_i$. To avoid introducing any folded triangles as well as to reduce computation, it is preferable that the search be limited to the one-ring neighborhood of the vertex. To ensure the existence of such a triangle in its one-ring neighborhood for proper projection, we scale the displacement such that its projection lies within the one-ring neighborhood of the vertex. Clearly, the scaling is performed with respect to the old configuration of the mesh.

The scaling is performed in two steps. Firstly, an optimal triangle is found within the one-ring of the vertex such that the point $\mathbf{p}=\mathbf{v}+\mathbf{d}$ lies in the plane of one of its offsets. There is no ambiguity in choosing such a triangle if point $\mathbf{p}$ lies inside the offset of one of the triangles in one-ring neighborhood. In case it doesn't lie inside any offset of all triangles in the one-ring neighborhood, we choose an appropriate triangle based on distance. That is, we compute the distance of $\mathbf{p}$ to the boundaries of offsets and choose that triangle whose offset is nearest to $\mathbf{p}$. Secondly, the displacement is scaled repeatedly by a constant factor α until the rescaled displacement lies in the interior of another offset of the optimal triangle.

However, this scaling doesn't prevent mesh folding due to concurrent movement of vertices. To avoid such folding, an asynchronous step-size control is used. We determine a relaxation factor α_v for each vertex and then add $\alpha_v\mathbf{d}_v$ to the vertex. For detailed description of asynchronous step-size control, see [30].

5.2.3 Safeguards and Limiters

While higher-degree polynomials may deliver better accuracy for sufficiently resolved meshes, they may cause large variations and even spikes in the reconstructed surfaces. This may be due to a combination of the facts that higher-degree polynomials are more oscillatory and tend to be more sensitive to perturbations.

To achieve robustness, our method utilizes a number of safeguards and geometric limiters. First, our method checks the condition number of the linear system for the polynomial fitting. Since we use a variant of reduced QR factorization to solve the linear system, we can approximate the condition number with relative ease. If the condition number is too large (e.g., $\geq 10^6$ with double-precision floating-point arithmetic), we then correct it by either decreasing the degree of the polynomial or increasing the number of points in the stencil. This *a priori* checking helps to capture cases where the stencil does not have enough points or has a poor geometrical configuration locally.

Second, we check the distance from the reconstructed point to the original surface. If the distance is too large compared to the average edge length in the neighborhood, we reject the high-order reconstruction and resort to a low-order fitting. This *a posteriori* checking allows us to capture cases where the large errors may be due to under-resolved meshes at high-curvature regions, which tend to be beyond the asymptotic regime of the high-degree polynomials. These safeguards are sometimes referred to as *limiters*, and their implementations may depend on the specific situations. We next describe a geometric limiter for remeshing, especially for cases where quadratic fittings are used.

Geometric Limiter

The input surface mesh may be under-resolved in some regions and hence is beyond the asymptotic regime for high-degree fittings. In these cases, high-degree (and sometimes even quadratic) polynomials may lead to large errors. We identify such problematic regions by defining an error indicator. Specifically, for each vertex we compute the average edge length of all the incident edges and compare the high-order displacement against it. Let d_{hisurf} and e_{avg} denote the high-order displacement and the average edge length of incident edges, respectively. If $d_{hisurf} > \gamma e_{avg}$, where γ is a user-specified parameter, then we reduce the order locally by computing a new displacement using a point-based, volume-conserving smoothing algorithm [37]. In particular, we move a vertex $\mathbf{x}$ to $\mathbf{x} + \mathbf{dx}$, where where $\mathbf{dx} = \mathbf{dx}^s + h\hat{\mathbf{n}}$. The first part of the displacement, $\mathbf{dx}^s$ is obtained from a general smoothing scheme such as Laplacian smoothing. The second part, $h\hat{\mathbf{n}}$ corrects the volume changed by the first part. The normal $\hat{\mathbf{n}}$ is the normalized sum of normals of all triangles inci-

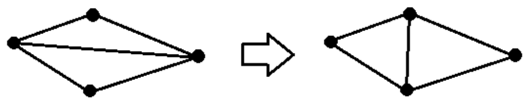


Figure 5.4: Illustration of edge flipping.

dent on $\mathbf{v}$ and $h = -\mathbf{dx}^s \cdot \hat{\mathbf{n}}$. More details of the algorithm can be found in [37].

5.2.4 Mesh Optimization and Untangling by Edge Flipping

The smoothing algorithm described above does not change the connectivity of the mesh. The energy (5.2.1) in variational optimization is convex with respect to the position of each vertex [30], and in practice the energy decreases close to the minimum quite rapidly. However, this minimum can be further reduced if the mesh connectivity is allowed to change. We modify the mesh connectivity using edge flipping, as illustrated in Fig. 5.4. In general, an edge flipping should be performed only if it preserves the topology of the surface (in particular, the new edge must not have already existed in the mesh). In addition, we use the following two criteria:

Energy-Reduction Edge Flipping: Flip an edge if it would decrease the sum of the energy of its incident triangles.

Valence-Improvement Edge Flipping: Flip an edge if after flipping the difference between the maximum and minimum valences[1] among the vertices of the two triangles is smaller than that before flipping.

The first strategy is a local strategy, and it is easy to understand. Its goal is simply to reduce the energy for each edge flipping, and therefore the total energy would never increase. The second strategy is non-local, and it may be counter-intuitive, because such a flipping may in fact increase the energy. Its motivation is the following: For a vertex in a mesh for a smooth surface, if it has a too high (> 7) or too low (< 5) valence, all its incident triangles are far from equilateral. Therefore, our strategy is to decrease the gap between the maximum and minimum valences, so that the energy in the whole neighborhood may be reduced. In our experience, repeatedly performing valence-improvement edge flipping tend to produce a mesh without high or low valences, and in turn allows much better mesh quality.

[1]The valence of a vertex is the number of its incident edges, which is equal to the number of incident triangles for a closed surface mesh.

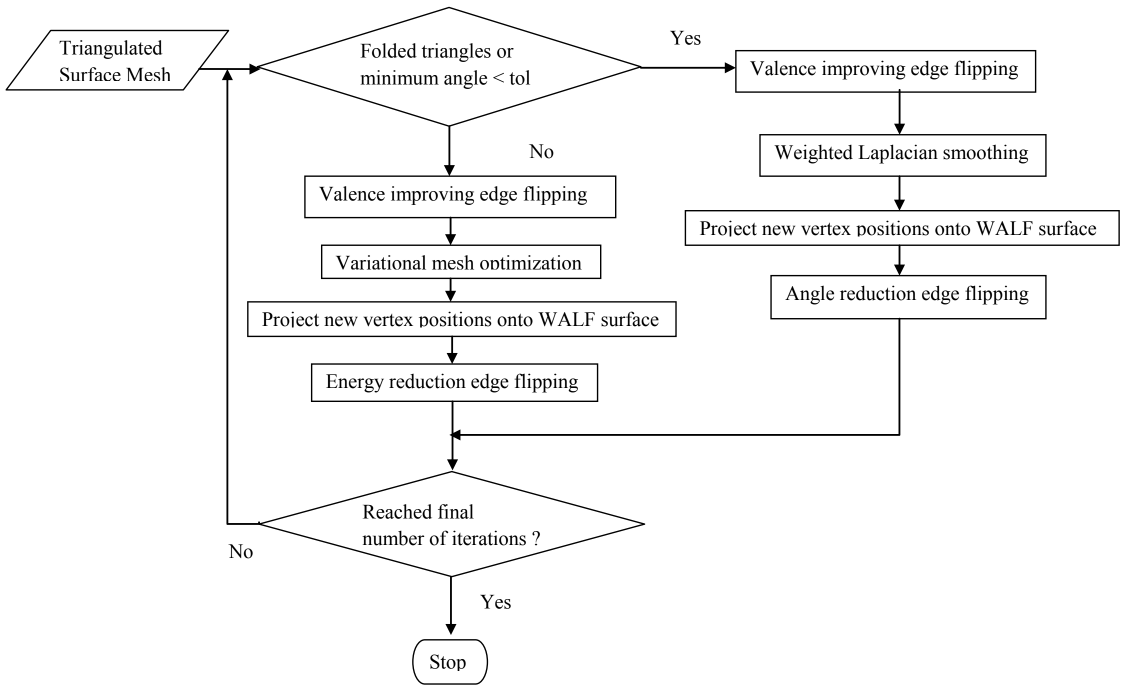
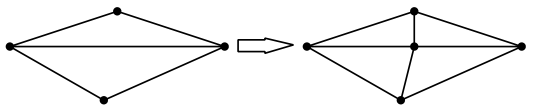

Triangulated Surface Mesh
Folded triangles or minimum angle < tol
Yes
No
Valence improving edge flipping
Weighted Laplacian smoothing
Project new vertex positions onto WALF surface
Angle reduction edge flipping
Valence improving edge flipping
Variational mesh optimization
Project new vertex positions onto WALF surface
Energy reduction edge flipping
Reached final number of iterations ?
No
Yes
Stop

Figure 5.5: Overview of key steps in mesh optimization and untangling with high-order surface reconstruction.

For each flipping strategy, we repeatedly perform the flipping until no further improvement is possible. Because valence-improvement edge flipping may not directly improve mesh quality, we perform it *before* the variational mesh optimization, and perform energy-reduction edge flipping *after* variational mesh optimization.

Similar to variational optimization, when coupling the weighted Laplacian smoothing with edge flipping, we also use two heuristics for edge flipping. The first is valence improvement edge flipping, as we described above. The second criterion is flipping based on the reducing angles equivalent to Delaunay edge flipping condition in 2-D. Furthermore, this operation also tends to flip an edge of a folded triangle. We perform angle-reduction edge flipping after mesh smoothing. Similar to variational optimization, we perform valence improvement edge flipping before mesh smoothing, and perform angle-reduction edge flipping after mesh smoothing. In addition, the vertices are projected onto high-order surface reconstruction after mesh smoothing, to preserve the high-order accuracy. Note that during the algorithm, we do not need to check explicitly whether individual triangles are folded. When simply repeated operations, the mesh becomes far from folding (namely, with 5 degrees). Figure 5.5 shows an algorithm combining mesh optimization and untangling with high-order surface reconstruction.


(a) Edge splitting

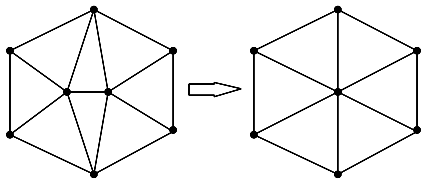

(b) Edge contraction

Figure 5.6: Illustrations of edge splitting and edge contraction.

5.3 High-Order Surface Mesh Adaptation

The method we described in the previous section have one major limitation. That is, it does not change the number of vertices in the mesh. However, it may be necessary to change the number of vertices for an evolving geometry. We further extend our techniques to accommodate mesh adaptation (refinement and coarsening) and couple the mesh modification operations with high-order point projection. We perform mesh adaptation primarily using two operations: edge splitting and edge contraction. To avoid thrashing, we must consider the two operations together to ensure consistency. We use a quality driven adaptivity [27] to obtain optimal aspect ratios of the triangulation.

Edge Splitting and Edge Contraction

First, let us consider edge splitting, which is the easier of the two. Given two adjacent triangular elements, edge splitting inserts a new vertex on their shared edge, as illustrated in Fig. 5.6(a). Similar to the approach in [27], we consider two criteria to determine whether an edge requires splitting:

Absolute Longness: The edge is the longest among its incident triangles and is longer than a provided threshold L.

Relative Longness: The edge is longer than a desired edge length l $(< L)$, one of its opposite angles is close to π (greater than provided θ_l), and the shortest edge among its incident triangles is no shorter than a provided threshold s $(< l)$.

The process of edge splitting abides by these criteria to help optimize element quality and consistency in size. The process of edge splitting occurs in decreasing order of edge lengths throughout the mesh.

For mesh coarsening, we perform edge contraction, which removes a vertex, as illustrated in Fig. 5.6(b). As in [27], we consider the following four criteria to determine whether an edge should be contracted:

Absolute small angle: the opposite angle in an incident triangle of the edge in question is smaller than a threshold θ_s, and the triangle's longest edge is shorter than a desired edge length l.

Relative shortness: The edge in question is shorter than a fraction r of the longest edge of its incident triangles.

Absolute small triangle: The edge in question is the shortest in its incident triangles and the longest edge of its incident triangles is shorter than a given threshold S.

Relative small triangle: The longest edge in its incident triangles is shorter than a fraction R of the longest edge in the mesh and also shorter than the desired edge length l.

The process of edge contraction abides by these criteria to help optimize element quality and consistency in size. The process of edge contraction occurs in increasing order of edge lengths throughout the mesh. While the first two criteria help to remove poor shaped triangles, the latter two criteria help to remove triangles that are too small, preserving the overall mesh quality as it evolves. When contracting an edge, its two incident vertices merge at a new location. To prevent mesh folding, we reject any contractions that would lead to topological changes or an inversion of normals on any triangle. Contracting the shortest edges first helps to avoid the need for such rejections. These parameters and criteria abide by Jiao et al. [27].

Preserving High-Order Accuracy

Just as mesh optimization, the mesh adaptation operations need to preserve the accuracy of the geometry. For edge splitting we first insert a new point onto an edge, and then we project the point onto a high-order reconstruction based on WALF or CMF. For edge contraction, we first replace the two vertices by a new point on the edge and then we project the point onto a high-order reconstruction. For high-order point projection either WALF or CMF based surface reconstruction can be used.

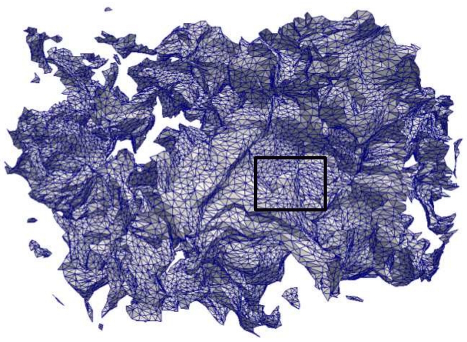
5.4 Numerical Experiments

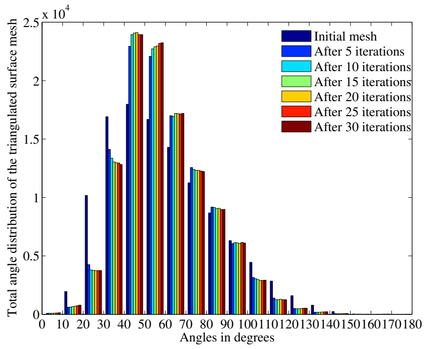
5.4.1 Effectiveness of Mesh Smoothing and Untangling

We demonstrate the improvement in mesh quality of an input surface mesh using our smoothing algorithm. To show the robustness of our method, we choose a surface which is a part of the solution of a two phase fluid-mixing simulation using the front-tracking method. The simulation was done using the front-tracking code FronTier and was performed in parallel on 64 processors. However, mesh improvement was performed on a single processor. The surface is very complicated and has a wide range of geometrical structures including regions of poor configuration. The total number of points and triangles of the mesh are 18,642 and 38,072, respectively. Figure 5.7 shows the initial mesh, a closeup of the region in the rectangle before and after smoothing using quadratic fitting. It also shows the comparison of angle distribution of the initial mesh with the redistributed mesh after every 5 iterations. The smoothing algorithm improves the overall quality of the mesh significantly. However, the algorithm introduces a small amount of triangles with very small angles (between 0 and 10) which didn't exist in the input mesh. This happens due to the application of weighted Laplacian smoothing algorithm for the first few iterations since the input mesh contains a significant number of folded triangles (about 2%) in the input mesh. Weighted Laplacian smoothing is not a good mesh quality improving algorithm and as a result introduces triangles with very small angles. Later isometric smoothing is not able to recover all the bad quality triangles. This also explains the decrease in mesh quality of the tube-like structure near the lower left of Fig. 5.7(c) and (d).

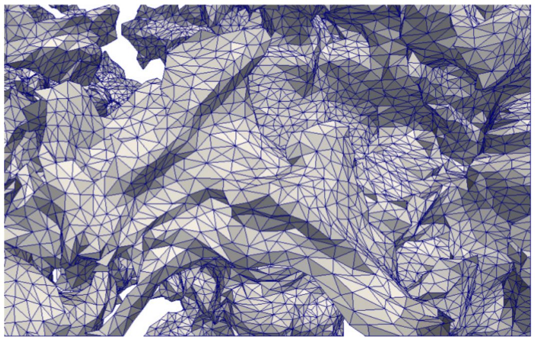
In Fig. 5.8, the droplet inside the rectangle is represented by a very coarse mesh. The Vandermonde matrices constructed at its vertices have very high condition numbers, approximately to the order of 10^6. The middle subfigure shows the droplet after one iteration of smoothing using CMF without using the geometric limiter described in 5.2.3. Clearly, the high-degree polynomial fitting introduces artificial spikes. These kind of situations are resolved by reducing the degree with the application of geometrical limiters. This can be seen in the right subfigure, which is the result after one iteration of smoothing using the geometric limiter. The effect of geometrical limiters can also be seen in the tube-like structure near the lower left of Fig. 5.7(c) and (d) and is an under-resolved region. The geometrical limiters are robust enough to allow the degree of fitting to be reduced properly and not introduce spikes. However, the volume is not conserved as the algorithm doesn't ensure global volume conservation.

The number of points where the reduction of order takes place is a very small percentage of the total number of points in the mesh. In fact, the number of such points


(a) The initial mesh



(b) Histogram of angle-distribution


(c) Before smoothing

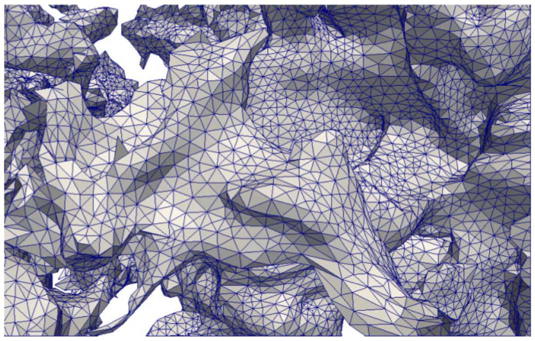

(d) After smoothing

Figure 5.7: The sub-figures (a), (c) and (d) show the initial mesh, closeup of the region inside the rectangle before and after smoothing, respectively. The smoothing algorithm is applied for 30 iterations. Sub-figure (b) shows the distribution of angles of the redistributed surface mesh.

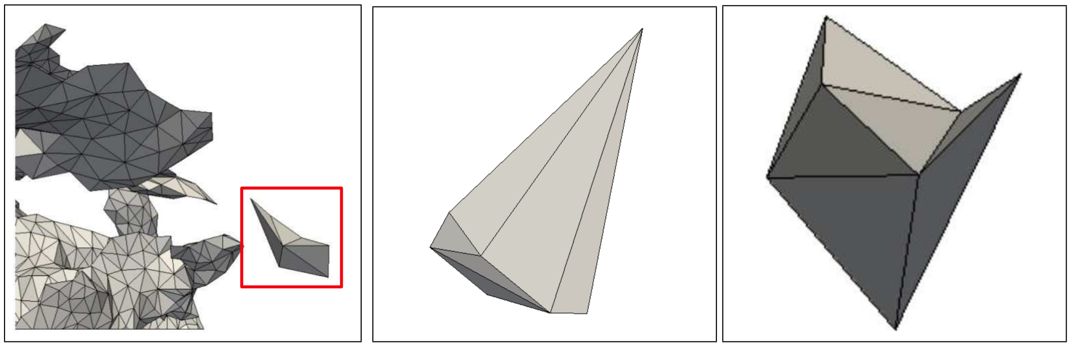


Figure 5.8: The left figure is a droplet in the initial surface mesh. The middle figure shows the configuration after applying 1 iteration of smoothing using CMF without using the geometric limiter. The right figure shows the result of smoothing when geometric limiter is used.

is reduced significantly with each iteration. For example, for the current mesh, the number of points whose degree is reduced during the first iteration is 195 points (about 1% of the total number of points), whereas for the tenth iteration that number was reduced to 74 points (about 0.4%). The distribution of such points in the mesh are illustrated in figure 5.9.

5.4.2 Effectiveness of Mesh Untangling

One strength of our proposed methodology is the ability to untangle mildly folded triangles. In the context of high-order mesh generation, we observe that occasionally, even the small perturbation of projecting vertices onto the high-order surface reconstruction can cause a few very poor-quality triangles (which may be present in the initial mesh) to fold. To demonstrate the effectiveness of mesh untangling, we construct a much more severe case: we start with a very poor-quality triangular mesh for an ellipsoid mesh generated using marching cubes, randomly perturb the vertices by up to the length of the background grid, and then project the perturbed points back onto the ellipsoid.

The initial mesh had 47 folded triangles, a few of which were high-lighted in the left image in Fig. 5.10. The combination of poor mesh quality, anisotropy of the geometry, and folded triangles makes the problem difficult to handle by ad hoc techniques. After only three iterations, our method untangled the mesh and produced a mesh with minimum angles of 18 degrees. After a few more iterations, the mesh converged to a high-quality triangulation with a minimum angle of 37.6 degrees and a maximum angle of 98.1 degrees, as shown in the right image in Fig. 5.10. In addition, the surface normal and curvatures of the resulting mesh are still accurate.

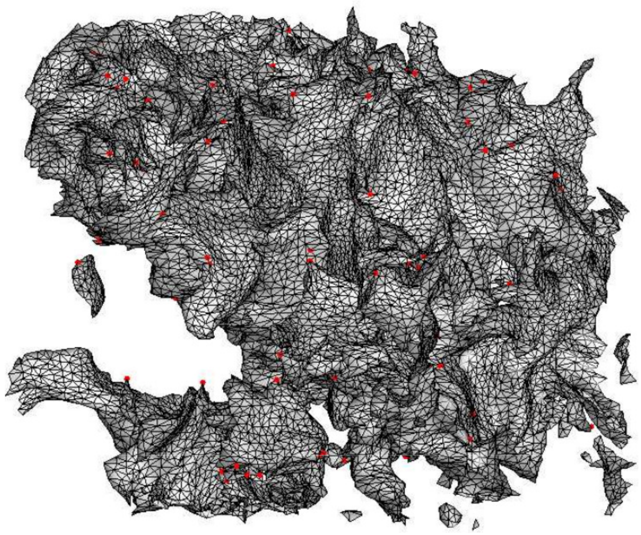
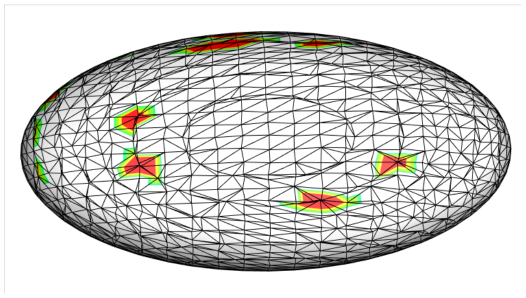


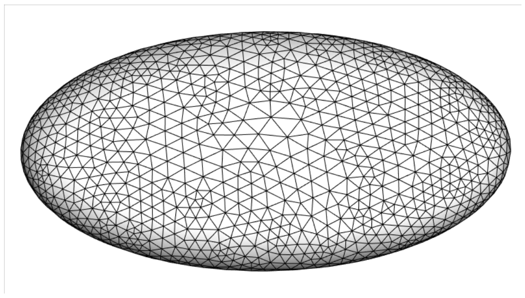
Figure 5.9: Distribution of points where the order of fitting was reduced after using the geometrical limiter.

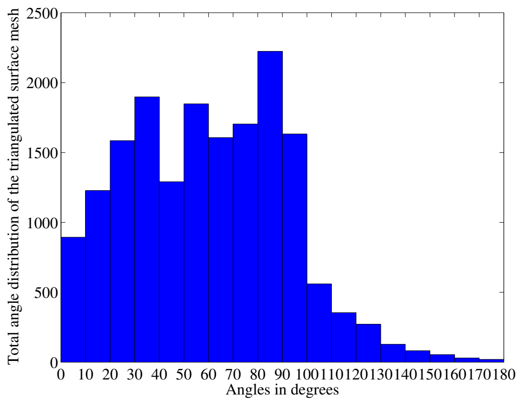
5.4.3 Effectiveness of Mesh Optimization

To demonstrate the effectiveness of our optimization technique, Fig. 5.11 shows a relatively simple example, where the mesh has 1562 vertices and 3120 triangles, both before and after mesh optimization. The initial mesh was generated from subdividing the quadrilaterals in a 40×40 logically rectangular grid in the $\theta\phi$-domain of spherical coordinate system, which result in high-valence vertices at the poles. Before optimization, the minimum and maximum angles were 3.2 and 140.6 degrees, respectively. After repeatedly performing variational mesh optimization and edge flipping, the minimum and maximum angles were 29.9 and 109.1 degrees, respectively. We used quadratic fittings in the remeshing, so that the resulting points are third-order accurate. When the normal and curvature computation algorithms in [31] are used, the normal and curvature would be second- and first-order accurate, respectively. Because of the high mesh quality and high accuracy, the resulting mesh can be used for third-order generalized finite difference methods, and also can be subdivided to generate quadratic elements for third-order finite elements methods without producing any negative Jacobians. For even higher-order methods, we can simply replace the quadratic fitting by cubic or higher order fittings in our point-projection procedure.

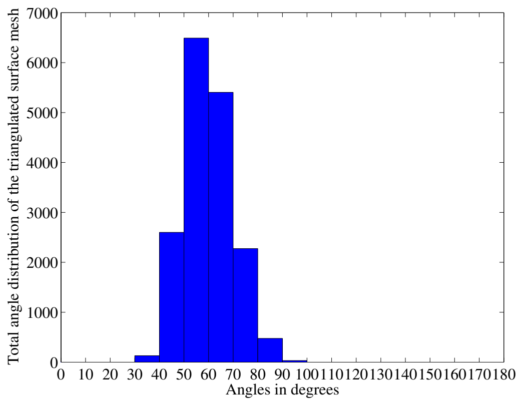
In Fig. 5.11, the mesh was optimized by the combination of these flipping strategies with variational mesh optimization, to eliminate valence-40 vertices and obtain a mesh with valences between 5 and 7. Our procedure not only improves the mesh quality (in this case, eliminating the high-valence vertices and improving the min-


(a) Initial mesh with folded triangles.


(b) Untangled and optimized mesh.



(c) Angles of initial mesh.



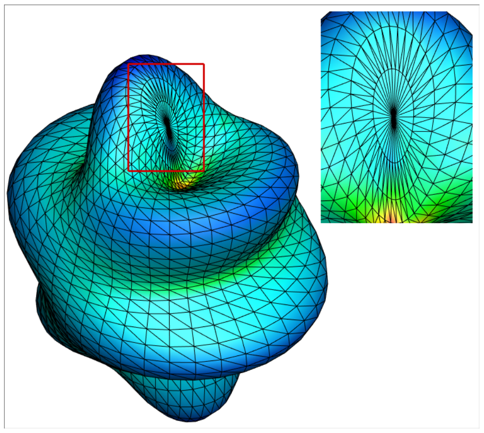
(d) Angles of optimized mesh.

Figure 5.10: Example of untangling and optimization of a poor-quality surface mesh. The initial mesh had 47 folded triangles (a few are highlighted in (a)) and a wide range of angles. After optimization, the mesh had excellent mesh quality, accurate curvatures, and no folding.

imum angle from 3.2 degrees to 29.9 degrees), but also preserves the geometry to third-order accuracy, so that both surface normals and curvatures can converge after remeshing.

5.4.4 Numerical Accuracy with Mesh Optimization

We now report the orders of convergence for computing differential quantities (normal and mean curvature) of an optimized mesh. We use a torus (with inner radius 0.7 and outer radius 1.3) and an ellipsoid (with semi-axes 1, 2, and 3) as test geometries since they can be represented analytically making it possible to compute the error with respect to exact value. For each geometry, we first generated a set of poor-quality meshes using marching cubes. These meshes were then optimized using our variational optimization, smoothing and edge flipping techniques. Finally,


(a) Initial mesh with high-valence vertices.

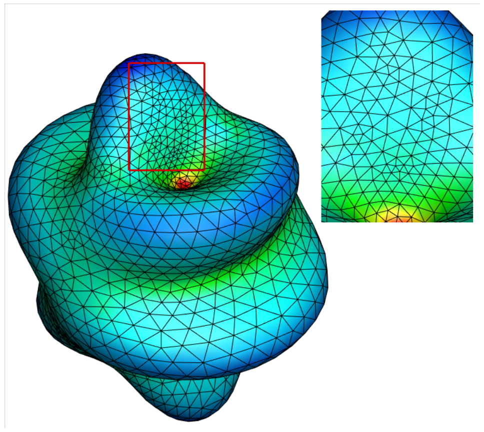

(b) Optimized mesh with high accuracy.

Figure 5.11: An example of remeshing a spherical harmonic surface to produce a high-quality mesh while preserving geometric accuracy. Images are color coded by mean curvatures. The initial mesh (left) has very high-valence vertices (as highlighted in the inlet, at the poles of the spherical coordinate system) and a minimum angle of 3.2 degrees. The optimized mesh (right) has a minimum angle of 29.9 degrees, with accurate surface normals and curvatures.

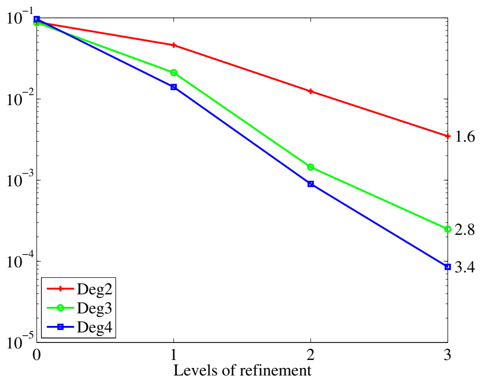
the differential quantities were computed at each vertex of the optimized meshes. For the mesh convergence study, we generated four meshes for each set of our test meshes, and numbered these meshes from the coarsest (mesh 1) to the finest (mesh 4). The average edge lengths are approximately halved between adjacent mesh resolutions. Let v be the total number of vertices. Let $\tilde{\mathbf{n}}_i$ and $\hat{\mathbf{n}}_i$ denote the computed and exact unit normal at the ith vertex, and $\tilde{k}_i$ and k_i denote the curvatures at the ith vertex, respectively. We estimate the L_∞ errors in normals and (mean or Gaussian) curvatures as

$$\begin{aligned} \text{error in normal} &= \max_i \|\tilde{\mathbf{n}}_i - \hat{\mathbf{n}}_i\|, \\ \text{error in curvature} &= \max_i |\tilde{k}_i - k_i|, \end{aligned} \tag{5.4.1}$$

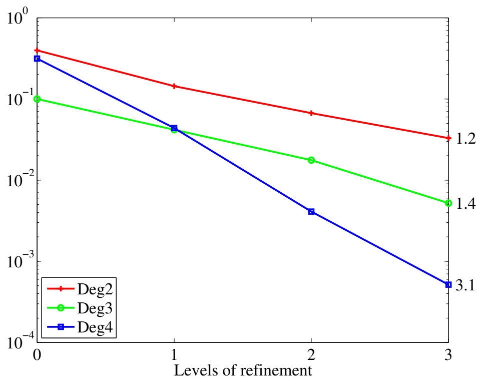
and compute the average convergence rate as

$$\text{average convergence rate} = \frac{1}{3}\log_2\left(\frac{\text{error of mesh 1}}{\text{error of mesh 4}}\right). \tag{5.4.2}$$

In Fig. 5.12, the horizontal axis corresponds to the level of mesh refinement, and the vertical axis corresponds to the L_∞ errors in logarithmic scale. In the legends, the "degree" indicates the degree of polynomial fittings used for point projection during mesh optimization. We show the average convergence rates along the right of the plots for each curve.



(a) L_∞ errors in normals.



(b) L_∞ errors in mean curvatures.

Figure 5.12: Errors and orders of convergence of normals and curvatures after mesh optimization for an ellipsoid.

Theoretically, the order of convergence of normal and mean curvature (which are first and second order differential quantities) should be d and $d-1$, respectively for WALF reconstructed surface mesh using degree d polynomial fittings. Fig. 5.12 shows that the optimized meshes preserved the accuracy of the points and as a result achieved the theoretical orders of convergence for the differential quantities.

5.4.5 Accuracy Comparison of Mesh Adaptation

For mesh adaptation, 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., [21]). Such an approach has only second order accuracy. Another approach taken by Frey [18] was to construct a G^1 continuous surface using Walton's method [50], but our experiments have shown that Walton's method is at most second order accurate despite its G^1 continuity. We compare between different point projection methods for mesh adaptation, namely,

1. Linear : Points are projected onto the linear triangle,
2. Walton's method : Points are projects onto a G^1 quartic patch [50], and
3. WALF : Points are projected onto WALF reconstructed surfaces with degrees 2, 3, and 4.

We use torus as a test geometry and compute the mean and Gaussian curvatures of the adapted mesh using the mentioned point projection methods. Tables 5.1 compares the L_∞ errors of mean curvatures and Gaussian curvatures for the adapted

Table 5.1: L_∞ errors and orders of convergence of mean curvature and Gaussian curvature on a torus after mesh adaption using surface reconstructed based on WALF and other alternatives.

Method	# of vertices	mean curvature			Gaussian curvature		
		Mesh 1	Mesh 2	Mesh 3	Mesh 1	Mesh 2	Mesh 3
Linear	L_∞ error	0.6	0.35	0.83	2.29	1.64	3.09
	Order	n/a	0.77	-1.2	n/a	0.48	-0.91
Walton's	L_∞ error	0.6	0.29	0.21	2.27	1.40	0.96
	Order	n/a	1	0.44	n/a	0.7	0.55
WALF deg 2	L_∞ error	0.6	0.27	0.11	2.29	1.36	0.52
	Order	n/a	1.1	1.3	n/a	0.74	1.4
WALF deg 3	L_∞ error	0.54	0.20	0.066	1.73	0.86	0.30
	Order	n/a	1.4	1.6	n/a	1	1.5
WALF deg 4	L_∞ error	0.45	0.085	0.0074	1.24	0.28	0.03
	Order	n/a	2.4	3.5	n/a	2.2	3.3

mesh of a torus. It is evident that the mean curvatures did not converge in L_∞ error for linear reconstruction, and barely converged for Walton's method, whereas it converged to high order for WALF. The numbers of vertices of the three meshes are 544, 1896, and 7528, respectively.

Chapter 6

Applications

In this chapter, we show applications of our framework to various applications such as computing Van der Waals Force, image-based surfaces, high-order finite element methods and solving geometric PDE's.

6.1 Computation of Van der Waals Force

We begin with an application of surface integrals to compute Van der Waals interaction forces between two microscopic bodies. The computation of Van der Waals force is of fundamental importance for understanding sintering, adhesion and fracture processes. We demonstrate the usefulness of our method (section 4.5.1) in improving the accuracy of the computed force. The Van der Waals force between two bodies of arbitrary geometry is equal to

$$\mathbf{F} = C\rho_1\rho_2 \int_{V_2}\int_{V_1} \nabla \frac{1}{s^6}\, dV_1\, dV_2 \tag{6.1.1}$$

where, ρ_i is the atomic density of body i for $i = 1,2$, C is the London–Van der Waals constant, and s is the distance between two atoms/molecules. In [55], a vector field $\mathbf{G}$ was defined such that $\nabla \cdot \mathbf{G} = C/s^6$, allowing reduction of the volume integral in Eq. (6.1.1) to a surface integral

$$\mathbf{F} = \rho_1\rho_2 \int_{S_2}\int_{S_1} (\mathbf{G}\cdot\hat{\mathbf{n}}_1)\cdot\hat{\mathbf{n}}_2\, dS_1\, dS_2, \tag{6.1.2}$$

where S_1 and S_2 are the surfaces of bodies 1 and 2, and $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ are the unit outward normal to S_1 and S_2, respectively. Specifically, $\mathbf{G} = \frac{1}{3}C\,(\mathbf{s}_1 - \mathbf{s}_2)\,/\|\mathbf{s}_1 - \mathbf{s}_1\|^6$, where $\mathbf{s}_1 \in S_1$ and $\mathbf{s}_2 \in S_2$.

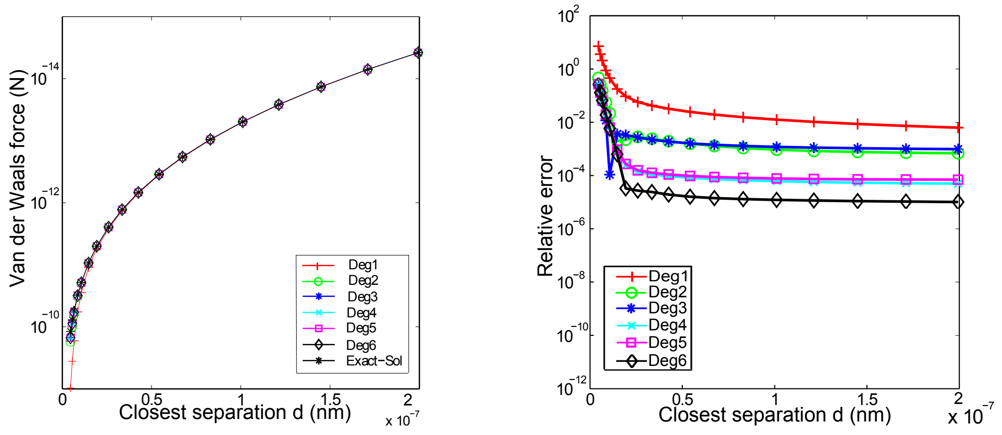


Figure 6.1: Analytical and numerical results for sphere-sphere interaction force calculation. Left: Van der Waals force at different separation distances. Right: relative errors of numerical results at different separation distances.

We assume that S_1 and S_2 are given by triangulations. Let $S_1 = \bigcup_{i=1}^{n_1} \sigma_i$ and $S_2 = \bigcup_{j=1}^{n_2} \tau_j$, where n_1 and n_2 are the numbers of triangles in S_1 and S_2, respectively. Therefore, (6.1.2) can be rewritten as

$$\mathbf{F} = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \rho_1 \rho_2 \int_{\tau_j} \int_{\sigma_i} (\mathbf{G} \cdot \hat{\mathbf{n}}_1) \cdot \hat{\mathbf{n}}_2 \, dS_1 \, dS_2, \tag{6.1.3}$$

i.e., the summation of double integrals over all pairs of triangles in S_1 and S_2. We use our algorithm to compute the double integrals using high-order reconstructions of the surface and the surface normals. Similar to [55], we show the results for the interaction force between two spherical bodies with radius $R_1 = 100$nm and $R_2 = 100$nm, and $\rho_1 = \rho_2 = 8.49 \times 10^{28}/\text{m}^3$ and $C = 4.5639 \times 10^{-78}\text{Jm}^6$.

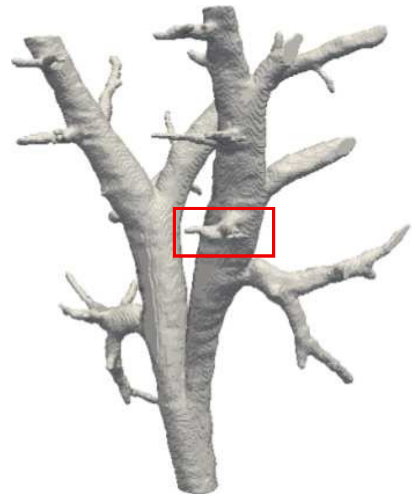
We calculate the Van der Waals force for 20 different separations, which are non-uniformly distributed from 5nm to 200nm. Figure 6.1 (left) shows the computed Van der Waals force for polynomial fittings of degrees from 1 to 6, along with the exact solutions. Figure 6.1 (right) shows the relative errors for different degrees. From these results, it is evident that the computed forces are more accurate with higher-degree fittings, and high-order reconstructions can significantly improve the accuracy compared with piecewise linear approximations.

6.2 Smoothing Biomedical Geometries from Medical Image Data

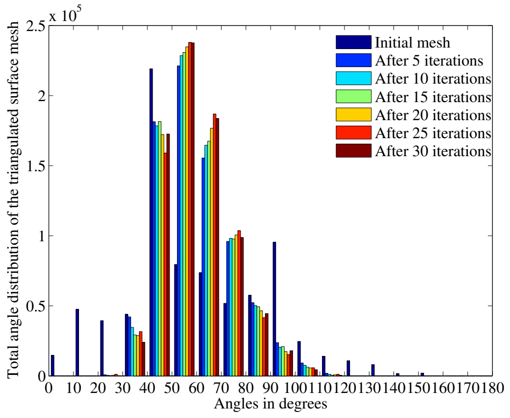
In this section, we apply our method to surface meshes of complex biomedical geometries extracted from medical images. As an example, we focus on a lung mesh to be used in numerical simulations of respiration, particle deposition and volatile gas metabolism. Surface meshes extracted from images tend to have stair-stepped features, as marching cubes places edges on the faces of image voxels. This jaggedness makes it unsuitable for the development of a volume mesh to be used in numerical simulations. Thus a smoothing algorithm which improves the quality of the mesh while maintaining the geometrical accuracy is highly desirable.

We use an extracted surface mesh of a rat lung image as an example. All animal work was done in accordance with a protocol approved by the Institutional Animal Care and Use Committee (IACUC) of Pacific Northwest National Laboratory. To acquire an image at total lung capacity (TLC), the rat (a male Sprague-Dawley rat approximately weighing 320 g) was euthanized by CO2 asphyxiation, then orally intubated with a 14-gauge catheter tube. The lungs was inflated to $\sim$25 cmH2O and held at that pressure with a constant supply of air regulated by a water column. The lung was imaged with micro-CT (GE eXplore CT120) using the following settings: 90 kVp, 40 mA, 16-ms exposure time, 900 projections over 360 degrees gantry rotation, and 2x2 binning. The gantry rotates at a rate of $\sim$0.019 rad/s, so total imaging time was about 5.5 min; animal positioning and scanner set-up required an additional $\sim$3 min. The image was reconstructed to 50 μm isotropic resolution. The airways of the TLC image were then semi-automatically segmented using intensity threshold-based approaches described in [9] in conjunction with interactive segmentation using Digital Data Viewer (DDV) (`http://cgc-code.org/`). A 3D median filter was applied to the segmented TLC airway data to improve surface boundary continuity. The surface mesh was then extracted using marching cubes.

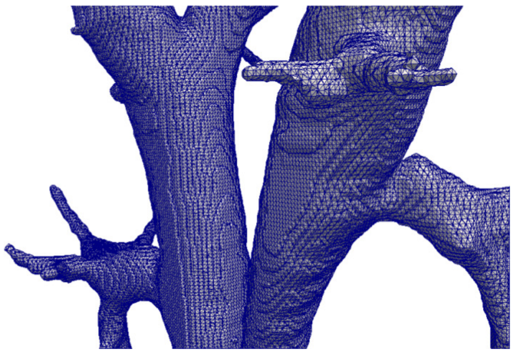
Figure 6.2 shows a surface plot of the initial mesh along with a close up of a branch (inside the rectangle) before and after 30 iterations of smoothing. As can be seen in Figure 6.2, the the stair-step features of the mesh are clearly attenuated, though not eliminated, and the triangle quality is significantly improved. We are currently investigating a method wherein the stencil size is locally adapted to the features of the geometry in an effort to achieve a smooth result without sacrificing geometric accuracy.



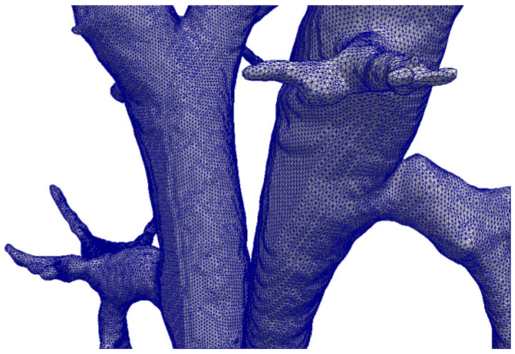
(a) Initial mesh



(b) Histogram of angle distribution



(c) Before smoothing



(d) After smoothing

Figure 6.2: The top-left figure shows a surface plot of the initial mesh. The bottom-left and bottom-right figures show the closeup of the portion inside the rectangle before and after 30 iterations of mesh smoothing. The closeup of the portion inside the red rectangle demonstrates the improved quality and smoothness of surface mesh after smoothing. The top-right shows the distribution of the angles of the triangulated surface mesh.

6.3 Improving High-Order Finite Elements

We apply our method to improving the quality of a given high-order (generally quadratic or cubic) isoparametric finite element mesh using the methods described in Sections 2.2 and 5.2. It is well known that the quality of a solution from the finite element method is highly dependent on the quality of the mesh. In general, any approximation of spatial derivatives of the physical domain requires a well defined diffeomorphism, which maps the reference element to a physical element. Poor-quality elements, especially high-order isoparametric elements, may violate this requirement, thereby compromising the efficiency of a numerical solver and the accuracy of the generated solution.

Our mesh-improvement algorithm works as follows: given a preliminary isoparametric surface mesh with possibly tangled elements, we first restrict the high-order elements down to the underlying linear elements, whose nodes are the corner vertices for each high-order element. We then optimize the linear mesh using the mesh smoothing algorithms described in section 5.2 for a total of 50 iterations. The elements of the resulting linear mesh are then enriched with the additional new nodes along the edges and interior of the element, by projecting these nodes onto a high-order reconstruction of the surface using WALF as described in [29].

Figure 6.3 demonstrates this mesh-improvement procedure with a torus mesh, which was initially obtained from marching cubes. The mesh contained $17,064$ vertices and $3,792$ quadratic 6-node triangles both before and after smoothing. To measure the mesh quality of isoparametric elements, we adopt the shape quality measure as defined in [19], which measures the deviation of an element from an equilateral planar triangle. However, in contrast to [19], we do not refer to an existing CAD model, but compute directly on the discrete geometry and identify inverted elements by computing approximate outward normals. Figure 6.3(c) and (d) show the shape quality measures of the mesh before and after improvement. It is clear that the element quality was greatly improved. Our high-order reconstruction techniques can potentially be coupled other techniques (such as those in [19, 32]) for optimizing isoparametric elements directly, which would move vertices tangentially along the surface. However, such a complicated method was not necessary in our experiments, since our mesh improvement delivered sufficiently good linear elements to ensure high-quality isoparametric elements.

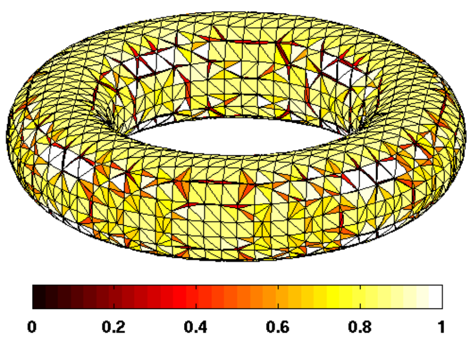
6.4 Geometric PDEs

We finally describe a usage of our techniques in the numerical solutions of geometric partial differential equations. Such problems appear in various applications, such as surface smoothing in computer-aided design [54] and the modeling of moving surfaces of materials [7]. As an example, we consider the solution of the mean-curvature flow over triangulated surfaces. The continuum formulations of these problems are as follows. Given a moving surface Γ, the coordinates $\mathbf{x}$ of points on Γ are functions of time t as well as some surface parametrization $\mathbf{u} = (u, v)$, which can be local instead of global parametrizations. Assume the surface is differentiable.

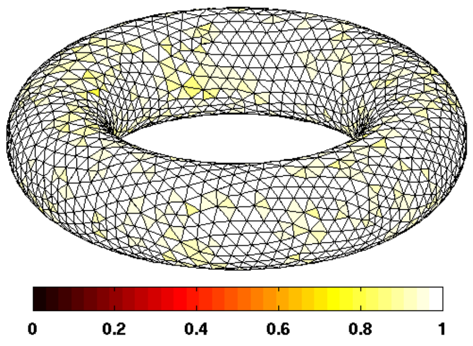
The *mean-curvature flow* is a second-order nonlinear PDE modeling the motion of the surface driven by the mean curvature, given by

$$\frac{\partial \mathbf{x}}{\partial t} = M\hat{\mathbf{n}}, \tag{6.4.1}$$

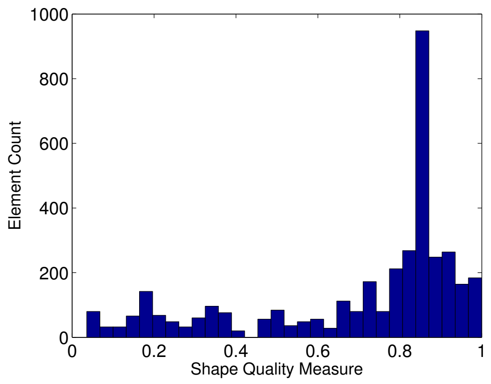
where M denotes the mean curvature and $\hat{\mathbf{n}}$ denotes the unit normal vector. The vector $\hat{\mathbf{n}}$ involves first-order partial derivatives of $\mathbf{x}$ with respect to the parameters $\mathbf{u}$, whereas M involves second-order partial derivatives of $\mathbf{x}$ with respect to $\mathbf{u}$. We discretize the problem in space using the generalized finite difference scheme, and discretize the equation in time using a semi-implicit scheme by evaluating the second-order terms over the new time step and evaluating the first-order terms over the current time step. As the surface evolves, the mesh may need to be adapted in order to maintain good spacing between the points. Utilizing adaptivity during evolution can help maintain mesh quality and ultimately increase the stability when trying to further evolve the mesh. Fig. 6.4 shows a comparison with and without mesh adaption for the evolution of an ellipsoid with semi-axes 1.5, 2, and 8. Without mesh improvement, the points become overly crowded at the top of the ellipsoid, which can severely undermine the time step requirement for the PDE solver. We optimize and adapt the mesh using our technique, so that the mesh quality and the order of accuracy are achieved simultaneously.



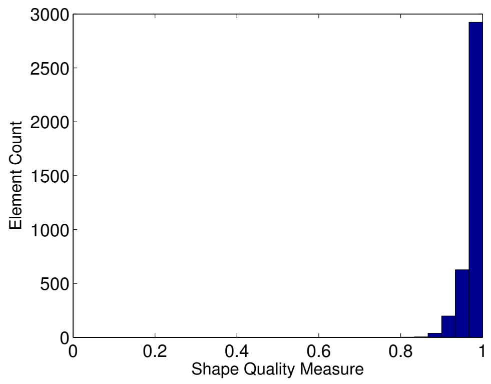
(a) Poor Quality Initial Mesh



(b) Improved Quality Optimized Mesh



(c) Initial element shape quality measures



(d) Improved element shape quality measures

Figure 6.3: Isoparametric (quadratic) element shape quality for a spherical harmonic surface (a,c) before and (b,d) after application of the mesh smoothing algorithm. Colors indicate the quality measure of each isoparametric element. Near-singular or folded elements are represented in black while near-ideal elements are colored white.

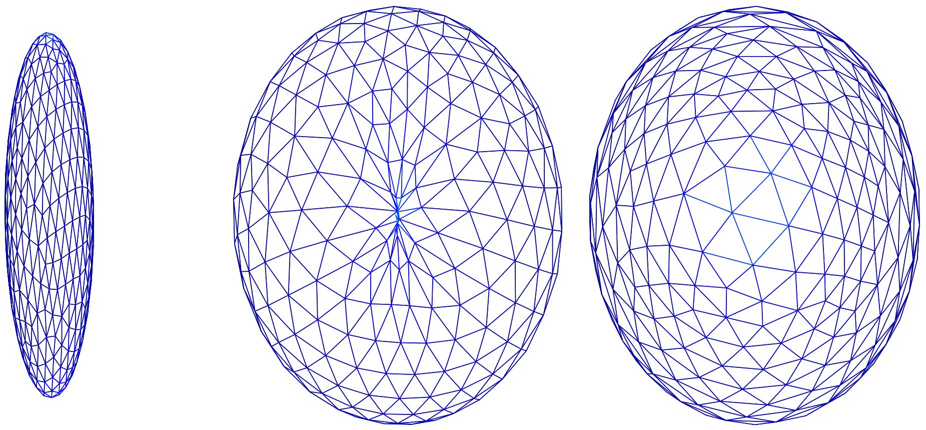


Figure 6.4: Evolution of an initial mesh (left) of an ellipsoid under mean curvature flow. The center and the right images show the top of the surface meshes after 0.14 seconds of evolution without and with mesh adaption, respectively.

Chapter 7

Conclusions and Future Work

In this dissertation, we have considered the problem of high-order approximations over discrete surfaces and their use in various mesh-based computations as well as in a few applications. We extended the underlying computational framework of high-order surface reconstruction to a function defined on the surface. This allowed us to compute high-order approximation of surface integrals over discrete meshes. We analyzed the problem theoretically along with numerical verification. We also integrated various mesh-based operations with high-order surface reconstruction to preserve geometrical accuracy, and develop algorithms for mesh smoothing, optimization, and adaptation. We improved the robustness of high-order reconstructions to allow remeshing of under-resolved meshes by introducing geometric limiters and untangling mildly folded triangles. All of our algorithms were developed on top a general, efficient and simple mesh data structure which was developed for general meshes with support for non-manifold features and mixed-dimensional entities. Finally, we show application of our methods to a variety of applications, specifically to Van der Waals force computation, geometric PDEs, high-order finite elements, complex meshes in fluid-mixing simultations and image-based biomedical surface meshes.

Our framework have a number of advantages. First of all, a least-squares based approach over interpolation allows greater flexibility and stability without any loss of accuracy. Secondly, it is based on local parameterization. Therefore, we do not require the input surface mesh to be globally parameterizable. Thirdly, it is independent of a CAD model representation. Thus it can be used in engineering applications such as complex fluid-mixing simulations or image-based surface mesh analysis where there is no CAD representation available. Finally, the basic formulation of the framework makes it largely independent of the mesh quality. That is, it does not depend on mesh quality measures such as angles, Jacobian, etc. except for degenerate arrangement of points. Because of this property, we can get high-order even for poor-quality meshes.

This work presents a number of lines for future work. First of all, the least-squares based framework for local polynomial fittings is based on the regularity assumption of the function to be approximated. An immediate question is how to bring together singularities and discontinuities as well as approximation issues associated with under-resolved regions of the mesh into the framework. For example, we have not considered singularities in the computation of surface integrals, which are very important for some numerical methods such as boundary integral methods. These are challenging problems and need to be studied more rigorously both theoretically and numerically.

Secondly, the current computational framework for high-order reconstruction computes a C^0 continuous support over the mesh. However, many applications do require high-order of smoothness of a variety of purposes. Though we observe from our numerical experiments that the increased level of smoothness does not necessarily result in accurate or even converging differential quantities, more theoretical and numerical studies should be done to draw some conclusions. Finally, our algorithms could be integrated with other numerical methods such as finite element methods, embedded boundary methods and conservative front tracking for moving interfaces, and be applied to applications such as fluid dynamics and fluid-structure interactions.

Bibliography

[1] T. Alumbaugh and X. Jiao. Compact array-based mesh data structures. In *Proceedings of 14th International Meshing Roundtable*, pages 485–504, 2005.

[2] M. W. Beall and M. S. Shephard. A general topology-based mesh data structure. *Int. J. Numer. Meth. Engrg.*, 40:1573–1596, 1997.

[3] 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.

[4] J. J. Benito, F. Ureña, and L. Gavete. Solving parabolic and hyperbolic equations by the generalized finite difference method. *Journal of Computational and Applied Mathematics*, 209(2):208–233, 2007.

[5] B. S. Bischoff, M. Botsch, S. Steinberg, S. Bischoff, L. Kobbelt, and R. Aachen. OpenMesh – a generic and efficient polygon mesh data structure. In *In OpenSG Symposium*, 2002.

[6] D. K. Blandford, G. E. Blelloch, D. E. Cardoze, and C. Kadow. Compact representations of simplicial meshes in two and three dimensions. In *Proceedings of 12th International Meshing Roundtable*, pages 135–146, 2003.

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

[8] D. Canino, L. D. Floriani, and K. Weiss. An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions. *Computer & Graphics Proc. of SMI Conf.*, 35:747–753, 2011.

[9] J. P. Carson, D. R. Einstein, K. R. Minard, M. V. Fanucchi, C. D. Wallis, and R. A. Corley. High-resolution lung airway cast segmentation with proper topology suitable for computational fluid dynamic simulations. *Comp. Med. Imag. Graph.*, 34(7):572–578, 2010.

[10] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Comput. Aid. Geom. Des.*, 22(2):121–146, 2005.

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

[12] B. Clark, N. Ray, and X. Jiao. Surface mesh optimization, adaption, and untangling with high-order accuracy. In *Proceedings of 21st International Meshing Roundtable*, San Jose, CA, 2012.

[13] R. Cools. An encyclopaedia of cubature formulas. *Journal of Complexity*, 19:445–453, 2003. Online database available at http://www.cs.kuleuven.ac.be/∼nines/research/ecf/ecf.html.

[14] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[15] V. Dyedov, D. R. Einstein, X. Jiao, A. P. Kuprat, J. P. Carson, and F. del Pin. Variational generation of prismatic boundary-layer meshes for biomedical computing. *International Journal for Numerical Methods in Engineering*, 79:907–945, 2009.

[16] V. Dyedov, N. Ray, D. Einstein, X. Jiao, and T. Tautges. AHF: Array-based half-facet data structure for mixed-dimensional and non-manifold meshes. In *Proceedings of 22nd International Meshing Roundtable*, Orlando, FL, October 2013.

[17] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30:1167–1202, 2000. Special Issue on Discrete Algorithm Engineering.

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

[19] A. Gargallo-Peiró, X. Roca, J. Peraire, and J. Sarrate. Defining quality measures for mesh optimization on parameterized cad surfaces. In *In Proceedings of the 21st International Meshing Roundtable*, pages 85–102, 2012.

[20] R. V. Garimella. MSTK – a flexible infrastructure library for developing mesh based applications. In *Proceedings of 13th International Meshing Roundtable*, pages 213–220, 2004.

[21] R. V. Garimella, M. J. Shashkov, and P. M. Knupp. Triangular and quadrilateral surface mesh quality optimization using local parametrization. *Comput. Meth. Appl. Mech. Engrg.*, 193:913–928, 2004.

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

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

[24] C. Gooch. Grummp version 0.5.0 users guide.

[25] B. M. Irons and S. Ahmad. *Techniques of Finite Elements*. Ellis Horwood Ltd, Chichester, UK, 1980.

[26] X. Jiao. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys.*, 220:612–625, 2007.

[27] X. Jiao, A. Colombi, X. Ni, and J. Hart. Anisotropic mesh adaptation for evolving triangulated surfaces. *Engrg. Comput.*, 26:363–376, 2010.

[28] X. Jiao and D. Wang. Reconstructing High-Order Surfaces for Meshing. In S. Shontz, editor, *Proceedings of the 19th International Meshing Roundtable*, pages 143–160. Springer Berlin Heidelberg, 2010.

[29] X. Jiao and D. Wang. Reconstructing high-order surfaces for meshing. *Engineering with Computers*, 28:361–373, 2012.

[30] X. Jiao, D. Wang, and H. Zha. Simple and effective variational optimization of surface and volume triangulations. *Engineering with Computers*, 27:81–94, 2011.

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

[32] A. Johnen, J.-F. Remacle, and C. Geuzaine. Geometric validity of curvilinear finite elements. *Journal of Computational Physics*, 233:359–372, 2013.

[33] F. Juretic and A. D. Gossman. Error analysis of the finite volume method with respect to mesh type. *Numerical Heat Transfer, Part B*, 57:414–439, 2010.

[34] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theo. Appl.*, 13:65–90, 1999.

[35] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. libMesh: A c++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22:237–254, 2006.

[36] M. Kremer, D. Bommes, and L. Kobbelt. OpenVolumeMesh – a versatile index based data structure for 3D polytopal complexes. *Proceedings of 21st International Meshing Roundtable*, pages 531–548, 2012.

[37] A. Kuprat, A. Khamayseh, D. George, and L. Larkey. Volume conserving smoothing for piecewise linear curves, surfaces, and triple lines. *J. Comput. Phys.*, 172:99–118, 2001.

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

[39] 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.

[40] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Trans. Graph.*, 12(1):56–102, Jan. 1993.

[41] D. Poirier, S. R. Allmaras, D. R. McCarthy, M. F. Smith, and F. Y. Enomoto. The CGNS system, 1998. AIAA Paper 98-3007.

[42] F. U. Prieto, J. J. B. Munoz, and L. G. Corvinos. Application of the generalized finite difference method to solve the advection-diffusion equation. *Journal of Computational and Applied Mathematics*, 235(7):1849 – 1855, 2011.

[43] N. Ray, T. Delaney, D. Einstein, and X. Jiao. Surface remeshing with robust high-order reconstruction. In *Engineering with Computers*, In press.

[44] N. Ray, D. Wang, X. Jiao, and J. Glimm. High-order numerical integration over discrete surfaces. *SIAM Journal on Numerical Analysis*, 50:3061–3083, 2012.

[45] E. S. Seol. *FMDB: Flexible Distributed Mesh Database For Parallel Automated Adaptive Analysis*. PhD thesis, Rensselaer Polytechnic Institute, 2005.

[46] H. Si. TetGen, a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator v1.4, 2006.

[47] D. Sieger and M. Botsch. Design, implementation and evaluation of the surface mesh data structure. In *In Proceedings of the 20th International Meshing Roundtable*, 2011.

[48] T. Tautges, R. Meyers, and K. Merkley. MOAB: A mesh-oriented database. Technical report, Sandia National Laboratories, 2004.

[49] The CGNS Steering Sub-committee. *The CFD General Notation System Standard Interface Data Structures*. AIAA, 2002.

[50] D. Walton. A triangular g1 patch from boundary curves. *Comput. Aid. Des.*, 28(2):113–123, 1996.

[51] D. Wang. Numerical differential geometry and its applications. 2011.

[52] D. Wang, B. L. Clark, and X. Jiao. An analysis and comparison of parameterization-based computation of differential quantities for discrete surfaces. *Comput. Aid. Geom. Des.*, 26:510–527, 2009.

[53] 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.

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

[55] P. Yang and X. Qian. A general accurate procedure for calculating molecular interaction force. *Journal of Colloid and Interface Science*, 337:594–605, 2009.

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