Accurate, Semi-Implicit Methods with Mesh Adaptivity for Mean Curvature and Surface Diffusion Flows Using Triangulated Surfaces

A Dissertation Presented
by

Bryan L. Clark

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

(Computational Applied Mathematics)

Stony Brook University

May 2012

Stony Brook University

The Graduate School

Bryan L. Clark

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

Dr. Xiangmin Jiao - Dissertation Adviser
Associate Professor, Applied Mathematics and Statistics

Dr. James Glimm - Chairperson of Defense
Distinguished Professor, Applied Mathematics and Statistics

Dr. Roman Samulyak - Committee Member
Associate Professor, Applied Mathematics and Statistics

Dr. Hong Qin - Outside Committee Member
Professor, Computer Science

This dissertation is accepted by the Graduate School

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Accurate, Semi-Implicit Methods with Mesh Adaptivity for Mean Curvature Flow and Surface Diffusion Using Triangulated Surfaces

by
Bryan L. Clark

Doctor of Philosophy
in
Applied Mathematics and Statistics
(Computational Applied Mathematics)

Stony Brook University
2012

Geometric partial differential equations (PDEs), such as mean-curvature flow and surface diffusion flow, are challenging to solve numerically due to their strong non-linearity and stiffness, when solved explicitly. Solving these high-order PDEs using explicit methods require very small time-steps to achieve stability, whereas using implicit methods result in complex nonlinear systems of equations that are expensive to solve. In addition, accurate spatial discretizations of these equations pose challenges in their own rights, especially on triangulated surfaces. We propose new methods for mean curvature flow and surface diffusion flow using triangulated surfaces. Our methods use a weighted least-squares approximation for improved accuracy and stability, and semi-implicit schemes for time integration for larger time-steps and higher efficiency. If mesh element quality is initially poor, or becomes poor through evolution under mean curvature flow or surface diffusion flow, we utilize mesh adaptivity to improve mesh quality and proceed further in evolution. Numerical experiments and comparisons demonstrate that our methods can achieve second-order convergence in errors for both mean-curvature flow and surface diffusion flow, with better accuracy and stability than both explicit schemes and alternative methods.

Contents

List of Figures

List of Tables

Acknowledgments

Special thanks to Dr. Xiangmin Jiao for his overall assistance and direction with this dissertation, Duo Wang for his contributions to the semi-implicit mean curvature flow and semi-implicit surface diffusion algorithms, and Tristan Delaney for his implementation of the finite element method for surface diffusion by Bänsch et al. [4]. Additionally, I would like to thank my family for their genuine belief, support, and encouragement, and my fiancee, Colleen, for being by my side every step of the way.

1 Introduction

Geometric partial differential equations (PDEs) on moving surfaces occur in various applications [15]. They are used in materials science, where strength and properties of materials require the mathematical modeling of the morphology of microstructure. Also, in evolving surfaces, such as grain boundaries, which separate differing orientations of the same crystalline phase. Another example is in image processing, to identify a dark shape in a light background in a 2D image, where a contour is evolved that wraps around the shape. As well as for surface smoothing in computer-aided design [70] and the modeling of moving surfaces of materials [10].

These geometric PDEs are challenging to solve, as they are often very nonlinear due to the presence of high-order geometric differential operators of a moving surface, and they are stiff in terms of the time-step constraints. The moving surface may be represented either implicitly (i.e. as the zero set of a field, such as in the level-set method), or be represented explicitly as a triangulated surface. For the latter case, the accurate discretization of the geometric differential operators pose a challenge in their own rights.

In this paper, we investigate the accurate and stable discretizations of two representative geometric PDEs, namely the mean-curvature flow and the surface diffusion flow, over triangulated surfaces. Please note that surface diffusion flow should not be confused with the general process involving the motion of molecules, and atomic clusters at solid material surfaces, also known as surface diffusion. Rather, surface diffusion flow is a nonlinear PDE modeling the motion of a surface driven by the surface Laplacian of the mean curvature [10, 2, 3, 5, 4, 16, 50]. With this understanding, we will move forward with the understanding that, in this paper, when referring to surface diffusion flow or surface diffusion, we are *always* talking about the nonlinear PDE for modeling the motion of a surface.

The continuum formulations of these problems are as follows. Given a moving surface Γ, the coordinates $\boldsymbol{x}$ of points on Γ are functions of time t as well as some surface parametrization $\boldsymbol{u} = (u, v)$, which can be local instead of global parametrization. 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

$$\frac{\partial \boldsymbol{x}}{\partial t} = M\hat{\boldsymbol{n}}, \tag{1.1}$$

where M denotes the mean curvature and $\hat{\boldsymbol{n}}$ denotes the unit normal vector. The vector $\hat{\boldsymbol{n}}$ involves first-order partial derivatives of $\boldsymbol{x}$ with respect to the parameters $\boldsymbol{u}$, whereas M involves second-order partial derivatives of $\boldsymbol{x}$ with respect to $\boldsymbol{u}$. This equation is analogous to the parabolic equations (such as the heat equation) in terms of its stiff time-step constraints for explicit schemes. If solved explicitly, the time-step must be second order to the minimum edge length of a triangulated surface.

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

$$\frac{\partial \boldsymbol{x}}{\partial t} = (\triangle_\Gamma M)\hat{\boldsymbol{n}}, \tag{1.2}$$

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

2 Literature Review

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

2.1 Geometric PDEs with Explicit Surfaces

Bänsch et al. [3] investigated the numerical solutions of surface diffusion on graphs. Later, Bänsch et al. [4] proposed a mixed finite element method (FEM) for solving the surface diffusion equation governed by the surface Laplacian of the mean curvature. Their method uses a semi-implicit time discretization to split the fourth order, highly nonlinear geometric PDE into four linear (up to second order) elliptic equations involving both scalar and vector forms of the curvature and velocity of the discrete surface. The discretization merely requires piecewise-linear elements which can be used to enforce a weak formulation of the system of PDEs which simplifies implementation. The resulting linear algebraic system of equations can be solved using a Schur complement approach. This allows the main system of equations to be reduced significantly. Effectively, only the velocities require solving, where the curvatures can be calculated separately if needed.

Xu and Zhang [70] provided a systematic framework to solve a class of geometric partial differential equations. The first-order derivatives, second-order partial derivatives, surface gradient, divergence and Laplacian are expressed as a linear combination of the vertex coordinates, thus a linear system is formed when the overall differential operator is approximated as a linear combination of the next step vertex coordinate. The quadratic polynomial is used to fit the local surface.

Deckelnick et al. [16] analyzes a fully discrete numerical scheme for approximating the evolution of graphs for surfaces evolving by anisotropic surface diffusion. The scheme uses second order operator splitting for the nonlinear geometric fourth order equation, which produces two coupled spatially second order problems, which are then approximated using linear finite elements. With the time discretization being semi-implicit, they are able to prove error bounds and numerically test calculations that confirm at least first-order convergence in errors.

2.2 Geometric PDEs with Implicit Surfaces

Osher and Sethian [53] devised an algorithm for front propagation with curvature-dependent speed, the equation is treated as a Hamilton-Jacobi equation with a viscosity term and is numerically solved using techniques from hyperbolic conservation laws. While Chopp and Sethian [12] solved the surface diffusion equation, with the fourth order operator calculated through a hybrid narrow band approach near the interface. Methods based on level set can easily handle singularities during propagation and a non-smooth initial front, but no accuracy analysis is given. Additionally, the time-step for these explicit methods has to be very small to ensure stability.

Smereka [59] introduced a semi-implicit scheme for mean curvature flow based on the level set framework. The idea is to separate the linear term from the mean curvature expression and apply the implicit scheme, while the nonlinear term is evaluated at the current time-step. The finite difference formula is used on a higher dimensional regular grid to approximate the nonlinear differential operator.

Glimm et al. [31, 32, 34, 28, 29, 30, 33] developed a front tracking method, for the propagation of a moving interface. Front tracking works by moving marker particles which represent the interface. These particles are located only on the interface and are connected to each other to form piecewise linear segments (2D) or a triangulated mesh (3D). It is significantly faster than other particle methods, since fewer particles are used per cell than in typical particle method simulations.

Deckelnick et al. [15] describes the mathematical formulations for mean curvature flow and surface diffusion flow using a parametric approach, a level set method and a phase field method, as well as each methods advantages and disadvantages. For each approach, first-order error convergence results are numerically calculated.

2.3 Mesh Adaptivity

An ideal mesh has elements with similar size and shape. For a triangulated mesh, this means that all elements are close to equilateral triangles and are consistent in size. When not presented with such a mesh, mesh adaptivity can be used to reform the mesh into a more ideal form. Mesh adaptivity is the process of taking a mesh of "poor" quality elements and adjusting the vertex positions and element assignments as to optimize the mesh for calculation purposes. In mean curvature flow and surface diffusion flow, this helps greatly in maintaining stability and accuracy as the surface evolves.

Jiao et al. [43] devised parameters and techniques to improve mesh quality through:

Vertex redistribution - The relocation of vertices along the surface to improve overall mesh quality.

Edge flipping - The reassignment of edges to more optimal pairings for element quality.

Edge contraction - The deletion of edges to remove poor-quality or “small” elements.

Edge splitting - The insertion of edges to improve mesh element quality and split “large” elements.

Bänsch et al. [4] outlines several adaptivity schemes to improve mesh quality and avoid defects associated with clustered vertices and singularities. These include:

Mesh regularization - Redistributing vertices in a volume-preserving manner to ensure that supports of basis functions are of nearly equal size.

Time adaptivity - Translation of nodes of lengths larger than the local mesh-size can create unwanted mesh distortion or node-crossings. Time adaptivity allows for appropriate step size such that mesh distortions are prevented.

Spatial adaptivity - Spatial adaptivity allows for ease of computation by decreasing the number of nodes in smooth regions while providing higher resolution of sharp edges or corners.

Angle-width control - Surfaces of disparate aspect ratios often develop singularities or “pinch-offs” when evolved under surface diffusion. Near singularities, elements become extremely degenerate by developing extremely large angles. Angle width control simply bisects elements with these large angles to provide better accuracy near singularities.

3 Preliminaries

In this section, we give the continuous formulae for normal and mean curvature for a parametric surface, both under global and local coordinate system. For a comprehensive list of differential quantities formulae and their derivations, see [65]. Assume the surface is parametrized as $\boldsymbol{x}(u,v) = [x_1(u,v), x_2(u,v), x_3(u,v)]$, here a local parametrization uv around each vertex suffices since normal and curvature are local property of the surface. In section 3.9, we describe how to locally parametrize a discrete surface and calculate differential quantities at each vertex.

3.1 Formulae Under Global Coordinate System

Let $\boldsymbol{x}$ be the global coordinate values of the surface vertices, the Jacobian matrix of the surface is,

$$\boldsymbol{J} = \nabla_{\boldsymbol{u}} \boldsymbol{x} = [\boldsymbol{x}_u \,|\boldsymbol{x}_v] = \begin{bmatrix} \frac{\partial x_1}{\partial u} & \frac{\partial x_1}{\partial v} \\ \frac{\partial x_2}{\partial u} & \frac{\partial x_2}{\partial v} \\ \frac{\partial x_3}{\partial u} & \frac{\partial x_3}{\partial v} \end{bmatrix} \tag{3.1}$$

Let $\ell = \|\boldsymbol{x}_u \times \boldsymbol{x}_v\|_2$, the first fundamental form $\boldsymbol{G} = \boldsymbol{J}^T \boldsymbol{J}$ and the second fundamental form $\boldsymbol{B} = \begin{bmatrix} \hat{\boldsymbol{n}}^T\boldsymbol{x}_{uu} & \hat{\boldsymbol{n}}^T\boldsymbol{x}_{uv} \\ \hat{\boldsymbol{n}}^T\boldsymbol{x}_{uv} & \hat{\boldsymbol{n}}^T\boldsymbol{x}_{vv} \end{bmatrix}$, the surface normal and mean curvature are,

$$\boldsymbol{n} = \frac{1}{\ell}(\boldsymbol{J}_{:,1} \times \boldsymbol{J}_{:,2}) \tag{3.2}$$

$$M = \frac{1}{2}\text{tr}(\boldsymbol{G}^{-1}\boldsymbol{B}) \tag{3.3}$$

3.2 Formulae Under Local Coordinate System

Under local coordinate system for each vertex, the formulae can be simplified, the Jacobian matrix now is,

$$\boldsymbol{J} = \nabla_{\boldsymbol{u}}\boldsymbol{x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \end{bmatrix} \tag{3.4}$$

Let $\ell = \|\boldsymbol{x}_u \times \boldsymbol{x}_v\|_2 = \sqrt{1 + f_u^2 + f_v^2}$ and the Hessian matrix $\boldsymbol{H} = \begin{bmatrix} f_{uu} & f_{uv} \\ f_{vu} & f_{vv} \end{bmatrix}$, the local normal and curvature are simplified to,

$$\boldsymbol{n} = \frac{1}{\ell}\begin{bmatrix} -f_u \\ -f_v \\ 1 \end{bmatrix} \tag{3.5}$$

$$M = \frac{\text{tr}(\boldsymbol{H})}{2\ell} - \frac{(\nabla f)^T\boldsymbol{H}(\nabla f)}{2\ell^3} \tag{3.6}$$

Compared to the formula under global coordinate system, the mean curvature here has no normal $\boldsymbol{n}$ in it, this separation of normal and curvature is useful when we derive the semi-implicit scheme in section 5.1.

3.3 Reduction to Curves

For curve, there is only one parameter u, the normal and curvature under local coordinate system are,

$$\boldsymbol{n} = \frac{1}{\ell}\begin{bmatrix} -f_u \\ 1 \end{bmatrix} \tag{3.7}$$

$$\kappa = \frac{f_{uu}}{2\ell^3} \tag{3.8}$$

where $\ell = \sqrt{1 + f_u^2}$

3.4 Derivatives of Area Element

The derivatives of the area element ℓ occur frequently in the spatial discretization. Since $\ell = \sqrt{1 + f_u^2 + f_v^2}$, its first partial derivatives are given by

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

The second-partial derivations are given by

$$\begin{aligned}
\ell_{uu} &= \frac{f_u f_{uuu} + f_v f_{uuv} + f_{uu}^2 + f_{uv}^2}{\ell} - \frac{\left(f_u f_{uu} + f_v f_{uv}\right)^2}{\ell^3}, \\
\ell_{vv} &= \frac{f_v f_{vvv} + f_u f_{uvv} + f_{vv}^2 + f_{uv}^2}{\ell} - \frac{\left(f_v f_{vv} + f_u f_{uv}\right)^2}{\ell^3}, \\
\ell_{uv} &= \frac{f_{uv} f_{uu} + f_{vv} f_{uv} + f_u f_{uuv} + f_v f_{uvv}}{\ell} - \frac{\left(f_u f_{uu} + f_v f_{uv}\right)\left(f_u f_{uv} + f_v f_{vv}\right)}{\ell^3}.
\end{aligned}$$

Omitting the second-order terms in ∇f, we then obtain

$$\begin{aligned}
\ell_{uu} &\approx \frac{f_u f_{uuu} + f_v f_{uuv} + f_{uu}^2 + f_{uv}^2}{\ell}, \\
\ell_{vv} &\approx \frac{f_v f_{vvv} + f_u f_{uvv} + f_{vv}^2 + f_{uv}^2}{\ell}, \\
\ell_{uv} &\approx \frac{f_u f_{uuv} + f_v f_{uvv} + f_{uv}(f_{uu} + f_{vv})}{\ell}.
\end{aligned}$$

3.5 Surface Gradient

Given a scalar function ϕ, the *surface gradient*, denoted by$\nabla_\Gamma \phi$ or $\nabla_{\boldsymbol{x}} \phi$, has a direction in the tangent space of Γ along which ϕ increases the most rapidly, and its magnitude is the maximum rate of change of ϕ. By the chain rule,$\nabla \boldsymbol{u} = \boldsymbol{J}^T \nabla_{\boldsymbol{x}} \phi$, the surface gradient is then

$$\nabla_{\boldsymbol{x}} \phi = \boldsymbol{J}^{+T} \nabla \boldsymbol{u}, \tag{3.9}$$

where

$$\boldsymbol{J}^{+} = \left(\boldsymbol{J}^T \boldsymbol{J}\right)^{-1} \boldsymbol{J}^T$$

is the pseudoinverse of $\boldsymbol{J}$.

Note that in the tensor notation for curvilinear coordinate systems, the matrix $\boldsymbol{G} = \boldsymbol{J}^T\boldsymbol{J}$ is known as the *covariant metric tensor*, whereas $\boldsymbol{G}^{-1} = \left(\boldsymbol{J}^T\boldsymbol{J}\right)^{-1}$ is the *contra variant metric tensor*, and their components, denoted by g_{ij} and g^{ij} respectively, are the *covariant* and *contravariant components*, respectively. Instead of using g_{ij} and g^{ij}, we choose to use $\boldsymbol{J}$ and $\boldsymbol{J}^+$ in our analysis, because the latter are transformation matrices and can lead to more concise formulae and clearer derivations.

3.6 Surface Divergence

Given a small surface patch S on Γ, let ∂S denote the boundary curve of S, and $d\boldsymbol{a} \equiv \boldsymbol{n}ds$ is orthogonal to ∂S and tangent to Γ. The surface divergence is equal to

$$\nabla_\Gamma \cdot \boldsymbol{f} = \lim_{A\to 0} \frac{1}{A} \oint_{\partial S} \boldsymbol{f} \cdot d\boldsymbol{a}, \text{ where } A = \text{area of } S. \tag{3.10}$$

Using the matrix notation, the surface divergence can be expressed as follows.

For a vector-valued function $\boldsymbol{f}(\boldsymbol{u}) : \mathbb{R}^2 \to \mathbb{R}^3$ on S, the surface divergence is

$$\nabla_\Gamma \cdot \boldsymbol{f} = \text{tr}\left(\boldsymbol{J}^+\nabla_{\boldsymbol{u}}\left(\boldsymbol{J}\boldsymbol{J}^+\boldsymbol{f}\right)\right). \tag{3.11}$$

Let $\boldsymbol{n}$ denote unit outward normal to Γ. Consider the decomposition of $\boldsymbol{f}$ into tangential and normal components

$$\boldsymbol{f} = \boldsymbol{J}\boldsymbol{J}^+\boldsymbol{f} + \boldsymbol{n}\boldsymbol{n}^T\boldsymbol{f}.$$

In (Equation 3.10), note that

$$\lim_{A\to 0} \frac{1}{A} \oint_{\partial S} \left(\boldsymbol{n}\boldsymbol{n}^T \boldsymbol{f} \right) \cdot d\boldsymbol{a} = 0,$$

which can be easily shown from the Taylor series expansions of $\boldsymbol{n}$. Then

$$\nabla_\Gamma \cdot \boldsymbol{f} = \nabla_\Gamma \cdot \left(\boldsymbol{J}\boldsymbol{J}^+ \boldsymbol{f} \right).$$

At any point $\boldsymbol{x}_0$ on Γ, let $\boldsymbol{J}_0$ denote the Jacobian matrix at the point, and consider the local uv coordinate system of the tangent space at $\boldsymbol{x}_0$ with the columns of $\boldsymbol{J}_0$ as the base vectors. Since the two columns of $\boldsymbol{J}_0$ are in general not orthogonal, consider its QR factorization $\boldsymbol{Q}_0\boldsymbol{R}_0$, and the local $\xi\eta$ coordinate system with the columns of $\boldsymbol{Q}_0$ as base vectors. Then $\boldsymbol{J}_0^+ = \boldsymbol{R}_0^{-1}\boldsymbol{Q}_0^T$. Let

$$\boldsymbol{\varphi} = \boldsymbol{Q}_0^T \boldsymbol{J}\boldsymbol{J}^+ \boldsymbol{f} = \boldsymbol{R}_0 \boldsymbol{J}_0^+ \boldsymbol{J}\boldsymbol{J}^+ \boldsymbol{f}.$$

From the divergence theorem, we have

$$\nabla_\Gamma \cdot \boldsymbol{f} = \frac{\partial \varphi_1}{\partial \xi} + \frac{\partial \varphi_2}{\partial \eta} = \operatorname{tr}\left(\nabla_{\boldsymbol{\xi}} \boldsymbol{\varphi} \right),$$

where

$$\nabla_{\boldsymbol{\xi}} \boldsymbol{\varphi} = \left(\nabla_{\boldsymbol{u}} \boldsymbol{\varphi} \right) \boldsymbol{J}_0^+ \boldsymbol{Q}_0 = \left(\nabla_{\boldsymbol{u}} \boldsymbol{\varphi} \right) \boldsymbol{R}_0^{-1} = \boldsymbol{R}_0 \boldsymbol{J}_0^+ \left(\nabla_{\boldsymbol{u}} \left(\boldsymbol{J}\boldsymbol{J}^+ \boldsymbol{f} \right) \right) \boldsymbol{R}_0^{-1}.$$

Therefore,

$$\nabla_\Gamma \cdot \boldsymbol{f} = \operatorname{tr}\left(\boldsymbol{R}_0 \boldsymbol{J}_0^+ \left(\nabla_{\boldsymbol{u}} \left(\boldsymbol{J}\boldsymbol{J}^+ \boldsymbol{f} \right) \right) \boldsymbol{R}_0^{-1} \right) = \operatorname{tr}\left(\boldsymbol{J}^+ \left(\nabla_{\boldsymbol{u}} \left(\boldsymbol{J}\boldsymbol{J}^+ \boldsymbol{f} \right) \right) \right)$$

at point $\boldsymbol{x}_0$.

Note that if $\boldsymbol{f}$ is tangent to Γ, then $\boldsymbol{f} = \boldsymbol{J}\boldsymbol{J}^{+}\boldsymbol{f}$, and we can further simplify (Equation 3.11) to

$$\nabla_\Gamma \cdot \boldsymbol{f} = \mathrm{tr}\left(\boldsymbol{J}^{+}\left(\nabla_{\boldsymbol{u}}\boldsymbol{f}\right)\right) = \boldsymbol{J}^{+}_{1,:}\boldsymbol{f}_u + \boldsymbol{J}^{+}_{2,:}\boldsymbol{f}_v, \tag{3.12}$$

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

In the literature, the surface divergence is sometimes defined as [37, page 21]

$$\nabla_\Gamma \cdot \boldsymbol{f} = \left(\nabla_{\boldsymbol{x}} - \boldsymbol{n}\boldsymbol{n}^T\nabla_{\boldsymbol{x}}\right) \cdot \left(\boldsymbol{f} - \boldsymbol{n}\boldsymbol{n}^T\boldsymbol{f}\right), \tag{3.13}$$

where $\nabla_{\boldsymbol{x}}$ is the gradient operator with respect to $\boldsymbol{x}$, or as [55, 70]

$$\nabla_\Gamma \cdot \boldsymbol{f} = \frac{1}{\sqrt{g}}\nabla_{\boldsymbol{u}} \cdot \left(\sqrt{g}\boldsymbol{J}^{+}\boldsymbol{f}\right), \tag{3.14}$$

where $g = \det(\boldsymbol{J}^T\boldsymbol{J})$. It can be shown that these definitions are equivalent. However, (Equation 3.12) has the simplest form and will allow dramatically simpler derivations in later sections.

3.7 Surface Laplacian

The *surface Laplacian* of a scalar function φ, denoted by $\Delta_\Gamma\varphi$, is given by the surface divergence of the surface gradient, i.e.,

$$\Delta_\Gamma\varphi = \nabla_\Gamma \cdot \nabla_\Gamma\varphi.$$

Because $\nabla_\Gamma \varphi$ is in the tangent space by definition, from (Equation 3.9) and (Equation 3.12), we then have

$$\Delta_\Gamma \phi = \text{tr}\left(\boldsymbol{J}^{+}\nabla_{\boldsymbol{u}}\left(\boldsymbol{J}^{+T}\nabla_{\boldsymbol{u}}\varphi\right)\right) = \text{tr}\left(\boldsymbol{J}^{+}\nabla_{\boldsymbol{u}}\left(\boldsymbol{J}^{+T}\nabla_{\boldsymbol{u}}\varphi\right)\boldsymbol{J}\right).$$

Using the product rule,

$$\nabla_{\boldsymbol{u}}\left(\boldsymbol{J}^{+T}\nabla_{\boldsymbol{u}}\varphi\right) = \underbrace{\left(\nabla^2_{\boldsymbol{u}}\varphi\right)}_{\text{Hessian of } \varphi} \boldsymbol{J}^{+} + \begin{bmatrix} (\nabla\varphi)^T \boldsymbol{J}^{+}_u \\ (\nabla\varphi)^T \boldsymbol{J}^{+}_v \end{bmatrix}$$

Therefore,

$$\Delta_\Gamma \varphi = \text{tr}\left(\boldsymbol{G}^{-1}\left(\left(\nabla(\nabla\varphi)\right)\boldsymbol{J}^{+}\boldsymbol{J} - \begin{bmatrix} (\nabla_\Gamma\varphi)^T \boldsymbol{J}_u \\ (\nabla_\Gamma\varphi)^T \boldsymbol{J}_v \end{bmatrix}\right)\right) = \text{tr}(\boldsymbol{G}^{-1}\boldsymbol{M}),$$

where

$$\boldsymbol{M} = \nabla(\nabla\varphi) - \begin{bmatrix} (\nabla_\Gamma\varphi)^T \boldsymbol{J}_u \\ (\nabla_\Gamma\varphi)^T \boldsymbol{J}_v \end{bmatrix}.$$

In the local coordinate system using a local height function, it can be further simplified to

$$\Delta_\Gamma \varphi = \text{tr}\left(\boldsymbol{G}^{-1}\left(\nabla^2\varphi - \frac{(\nabla\varphi)^T \nabla f}{\ell^2}\boldsymbol{H}\right)\right) \tag{3.15}$$

3.8 Calculation of Surface Laplacian of Mean Curvature

In chapter 4 we will make use of the surface Laplacian of mean curvature, $\triangle_\Gamma M$. To maintain simplicity in chapter 4, we will discuss our calculation of the surface Laplacian of mean curvature here. We consider the computation of $\triangle_\Gamma M$ in the local uv coordinate system, as given in (Equation 4.5). The mean-curvature M is given by

$$M = \frac{\text{tr}(\boldsymbol{H})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^3},$$

and hence its gradient is

$$\nabla M = \frac{\ell\left(\nabla \text{tr}(\boldsymbol{H})\right) - \text{tr}(\boldsymbol{H})\nabla \ell}{2\ell^2} - \frac{\ell \nabla \left((\nabla f)^T \boldsymbol{H} \nabla f\right) - (\nabla f)^T \boldsymbol{H} \nabla f \nabla \ell}{2\ell^4}.$$

Therefore,

$$\begin{aligned}\nabla^2 M =& \frac{\ell \nabla \left(\ell\left(\nabla \text{tr}(\boldsymbol{H})\right) - \text{tr}(\boldsymbol{H})\nabla \ell\right) - 2\nabla \ell \left(\ell\left(\nabla \text{tr}(\boldsymbol{H})\right) - \text{tr}(\boldsymbol{H})\nabla \ell\right)^T}{2\ell^3} - \\ & \frac{\nabla \left(\ell \nabla \left((\nabla f)^T \boldsymbol{H} \nabla f\right) - (\nabla f)^T \boldsymbol{H} \nabla f \nabla \ell\right)}{2\ell^4} + \\ & \frac{4\nabla \ell \left(\ell \nabla \left((\nabla f)^T \boldsymbol{H} \nabla f\right) - (\nabla f)^T \boldsymbol{H} \nabla f \nabla \ell\right)^T}{2\ell^5}.\end{aligned}$$

For simplicity, we consider only up to second-order accuracy in ∇f. In (Equation 4.5), ∇M is multiplied with ∇f, and $\nabla \ell$ is also in the order of ∇f. Therefore, we can omit the first-order terms in ∇f and approximate ∇M as

$$\nabla M \approx \frac{1}{2\ell} \nabla \text{tr}(\boldsymbol{H}), \tag{3.16}$$

and therefore in (Equation 4.5)

$$\frac{(\nabla M)^T \nabla f}{\ell^2} \approx \frac{(\nabla \text{tr}(\boldsymbol{H}))^T \nabla f}{2\ell^3}$$

is up to second-order accurate in ∇f.

For $\nabla^2 M$, omitting the second-order terms in ∇f, we obtain an approximation

$$\begin{aligned}\nabla^2 M \approx & \frac{\nabla\left(\ell\left(\nabla \text{tr}(\boldsymbol{H})\right) - \text{tr}(\boldsymbol{H})\nabla\ell\right)}{2\ell^2} - \frac{\nabla\ell\left(\nabla \text{tr}(\boldsymbol{H})\right)^T}{\ell^2} - \frac{\nabla^2\left((\nabla f)^T \boldsymbol{H}\nabla f\right)}{2\ell^3} & (3.17)\\ = & \frac{1}{\ell}\nabla^2\left(\text{tr}\left(\boldsymbol{H}\right)\right) - \frac{1}{2\ell^2}\nabla^2\left(\ell \text{tr}(\boldsymbol{H})\right) - \frac{1}{2\ell^3}\nabla^2\left((\nabla f)^T \boldsymbol{H}\nabla f\right), & (3.18)\end{aligned}$$

where the second equality above is because the first two-terms of (Equation 3.17) can be regrouped as

$$\begin{aligned}&\frac{\nabla\left(\ell\left(\nabla \text{tr}(\boldsymbol{H})\right) - \text{tr}(\boldsymbol{H})\nabla\ell\right)}{2\ell^2} - \frac{\nabla\ell\left(\nabla \text{tr}(\boldsymbol{H})\right)^T}{\ell^2}\\ =&\frac{1}{2\ell^2}\left(\nabla\left(\ell\left(\nabla \text{tr}(\boldsymbol{H})\right) - \text{tr}(\boldsymbol{H})\nabla\ell\right) - 2\nabla\ell\left(\nabla \text{tr}(\boldsymbol{H})\right)^T\right)\\ =&\frac{1}{2\ell^2}\left(\ell\nabla^2 \text{tr}(\boldsymbol{H}) - \text{tr}(\boldsymbol{H})\nabla^2\ell - \nabla \text{tr}(\boldsymbol{H})\left(\nabla\ell\right)^T - \nabla\ell\left(\nabla \text{tr}(\boldsymbol{H})\right)^T\right)\\ =&\frac{1}{2\ell^2}\left(2\ell\nabla^2 \text{tr}(\boldsymbol{H}) - \nabla^2(\ell \text{tr}(\boldsymbol{H}))\right).\end{aligned}$$

(Equation 3.18) is compact and conceptually simple. For computational purpose, we can expand (Equation 3.18) and further omit additional second order terms to obtain

$$\begin{aligned}
M_{uu} \approx & \frac{\ell \mathsf{tr}(\boldsymbol{H}_{uu}) - 2\ell_u \mathsf{tr}(\boldsymbol{H}_u) - \ell_{uu}\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \\
& \frac{(\nabla f_{uu})^T \boldsymbol{H} \nabla f + 2(\nabla f_u)^T \boldsymbol{H}_u \nabla f + (\nabla f_u)^T \boldsymbol{H} \nabla f_u}{\ell^3}, \\
M_{uv} \approx & \frac{\ell \mathsf{tr}(\boldsymbol{H}_{uv}) - \ell_u \mathsf{tr}(\boldsymbol{H}_v) - \ell_v \mathsf{tr}(\boldsymbol{H}_u) - \ell_{uu}\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \\
& \frac{(\nabla f_{uv})^T \boldsymbol{H} \nabla f + (\nabla f_u)^T \boldsymbol{H}_v \nabla f + (\nabla f_v)^T \boldsymbol{H}_u \nabla f + (\nabla f_u)^T \boldsymbol{H} \nabla f_v}{\ell^3} \\
M_{vv} \approx & \frac{\ell \mathsf{tr}(\boldsymbol{H}_{vv}) - 2\ell_v \mathsf{tr}(\boldsymbol{H}_v) - \ell_{vv}\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \\
& \frac{(\nabla f_{vv})^T \boldsymbol{H} \nabla f + 2(\nabla f_v)^T \boldsymbol{H}_v \nabla f + (\nabla f_v)^T \boldsymbol{H} \nabla f_v}{\ell^3}.
\end{aligned}$$

In the above, $\boldsymbol{H}_{\xi\eta}$ are the only terms that involve fourth-order derivatives, and the terms $\ell_{\xi\eta}$, $\boldsymbol{H}_{\xi}$, and $\nabla f_{\xi\eta}$ involve third-order derivatives, where ξ is u or v and so is η. All the other terms involve first- or second-order derivatives.

Finally, for completeness, we also give the exact formulae for the components of ∇M and $\nabla^2 M$ as

$$\begin{aligned}
M_u = & \frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_u \nabla f}{2\ell^3} - \ell_u \left(\frac{\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^3} \\
M_v = & \frac{\mathsf{tr}(\boldsymbol{H}_v)}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_v \nabla f}{2\ell^3} - \ell_v \left(\frac{\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \frac{(\nabla f_v)^T \boldsymbol{H} \nabla f}{\ell^3}, \\
M_{uu} = & \frac{\mathsf{tr}(\boldsymbol{H}_{uu})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{uu} \nabla f}{2\ell^3} - \ell_u \left(\frac{\mathsf{tr}(\boldsymbol{H}_u)}{\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{\ell^4} \right) - \\
& \ell_{uu} \left(\frac{\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \frac{(\nabla f_{uu})^T \boldsymbol{H} \nabla f + 2(\nabla f_u)^T \boldsymbol{H}_u \nabla f}{\ell^3} - \\
& \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f_u}{\ell^3} + \ell_u^2 \left(\frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) + 6\ell_u \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4},
\end{aligned}$$

$$
\begin{aligned}
M_{vv} =& \frac{\operatorname{tr}(\boldsymbol{H}_{vv})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{vv} \nabla f}{2\ell^3} - \ell_v \left(\frac{\operatorname{tr}(\boldsymbol{H}_v)}{\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_v \nabla f}{\ell^4} \right) - \\
& \ell_{vv} \left(\frac{\operatorname{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \frac{(\nabla f_{vv})^T \boldsymbol{H} \nabla f + 2(\nabla f_v)^T \boldsymbol{H}_v \nabla f}{\ell^3} - \\
& \frac{(\nabla f_v)^T \boldsymbol{H} \nabla f_v}{\ell^3} + \ell_v^2 \left(\frac{\operatorname{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) + 6\ell_v \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4},
\end{aligned}
$$

$$
\begin{aligned}
M_{uv} =& \frac{\operatorname{tr}(\boldsymbol{H}_{uv})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{uv} \nabla f}{2\ell^3} - \ell_u \left(\frac{\operatorname{tr}(\boldsymbol{H}_v)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_v \nabla f}{2\ell^4} \right) - \\
& \ell_v \left(\frac{\operatorname{tr}(\boldsymbol{H}_u)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{2\ell^4} \right) - \ell_{uv} \left(\frac{\operatorname{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \\
& \frac{(\nabla f_{uv})^T \boldsymbol{H} \nabla f + (\nabla f_u)^T \boldsymbol{H}_v \nabla f + (\nabla f_v)^T \boldsymbol{H}_u \nabla f + (\nabla f_u)^T \boldsymbol{H} \nabla f_v}{\ell^3} + \\
& \ell_u \ell_v \left(\frac{\operatorname{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) + 3\ell_u \frac{(\nabla f_v)^T \boldsymbol{H} \nabla f}{\ell^4} + 3\ell_v \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4}.
\end{aligned}
$$

We do not use these formulae directly in our computations for efficiency purpose. Nevertheless, these formulae are useful for numerical verification of the accuracy of the our approximate formulae above.

3.9 Vertex-Based Polynomial Fittings

We approximate various differential operators based on local polynomial fittings. We have successfully used these techniques previously to compute differential quantities of discrete surfaces (such as normals and curvatures) to high-order accuracy [46, 65]. For more details on the theoretical background, readers are referred to [46] and references therein.

3.9.1 Local Polynomial Fitting

Local polynomial fittings, also known as *Taylor polynomials* in numerical analysis [40], are based on the well-known Taylor series expansions about a point. We are primarily concerned with surfaces, so the local fitting is basically an interpolation or approximation to a neighborhood of a point P under a local parametrization (with parameters u and v), where P corresponds to $u = 0$ and $v = 0$. The polynomial fitting may be defined over the global xyz coordinate system or a local uvw coordinate system. In the former, the neighborhood of the surface is defined by the *coordinate function* $\mathbf{f}(u, v) = [x(u, v), y(u, v), z(u, v)]$. 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 $[u, v]^T$. Let $\varphi(\mathbf{u})$ denote a smooth bivariate function, which may be the local height function or the x, y, or z component of the coordinate function for a parametric surface. Let c_{jk} be a shorthand for $\frac{\partial^{j+k}}{\partial u^j \partial v^k}\varphi(\mathbf{0})$. Let d be the desired degree of the polynomial fitting, typically ≤ 6. If $\varphi(\mathbf{u})$ has $d + 1$ continuous derivatives, it can be approximated to $(d + 1)$st order accuracy about the origin $\mathbf{u}_0 = [0, 0]^T$ by

$$\varphi(\mathbf{u}) = \underbrace{\sum_{p=0}^{d} \sum_{j,k\geq 0}^{j+k=p} c_{jk} \frac{u^j v^k}{j!k!}}_{\text{Taylor polynomial}} + \underbrace{\sum_{j,k\geq 0}^{j+k=d+1} \frac{\partial^{j+k}}{\partial u^j \partial v^k} \varphi(\tilde{u}, \tilde{v}) \frac{\tilde{u}^j \tilde{v}^k}{j!k!}}_{\text{remainder}}, \tag{3.19}$$

where $0 \leq \tilde{u} \leq u$ and $0 \leq \tilde{v} \leq v$.

Suppose we have a set of data points, say $[u_i, v_i, \varphi_i]^T$ for $i = 1, \ldots, m - 1$, sampled from a neighborhood near P on the surface. Substituting each given point into (Equation 3.19), we obtain an approximate equation

$$\sum_{p=0}^{d} \sum_{j,k \geq 0}^{j+k=p} \left(\frac{u_i^j v_i^k}{j!k!} \right) c_{jk} \approx \varphi_i, \tag{3.20}$$

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. Note that one could force the polynomial to pass through point P by setting $c_{00} = 0$ and removing its corresponding equation, 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.

Let us denote the rectangular linear system obtained from (Equation 3.20) as

$$\boldsymbol{Ac} \approx \boldsymbol{f}, \tag{3.21}$$

where $\boldsymbol{c}$ is an n-vector composed of c_{jk}, $\boldsymbol{A}$ is $m \times n$, known as a *generalized Vandermonde matrix* and $\boldsymbol{f}$ is an m-vector composed of f_i.

The above formulations can be easily adapted to curves in 2-D or 3-D, by using the univariable instead of the bivariable version of Taylor series expansions. For a curve in 3-D, the parametrization has only one parameter, u, and the local height function has two components. When applied to a surface mesh, the point P is typically a vertex with some k-ring neighborhood. Following Jiao and Zha [46], we allow k to have 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 neighborhood* is the union of the 1.5-ring neighbors of the k-ring neighbor vertices.

We typically choose k to be $(d+1)/2$ (for non-noisy surface) or $d/2+1$ (for noisy surface), but may also enlarge k if there are fewer than 1.5 times the required number of points in the k-ring.

3.9.2 Weighted Least Squares Approximation

Numerically, (Equation 3.21) can be solved using the framework of *weighted linear least squares* [36, p. 265], i.e., to minimize a weighted norm (or semi-norm),

$$\min_{\boldsymbol{c}} \|\boldsymbol{A}\boldsymbol{c} - \boldsymbol{f}\|_{\boldsymbol{\Omega}} = \min_{\boldsymbol{c}} \|\boldsymbol{\Omega}(\boldsymbol{A}\boldsymbol{c} - \boldsymbol{f})\|_2, \tag{3.22}$$

where $\boldsymbol{\Omega}$ is a *weighting matrix*. Typically, $\boldsymbol{\Omega}$ 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 $\boldsymbol{A}$. It is desirable to assign lower priorities to points that are farther away from the origin or whose normals differ substantially from the w direction of the local coordinate frame.

The formulation (Equation 3.22) is equivalent to the linear least squares problem

$$\tilde{\boldsymbol{A}}\boldsymbol{c} \approx \tilde{\boldsymbol{f}}, \text{ where } \tilde{\boldsymbol{A}} = \boldsymbol{\Omega}\boldsymbol{A} \text{ and } \tilde{\boldsymbol{f}} = \boldsymbol{\Omega}\boldsymbol{f}. \tag{3.23}$$

In general, $\tilde{\boldsymbol{A}}$ is $m \times n$ and $m \geq n$. A technical difficulty is that this linear system may be very ill-conditioned (i.e., the singular values of $\tilde{\boldsymbol{A}}$ may differ by orders of magnitude) due to a variety of reasons, such as poor scaling, insufficient number of points, or degenerate arrangements of points [47]. The conditioning number of $\tilde{\boldsymbol{A}}$ can be improved by using a scaling matrix $\boldsymbol{S}$, changing the problem:

$$\min_{\boldsymbol{y}} \|\boldsymbol{B}\boldsymbol{y} - \boldsymbol{f}\|_2, \text{ where } \boldsymbol{B} = \tilde{\boldsymbol{A}}\boldsymbol{S} \text{ and } \boldsymbol{y} = \boldsymbol{S}^{-1}\boldsymbol{c}. \tag{3.24}$$

We chose $\boldsymbol{S}$ to be a diagonal matrix. Let $\tilde{\mathbf{v}}_i$ denote the ith column of $\tilde{\boldsymbol{A}}$. The ith diagonal entry of $\boldsymbol{S}$ is chosen to be $\|\tilde{\mathbf{v}}_i\|_2$, which approximately minimizes the condition number of $\tilde{\boldsymbol{A}}\boldsymbol{S}$ [36, p. 265].

3.9.3 Accuracy and Stability of Least Squares Polynomial Fittings

The local least squares polynomial fitting provides us accurate approximation to the exact surface, established by the following proposition by Jiao and Zha [46]:

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 scaled matrix $\boldsymbol{B} = \tilde{\boldsymbol{A}}\boldsymbol{S}$ in (Equation 3.24) is bounded. The degree-d weighted least squares fitting approximates c_{jk} to $O(h^{d-j-k+1})$.

3.10 Mesh Regularization

Mesh regularization redistributes surface nodes tangentially along the surface to ensure that the supports of basis functions are nearly uniform locally. During mesh regulation, the line beginning from the average of the barycenters of the triangular elements neighboring vertex $\boldsymbol{z}_i$ and pointing in the direction of a weighted-average of the unit normals of the neighboring elements. Then the vertex $\boldsymbol{z}_i$ is moved onto this line in a manner such that the volume of the (closed) surface is unchanged. This process is explained in more detail in section 6.4.

4 Explicit Methods and Spatial Discretization

In this section, we consider explicit methods of mean-curvature flow as well as surface diffusion. We first describe an explicit method, which is relatively simple and will be used as a reference of our comparison. Because the temporal discretization of explicit methods are straightforward, the primary focus of this section is on spatial discretization of the geometric differential operators.

4.1 Mean-Curvature Flow

Since the surface is a triangulated, we use the method of lines to discretize the PDE to obtain a system of ODEs. In particular, at each vertex of the triangulation, we have an ODE

$$\frac{d\boldsymbol{x}_i}{dt} = M_i \hat{\boldsymbol{n}}_i, \tag{4.1}$$

where $\boldsymbol{x}_i$, $\hat{\boldsymbol{n}}_i$ and M_i denote the position vector, the unit normal vector and the mean curvature at the ith vertex of the mesh, respectively. In section 3.2, the mean curvature is computed as

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

where $\boldsymbol{H}$ is the Hessian of the local height function with respect to the local parametrization, and contains the second-order terms. We approximate $\hat{\boldsymbol{n}}_i$ and M_i numerically using a local polynomial fitting at the ith vertex, as described in the previous section. In particular, we use cubic fittings over a 2-ring. The system of ODEs (Equation 4.1) are then discretized using the forward Euler scheme, and we obtain the equation

$$\boldsymbol{x}_i^{(n+1)} = \boldsymbol{x}_i^{(n)} + \triangle t M_i^{(n)} \hat{\boldsymbol{n}}_i^{(n)}, \tag{4.2}$$

where the superscripts denote the indices of the time-step of the corresponding variables.

4.2 Surface Diffusion

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

$$\frac{d\boldsymbol{x_i}}{dt} = \left(\triangle_\Gamma M\right)_i \hat{\boldsymbol{n}}_i, \tag{4.3}$$

where $\triangle_\Gamma$ denotes the surface Laplacian operator. Using the forward Euler scheme, we obtain the equation

$$\boldsymbol{x}_i^{(n+1)} = \boldsymbol{x}_i^{(n)} + \triangle t \left(\triangle_\Gamma M\right)_i^{(n)} \hat{\boldsymbol{n}}_i^{(n)}. \tag{4.4}$$

To solve (Equation 4.4), the key step is to discretize the surface Laplacian of the mean curvature. As described in section 3.7, the surface Laplacian of a function φ is given by (Equation 3.15). Substituting the mean curvature as φ into (Equation 3.15), we obtain the continuum formula for $\triangle_\Gamma M$ in the local coordinate system as

$$\triangle_\Gamma M = \text{tr}\left(\boldsymbol{G}^{-1}\left(\boldsymbol{H}_M - \frac{(\nabla M)^T \nabla f}{\ell^2}\boldsymbol{H}\right)\right), \tag{4.5}$$

where $\boldsymbol{H}_M$ and $\boldsymbol{H}$ denote the Hessian matrices of M and f, respectively, i.e.,

$$\boldsymbol{H}_M = \begin{bmatrix} M_{uu} & M_{uv} \\ M_{uv} & M_{vv} \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{H} = \begin{bmatrix} f_{uu} & f_{uv} \\ f_{uv} & f_{vv} \end{bmatrix},$$

and

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

In the local coordinate system, the mean curvature is given by

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

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

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

It is easy to show that

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

where $\boldsymbol{H}_u$ involves third-order derivatives with respect to u and v. Similarly,

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

where $\boldsymbol{H}_v$ involves third-order derivatives with respect to u and v. In summary, we can denote the gradient as

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

where $\nabla \boldsymbol{H}$ denotes a third-order tensor.

Let $\langle \nabla f, \nabla f \rangle_{\boldsymbol{H}} = \langle \nabla f, \nabla f \rangle_{\boldsymbol{H}}$

To obtain the second partial derivatives, note that

$$M_{uu} = A_u - \ell_u B_u - \ell_{uu} B - C_u, \tag{4.6}$$

where

$$\begin{aligned}
A_u &= \frac{\mathsf{tr}(\boldsymbol{H}_{uu})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{uu} \nabla f}{2\ell^3} - \ell_u \left(\frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{2\ell^4} \right) - \frac{(\nabla f_u)^T \boldsymbol{H}_u \nabla f}{\ell^3} \\
&\approx \frac{\mathsf{tr}(\boldsymbol{H}_{uu})}{2\ell} - \ell_u \frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell^2} - \frac{(\nabla f_u)^T \boldsymbol{H}_u \nabla f}{\ell^3} \\
B_u &= \frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{2\ell^4} - \ell_u \left(\frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) - \frac{3(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} \\
&\approx \frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell^2} - \ell_u \frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{3(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} \\
C_u &= \frac{(\nabla f_{uu})^T \boldsymbol{H} \nabla f + (\nabla f_u)^T \boldsymbol{H}_u \nabla f + (\nabla f_u)^T \boldsymbol{H} \nabla f_u}{\ell^3} - 3\ell_u \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} \\
\ell_{uu} &= \frac{f_u f_{uuu} + f_v f_{uuv}}{\ell} + \frac{f_{uu}^2 + f_{uv}^2 + (f_u f_{uv} - f_v f_{uu})^2}{\ell^3},
\end{aligned}$$

Substituting them into (Equation 4.6) and regrouping, we then obtain

$$\begin{aligned}
M_{uu} =& \frac{\mathsf{tr}(\boldsymbol{H}_{uu})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{uu} \nabla f}{2\ell^3} - \ell_u \left(\frac{\mathsf{tr}(\boldsymbol{H}_u)}{\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{\ell^4} \right) - \\
& \ell_{uu} \left(\frac{\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \frac{(\nabla f_{uu})^T \boldsymbol{H} \nabla f + 2(\nabla f_u)^T \boldsymbol{H}_u \nabla f}{\ell^3} + \\
& \ell_u^2 \left(\frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) + 6\ell_u \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} - \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f_u}{\ell^3} \\
\approx& \frac{\mathsf{tr}(\ell \boldsymbol{H}_{uu} - 2\ell_u \boldsymbol{H}_u - \ell_{uu} \boldsymbol{H})}{2\ell^2} - \frac{(\nabla f_{uu})^T \boldsymbol{H} \nabla f + 2(\nabla f_u)^T \boldsymbol{H}_u \nabla f + (\nabla f_u)^T \boldsymbol{H} \nabla f_u}{\ell^3}.
\end{aligned}$$

Symmstricically, we obtain

$$\begin{aligned}
M_{vv} =& \frac{\mathsf{tr}(\boldsymbol{H}_{vv})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{vv} \nabla f}{2\ell^3} - \ell_v \left(\frac{\mathsf{tr}(\boldsymbol{H}_v)}{\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_v \nabla f}{\ell^4} \right) - \\
& \ell_{vv} \left(\frac{\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \frac{(\nabla f_{vv})^T \boldsymbol{H} \nabla f + 2(\nabla f_v)^T \boldsymbol{H}_v \nabla f}{\ell^3} + \\
& \ell_v^2 \left(\frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) + 6\ell_v \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} - \frac{(\nabla f_v)^T \boldsymbol{H} \nabla f_v}{\ell^3}.
\end{aligned}$$

To obtain M_{uv}, note that

$$\begin{aligned}
A_v &= \frac{\mathsf{tr}(\boldsymbol{H}_{uv})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{uv} \nabla f}{2\ell^3} - \ell_v \left(\frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{2\ell^4} \right) - \frac{(\nabla f_v)^T \boldsymbol{H}_u \nabla f}{\ell^3} \\
B_v &= \frac{\mathsf{tr}(\boldsymbol{H}_v)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_v \nabla f}{2\ell^4} - \ell_v \left(\frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) - \frac{3(\nabla f_v)^T \boldsymbol{H} \nabla f}{\ell^4} \\
C_v &= \frac{(\nabla f_{uv})^T \boldsymbol{H} \nabla f + (\nabla f_u)^T \boldsymbol{H}_v \nabla f + (\nabla f_u)^T \boldsymbol{H} \nabla f_v}{\ell^3} - 3\ell_v \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} \\
\ell_{uv} &= \frac{f_u f_{uuv} + f_v f_{uvv}}{\ell} + \frac{f_{uv} f_{uu} + f_{vv} f_{uv} + (f_u f_{vv} - f_v f_{uv})(f_u f_{uv} - f_v f_{uu})}{\ell^3}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
M_{uv} =& A_v - \ell_u B_v - \ell_{uv} B - C_v \\
=& \frac{\mathsf{tr}(\boldsymbol{H}_{uv})}{2\ell} - \frac{(\nabla f)^T \boldsymbol{H}_{uv} \nabla f}{2\ell^3} - \ell_u \left(\frac{\mathsf{tr}(\boldsymbol{H}_v)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_v \nabla f}{2\ell^4} \right) - \\
& \ell_v \left(\frac{\mathsf{tr}(\boldsymbol{H}_u)}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H}_u \nabla f}{2\ell^4} \right) - \ell_{uv} \left(\frac{\mathsf{tr}(\boldsymbol{H})}{2\ell^2} - \frac{3(\nabla f)^T \boldsymbol{H} \nabla f}{2\ell^4} \right) - \\
& \frac{(\nabla f_{uv})^T \boldsymbol{H} \nabla f + (\nabla f_u)^T \boldsymbol{H}_v \nabla f + (\nabla f_v)^T \boldsymbol{H}_u \nabla f}{\ell^3} + \\
& \ell_u \ell_v \left(\frac{\mathsf{tr}(\boldsymbol{H})}{\ell^3} - \frac{6(\nabla f)^T \boldsymbol{H} \nabla f}{\ell^5} \right) + \\
& 3\ell_u \frac{(\nabla f_v)^T \boldsymbol{H} \nabla f}{\ell^4} + 3\ell_v \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f}{\ell^4} - \frac{(\nabla f_u)^T \boldsymbol{H} \nabla f_v}{\ell^3}.
\end{aligned}$$

To obtain a second-order approximation, we approximate $\triangle_\Gamma M$ as follows.

$$\triangle_\Gamma M \approx \mathsf{tr} \left(\boldsymbol{G}^{-1} \left(\tilde{\boldsymbol{H}}_M - \frac{\left(\tilde{\nabla} M\right)^T \nabla f}{\ell^2} \boldsymbol{H} \right) \right), \tag{4.7}$$

Omitting the terms that are second order in f_u and f_v, we obtain

$$\begin{aligned}\frac{\left(\tilde{\nabla} M\right)^T \nabla f}{\ell^2} \approx& \frac{\left(\nabla \mathsf{tr}(\boldsymbol{H})\right) \nabla f}{2\ell^3},\\ \tilde{M}_{st} \approx& \frac{2\ell \mathsf{tr}\left(\boldsymbol{H}_{st}\right) - \left(\ell \boldsymbol{H}\right)_{st}}{2\ell^2} - \frac{1}{2\ell^3}\left(\left(\nabla f\right)^T \boldsymbol{H} \nabla f\right)_{st},\end{aligned}$$

where s is u or v, and similarly for t. The terms involving ℓ are:

$$\begin{aligned}\ell &= \sqrt{1+f_u^2+f_v^2}\\ \ell_u &= \frac{f_u f_{uu} + f_v f_{uv}}{\ell}\\ \ell_v &= \frac{f_u f_{uv} + f_v f_{vv}}{\ell}\\ \ell_{uu} &\approx \frac{f_u f_{uuu} + f_v f_{uuv} + f_{uu}^2 + f_{uv}^2}{\ell}\\ \ell_{vv} &\approx \frac{f_v f_{vvv} + f_u f_{uvv} + f_{vv}^2 + f_{uv}^2}{\ell}\\ \ell_{uv} &\approx \frac{f_u f_{uuv} + f_v f_{uvv} + f_{uv}(f_{uu} + f_{vv})}{\ell}.\end{aligned}$$

The terms involving

$$\left(\left(\nabla f\right)^T \boldsymbol{H} \nabla f\right)_{st} \approx 2((\nabla f_{st})^T \boldsymbol{H} \nabla f + (\nabla f_s)^T \boldsymbol{H}_t \nabla f + (\nabla f_s)^T \boldsymbol{H}_t \nabla f + (\nabla f_s)^T \boldsymbol{H} \nabla f_t).$$

In summary,

$$\nabla^2 M \approx \frac{1}{\ell}\nabla^2\left(\mathsf{tr}\left(\boldsymbol{H}\right)\right) - \frac{1}{2\ell^2}\nabla^2\left(\ell \boldsymbol{H}\right) - \frac{1}{2\ell^3}\nabla^2\left(\left(\nabla f\right)^T \boldsymbol{H} \nabla f\right).$$

Note that in the above, the first two rows involve fourth-order and third-order derivatives with respect to u and v, while the last row involves only second-order derivatives.

4.3 Limitations of Explicit Methods

The mean-curvature flow is analogous to a heat equation, and therefore the stability constraint requires the time-step Δt to be proportional to the square of the local mesh resolution for this explicit scheme. However, because the PDE is given on an unstructured moving mesh, the CFL condition cannot be defined as clearly as for a heat equation on an Eulerian mesh. We will investigate the stability conditions experimentally in chapter 7. For these reasons, we consider deriving semi-implicit methods of the two equations, as described in chapter 5.

5 Semi-Implicit Methods

5.1 Semi-Implicit Method for Mean-Curvature Flow

In this section, we consider the numerical solution of the mean-curvature flow. The continuum form of the flow is given by (Equation 1.1). We will consider both explicit as well as semi-implicit schemes. The explicit scheme is simple and direct given the formulae in chapter 3 and the fitting method in section 3.9. Mean curvature is a nonlinear term, so a fully implicit scheme would be complicated. We devise our semi-implicit scheme by evaluating all the first order derivatives in the current time-step and all the second order derivatives in the future time-step to avoid the nonlinear issue.

To overcome the time-step restriction imposed by the CFL condition, we consider the implicit scheme. A fully implicit scheme for mean curvature flow would be,

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \cdot M_i^{(n+1)} \mathbf{n}_i^{(n+1)} \tag{5.1}$$

Where both normal and mean curvature are in the next time-step, this is possible if the mean curvature and normal can be expressed as linear combinations of vertices coordinates.

Recall the least square fitting equation at a vertex,

$$\boldsymbol{A}\boldsymbol{c} \approx \boldsymbol{f}, \tag{5.2}$$

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

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

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

$$f_u = \sum_{j\in N(i)} C_{2,j} f_j \qquad\qquad f_v = \sum_{j\in N(i)} C_{3,j} f_j$$

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

Now we can express the differentials as linear combinations of vertices coordinates. The fully implicit scheme imposes a challenge since the term of $M_i^{(n+1)}\mathbf{n}_i^{(n+1)}$ is nonlinear, so we use the explicit normal with implicit mean curvature,

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \cdot M_i^{(n+1)}\mathbf{n}_i^{(n)} \tag{5.3}$$

Now we look at the formula for mean curvature M_i,

$$global : M = \frac{1}{2}\text{tr}(\boldsymbol{G}^{-1}\boldsymbol{B}) \tag{5.4}$$

$$local : M = \frac{\text{tr}(\mathbf{H})}{2\ell} - \frac{(\nabla f)^T\mathbf{H}(\nabla f)}{2\ell^3} \tag{5.5}$$

For the global formula, the normal $\boldsymbol{n}$ is embedded in the second fundamental form $\boldsymbol{B}$, the implicit term $M_i^{(n+1)}$ requires $\mathbf{n}_i^{(n+1)}$, the term we try to avoid. So we use the formula under local coordinate system. The local mean curvature is still composed of the product of first and second order terms, we can make it linear with respect to the the second order derivatives only, and the first order derivatives ℓ and ∇f remain explicit. The corresponding mean curvature term is,

$$\widetilde{M}_i^{(n+1)} = \frac{\text{tr}(\widetilde{\mathbf{H}})}{2\ell} - \frac{(\nabla f)^T\widetilde{\mathbf{H}}(\nabla f)}{2\ell^3} = \frac{f_{uu} + f_{vv}}{2\ell} - \frac{f_u^2\widetilde{f_{uu}} + 2f_uf_v\widetilde{f_{uv}} + f_v^2\widetilde{f_{vv}}}{2\ell^3} \tag{5.6}$$

where ˜ indicates an implicit term. Using the coefficient matrix C we have,

$$\widetilde{M}_i^{(n+1)} = \frac{1}{2\ell^3}\sum_{j\in N(i)}\Big((1+f_v^2)C_{4,j} - 2f_uf_vC_{5,j} + (1+f_u^2)C_{6,j}\Big)\cdot(\mathbf{x}_j^{(n+1)} - \mathbf{x}_i^{(n+1)})\cdot\mathbf{n}_i^{(n)}$$

Thus the semi-implicit scheme,

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \cdot \widetilde{M}_i^{(n+1)}\mathbf{n}_i^{(n)} \tag{5.7}$$

This is a linear relation with respect to the future vertices coordinates, we can aggregate these equations over all vertices to form a global linear system,

$$\boldsymbol{B}^{(n)}\boldsymbol{\phi}^{(n+1)} = \boldsymbol{\phi}^{(n)}$$

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

5.2 Semi-Implicit Method for Surface Diffusion

The semi-implicit scheme is as follows:

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \left(\triangle_\Gamma \widetilde{M}^{(n+1)}\right)_i^{(n)} \mathbf{n}_i^{(n)}$$

The semi-implicit scheme uses explicit first and second order differentials, with implicit 3rd and 4th order differentials:

$$\mathbf{x}_i^{(n+1)} - \mathbf{x}_i^{(n)} = \triangle t \left(\text{tr}\left(\mathbf{G}^{-1}\left(\widetilde{\mathbf{H}}_M - \frac{\left(\nabla \widetilde{M}\right)^T \nabla f}{\ell^2}\mathbf{H}\right)\right)\right)_i^{(n)} \mathbf{n}_i^{(n)}$$

The CFL condition for surface diffusion is much more strict than the mean curvature flow, which is why we devised a semi-implicit scheme. It can be seen that M_u and M_v are linear with respect to the third order differentials (f_{uuu}, f_{uuv}, f_{uvv} and f_{vvv}); M_{uu}, M_{uv} and M_{vv} are linear with respect to both third and fourth order differentials (f_{uuuu}, f_{uuuv}, f_{uuvv}, f_{uvvv} and f_{vvvv}), because the sum of orders (of differential) for M is 4, taking twice derivative, the sum of order is 6, so a 3rd and a 4th order differential can not appear in the same term. Similar to the semi-implicit scheme for mean curvature flow, we construct the semi-implicit scheme for surface diffusion using explicit 1st and 2nd order differentials, implicit 3rd and 4th order differentials. The explicit and implicit terms are,

$$
\begin{aligned}
f_{uuu} &= 6c_{30} & f_{uuu} &= 6\sum_{j\in N(i)} C_{7,j} f_j \\
f_{uuv} &= 2c_{21} & f_{uuv} &= 2\sum_{j\in N(i)} C_{8,j} f_j \\
f_{uvv} &= 2c_{12} & f_{uvv} &= 2\sum_{j\in N(i)} C_{9,j} f_j \\
f_{vvv} &= 6c_{03} & f_{vvv} &= 6\sum_{j\in N(i)} C_{10,j} f_j \\
f_{uuuu} &= 24c_{40} & f_{uuuu} &= 24\sum_{j\in N(i)} C_{11,j} f_j \\
f_{uuuv} &= 6c_{31} & f_{uuuv} &= 6\sum_{j\in N(i)} C_{12,j} f_j \\
f_{uuvv} &= 4c_{22} & f_{uuvv} &= 4\sum_{j\in N(i)} C_{13,j} f_j \\
f_{uvvv} &= 6c_{13} & f_{uvvv} &= 6\sum_{j\in N(i)} C_{14,j} f_j \\
f_{vvvv} &= 24c_{04} & f_{vvvv} &= 24\sum_{j\in N(i)} C_{15,j} f_j
\end{aligned}
$$

5.3 Description of Algorithm

1. First, if not given, compute normals at each vertex.
2. For each vertex, construct the stencil and calculate all the differentials using weighted least square polynomial fittings.

For explicit scheme:

- Calculate normal, and mean curvature or surface diffusion from the local polynomial differentials.
- Propagate the surface using the explicit scheme.

For semi-implicit scheme:

- Aggregate over all the vertices to form the global linear system through the coefficient matrix, $\boldsymbol{C}$.
  - Note: The global linear system's coefficient matrix $\boldsymbol{A}$ will be sparse, consisting of submatrices, $\boldsymbol{A_i}$.
- Solve the linear system to get the updated surface mesh.
  - Note: For memory conservation when solving, matrix $\boldsymbol{A}$ should be stored using a sparse matrix format $\boldsymbol{A}_{sparse} = [\boldsymbol{v} \,|\, \boldsymbol{i} \,|\, \boldsymbol{j}]$. Where each row of matrix $\boldsymbol{A}_{sparse}$ contains a non-zero value, v_i, along with its row (i_i) and column (j_i) in matrix $\boldsymbol{A}$.

6 Mesh Adaptivity

As will be seen from our experimental results, our semi-implicit mean curvature flow and surface diffusion methods have many strengths, such as accuracy, stability and allowance for large time-steps. However, our testing for these methods rely on a decent mesh element quality. In a triangulated surface, while the most desirable element would be a triangle that is equilateral, a less desirable element would be a triangle with relatively acute or obtuse angles, or relatively long or short edges. Additionally, less desirable elements might be too small or too large relative to the overall mesh. We use mesh adaptivity to correct these issues.

There are primarily two cases in which the mesh quality becomes an issue during mean curvature flow or surface diffusion. The first is the initial mesh. Many times our initial mesh might have low quality elements due to how it is obtained. In such a case, our goal is to improve the mesh element quality using mesh adaptivity before evolving the mesh under mean curvature flow or surface diffusion using our semi-implicit methods as can be seen in Figure 6.1.

The second case for use of mesh adaptivity is after evolving a mesh under mean curvature

Figure 6.1: The evolution of a closed torus (major radius = 1, minor radius = 0.25) under mesh adaptivity utilizing edge splitting, edge contraction, edge flipping, and high-order surface reconstruction.

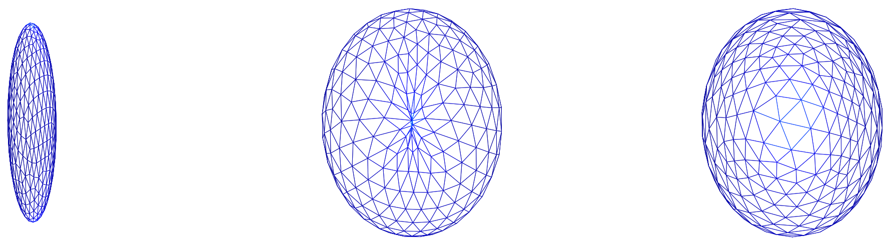

Figure 6.2: The evolution of a closed ellipsoid mesh (axes = 1.5, 2, 8) under mean curvature flow using our semi-implicit method. (left) surface before evolution. (center) the top of the surface after 0.14 seconds of evolution. (right) the top of the surface after 0.14 seconds of evolution with adaptivity.

flow or surface diffusion using our semi-implicit method for a duration of time. This can potentially lead to very large or very small elements which pose stability issues. Utilizing adaptivity during evolution can help maintain mesh quality and ultimately increase the stability when trying to further evolve the mesh, as can be seen in Figure 6.2.

6.1 Edge Splitting

The first tool at our disposal is edge splitting. Provided two adjacent triangular elements, edge splitting inserts a new vertex on their shared edge, as can be seen in Figure 6.3. There are two criterion which we use to determine if 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 $s < l$.

These parameters and criteria abide by Jiao et al. [43] and must be chosen consistently with edge contraction.

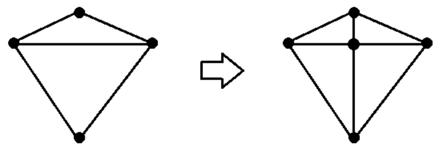

Figure 6.3: An example of edge splitting.

The process of edge splitting abides by this criterion to help optimize element quality and consistency in size. The process of edge splitting occurs in decreasing order of edge lengths throughout the mesh. Different from Jiao et al. [43], the new vertex is an average of the weighted polynomial fittings [65, 46] based on the vertices incident of the existing edge, and its opposite vertices, which allows us to maintain second-order convergence in our calculations. This process helps to preserve smoothness.

6.2 Edge Contraction

Edge contraction is another very useful technique for improving mesh quality. Provided an undesirably short edge, small angle or small element, edge contraction removes a desired edge and reassigns its incident vertices to a new location. This is done using the criterion set by Jiao et al. [43]:

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*.

These parameters and criteria abide by Jiao et al. [43] and must be chosen consistently with edge splitting.

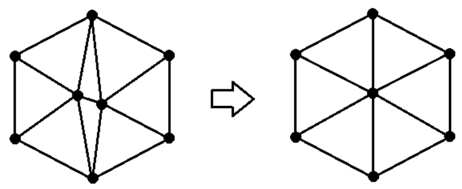

Figure 6.4: An example of edge contraction.

The process of edge contraction abides by this criterion 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. Slightly different from Jiao et al. [43], the new vertex is a weighted polynomial fitting [65, 46] based on the vertices incident to the contracted edge, which allows us to maintain second-order convergence in our calculations. This process helps to preserve smoothness. 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.

6.3 Edge Flipping

Edge flipping is the process of reassigning the connectivity of existing points such that an edge incident of two triangular faces is reassigned to instead connect that edge's opposite vertices, as can be seen in Figure 6.5. Provided an edge *uv* with opposite vertices *p* and *q*, the edge is flipped when satisfying the Delaunay flipping criterion (i.e. $\angle upv + \angle uqv > \pi$) as outlined by Jiao et al. [43].

Flipping edges helps improve element quality without any change in vertex positioning. The process of edge flipping occurs in decreasing order of maximum opposite angles, which avoids an infinite flipping of edges. Provided a mesh with sharp features, we reject any

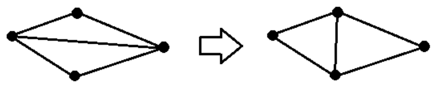

Figure 6.5: An example of edge flipping.

flips on such ridges.

6.4 Mesh Regularization

Mesh regularization is important because it redistributes vertices on a surface tangentially so that their neighboring elements are of similar size. This has the effect of "spreading out" vertices which can become clustered around one another and provides for better accuracy when evolving a mesh. Each vertex is moved to a position along the ray emanating from the average of the barycenter of the neighboring elements such that the total volume enclosed in the mesh is conserved. This will naturally allow you to introduce the parameter t whose purpose is to find the new vertex position on the ray which conserves volume. This is done using the method for mesh regularization outlined in Bänsch et al. [3]. Given a closed, triangulated surface $\Gamma(t)$, at each node:

1. Calculate the weighted average unit-normal for the node z. The weights are chosen to be the area of each element neighboring z,

$$\boldsymbol{\nu}_z = \frac{1}{\sum |T|} \sum \boldsymbol{\nu}_T \|T\|$$

2. Calculate the average of the barycenter, denoted as $\tilde{z}$, of each of the neighboring elements of z.

3. Determine the new position, $\hat{z}$, which has the form $\hat{z} = \tilde{z} + t\boldsymbol{\nu}_z$. By enforcing volume conservation between the original mesh and the mesh generated by replacing node z with $\hat{z}$, the parameter t can be calculated to be

$$t = \frac{\sum (z - \tilde{z}) \times (z_2 - \tilde{z}) \cdot (z_3 - \tilde{z})}{\sum \boldsymbol{\nu}_z \times (z_2 - \tilde{z}) \cdot (z_3 - \tilde{z})} \tag{6.1}$$

7 Numerical Experiments and Comparisons

In this section, we perform numerical experiments to verify our preceding analysis. The experimental results can be affected by a number of factors, such as input errors, surface topology, mesh connectivity, and methods used.

Our convergence tests will focus on our semi-implicit methods and their rate of convergence with and without adaptivity. These tests will measure the error in the total surface area and the error in the encapsulated volume on a variety of meshes. These values will then produce a rate of convergence for each measure of error. Overall, we want to see our semi-implicit methods achieve second-order convergence in surface area errors and encapsulated volume errors. We also hope to maintain second-order convergence in surface area errors and encapsulated volume errors when working with less ideal meshes, by use of mesh adaptivity.

Our comparisons will also be performed on a variety of meshes. First we will compare our semi-implicit methods with their explicit counterparts. Our second comparison will evaluate results using our semi-implicit methods with and without adaptivity. Our final comparison will directly examine the differences in results between our semi-implicit method with adaptivity for surface diffusion and the finite element method for surface diffusion presented by Bänsch et al. [4].

For our surface area calculation, we simply sum the surface area of all triangular elements. For our volume calculation, we connect all mesh vertices to the origin, forming tetrahedrons. We then sum the volume of all tetrahedrons to obtain the total volume encapsulated by our surface mesh.

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

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

R_{exact} is the exact radius of the sphere we expect to obtain after t time-steps of size $\triangle t$, from an initial sphere with radius of $R_{initial}$. For non-sphere mean curvature flow tests and surface diffusion tests, there is not an exact value for comparison, so we evolve a super-refined mesh to get an approximation of our errors. To measure convergence, we use the following equation:

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

7.1 Comparison of Explicit and Semi-Implicit Methods

In this section, we will conduct three comparisons between our semi-implicit methods and their explicit counterparts. The first will be a comparison for mean curvature flow. This comparison will measure the L^∞ error in the placement of the vertices, the surface area error and the encapsulated volume error of a spherical mesh with radius $= 1$. The second comparison will be for surface diffusion. This comparison will measure the surface area error and the encapsulated volume error of an ellipsoid mesh with axes 4.0, 6.0 and 8.0. The third will be a runtime comparison, which will test the difference between the explicit and semi-implicit methods' runtimes when producing converging results correlating to real-time.

7.1.1 Mean Curvature Comparison

Here we compare our semi-implicit mean curvature flow method with its explicit counterpart on a spherical surface with a radius of 1. We will use three levels of refinement for our initial mesh, with 368, 1,450, and 5,804 vertices. In these tests, we will measure L^∞ error, surface area error and encapsulated volume error. Overall, we want to see our semi-implicit method produce the same accuracy, and maintain higher stability than its explicit counterpart, while achieving second-order convergence in surface area errors and encapsulated volume errors. We obtain the results seen in Figure 7.1.

Because of their differences in stability, the explicit method cannot be compared effectively with our semi-implicit method using the same size time-step, $\triangle t$. As can be seen in Figure 7.1, the explicit method after 5 time-steps at $\triangle t = 10^{-3}$ diverges in both L^∞ errors and surface area errors, opposed to our semi-implicit method which converges in both cases. The results for the encapsulated volume errors were excluded because of their similarity to the surface area errors. Since the explicit method is less stable than our semi-implicit method, it requires a much smaller time-step of $\triangle t = 10^{-7}$ in order to produce convergent results in the surface area error and volume error. To create an equivalent real-time measure of 0.005 seconds, we must use 50000 time-steps with $\triangle t = 10^{-7}$. By shrinking the time-step, the explicit method's surface area error converges with a rate of 2.04, with similar results in its volume error, equal to the convergence rate of our semi-implicit method. Though, even with comparable results in accuracy, the major issue of

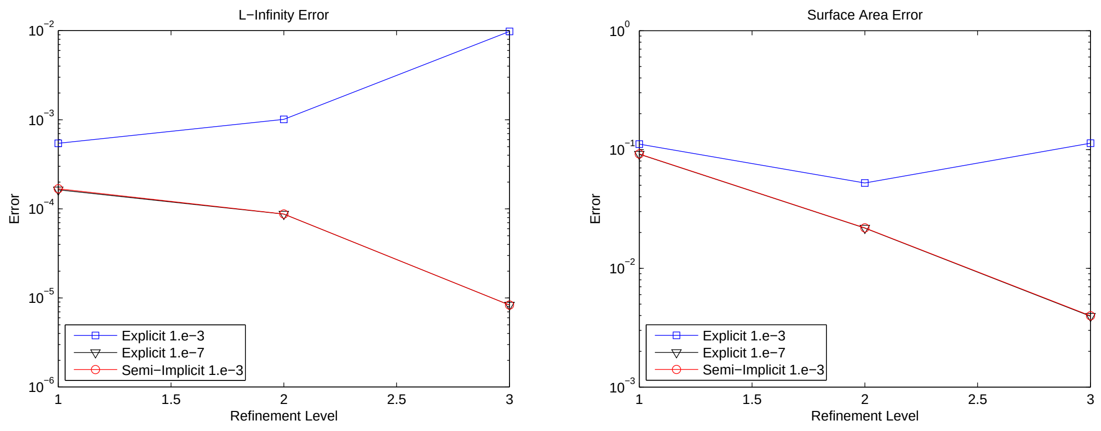

Figure 7.1: A comparison of our semi-implicit mean curvature flow method and its explicit counterpart on a spherical mesh of radius = 1 after 0.005 seconds. Compared are the explicit method after 5 time-steps at $\triangle t = 10^{-3}$ and 50000 time-steps at $\triangle t = 10^{-7}$, and our semi-implicit method for 5 time-steps at $\triangle t = 10^{-3}$, with refinement levels of 368, 1450, and 5804 vertices. (left) Displays the L^{∞}error of both methods, with the explicit method ($\triangle t = 10^{-3}$) diverging, and both the explicit method ($\triangle t = 10^{-7}$) and our semi-implicit method converging. (right) Displays the surface area error of both methods with the explicit method ($\triangle t = 10^{-3}$) diverging, the explicit method ($\triangle t = 10^{-7}$) and our semi-implicit method ($\triangle t = 10^{-3}$) converging with a rate of 2.04.

requiring such small time-steps for explicit method ultimately results in very long runtimes, as will be explored in subsection 7.3.2. Overall, we see our semi-implicit method produce the same accuracy, while maintaining much higher stability than its explicit counterpart, while achieving second-order convergence in surface area errors and encapsulated volume errors.

7.1.2 Surface Diffusion Comparison

Our next comparison will be for surface diffusion on an ellipsoid mesh with an axes 4.0, 6.0, and 8.0. We will use three levels of refinement for our initial mesh, with our number of vertices being 368, 1450, and 5804. For all meshes, we will perform 22 time-steps at $\triangle t = 5 \times 10^{-6}$. Ideally, surface diffusion produces no change in volume, so we will not display a comparison in volume errors. However, we would like to state that in our testing we saw that our method produced changes in volume always less than 1.0%. However, we still want to see our semi-implicit method produce the same accuracy, and maintain higher stability than its explicit counterpart, while achieving second-order convergence in surface area errors. Our results can be seen in Figure 7.2

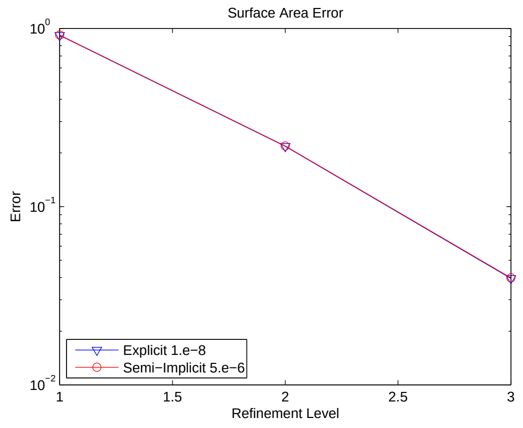

Figure 7.2: A comparison of explicit and semi-implicit surface diffusion methods on an ellipsoid mesh with axes 4.0, 6.0, and 8.0, with refinement levels of 368, 1450, and 5804 vertices after 1.1×10^{-4} seconds. Above, we display the surface area error of both the explicit method ($\triangle t = 10^{-8}$) and our semi-implicit method ($\triangle t = 5 \times 10^{-6}$) converging with a rate of 2.34.

Once again, we see the explicit method unable to achieve the same stability as our semi-implicit method. In Figure 7.2 we can see that our semi-implicit method produces second-order convergence rates for surface area errors, but is able to take much larger time-steps. If we increase the time-step slightly for the explicit method, we would see the explicit method's failure to converge in surface area error. What we are yet to examine is just how much this affects the runtime of the explicit method; this is explored in the following, subsection 7.3.2.

7.1.3 Runtime Comparison

Here we perform two separate runtime tests. Both tests will be performed with compiled C code on a 2.27 GHz dual core processor with 4 GB of RAM. The first test will compare the explicit and our semi-implicit mean curvature flow methods on a spherical mesh of radius $= 1$, after 5 time-steps at $\triangle t = 10^{-3}$ which totals to 0.005 seconds in real-time. The results can be seen in Table 7.1 .

Table 7.1: A runtime comparison of explicit and semi-implicit mean curvature flow methods on a spherical mesh of radius $= 1$, after 0.005 seconds. Both methods perform 5 time-steps at $\Delta t = 10^{-3}$.

ref	$npts$	sec_{exp}	$steps \times \Delta t$	$sec_{semi-imp}$	$steps \times \Delta t$
1	368	1.88	$5{\times}10^{-3}$	2.26	$5{\times}10^{-3}$
2	1450	7.22	$5{\times}10^{-3}$	10.90	$5{\times}10^{-3}$
3	5804	29.20	$5{\times}10^{-3}$	52.40	$5{\times}10^{-3}$

While Table 7.1 clearly shows the explicit method produces faster runtimes than our semi-implicit method, what is does not show is the explicit method's failure to converge in surface area error and encapsulated volume error (refer to subsection 7.2.1) with a time-step ($\triangle t$) of 10^{-3}. To compare runtimes effectively, we must compare runtimes in which both methods produce convergent results for the surface area errors and encapsulated volume errors. To obtain convergent results with the explicit method, we had to shrink the time-step ($\triangle t$) to 10^{-7}. Ultimately, we still want to evolve the mesh under mean curvature flow for 0.005 seconds in real-time, which means the explicit method will now require 50000 time-steps (t). This is a big advantage of our semi-implicit method. More than just an increase in accuracy and second-order convergence in errors, our semi-implicit method's stability allows for significantly larger time-steps, resulting in extremely shorter runtimes.

Table 7.2: A runtime comparison of explicit and semi-implicit mean curvature flow methods on a spherical mesh of radius $= 1$, after 0.005 seconds. To produce convergent results the explicit method performs 50000 time-steps at $\Delta t = 10^{-7}$ where our semi implicit method performs only 5 time-steps at $\Delta t = 10^{-3}$.

ref	$npts$	sec_{exp}	$steps \times \Delta t$	$sec_{semi-imp}$	$steps \times \Delta t$
1	368	1.84$\times 10^4$	50000$\times 10^{-7}$	2.26	5$\times 10^{-3}$
2	1450	7.86$\times 10^4$	50000$\times 10^{-7}$	10.90	5$\times 10^{-3}$
3	5804	3.28$\times 10^5$	50000$\times 10^{-7}$	52.40	5$\times 10^{-3}$

Table 7.2 exhibits the importance of the stability of our semi-implicit method. To obtain convergent results in errors after 0.005 seconds of evolution under mean curvature flow in real-time for all three refinement levels, the runtime totaled to over 3 days for the explicit method, where our semi-implicit method took slightly more than 1 minute.

7.2 Convergence Results for Semi-Implicit Methods

In this section we will examine the accuracy of our semi-implicit mean curvature flow and surface diffusion methods. Varying by test, we will measure the L^∞ errors, surface area errors, and encapsulated volume errors. By these errors we will then determine the convergence rates.

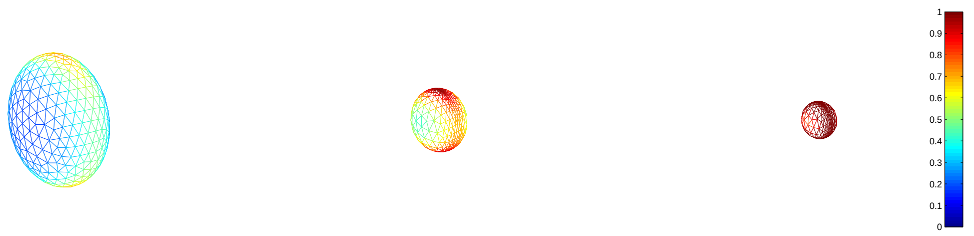

Figure 7.3: The evolution of a closed ellipsoid mesh (axes = 4.0, 6.0, 8.0) under mean curvature flow using our semi-implicit method. The colormap reflects the mean curvature. (left) surface before evolution. (center) surface after 3.00 seconds of evolution. (right) surface after 3.75 seconds of evolution.

7.2.1 Mean Curvature Flow Convergence Results

This test will examine the accuracy and stability of our semi-implicit method. We will work with a spherical mesh of radius = 1.0, an ellipsoid mesh with axes of length 4.0, 6.0, and 8.0, and a torus mesh with major radius = 1.0, and minor radius = 0.25. For each test, we will measure the surface area error and the encapsulated volume error. For our spherical mesh test, we will also measure L^∞ error.

Overall, we want to achieve second-order convergence in surface area errors and encapsulated volume errors. For our sphere test, we use 50 time-steps at $\triangle t = 10^{-3}$ over refinement levels of 368, 1450, and 5804 vertices to obtain the results shown in Table 7.3

As can be seen in Table 7.3 both surface area errors and volume errors exhibit greater than second-order convergence in all cases.

For our ellipsoid test, we will use three levels of refinement for our initial mesh, with 368, 1,450, and 5,804 vertices. For each refinement level, we will use 50 time-steps at

Table 7.3: L^{∞}errors of vertex placement, surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on a spherical mesh of radius = 1, after 50 time-steps at $\triangle t = 10^{-3}$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O	L^{∞}_{err}
1	368	9.17×10^{-2}	n/a	5.26×10^{-2}	n/a	1.68×10^{-4}
2	1450	2.19×10^{-2}	2.07	1.27×10^{-2}	2.05	8.76×10^{-5}
3	5804	3.98×10^{-3}	2.46	2.47×10^{-3}	2.36	8.33×10^{-6}

$\triangle t = 10^{-3}$. For our ellipsoid test, we obtain the results shown in Table 7.4

Table 7.4: Surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on an ellipsoid mesh with axes 4.0, 6.0 and 8.0, after 50 time-steps at $\triangle t = 10^{-3}$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O
1	368	4.78×10^{-1}	n/a	4.84×10^{-1}	n/a
2	1450	1.18×10^{-1}	2.02	1.16×10^{-1}	2.07
3	5804	2.52×10^{-2}	2.22	2.32×10^{-2}	2.32

While results in Table 7.4 are based on approximations, they are very close to the true errors and convergence rates. As can be seen, our obtained errors lead to a greater than second-order convergence in all cases.

Table 7.5: Surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on a torus mesh with major radius = 1.0, and minor radius = 0.25 after 20 time-steps at $\triangle t = 10^{-3}$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O
1	328	8.14×10^{-2}	n/a	6.11×10^{-2}	n/a
2	1348	2.65×10^{-2}	1.62	1.60×10^{-2}	1.93
3	5269	6.82×10^{-3}	1.96	3.70×10^{-3}	2.11

With such success evolving both a sphere and ellipsoid under mean curvature flow using our semi-implicit mean curvature flow method, we wanted to see how our method would handle a more complicated topology. Table 7.5 shows the convergence results for our semi-implicit mean curvature flow method on a torus mesh with major radius = 1.0, and minor radius = 0.25 after 20 time-steps at $\triangle t = 10^{-3}$. As can be seen, our surface area errors still maintain a rate of convergence near 2.00 as the mesh refines.

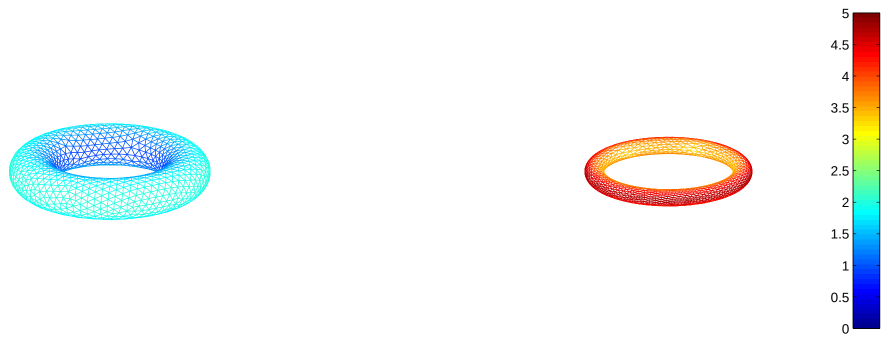

Figure 7.4: The evolution of a closed torus mesh (major radius = 1.0, minor radius = 0.25) under mean curvature flow using our semi-implicit method. The colormap reflects the mean curvature. (left) surface before evolution. (right) surface after 0.09 seconds of evolution.

7.2.2 Surface Diffusion Convergence Results

This test will examine the accuracy and stability of our semi-implicit method for surface diffusion. We will work with the same ellipsoid from before, with axes of length 4.0, 6.0, and 8.0. We will use three levels of refinement for our initial mesh, with our number of vertices being 368, 1,450, and 5,804. For all refinement levels, we will use 50 time-steps at $\triangle t = 10^{-6}$. As with our semi-implicit mean curvature flow method, here we want to achieve second-order convergence in surface area errors. We do not measure volume, as it is ideally unchanged in surface diffusion. We make note that volume changes were less than 1.0% in our experimentation. With our ellipsoid test, we obtain the results seen in Table 7.6

Table 7.6: Surface area errors and convergence rates for our semi-implicit surface diffusion method on an ellipsoid mesh with axes 4.0, 6.0 and 8.0 after 50 time-steps at $\triangle t = 10^{-6}$.

ref	$npts$	$area_{err}$	O
1	368	8.97×10^{-1}	n/a
2	1450	2.20×10^{-1}	2.03
3	5804	4.41×10^{-2}	2.32

As can be seen in Table 7.6, our obtained errors produce higher than second-order convergence.

7.3 Comparison of Semi-Implicit Methods with and without Adaptivity

We will conduct two comparisons in this section. First, we will compare the surface area errors and the encapsulated volume errors of our method with and without adaptivity on a spherical mesh with radius = 1.0. This will confirm that we do not sacrifice accuracy when using mesh adaptivity. The second test will be a runtime comparison on an ellipsoid mesh (axes = 1.5, 2.0, and 8.0). Here, we will show the worth of stability in its ability to use larger time-steps when evolving a poor quality mesh.

7.3.1 Convergence Comparison

Here we compare accuracy of our semi-implicit mean curvature flow method with and without adaptivity on a spherical surface with radius $= 1.0$. We will use three levels of refinement for our initial mesh, with 368, 1,450, and 5,804 vertices. In these tests, we compare surface area error and encapsulated volume error. Overall, we want to see if the accuracy of our semi-implicit method is preserved when using mesh adaptivity. We obtain the results seen in Figure 7.5.

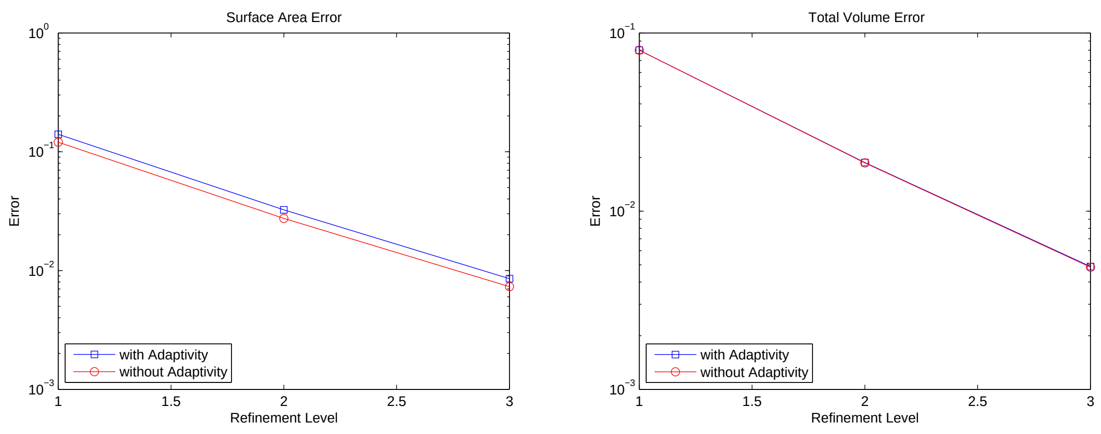

Figure 7.5: A comparison of our semi-implicit mean curvature flow method with and without adaptivity on a spherical mesh with radius $= 1$, after 10 time-steps at $\triangle t = 10^{-3}$ with refinement levels of 368, 1450, and 5804 vertices. (left) displays the surface area errors. Our semi-implicit method converges with a rate of 1.91 without adaptivity, and a rate of 1.92 with adaptivity. (right) displays the encapsulated volume errors. Our semi-implicit method converges with a rate of 1.94 without adaptivity, and a rate of 1.94 with adaptivity.

As can be seen in Figure 7.5, with or without adaptivity, our method produces nearly identical results. Knowing this, we can further examine our semi-implicit method with mesh adaptivity on meshes of poor quality with varying curvature.

7.3.2 Runtime Comparison

Here we perform a runtime test with compiled C code on a 2.27 GHz dual core processor with 4 GB of RAM. This test will compare the runtimes of our semi-implicit mean curvature flow method with and without adaptivity on an ellipsoid mesh with axes = 1.5, 2.0, and 4.0 with refinement levels of 480, 1,896, and 7,680 vertices. Because of the meshes poor element quality, without mesh adaptivity, we will have to use very small time-steps to maintain stability and produce convergence results ($\triangle t = 10^{-7}$). Whereas, with mesh adaptivity we are able to use much larger time-steps and still maintain stability and produce convergence results ($\triangle t = 10^{-2}$). We will run both methods to 0.22 seconds in real-time. Because our method without adaptivity will require so many time-steps, we will run the test for 10 steps at $\triangle t = 10^{-7}$, then multiply our results by 220,000. The results can be seen in Table 7.7 .

Table 7.7: A runtime comparison of our semi-implicit mean curvature flow methods with and without mesh adaptivity on an ellipsoid mesh with axes = 1.5, 2.0, and 4.0 after 0.22 seconds. Step size is adjusted to maintain stability.

ref	$npts$	$sec_{w/adapt}$	$steps \times \Delta t$	$sec_{w/o\,adapt}$	$steps \times \Delta t$
1	480	1.19×10^{1}	22×10^{-2}	4.21×10^{5}	$2{,}200{,}000\times10^{-7}$
2	1896	7.61×10^{1}	22×10^{-2}	1.84×10^{6}	$2{,}200{,}000\times10^{-7}$
3	7680	1.02×10^{3}	22×10^{-2}	3.30×10^{7}	$2{,}200{,}000\times10^{-7}$

Table 7.7 exhibits the importance of the stability of our semi-implicit method through use of mesh adaptivity. Though our runtimes for our semi-implicit method without adaptivity are approximations, they clearly exhibit the importance of using mesh adaptivity to increase stability: To obtain convergent results in errors after 0.22 seconds of evolution under mean curvature flow in real-time for all three refinement levels, the runtime totaled to over a year

when not using mesh adaptivity, where our semi-implicit method with adaptivity took less than 19 minutes.

7.4 Convergence Results of Semi-Implicit Methods with Adaptivity

In this section, we will conduct four tests examining our semi-implicit methods with adaptivity. The first and second tests will measure the accuracy of our semi-implicit method for mean curvature flow, using mesh adaptivity. We will measure the surface area errors and the encapsulated volume errors on ellipsoid meshes with both high- and poor-quality elements. The third and fourth tests will measure the accuracy of our semi-implicit method for surface diffusion, using mesh adaptivity. We will measure the surface area errors and the encapsulated volume errors on ellipsoid meshes with both high- and poor-quality elements.

7.4.1 Mean Curvature Flow with Adaptivity Convergence Results

Here we will examine our semi-implicit mean curvature flow method with mesh adaptivity on meshes that otherwise would cause instability in our standard method. We will use two different meshes in this comparison to exhibit the different ways mesh adaptivity can be utilized with our semi-implicit mean curvature flow method.

7.4.1.1 High-Quality Mesh Convergence - Mean Curvature Flow

For this test, we will use an ellipsoid mesh (axes = 1.5, 2.0, 8.0) with high quality elements. For a triangular mesh, this means that the elements are all close to equilateral triangles. Normally, this means that our semi-implicit mean curvature flow method would be very stable. However, the shape of the mesh causes an issue. Because of its elongated shaped, as the mesh evolves, vertices become crowded and the element quality diminishes. Consequently, the mesh is entirely unable to evolve past a certain point. Even lessening the time-step cannot correct the issue, as elements will continue to decrease in size and quality, disallowing our semi-implicit mean curvature flow method to work properly. This can be corrected if we utilize mesh adaptivity. By doing so, this allows the mesh to evolve without the elements becoming crowded, as can be seen in Figure 6.2. The accuracy and convergence over three refinement levels of 368, 1,450, and 5,804 vertices can be seen in Table 7.8.

Table 7.8: Surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on an ellipsoid mesh with axes 1.5, 2.0 and 8.0, after 22 time-steps at $\triangle t = 10^{-2}$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O
1	368	9.18×10^{-1}	n/a	4.20×10^{-1}	n/a
2	1450	2.18×10^{-1}	2.08	1.03×10^{-1}	2.02
3	5804	4.21×10^{-2}	2.37	2.13×10^{-2}	2.28

7.4.1.2 Poor-Quality Mesh Convergence - Mean Curvature Flow

Mesh adaptivity can also be used if we have a poor quality mesh to start with. Figure 7.6 clearly shows how mesh adaptivity drastically improves element quality, which then allows for bigger time-steps when evolving the mesh under mean curvature flow using our semi-implicit method. For this test we use an ellipsoid (axes 1.5, 2.0, and 4.0) with refinement levels of 480, 1,896, and 7,680 vertices. To evolve 0.22 seconds in real-time, our semi-implicit mean curvature flow method must decrease its time-steps to $\triangle t = 10^{-7}$ to maintain stability, requiring 2200000 steps. Whereas, if we utilized mesh adaptivity, we only need to take 22 steps at $\triangle t = 10^{-2}$. In doing so, we significantly lessen our runtime (subsection 7.3.2) while maintaining stability and producing very accurate results, as can be seen in Table 7.9 .

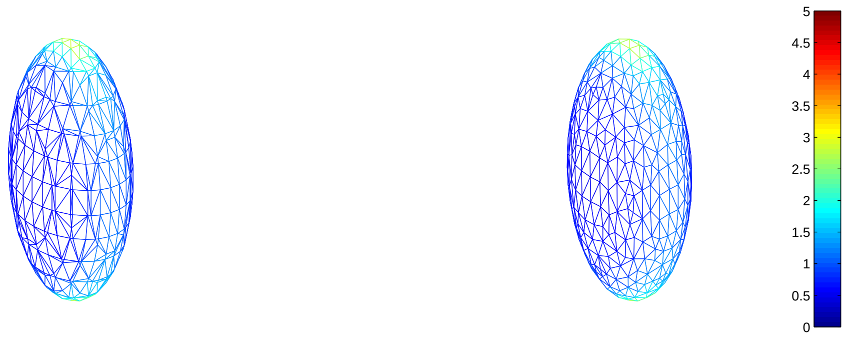

Figure 7.6: The evolution of a closed ellipsoid mesh (axes = 1.5, 2, 4) using mesh adaptivity. (left) surface before evolution. (right) surface after evolution.

Table 7.9: Surface area errors, encapsulated volume errors and convergence rates for our semi-implicit mean curvature flow method on an ellipsoid mesh with axes 1.5, 2.0 and 4.0, after 22 time-steps at $\triangle t = 10^{-2}$.

ref	$npts$	$area_{err}$	O	$volume_{err}$	O
1	480	8.76×10^{-1}	n/a	3.75×10^{-1}	n/a
2	1896	2.00×10^{-1}	2.13	9.22×10^{-2}	2.02
3	7680	1.78×10^{-2}	2.41	1.78×10^{-2}	2.38

7.4.2 Surface Diffusion with Adaptivity Convergence Results

Here we will examine our semi-implicit surface diffusion method with mesh adaptivity on two meshes. These meshes are of an ellipsoid with axes 2.0, 2.0 and 3.0.

7.4.2.1 High-Quality Mesh Convergence - Surface Diffusion

The first mesh we will use for this test has elements of high quality. For a triangular mesh, this means that the elements are all close to equilateral triangles. The accuracy and convergence rates over three refinement levels of 368, 1,450, and 5,804 vertices after 0.05 seconds of evolution can be seen in Table 7.10.

Table 7.10: Surface area errors and convergence rates for our semi-implicit surface diffusion method on an ellipsoid mesh with axes 2.0, 2.0 and 3.0, after 100 time-steps at $\triangle t = 10^{-4}$.

ref	$npts$	$area_{err}$	O
1	368	1.26×10^{-1}	n/a
2	1450	3.12×10^{-2}	2.01
3	5804	6.08×10^{-3}	2.36

7.4.2.2 Poor-Quality Mesh Convergence - Surface Diffusion

As can be seen in Table 7.10, with the inclusion of mesh adaptivity, our semi-implicit surface diffusion method maintains second-order convergence in surface area errors on a surface with high quality elements. Additionally, we will examine our semi-implicit surface diffusion method with mesh adaptivity on a mesh with poor quality elements. We will use another ellipsoid mesh with axes 2.0, 2.0 and 3.0. Again, we are able to maintain second-order convergence over three refinement levels of 480, 1,896, and 7,680 vertices after 0.05 seconds of evolution, as can be seen in Table 7.11 .

Table 7.11: Surface area errors and convergence rates for our semi-implicit surface diffusion method on an ellipsoid mesh with axes 2.0, 2.0 and 3.0, after 100 time-steps at $\triangle t = 10^{-4}$.

ref	$npts$	$area_{err}$	O
1	480	1.48$\times 10^{-1}$	n/a
2	1896	3.15$\times 10^{-2}$	2.23
3	7680	5.12$\times 10^{-3}$	2.62

7.5 Comparison of FEM and Semi-Implicit Method with Adaptivity

In addition to exhibiting the accuracy and rate of convergence of our semi-implicit method with use of mesh adaptivity, we also want to show how our method compares to other existing methods. Bänsch et al. [4] proposed a mixed finite element method (FEM) for solving the surface diffusion equation governed by the surface Laplacian of the mean curvature. Their method uses a semi-implicit time discretization to split the fourth order, highly nonlinear geometric PDE into four linear (up to second order) elliptic equations involving both scalar and vector forms of the curvature and velocity of the discrete surface. The discretization merely requires piecewise-linear elements which can be used to enforce a weak formulation of the system of PDEs which simplifies implementation. The resulting linear algebraic system of equations can be solved semi-implicitly utilizing a Schur complement approach. For mesh adaptivity, Bänsch et al. [4] outlines several schemes to improve mesh quality and avoid defects associated with clustered vertices and singularities, which include mesh regularization, time adaptivity, spatial adaptivity, and angle-width control. In this comparison, only mesh regularization and time adaptivity are implemented as they improve mesh quality but do not affect mesh connectivity, allowing analysis of convergence rates to be done in a consistent manner.

7.5.1 High-Quality Mesh Comparison

The mesh we will use for this test has elements of good quality. For a triangular mesh, this means that the elements are all close to equilateral triangles. Since ideal surface diffusion flow maintains its volume, we will measure accuracy and rate of convergence using only surface area. Please note that through evolution via surface diffusion flow, both methods result in volume changes less than 1.0%. We evolve an ellipsoid mesh (axes = 2.0, 2.0 and 3.0) over 0.05 seconds using 100 time-steps at $\triangle t = 5 \times 10^{-4}$. The accuracy and convergence over three refinement levels of 368, 1,450, and 5,804 vertices can be seen in Figure 7.7.

Figure 7.7 clearly shows the strength of our method. While Bänsch et al. [4] mixed finite element method (FEM) converges with a rate of 1.03, our method achieves a convergence rate of 2.36.

7.5.2 Poor-Quality Mesh Comparison

Our second mesh will be one of “poor” quality, meaning that its elements will not be

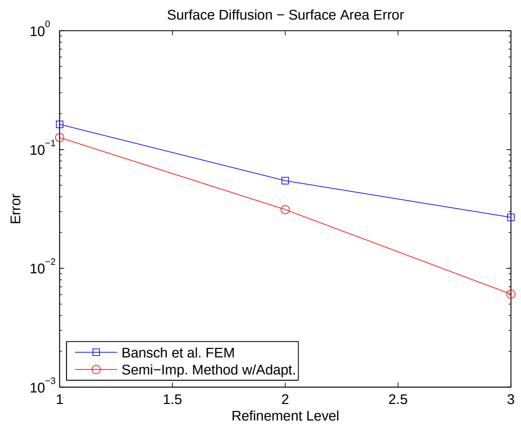

Figure 7.7: A comparison of our semi-implicit mean curvature flow method with adaptivity and Bänsch et al. [4] mixed finite element method (FEM) on a "good" quality ellipsoid mesh with axes 2.0, 2.0, and 3.0, after 0.05 seconds. Displayed are the surface area errors, with Bänsch et al. [4] method converging with a rate of 1.03, and our semi-implicit method with adaptivity converging with a rate of 2.36.

consistent and will vary significantly in angle values and size. This will exhibit a different way mesh adaptivity can be utilized. Again, we evolve an ellipsoid mesh (axes = 2.0, 2.0 and 3.0) over 0.05 seconds using 100 time-steps at $\triangle t = 5 \times 10^{-4}$. The accuracy and convergence over three refinement levels of 480, 1896, and 7680 vertices can be seen in Figure 7.7.

As stated in Bänsch et al. [4], poor quality meshes are handled well with their mesh adaptivity techniques. As a result, we see stability and accuracy maintained in their results. However, with our mesh adaptivity techniques, our method successfully maintains stability and accuracy as well, while producing much stronger convergence rates. As can be seen in Figure 7.8, Bänsch et al. [4] method's surface area errors converge with a rate of 1.09, where our semi-implicit method with mesh adaptivity produces a convergence rate of 2.62.

While we would like to report a runtime comparison, we are unable to do so justifiably,

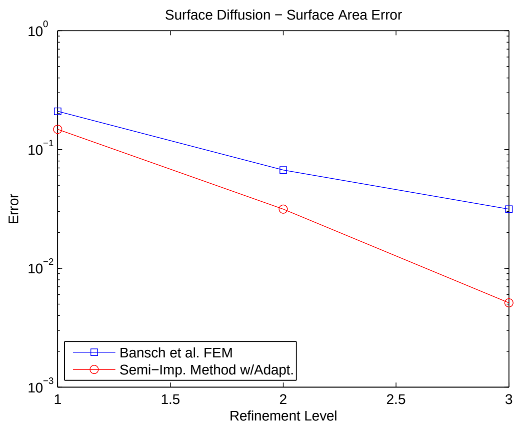

Figure 7.8: A comparison of our semi-implicit surface diffusion method with adaptivity and Bänsch et al. [4] mixed finite element method (FEM) on a "poor" quality ellipsoid mesh with axes 2.0, 2.0, and 3.0, after 0.05 seconds. Displayed are the surface area errors, with Bänsch et al. [4] method converging with a rate of 1.09, and our semi-implicit method with adaptivity converging with a rate of 2.62.

since this is our own implementation of Bänsch et al. [4] finite element method. We cannot guarantee runtimes to effectively represent their code. However, we would like to mention that in our experimentation, Bänsch et al. [4] method's runtime was not significantly different from ours.

8 Conclusions and Future Work

Many methods have tackled mean curvature flow and surface diffusion. To solve these equations, some have used level set methods [12, 53]. Osher and Sethian [53] devised an algorithm for front propagation with curvature-dependent speed, where the equation is treated as a Hamilton-Jacobi equation with a viscosity term and is numerically solved using techniques from hyperbolic conservation laws. While Chopp and Sethian [12] solved the surface diffusion equation, with the fourth order operator calculated through a hybrid narrow band approach near the interface. Such methods based on level set can easily handle singularities during propagation and a non-smooth initial front, but fail to provide accuracy analysis. Additionally, the time-step for these explicit methods has to be very small to ensure stability.

Other methods use semi-implicit methods to address the limitation of small time-steps in the explicit approach [59, 4]. Smereka [59] introduced a semi-implicit scheme for mean curvature flow based on the level set framework, while Bänsch et al. [4] proposed a semi-implicit finite element method for solving the diffusion equation governed by the surface Laplacian of the mean curvature. While both methods allow for larger time-steps, both fail to provide substantial accuracy analysis or proof of second-order convergence. The shortcoming that can be seen clearly in all existing methods is that they fail to meet all three basic goals: substantial accuracy analysis, allowance for relatively large time-steps, and proof of second-order convergence. This has motivated the creation of our semi-implicit method which achieves all of these things.

For substantial accuracy analysis we use a continuous fitting method where once we have the local polynomial, we can use the continuous algebraic formulae to calculate various differential operators of both the surface itself and functions defined on the surface. Such a method provides robust and provable convergent results compared to discrete schemes. Additionally, our method also allows for relatively large time-steps through its semi-implicit form. While explicit methods are restricting by their need for very small time-steps, and implicit methods are costly due to their non-linear nature, we find a compromise between the two which allows for larger time-steps, which ultimately reduces runtime. Our experimental results show clearly that our semi-implicit methods also achieve second-order convergence. This is very significant.

With the addition of mesh adaptivity, we are able to maintain second-order convergence for meshes of varying quality and through the issue of crowding vertices due to the evolution process. This allows us to compete with other methods, such as Bänsch et al. [4] which also include the use of mesh adaptivity. Because of this, we are able to make a direct comparison between both methods. Our results exhibit the superiority of our method, as it was able to achieve second-order convergence, where Bänsch et al. [4] method only produced first-order convergence.

With all of our goals met, we look forward to improving our methods in our future work. For both our semi-implicit mean curvature flow method and our semi-implicit surface diffusion method we are exploring the idea of combining our generalized finite difference method framework with that of a finite element method structure. By doing so, we would hope to achieve a method which maintains second-order convergence, while also preserving matrix symmetry in calculations. The result would be a method comparable to our current method in terms of accuracy, but would allow for even larger time-steps.

Bibliography

[1] E. Bänsch. Finite element discretization of the navier-stokes equations with a free capillary surface. *Numer. Math.*, 88:203–235, 2001.

[2] E. Bänsch, P. Morin, and R.H. Nochetto. Finite element methods for surface diffusion. *Free Boundary Problems, International Series of Numerical Mathematics*, 147:53–63, 2003.

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

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

[5] A.J. Bernoff, A.L. Bertozzi, and T.P. Witelski. Axisymmetric surface diffusion: dynamics and stability of self-similar pinchoff. *J. Stat. Phys.*, 93:725–776, 1998.

[6] H. Borouchaki and P. Frey. Adaptive triangular-quadrilateral mesh generation. *Int. J. Numer. Meth. Engr.*, pages 915–934, 1998.

[7] K.A. Brakke. The Surface Evolver. *Experimental Mathematics*, 1(2):141–165, 1992.

[8] K.A. Brakke. Surface Evolver. *Philosophical Transactions of the Royal Society A Mathematical Physical and Engineering Sciences*, 354(1715):2143–2157, 1996.

[9] G. Buscaglia and E. Dari. Anistropic mesh optimization and its application in adaptivity. *Int. J. Numer. Meth. Engr.*, 40:4119–4136, 1997.

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

[11] Y. Chen, Y. Giga, and S. Goto. Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations. *Proc Japan Acad Ser A Math Sci*, 65(3):207–210, 1991.

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

[13] B.D. Coleman, R.S. Falk, and M. Moakher. Space-time finite element methods for surface diffusion with applications to the theory of stability of cylinders. *SIAM J. Sci. Comp.*, 17:1434–1448, 1996.

[14] K. Deckelnick, G. Dziuk, and C. Elliott. Error analysis of a semidiscrete numerical scheme for diffusion in axially symmetric surfaces. *SIAM J. Numer. Anal.*, 41:2161–2179, 2003.

[15] K. Deckelnick, G. Dziuk, and C. Elliott. Computation of geometric partial differential equations and mean curvature flow. *Acta Numerica*, 14:139–232, 2005.

[16] K. Deckelnick, G. Dziuk, and C. Elliott. Fully Discrete Finite Element Approximation for Anisotropic Surface Diffusion of Graphs. *SIAM J Num Anal*, 43(33):1112, 2005.

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

[18] V. Dolejsi. Anisotropic mesh adaptation for finite volume and finite element methods on triangular meshes. *Comput. Vis. Sci.*, 1:165–178, 1998.

[19] J. Du, B. Fix, J. Glimm, X. Jia, X. Li, Y. Li, and L. Wu. A simple package for front tracking. *J. Comput. Phys.*, 213:613–628, 2006.

[20] G. Dziuk. An algorithm for evolutionary surfaces. *Numer. Math.*, 58:603–611, 1991.

[21] C.M. Elliott and H. Garcke. Existence results for diffusive surface motion laws. *Adv. Math. Sci. Appl.*, 7:465–488, 1997.

[22] L.C. Evans and J. Spruck. Motion of Level Sets by Mean Curvature IV. *Journal of Differential Geometry*, 33(1):635–681, 1991.

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

[24] M.S. Floater and K. Hormann. *Surface parameterization: a tutorial and survey*, pages 157–186. Springer Verlag, 2005.

[25] P.J. Frey. About Surface Remeshing, 2000.

[26] R.V. Garimella and B.K. Swartz. Curvature estimation for unstructured triangulations of surfaces, 2003.

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

[28] J. Glimm, M.J. Graham, J.W. Grove, X.-L. Li, T.M. Smith, D. Tan, F. Tangerman, and Q. Zhang. Front tracking in two and three dimensions. *Comput. Math. Appl.*, 35(7):1–11, 1998.

[29] J. Glimm, J.W. Grove, X.-L. Li, K.-M. Shyue, Q. Zhang, and Y. Zeng. *Three dimensional front tracking*, volume 19. 1998.

[30] J. Glimm, J.W. Grove, X.-L. Li, and N. Zhao. *Simple front tracking*, volume 238, pages 138–149. Amer. Math. Soc., Providence, RI, 1999.

[31] J. Glimm, E. Isaacson, D. Marchesin, and O. McBryan. Front tracking for hyperbolic systems. *Adv. Appl. Math.*, 2:91–119, 1981.

[32] J. Glimm, C Klingenberg, and O. McBryan. Front tracking and two-dimensional riemann problems. *Adv. Appl. Math.*, 6:259–290, 1985.

[33] J. Glimm, X.-L. Li, Y.-J. Liu, Z.L. Xu, and N. Zhao. Conservative front tracking with improved accuracy. *SIAMJNA*, pages 1926–1947, 2003.

[34] J. Glimm, O. McBryan, R. Menikoff, and D. Sharp. Front tracking applied to rayleigh-taylor instability. *SIAM J. Sci. Stat. Comput.*, pages 230–251, 1986.

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

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

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

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

[39] B. Hamann. Curvature approximation for triangulated surfaces. In *Geometric modelling: Springer Computing Supplementum*, pages 139–153. 1993.

[40] M. T. Heath. *Scientific Computing: An Introductory Survey*. "McGraw–Hill", "New York", "2nd" edition, "2002".

[41] P.S. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. *Comput. Geom.*, pages 49–65, 1999.

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

[43] X. Jiao, A. Colombi, X. Ni, and J. Hart. *"Anisotropic Mesh Adaptation for Evolving Triangulated Surfaces"*. 2006.

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

[45] X. Jiao, D. Wang, and H. Zha. Simple and effective variational optimization of surface and volume triangulations. *Engineering with Computers*, 2010.

[46] 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.

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

[48] T. Langer, A.G. Belyaev, and H.P. Seidel. Analysis And Design Of Discrete Normals And Curvatures. Technical report, Max-Planck-Institut Fur Informatik, 2005.

[49] X. Li, M.S. Shephard, and M.W. Beall. 3d anistropic mesh adaptation by mesh modification. *Comput. Methods Appl. Mech. Engrg.*, pages 4915–4950, 2005.

[50] U.F. Mayer. Numerical solutions for the surface diffusion flow in three space dimensions. *Comput. Appl. Math.*, 20:361–379, 2001.

[51] M. Meyer, M. Desbrun andP. Schröder, and A.H. Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds, 2002.

[52] R.H. Nochetto, M. Paolini, and C. Verdi. A Dynamic Mesh Algorithm for Curvature Dependent Evolving Interfaces. *J. Comput. Phys.*, pages 296–310, 1996.

[53] S. Osher and J.A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J Comp Phys*, 79(1988):12–49, 1987.

[54] A. Razdan and M. Bae. Curvature estimation scheme for triangle meshes using biquadratic Bézier patches. 37:1481–1491, 2005.

[55] S. Rosenberg. *The Laplacian on a Riemannian Manifold*. Cambridge University Press, 1998.

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

[57] J.A. Sethian. Curvature and the Evolution of Fronts. *Communication of Mathematical Physics*, 101:487–499, 1985.

[58] J.A. Sethian. *Level Set Methods and Fast Marching Methods (Second Edition)*. Cambridge University Press, 1999.

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

[60] J.M. Sullivan. Curvature Measures for Discrete Surfaces. 2002.

[61] T. Surazhsky, E. Magid, O. Soldea, G. Elber, and E. Rivlin. A comparison of gaussian and mean curvatures estimation methods on triangular meshes. In *2003 IEEE international conference on robotics and automation (ICRA2003)*, pages 1021–1026, 2003.

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

[63] L.N. Trefethen, V. De Silva, and J. Langford. *Numerical Linear Algebra*. SIAM, 1997.

[64] A. Voigt. PDE's on surfaces – a diffuse interface approach. *Interface*, 2(2):1–15, 2005.

[65] D. Wang, B.L. Clark, and X. Jiao. An analysis and comparison of parameterization-based computation of differential quantities for discrete surfaces. *Computer Aided Geometric Design*, 26(5):510–527, 2009.

[66] B. White. Evolution of Curves and Surfaces by Mean Curvature. *ICM*, I:525–538, 2002.

[67] G. Xu. Convergence of discrete laplace-beltrami operators over surfaces. *Comput. Math. Appl.*, pages 347–360, 2004.

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

[69] G. Xu and Q. Zhang. G2G2 surface modeling using minimal mean-curvature-variation flow. *Computer-Aided Design*, 39(5):342–351, 2007.

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