SIAM J. SCI. COMPUT.
Vol. 31, No. 6, pp. 4330–4350
© 2009 Society for Industrial and Applied Mathematics

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

THE IMPLICIT CLOSEST POINT METHOD FOR THE NUMERICAL SOLUTION OF PARTIAL DIFFERENTIAL EQUATIONS ON SURFACES*

COLIN B. MACDONALD† AND STEVEN J. RUUTH‡

Abstract. Many applications in the natural and applied sciences require the solutions of partial differential equations (PDEs) on surfaces or more general manifolds. The closest point method is a simple and accurate embedding method for numerically approximating PDEs on rather general smooth surfaces. However, the original formulation is designed to use explicit time stepping. This may lead to a strict time-step restriction for some important PDEs such as those involving the Laplace–Beltrami operator or higher-order derivative operators. To achieve improved stability and efficiency, we introduce a new implicit closest point method for surface PDEs. The method allows for large, stable time steps while retaining the principal benefits of the original method. In particular, it maintains the order of accuracy of the discretization of the underlying embedding PDE, it works on sharply defined bands without degrading the accuracy of the method, and it applies to general smooth surfaces. It also is very simple and may be applied to a rather general class of surface PDEs. Convergence studies for the in-surface heat equation and a fourth-order biharmonic problem are given to illustrate the accuracy of the method. We demonstrate the flexibility and generality of the method by treating flows involving diffusion, reaction-diffusion, and fourth-order spatial derivatives on a variety of interesting surfaces including surfaces of mixed codimension.

Key words. closest point method, surface computation, implicit surfaces, partial differential equations, implicit time stepping, Laplace–Beltrami operator, biharmonic operator, surface diffusion

AMS subject classifications. 65M06, 58J35, 65M20

DOI. 10.1137/080740003

1. Introduction.

Partial differential equations (PDEs) are ubiquitous throughout the natural and applied sciences. Because analytical solutions are rarely possible, the practical importance of accurate and efficient numerical methods for PDEs cannot be overemphasized. There has been a great effort made to develop numerical methods for many important classes of PDEs. These efforts often focus on finding solutions in $\mathbb{R}^1$, $\mathbb{R}^2$, and $\mathbb{R}^3$. But problems involving general differential equations also arise on manifolds (such as a two-dimensional curved surface or a one-dimensional curved filament) in some embedding space (such as $\mathbb{R}^3$). For example, in material science one might wish to examine phase change of a material on a surface [36]. In biological modeling, one might study wound healing on skin [27] or the evolution of a pattern on an animal coat [24]. Computer graphics and image processing also frequently use PDEs on surfaces; for example, they might use such methods to place a texture on a surface [40], segment out objects defined in surface textures [37], or restore a damaged pattern [5] on a surface such as a vase.

PDEs on surfaces are traditionally handled by one of several techniques. These include finite element discretization on a triangulation of the surface [18, 9],

*Received by the editors November 6, 2008; accepted for publication (in revised form) July 28, 2009; published electronically December 16, 2009.
http://www.siam.org/journals/sisc/31-6/74000.html
†Department of Mathematics, University of California, Los Angeles, CA 90095 (cbm@math.ucla.edu). The work of this author was supported by a grant from NSERC Canada, an NSERC postdoctoral fellowship, and NSF grant CCF-0321917.
‡Department of Mathematics, Simon Fraser University, Burnaby, British Columbia, V5A 1S6 Canada (sruuth@sfu.ca). The work of this author was partially supported by a grant from NSERC Canada.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

discretizations based on some parameterization of the surface [11, 35, 16], and embedding techniques [6, 8, 7, 14, 15, 10, 25] which solve some embedding PDE in a small region near the surface. Naturally, each class of methods has its own advantages and disadvantages. The finite element approach can treat elliptic equations but is often less effective when convective or advective terms arise. Parameterization methods can be effective in certain specialized situations but have the deficiency of introducing distortions and singularities into the method through the parameterization [11]. Methods within the embedding class have the attractive feature of being able to work in standard Cartesian coordinates, using Cartesian grid methods. Frequently, however, such methods are built around level set representations of the surface, a property that complicates the treatment of general open surfaces or surfaces without orientation, because a standard level set representation needs a well-defined inside and outside. Furthermore, level-set-based embedding methods typically impose some artificial boundary conditions when computations are carried out in a small region surrounding the surface. The imposition of such artificial boundary conditions leads to low-order accurate results for some important PDEs [14].

Within the embedding class, a method (the closest point method) was developed and analyzed in [30] which uses a closest point representation of the underlying surface $\mathcal{S}$. A closest point representation assumes that for any point $\boldsymbol{x}$ in the embedding space $\mathbb{R}^d$ containing $\mathcal{S}$, a (possibly nonunique) point $\text{cp}(\boldsymbol{x}) \in \mathcal{S}$ is known which is closest to $\boldsymbol{x}$ in Euclidean distance. For smooth surfaces, the closest point representation is unique in a narrow band around the surface, the width of which depends on, for example, the principal curvatures. The closest point representation allows for general surfaces with boundaries and does not require the surface to have an inside/outside. The surface can be of any codimension or even of mixed codimension.

Given a closest point representation, a *closest point extension* which propagates data off the surface $\mathcal{S}$ into the surrounding space $\mathbb{R}^d$ is given by $u(\text{cp}(\boldsymbol{x}))$. Closest point extensions lead to simplified derivative calculations in the embedding space because they result in functions which are constant in the direction normal to the surface, at least within a neighborhood of a smooth surface. The relationship between surface derivatives and derivatives in the embedding space may be made precise using two fundamental principles [30]. Let ∇ denote the "standard" gradient in $\mathbb{R}^d$, and let $\nabla_{\mathcal{S}}$ denote the gradient intrinsic to the smooth surface $\mathcal{S}$; then the following hold:

PRINCIPLE 1 (gradients). *For points $\boldsymbol{x}$ on a smooth surface, $\nabla_{\mathcal{S}} u(\boldsymbol{x}) = \nabla\left(u(\text{cp}(\boldsymbol{x}))\right)$ because the function $u(\text{cp}(\boldsymbol{x}))$ is constant in the normal direction and therefore only varies along the surface. In other words, at points $\boldsymbol{x}$ on the surface,* intrinsic surface gradients $\nabla_{\mathcal{S}} u(\boldsymbol{x})$ *are the same as* gradients *of* $u(\text{cp}(\boldsymbol{x}))$.

PRINCIPLE 2 (divergence). *Let $\nabla_{\mathcal{S}}\cdot$ denote the divergence operator intrinsic to a smooth surface $\mathcal{S}$, and let $\boldsymbol{v}$ be any vector field on $\mathbb{R}^d$ that is tangent at $\mathcal{S}$ and also tangent at all surfaces displaced by a fixed distance from $\mathcal{S}$ (i.e., all surfaces defined as level sets of the distance function to $\mathcal{S}$). Then at points $\boldsymbol{x}$ on the surface $\nabla \cdot \boldsymbol{v}(\boldsymbol{x}) = \nabla_{\mathcal{S}} \cdot \boldsymbol{v}(\boldsymbol{x})$.*

Combinations of these two principles may be made to encompass very general second-order differential operators, including the Laplace–Beltrami operator $\Delta_{\mathcal{S}} = \nabla_{\mathcal{S}} \cdot \nabla_{\mathcal{S}}$, in-surface curvature, and nonlinear diffusion operators [30]. Indeed, even higher-order and more general derivative replacements may be considered by carrying out multiple closest point extensions as first described in [30] and demonstrated in practice in sections 5 and 6. Notice that the introduction of a closest point extension step leads us to consider standard PDEs in the embedding space for important classes

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

of surface PDEs. This in turn enables the use of standard discretizations for the underlying embedding PDEs.

The closest point method has been applied to a variety of time-dependent problems including in-surface advection, diffusion, reaction-diffusion, and Hamilton–Jacobi equations [30, 23, 22]. In these previous works, time stepping was performed explicitly and the numerical solution was propagated by alternating between two steps:

1. evolution of an embedding PDE on a regular Cartesian grid enveloping the surface using an explicit time-stepping scheme;
2. extension of the surface data to the surrounding grid points with a closest point extension; i.e., for grid point $\boldsymbol{x}$, assign $u(\boldsymbol{x}) = u(\mathrm{cp}(\boldsymbol{x}))$. As $\mathrm{cp}(\boldsymbol{x})$ is typically not a grid point, this extension is an interpolation step.

This explicit approach to the closest point method works very well for nonstiff problems such as arise from Hamilton–Jacobi equations. The method achieves high-order accuracy [23] using standard spatial discretizations and explicit Runge–Kutta time-stepping methods with time-step sizes $\Delta t = O(\Delta x)$, i.e., on the order of the spatial grid. It also avoids introducing artificial boundary conditions when computations are limited to a neighborhood of the surface [30, 23], thereby enabling the method to localize computations while maintaining the accuracy of the underlying discretization.

Unfortunately, explicit time-stepping methods are a poor choice for many important PDEs. For example, a severe stability time-step restriction may arise when spatially discretized diffusion, biharmonic, or other stiff terms appear. Such time-step restrictions can lead to inefficient or impractical algorithms (for example, when biharmonic terms arise, we might anticipate a step-size restriction like $\Delta t \sim \Delta x^4$). Severe time-step restrictions are also possible when local grid refinement is carried out because the time step will scale like some power of the minimal grid spacing. On the other hand, the use of implicit time-stepping methods typically leads to mild stability time-step restrictions, making them natural candidates for treating the stiff systems arising from certain PDEs. This motivates the development and analysis of an implicit closest point method.

The remainder of this paper unfolds as follows. In section 2, the implicit closest point method is presented. It uses implicit linear multistep or Runge–Kutta time-stepping schemes and allows for large time steps. When the implicit closest point method is applied to linear problems such as the in-surface heat equation, each time step requires a linear system solve: section 3 discusses the properties of that system. Numerical results in sections 4 and 5 demonstrate the high-order accuracy of the method on the in-surface heat equation and a biharmonic problem. Several applications are presented, including blurring an image on a triangulated surface, modeling heat flow on a complicated surface with components of varying codimension, in-surface pattern formation, and fourth-order "surface diffusion" for an interface on a surface. Section 6 extends the method to odd-order differential terms.

2. The implicit closest point method. In this section we derive the implicit closest point method. Before giving a more general construction, we illustrate the main ideas for the simple but important case of in-surface heat flow.

2.1. The closest point method for in-surface heat flow. To motivate the method, let $\mathcal{S}$ be a surface embedded in $\mathbb{R}^d$ and consider the in-surface heat equation

$$u_t = \Delta_{\mathcal{S}} u, \tag{2.1a}$$

$$u(0, \boldsymbol{x}) = u_0(\boldsymbol{x}). \tag{2.1b}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Based on the Principles 1 and 2, we may replace the Laplace–Beltrami operator $\Delta_{\mathcal{S}}$ by the standard Laplacian Δ in the embedding space $\mathbb{R}^d$, provided the argument of the right-hand side is replaced by the closest point operator $\mathrm{cp}(\boldsymbol{x})$. This leads to the *embedding PDE*

$$u_t(t,\boldsymbol{x}) = \Delta\left(u(t,\mathrm{cp}(\boldsymbol{x}))\right), \tag{2.2a}$$

$$u(0,\boldsymbol{x}) = u_0(\mathrm{cp}(\boldsymbol{x})), \tag{2.2b}$$

which has the property that the solutions of (2.1) and (2.2) agree on the surface [30].

The implicit closest point method is based on discretizing (2.2) directly. For ease of exposition, in this example we assume the surface $\mathcal{S}$ is a curve embedded in two dimensions (2D) and consider a second-order centered finite difference scheme applied to the Laplacian Δ in (2.2). This yields

$$\begin{aligned}\frac{\partial}{\partial t}u(t,x,y) = \frac{1}{\Delta x^2}\Big(&-4u(t,\mathrm{cp}(x,y)) + u(t,\mathrm{cp}(x+\Delta x,y)) + u(t,\mathrm{cp}(x-\Delta x,y))\\ &+ u(t,\mathrm{cp}(x,y+\Delta x)) + u(t,\mathrm{cp}(x,y-\Delta x))\Big) + O(\Delta x^2),\end{aligned} \tag{2.3a}$$

$$u(0,x,y) = u_0(\mathrm{cp}(x,y)), \tag{2.3b}$$

where x and y are still continuous variables (this is not yet a spatial discretization). The spatial discretization can be completed by choosing a grid $\{(x_i,y_j)\}$, approximating the solution with nodal values u_{ij}, and using interpolation to replace the value of $u(\mathrm{cp}(x_i,y_j))$ by a linear combination of nearby nodal values. As the value of $\mathrm{cp}(x_i,y_j)$ is known beforehand, the interpolation weights can be precomputed using barycentric Lagrange interpolation [4] (in a dimension-by-dimension fashion). These interpolation weights and the differentiation weights $\frac{1}{\Delta x^2}\{-4,1,1,1,1\}$ can then be combined into a matrix (see section 2.2 for details) which approximates $\Delta\left(u(t,\mathrm{cp}(x,y))\right)$. Thus, this method-of-lines approach yields a linear system of coupled ODEs in time for $u_{ij}(t) \approx u(t,x_i,y_j)$. The discretization may then be completed by applying standard implicit Runge–Kutta or linear multistep methods.

As we shall see in section 2.2.3, the system resulting from (2.3) is unstable. An alternate consistent formulation which benefits from improved diagonal dominance and improved stability arises when the redundant closest point operators are removed from the discretization (where diagonal terms appear in the discretization, consistency does not require an interpolation to the closest point on the surface). This leads to the following stabilized form:

$$\begin{aligned}\frac{\partial}{\partial t}u(t,x,y) = \frac{1}{\Delta x^2}\Big(&-4u(t,x,y) + u(t,\mathrm{cp}(x+\Delta x,y)) + u(t,\mathrm{cp}(x-\Delta x,y))\\ &+ u(t,\mathrm{cp}(x,y+\Delta x)) + u(t,\mathrm{cp}(x,y-\Delta x))\Big) + O(\Delta x^2),\end{aligned} \tag{2.4a}$$

$$u(0,x,y) = u_0(\mathrm{cp}(x,y)), \tag{2.4b}$$

where the only change is that the diagonal entries no longer involve the closest point operator cp. Note that (2.4) and the previous (2.3) agree at the surface because for points (x,y) on the surface we have $\mathrm{cp}(x,y) = (x,y)$. This ensures that the solution of (2.4) is consistent with that of (2.1).

2.2. Matrix formulation of the closest point method. The general matrix formulation of the closest point method introduces a matrix operator (the "extension matrix") to discretize all instances of the closest point operator. Our approach explicitly

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

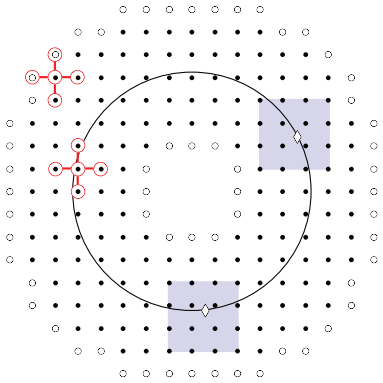


FIG. 2.1. *Example of the lists of grid points L (indicated by $\bullet$) and G (indicated by $\circ$) where the curve $\mathcal{S}$ is a circle. The interpolation stencil is a 4×4 grid (arising, for example, from using degree $p = 3$ barycentric Lagrange interpolation), and the shaded regions illustrate the use of this stencil at the two points on the circle indicated by $\diamond$. Five-point differentiation stencils are shown for two example points in L, in one case illustrating the use of ghost points.*

constructs the extension matrix. This enables the use of either direct or iterative matrix solvers and makes the analysis of the ODE system straightforward. The alternative approach to computing an iterative solution of the implicit equations without explicitly constructing the extension matrix has the advantage of being closer with respect to implementation to the explicit closest point method. This alternative approach is discussed briefly in section 3.2 and will be treated in detail as part of future work.

We now give the details on the general matrix formulation. Suppose the surface $\mathcal{S}$ is embedded in $\mathbb{R}^d$. Assume we have an interpolation scheme where the interpolated value is a linear combination of the values at neighboring grid points in, for example, a hypercube in $\mathbb{R}^d$. These neighboring grid points form the *interpolation stencil* of the scheme. Various interpolation techniques are possible. Similar to the explicit closest point method in [30], we will use barycentric Lagrange interpolation [4] of degree p, in a dimension-by-dimension fashion, so the hypercube has $p + 1$ points in each coordinate direction.

Now consider two discrete ordered lists of points in the embedding space. The first is $L = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$. This list contains every grid point which can appear in the interpolation stencil for some point on the surface $\mathcal{S}$.[1] The grid points in L form the computational band on which the numerical solution is defined. Specifically, to approximate the solution function u at all points in L we introduce the vector $\boldsymbol{u} \in \mathbb{R}^m$ with entries $u_i \approx u(\boldsymbol{x}_i)$ for each grid point $\boldsymbol{x}_i \in L$. The second list of points $G = \{\boldsymbol{x}_{m+1}, \boldsymbol{x}_{m+2}, \ldots, \boldsymbol{x}_{m+m_g}\}$ is disjoint from L and contains m_g additional "ghost points" along the edges of the computational band. These points will be used in the derivation below, but their values are not propagated in time. Figure 2.1 shows an example of the two lists L and G where the curve $\mathcal{S}$ consists of a circle embedded in $\mathbb{R}^2$. Algorithms for the construction of these lists are given in Appendix A.

Finally, we require two other vectors over the list L and G for use in the derivation. Let $u(\mathrm{cp}(L)) \in \mathbb{R}^m$ denote the vector with components $u(\mathrm{cp}(\boldsymbol{x}_i))$ for $\boldsymbol{x}_i \in L$, and let $u(\mathrm{cp}(G)) \in \mathbb{R}^{m_g}$ denote the vector with components $u(\mathrm{cp}(\boldsymbol{x}_{m+i}))$ for $\boldsymbol{x}_{m+i} \in G$.

[1] This intuitive condition is sufficient. A slightly smaller computational band is often possible by taking L to be the list of all interpolation nodes over just the interpolated points $\mathrm{cp}(\boldsymbol{x}_i), \boldsymbol{x}_i \in L \cup G$, rather than over all of the surface $\mathcal{S}$. This is the approach used in Appendix A to construct L.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

2.2.1. The extension matrix. The closest point extension is an operator which assigns a value of $u(\mathrm{cp}(\boldsymbol{x}))$ to $u(\boldsymbol{x})$. In a discrete setting, $\mathrm{cp}(\boldsymbol{x}_i)$ is generally not a grid point and interpolation is used to approximate $u(\mathrm{cp}(\boldsymbol{x}_i))$. Because each interpolation is a linear combination of values in $\boldsymbol{u}$, this operation can be expressed as a matrix multiplication using the *extension matrix* E defined as follows.

DEFINITION 1 (extension matrix E). *Given the vectors* $\boldsymbol{u}$, $u(\mathrm{cp}(L))$, *and* $u(\mathrm{cp}(G))$ *and an interpolation scheme as described above, the* extension matrix *is an* $(m+m_g)\times m$ *matrix operator* E *such that*

$$\begin{pmatrix} u(\mathrm{cp}(L)) \\ u(\mathrm{cp}(G)) \end{pmatrix} \approx \mathsf{E}\boldsymbol{u} = \underset{\mathsf{E}}{\square}\,\underset{\boldsymbol{u}}{\square}. \tag{2.5}$$

The nonzero entries in the i*th row of* E *consist of the weights in the interpolation scheme for* $u(\mathrm{cp}(\boldsymbol{x}_i))$. *That is, the components of the matrix* $\mathsf{E} = [\gamma_{ij}]$ *are*

$$\gamma_{ij} = \begin{cases} w_j & \text{if } \boldsymbol{x}_j \text{ is in the interpolation stencil for } \mathrm{cp}(\boldsymbol{x}_i), \\ 0 & \text{otherwise}, \end{cases}$$

with w_j *denoting the weight associated with the grid point* x_j *in the interpolation scheme for point* $\mathrm{cp}(\boldsymbol{x}_i)$.

2.2.2. The discrete differential operator. We are now in a position to introduce the discrete differential operator. Our exposition will treat the Laplace–Beltrami operator $\Delta_{\mathcal{S}}$ and the associated in-surface heat equation. More general operators are certainly possible; however, the Laplace–Beltrami operator is particularly instructive because its spatial discretization may be stiff and because it is the key building block for the nonlinear and high-order flows we shall consider in subsequent sections.

We begin by recalling that for points $\boldsymbol{x}$ on the surface, $\Delta_{\mathcal{S}} u(\boldsymbol{x})$ is consistent with $\Delta\left(u(\mathrm{cp}(\boldsymbol{x}))\right)$. We approximate the function $\Delta\left(u(\mathrm{cp}(\boldsymbol{x}))\right)$ with a vector of m values, with the *ith* entry corresponding to an approximation of $\Delta\left(u(\mathrm{cp}(\boldsymbol{x}_i))\right)$ for $\boldsymbol{x}_i \in L$. The vector is given by the right-hand side of

$$\Delta u(\mathrm{cp}(\boldsymbol{x})) \approx \boldsymbol{\Delta}_h \begin{pmatrix} u(\mathrm{cp}(L)) \\ u(\mathrm{cp}(G)) \end{pmatrix}, \tag{2.6}$$

where $\boldsymbol{\Delta}_h$ is an $m \times (m + m_g)$ matrix which approximates the Laplacian operator in $\mathbb{R}^d$ using a linear finite difference scheme (e.g., second-order or fourth-order centered differences). The finite difference scheme is applied at each point in L by taking a combination of neighboring points appearing in a *differentiation stencil*. Some of these neighboring grid points will be in L with corresponding values of $u(\mathrm{cp}(\boldsymbol{x}))$ in the vector $u(\mathrm{cp}(L))$; the remaining grid points will be ghost points in G, and hence $u(\mathrm{cp}(G))$ also appears in (2.6). For example, Figure 2.1 shows two differentiation stencils, one of which includes a ghost point.

From (2.5), approximations of the vectors $u(\mathrm{cp}(L))$ and $u(\mathrm{cp}(G))$ are obtained by left multiplying the vector $\boldsymbol{u}$ by the extension matrix E. Applying this to (2.6) yields

$$\Delta u(\mathrm{cp}(\boldsymbol{x})) \approx \boldsymbol{\Delta}_h \begin{pmatrix} u(\mathrm{cp}(L)) \\ u(\mathrm{cp}(G)) \end{pmatrix} \approx \boldsymbol{\Delta}_h \mathsf{E}\boldsymbol{u} = \underset{\boldsymbol{\Delta}_h}{\square}\,\underset{\mathsf{E}}{\square}\,\underset{\boldsymbol{u}}{\square} = \square = \widetilde{\mathsf{M}}\boldsymbol{u},$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

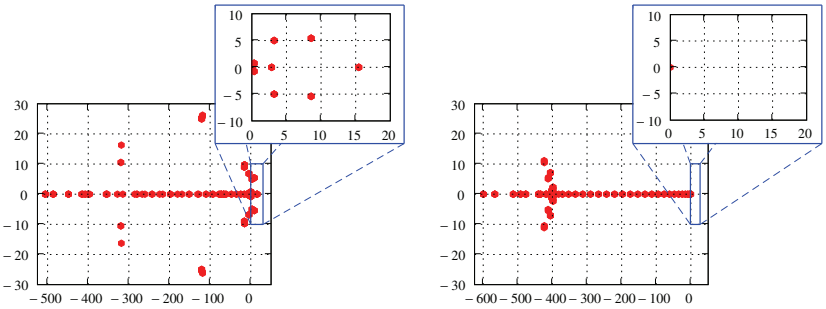
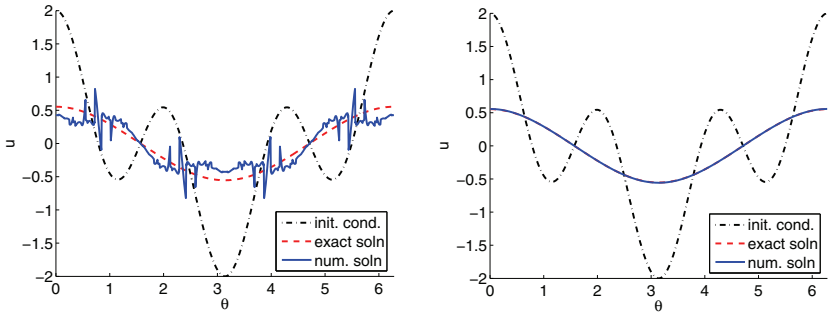


FIG. 2.2. *Spectra of the* $\widetilde{\mathsf{M}}$ *(left) and* M *(right) matrices. The geometry is a unit circle in* 2*D, using interpolation matrix* E *with biquartic interpolation* ($p = 4$) *and a mesh spacing of* $\Delta x = 0.1$. *Observe that* $\widetilde{\mathsf{M}}$ *has eigenvalues in the right half-plane.*



(a) $\widetilde{\mathsf{M}}$, unstable (b) M, stable

FIG. 2.3. *Stable and unstable solutions of the in-surface heat equation on a unit circle embedded in* 2*D, with* $\Delta x = 0.1$, *degree* $p = 4$ *interpolation, and backward Euler time stepping until* $t = 0.6$ *with* $\Delta t = \frac{1}{8}\Delta x$. *Note oscillations indicating instability in* (a), *whereas the numerical solution essentially overlaps the exact solution in* (b).

where $\widetilde{\mathsf{M}} = \boldsymbol{\Delta}_h \mathsf{E}$ is an $m \times m$ matrix. Thus, a spatial discretization of the in-surface heat equation $u_t = \Delta_{\mathcal{S}} u$ is

$$\frac{\partial}{\partial t}\boldsymbol{u} = \widetilde{\mathsf{M}}\boldsymbol{u}. \tag{2.7}$$

Note that the ghost points and $u(\mathrm{cp}(G))$ were used only in the derivation; the system (2.7) is defined on the list of grid points L.

2.2.3. Stabilizing the closest point method. By examining the spectra of $\widetilde{\mathsf{M}}$ given in Figure 2.2 (left), we find that the matrix has some eigenvalues with positive real parts. In the solution of the semidiscrete system (2.7), these positive eigenvalues correspond to exponential growth in the associated eigenvectors. But the Laplace–Beltrami operator is diffusive so these exponentially growing components correspond to instability, and in fact, this instability is observed in practice. For example, Figure 2.3(a) displays the oscillatory results obtained for an in-surface heat flow calculation using the implicit closest point method with $\widetilde{\mathsf{M}}$.

Fortunately, a stable modification of the implicit closest point method procedure is straightforward to construct: when approximating $\Delta_{\mathcal{S}} u$ at the point $\boldsymbol{x}_i$ with a finite difference scheme, we map only the neighboring points $\boldsymbol{x}_j$ of the stencil back to their

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

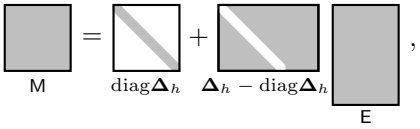
Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

closest points $\mathrm{cp}(\boldsymbol{x}_j)$ and use u_i itself instead of $u(\mathrm{cp}(\boldsymbol{x}_i))$ (see also (2.4)). This special treatment of the diagonal elements yields a new $m \times m$ matrix

$$\mathsf{M} = \mathrm{stab}(\boldsymbol{\Delta}_h, \mathsf{E}) := \mathrm{diag}\, \boldsymbol{\Delta}_h + (\boldsymbol{\Delta}_h - \mathrm{diag}\, \boldsymbol{\Delta}_h)\mathsf{E} \tag{2.8a}$$

or diagrammatically

(2.8b)



and thus the spatial discretization of the in-surface heat equation is

$$\frac{\partial}{\partial t}\boldsymbol{u} = \mathsf{M}\boldsymbol{u}. \tag{2.9}$$

As explained in section 2.1, this splitting eliminates redundant interpolations and does not impact the consistency of u at the surface. Intuitively, we expect improved stability from this splitting because the magnitude of the diagonal of the operator has generally increased relative to the off-diagonal elements (see [22, section 3.3.3]). Indeed, a study of the spectra of M for a variety of surfaces [22, section 3.3.1] indicates that all eigenvalues of M have negative real parts. See Figure 2.2 (right) for the spectra of the heat equation on a unit circle in 2D and Figure 2.3(b) for the (stable) results from a related numerical experiment.

Remark. The two steps of the explicit closest point method for solving $u_t = \Delta_{\mathcal{S}} u$ can also be formulated using the matrices $\boldsymbol{\Delta}_h$ and E as follows: (a) perform a forward Euler time step $\tilde{\boldsymbol{u}}^{n+1} = \boldsymbol{u}^n + \Delta t \boldsymbol{\Delta}_h \boldsymbol{u}^n$; and (b) perform a closest point extension $\boldsymbol{u}^{n+1} = \mathsf{E}\tilde{\boldsymbol{u}}^{n+1}$. Combining these two steps yields

$$\boldsymbol{u}^{n+1} = \mathsf{E}\boldsymbol{u}^n + \Delta t \boldsymbol{\Delta}_h \mathsf{E}\boldsymbol{u}^n,$$

which is notably *not* the forward Euler time-stepping method applied to either (2.7) or (2.9). However, it suggests another possible implicit discretization based on backward Euler:

$$\boldsymbol{u}^{n+1} = \mathsf{E}\boldsymbol{u}^n + \Delta t \boldsymbol{\Delta}_h \mathsf{E}\boldsymbol{u}^{n+1}. \tag{2.10}$$

Unfortunately, numerical experiments indicate that (2.10) is often unstable when time steps larger than those possible with forward Euler are used.

2.3. Truncation error analysis. The error from approximating the Laplace–Beltrami operator using the implicit closest point method has two components: one arising from the polynomial interpolation and a second arising from the discretization of the Laplacian operator $\boldsymbol{\Delta}_h$.

Assume that p is the degree of interpolation to be used in the closest point extension procedure. Also assume an order-q discretization of the underlying Laplacian operator. In [30] it is shown that if an interpolation of degree $p \geq q + 1$ is chosen, then we maintain the order of accuracy of the underlying spatial discretization; i.e., we obtain an $O(\Delta x^q)$ truncation error. Conversely, if $p < q + 1$, the error will be dominated by the interpolation error, and we obtain an $O(\Delta x^{p-1})$ truncation error. Notice, in particular, that the method will be inconsistent if linear interpolation is chosen. Section 4.1 compares these analytically derived truncation errors with numerical experiments.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 3.1
Condition numbers for the time-stepping matrix $\mathsf{A}_{BE} = \mathsf{I} - \Delta t\mathsf{M}$ *with* $\Delta t = \frac{1}{4}\Delta x$ *for various* Δx. *The curve* $\mathcal{S}$ *is a unit circle in* 2*D, and the interpolation is of degree* $p = 4$.

Δx	$\mathrm{cond}(\mathsf{A}_{\mathrm{BE}})$
0.4	94.1
0.2	281
0.1	664
0.05	1520
0.025	3114
0.0125	6289
0.00625	12703
0.003125	25984

3. Time stepping and matrix properties. The temporal discretization of the semidiscrete problem (2.9) can be performed using implicit linear multistep methods[2] such as the backward differentiation formulas (BDF methods) [12]. For each time step, the BDF methods perform a single linear system solve of the form $\mathsf{A}\boldsymbol{u} = \boldsymbol{b}$ with $\mathsf{A} = \mathsf{I} - \gamma\Delta t\mathsf{M}$ for some value of γ that depends on the particular method. For example, the BDF2 applied to (2.9) leads to $[\mathsf{I} - \frac{2}{3}\Delta t\mathsf{M}]\boldsymbol{u}^{n+1} = \frac{4}{3}\boldsymbol{u}^n - \frac{1}{3}\boldsymbol{u}^{n-1}$. We denote the time-stepping matrices corresponding to the backward Euler and the BDF schemes as A_{BE}, $\mathsf{A}_{\mathrm{BDF2}}$, and $\mathsf{A}_{\mathrm{BDF4}}$. The structure and properties of these matrices are important for understanding the behavior of direct and iterative methods for solving the linear systems.

3.1. Matrix properties. As previously noted in section 2.2.3 the eigenvalues of M have negative real parts, leading to stability when the evolution is treated with an appropriate (e.g., A-stable) time-stepping scheme.

Table 3.1 lists the condition number for the time-stepping matrix $\mathsf{A}_{\mathrm{BE}} = \mathsf{I} - \Delta t\mathsf{M}$ for various Δx. Note that with $\Delta t = O(\Delta x)$, the condition number of the time-stepping matrix A_{BE} is moderate and only doubles when Δx is halved. Similar results are obtained for the time-stepping matrices $\mathsf{A}_{\mathrm{BDF2}}$ and $\mathsf{A}_{\mathrm{BDF4}}$. The time-stepping matrices thus appear to be well-conditioned. As discussed in section 3.2, this is important for solving the linear system at each time step.

Figure 3.1 shows the sparsity structure of the $\boldsymbol{\Delta}_h$, E, and M matrices for various surfaces, values of the grid spacing Δx, and degree of interpolation p. The effects of the ghost points G are noticeable on the right of $\boldsymbol{\Delta}_h$ and the bottom of E. Note that the composed matrix M has a band-limited profile; this feature is related to the ordering of the lists L and G chosen by the banding algorithm in Appendix A. Table 3.2 shows some sparsity properties of M including the number of nonzero elements and the bandwidth. The increasing sparsity as Δx decreases demonstrates the importance of iterative solvers as discussed further the next section.

3.2. Solving the linear systems. Performing time stepping for the implicit closest point method results in large linear systems of the form $\mathsf{A}\boldsymbol{u} = \boldsymbol{b}$. As noted above, the time-stepping matrices are well-conditioned for the BDF schemes, and thus direct solves based on variations of LU factorization work well, at least for small systems such as arise in 2D with a moderate or large grid spacing Δx. The MATLAB backslash operator or the sparse solver scipy.sparse.linalg.spsolve in SciPy [20] is used in many of the calculations in section 4.

[2]Implicit Runge–Kutta methods are also straightforwardly accommodated, particularly for the case of diagonally implicit schemes [21, 17].

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

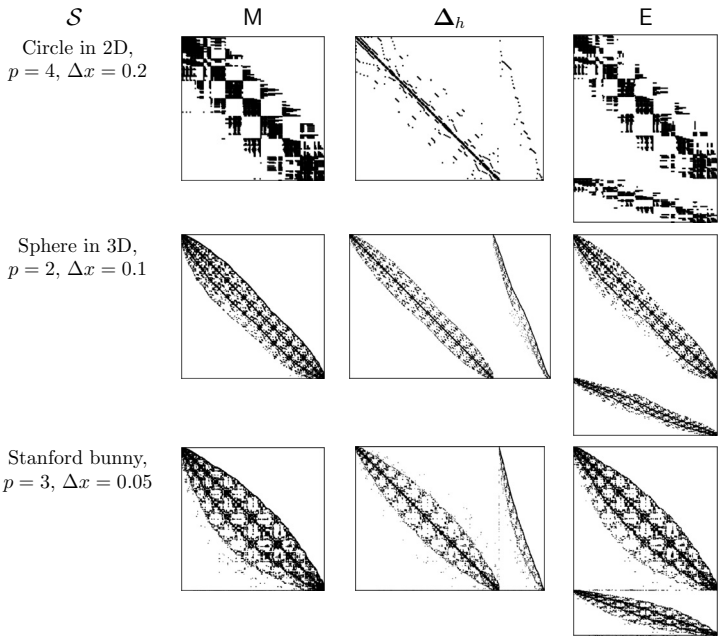


FIG. 3.1. *Sparsity structure of the* M, $\boldsymbol{\Delta}_h$, *and* E *matrices (from left to right) for the Laplace–Beltrami operator on various surfaces using degree-p interpolation.*

TABLE 3.2

Sparsity properties of the matrix M *for a unit circle in* 2*D. An increase in* Δx *or in the degree of interpolation p leads to an increase in the percentage of nonzero entries. Roughly the same relation is reflected in the bandwidth as a percentage of total width.*

Δx	Nonzero entries (percentage)				Bandwidth (percentage)			
	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
0.2	12.8	16.3	18.2	21.3	31.7	55.4	51.5	65.6
0.1	6.5	8.0	9.0	10.5	15.8	21.4	27.2	34.1
0.05	3.3	3.9	4.5	5.2	8.8	11.0	18.2	16.3
0.025	1.6	1.9	2.3	2.6	5.6	6.4	8.6	8.1
0.0125	0.8	1.0	1.1	1.3	2.9	2.8	4.5	3.9

The time-stepping matrices become increasingly sparse as Δx decreases (e.g., Table 3.2), potentially causing direct methods to become inefficient due to "fill-in" from the LU factorization. For large systems, particularly in 3D, iterative schemes are more efficient both in terms of memory and CPU time. It was shown experimentally in [22, sections 3.3.3 and 3.3.4] that the backward Euler time-stepping matrix $\mathsf{I} - \Delta t\mathsf{M}$ associated with the implicit closest point method matrix M is diagonally dominant and (unsymmetric) positive definite [13] for time steps of $\Delta t = O(\Delta x^2)$: this guarantees convergence for many iterative schemes. Naturally, in practice we wish to use larger time steps $\Delta t = O(\Delta x)$, and for these step sizes, the backward Euler matrix $\mathsf{I} - \Delta t\mathsf{M}$ is indefinite and not diagonally dominant. It is interesting to note that in the examples we have considered [22, section 3.3.1] the spectral radii of the Jacobi and Gauss–Seidel iteration matrices are all strictly less than one, and thus we can expect convergence for those methods. In many of the examples we provide in section 4, the GMRES algorithm [38] is used instead, as it seems to converge well in practice. In these cases,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

we chose GMRES subroutines available within either SciPy (scipy.sparse.linalg.gmres) or MATLAB (gmres).

We remark that it is not necessary to explicitly construct the matrices M or A in order to use these iterative schemes. For example, GMRES requires only a subroutine that computes the product $\mathsf{A}\boldsymbol{u}$. However, in the numerical results that follow, we have elected to explicitly construct and use the matrices M, A_{BE}, $\mathsf{A}_{\mathrm{BDF2}}$, and $\mathsf{A}_{\mathrm{BDF4}}$.

4. Numerical results.

4.1. Numerical convergence studies.

The convergence of the implicit closest point method is tested on two problems.

Two-dimensional test problem. The problem consists of the in-surface heat equation (2.1) on a unit circle in $\mathbb{R}^2$ with initial conditions $u(0,\theta) = \cos\theta + \cos 3\theta$. The exact solution is $u(t,\theta) = e^{-t}\cos(\theta) + e^{-9t}\cos(3\theta)$. In the tests below, the problem is solved discretely in time until $t = \frac{1}{2}$ using BDF schemes. The exact solution is used as the starting procedure. Linear systems are solved directly with MATLAB's backslash operator.

Three-dimensional test problem. The problem consists of the in-surface heat equation on a unit sphere in $\mathbb{R}^3$ with the exact solution given as a series of spherical harmonics $u(t,\theta,\phi) = \frac{20}{3\pi}\sum_{l=1}^{\infty} e^{-l^2/9}e^{-tl(l+1)}Y_{ll}(\theta,\phi)$, where Y_{lm} represents the degree l order m real spherical harmonic. In practice, the rapid decay of the coefficients means that the series can be truncated after only a few terms. The problem is evolved in time until $t = \frac{1}{2}$ using BDF schemes and the exact solution as a starting procedure. Linear system solves are performed in MATLAB with gmres using a tolerance of 1×10^{-10}.

4.1.1. Heat equation, second-order differences in space.

Figure 4.1 shows the results of a numerical convergence study on these two problems using second-order finite differences, the second-order BDF2 scheme with $\Delta t = \frac{1}{8}\Delta x$, and various degrees p of interpolating polynomials. We note clear second-order convergence using $p \geq 3$. The results are consistent with the analysis in section 2.3. The computational results appear identical for $p \geq 4$ because the error is dominated by the Laplacian discretization error term and the interpolation error term is insignificant. Degree $p = 3$ still exhibits second-order convergence but with a larger error constant which suggests that the interpolation error dominates. Note that $p = 2$ behaves surprisingly well (often better than the predicted first-order). As expected, degree $p \leq 1$ interpolation appears inconsistent.

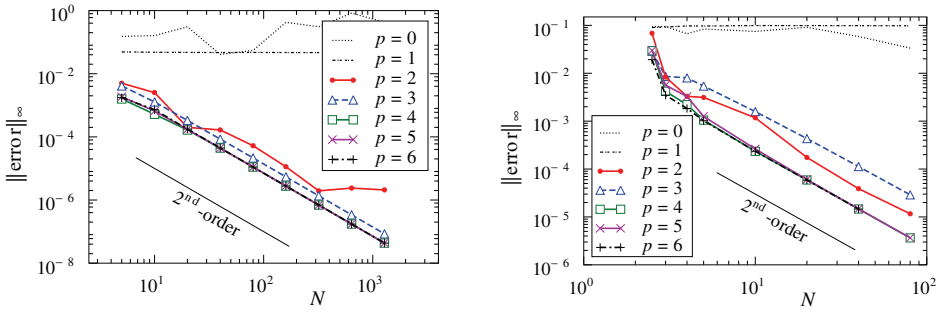


FIG. 4.1. *Numerical convergence study results for the in-surface heat equation on a circle (left) and a sphere (right) using second-order finite differences, BDF2 time stepping, and degree p interpolation. Here* $N = \frac{1}{\Delta x}$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

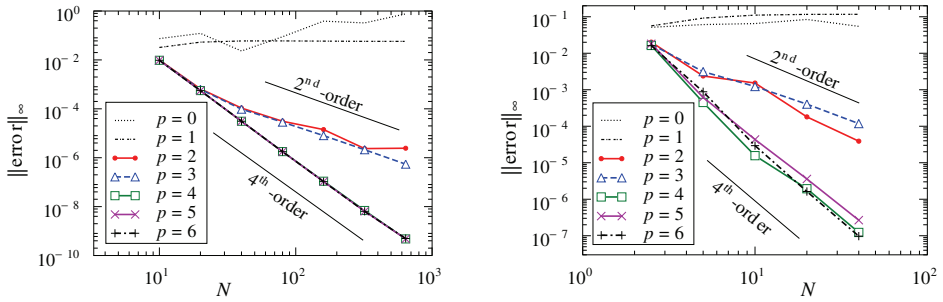


FIG. 4.2. *Numerical convergence study results for the in-surface heat equation on a circle (left) and a sphere (right) using fourth-order finite differences, BDF4 time stepping, and degree p interpolation. Here* $N = \frac{1}{\Delta x}$.

Larger values for the interpolation degree p result in larger interpolation stencils which translate into denser matrices (see Table 3.2). For second-order finite differences, degree $p = 3$ or $p = 4$ are both good choices as they exhibit clear second-order convergence, although the error for $p = 3$ is typically larger than for $p = 4$. Degree $p \geq 5$ would be excessive for second-order calculations as no benefit in accuracy results from the extra work required. Degree $p = 2$ may be appealing for applications where calculation speed is more important than accuracy.

4.1.2. Heat equation, fourth-order differences in space. The convergence of the implicit closest point method is next tested using a fourth-order centered finite difference stencil for $\mathbf{\Delta}_h$. Figure 4.2 shows the results of a numerical convergence study on the two test problems above, where time stepping is performed using the fourth-order BDF4 scheme with $\Delta t = \Delta x$ (on the circle) and $\Delta t = \frac{1}{4}\Delta x$ (on the sphere). We note again that $p = 0$ and $p = 1$ degree interpolation appear inconsistent. Degree $p = 2$ or $p = 3$ produce second-order results. Degree $p \geq 4$ produces fourth-order convergence. Note that on this problem, $p = 2$ and $p = 4$ performed roughly one order better than predicted by section 2.3.

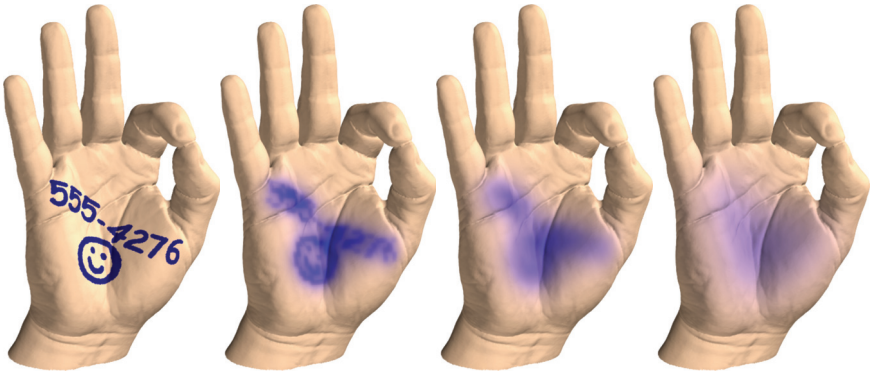
4.2. Heat equation on surfaces. The computations in this section were performed in Python [42] using SciPy [20] and NumPy [26] for numerical calculations. Visualizations use VTK [33].

Throughout this section, closest point representations of surfaces are obtained from complicated triangulated surfaces using the method described in [23]. Briefly, for each triangle in a triangulation, the method determines all nodes that are within the bandwidth of the triangle (for computations using the closest point method, nodes that are further away cannot be influenced by that part of the surface). For each node, this gives a list of triangles. This list is sufficiently small that it is efficient to directly examine each member to determine the closest triangle (and hence the closest point on the surface). Note that this method scales linearly with the number of triangles under the assumption that the closest point function is needed only on a narrow computational band.

4.2.1. Heat equation on Laurent's Hand. Figure 4.3 shows the results of blurring an image on the surface of Laurent's Hand [31] by solving several steps of the in-surface heat equation. In this example, the phone number written on the hand is completely indistinguishable after only a few steps. Note that the closest point method could also be used to apply an in-surface unsharp mask or other image processing technique to attempt to recover the phone number from the blurred image.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php



(a) $t = 0$, init. cond. (b) $t = \frac{1}{400}$, step 1 (c) $t = \frac{1}{200}$, step 2 (d) $t = \frac{3}{200}$, step 6

FIG. 4.3. *Blurring on the surface of Laurent's Hand* [31] *by solving the in-surface heat equation. The phone number becomes increasingly blurred with each time step. Here* $\Delta x = \frac{1}{40}$ *and the length of the hand is about* $80\Delta x$*. Time stepping is performed with backward Euler and* $\Delta t = \frac{\Delta x}{10}$*. The closest point extension uses degree* $p = 2$ *interpolation.*

4.2.2. Heat equation on a complicated domain. Consider a composite domain consisting of Hui's Pig [19] connected by the tail to a filament (an infinitesimally thin wire) which is also connected to a sphere. This surface is therefore comprised of components of mixed codimension: the pig and sphere are codimension-one, whereas the filament is codimension-two. The closest point representation is straightforward to obtain for this type of composite domain: simply compute the closest point in each of the three components and then return the closest of the three.

A heat source is applied under the sphere. The temperature u over the surface of the domain is modeled according to

$$u_t = k(\boldsymbol{x})\Delta_{\mathcal{S}} u + \gamma(T - u)g(\boldsymbol{x}), \tag{4.1a}$$

where the heat coefficient $k(\boldsymbol{x})$ is chosen as

$$k(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in \text{pig}, \\ 2 & \boldsymbol{x} \in \text{sphere}, \\ 10 & \boldsymbol{x} \in \text{filament} \end{cases} \tag{4.1b}$$

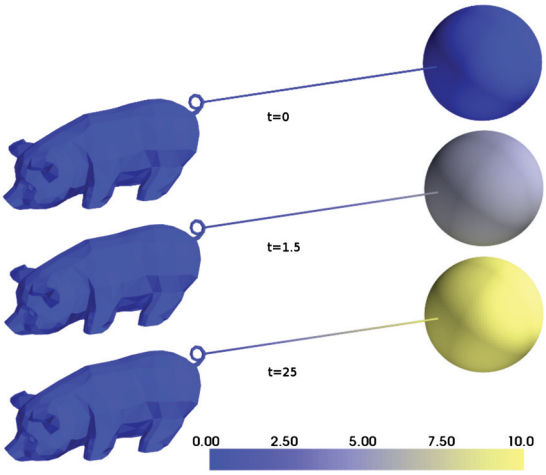
and the forcing term consists of Newton's law of heating and cooling with constant $\gamma = 30$ and maximum temperature $T = 10$. The function $g(\boldsymbol{x})$ is a Gaussian blob in $\mathbb{R}^3$ which localizes the heat source directly under the sphere. Initially the temperature is set to zero everywhere. The problem is solved using $\Delta x = 0.05$ and degree $p = 3$ interpolation. Time stepping is performed to time $t = 25$ using the BDF2 scheme with $\Delta t = 2\Delta x$ and a single step of backward Euler as a starting procedure. Thus, only 250 steps are used in contrast with the roughly 10,000 steps required for explicit time stepping for this problem.

Figure 4.4 shows the initial conditions and heat distribution at various times. This example demonstrates the great flexibility of the method with respect to geometry: heat flow over a complicated geometry of variable codimension is computationally no more complicated than for a sphere.

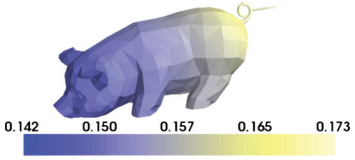
4.2.3. Pattern formation. The solutions of reaction-diffusion equations can exhibit rich pattern forming behavior from random initial conditions. The application

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php



(a) Initial conditions (top), solution at $t = 1.5$ (middle), and at $t = 25$ (bottom).



(b) Rescaled image showing the heat distribution on the pig at $t = 25$.

FIG. 4.4. *Numerical solution of the in-surface heat equation on the composite pig-wire-sphere domain. At $t = 25$, the temperature is roughly constant at $u = 10$ on the sphere. The pig has warmed up only slightly, and there is a temperature gradient in the wire.*

of reaction-diffusion equations on surfaces has been investigated in previous studies, such as [40] (by computing directly on a surface mesh) and [6] (using a level set representation of the surface). In this section, the implicit closest point method is applied to pattern formation problems.

A possible set of reaction-diffusion equations for two chemicals u and v is

$$u_t = f(u,v) + \nu_u \Delta_{\mathcal{S}} u, \qquad u(0,\boldsymbol{x}) = u_0(\boldsymbol{x}), \tag{4.2a}$$

$$v_t = g(u,v) + \nu_v \Delta_{\mathcal{S}} v, \qquad v(0,\boldsymbol{x}) = v_0(\boldsymbol{x}), \tag{4.2b}$$

where f and g are nonlinear reaction terms [39]. The chemicals locally react with one another while at the same time diffusing spatially at different rates. We consider one particular form of (4.2) known as the Brusselator [28, 43] which has

$$f(u,v) = a - (b-1)u + u^2 v, \qquad g(u,v) = bu - u^2 v.$$

The implicit closest point method can be applied to (4.2) using implicit-explicit (IMEX) time-stepping schemes [3, 2, 29]. Using this approach, the nonlinear reaction terms are treated explicitly and the linear diffusion terms are treated implicitly. Notice that this decouples the implicit system solves for u and v and we solve two systems of the form $\mathsf{A}_{\mathrm{BDF2}}\boldsymbol{u} = \boldsymbol{b}_1$ and $\mathsf{A}_{\mathrm{BDF2}}\boldsymbol{v} = \boldsymbol{b}_2$ to advance to the next time step. The SBDF2 scheme [3] is known to work well for pattern formation problems [29], and it is used here with one step of IMEX Euler as a starting method. For initial conditions,

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

(a) $\nu_u = \frac{5}{900}$, $t = 25$ (b) $\nu_u = \frac{3.8}{900}$, $t = 15$ (c) $\nu_u = \frac{2.5}{900}$, $t = 15$

FIG. 4.5. *Patterns formed by the Brusselator on the surface of the Stanford Bunny for different values of ν_u with $a = 3$, $b = 10.2$, and $\nu_v = 10/900$. Darker color indicates high concentrations of u. The computation uses $\Delta t = \Delta x = 0.05$ with degree $p = 2$ interpolation, and the bunny is scaled to be approximately 2 units long.*

small random perturbations around the zero-diffusion ($\mu_u = \mu_v = 0$) steady state of $u = a$ and $v = \frac{b}{a}$ are used.

Depending on the values of the coefficients a, b, ν_u, and ν_v, the Brusselator exhibits different types of pattern forming behavior [43]. Figure 4.5 shows the results of three simulations of the Brusselator on the surface of the Stanford Bunny [41] with $a = 3$, $b = 10.2$, $\nu_v = 10/900$, and various values of ν_u. This choice of parameters closely matches those used for planar calculations in [43, Figure 2(a)],[3] and indeed we notice the same transition from "honeycomb" (Figure 4.5(a)) to stripes (Figure 4.5(b)) to roughly hexagonal patterns of spots (Figure 4.5(c)).

5. Biharmonic problems. Consider the fourth-order problem

$$u_t = -\nabla_{\mathcal{S}}^4 u, \tag{5.1}$$

where $\nabla_{\mathcal{S}}^4$ is the biharmonic operator intrinsic to the surface. A naive approach to solving this problem via the closest point method would introduce a single extension matrix into the discretization. Unfortunately, when the gradient and divergence operators are combined into a biharmonic operator using a single extension matrix, the hypotheses for the principles in section 1 are violated. We therefore expect that this approach will not work, and indeed, numerical experiments demonstrate the inconsistency of the approach [22].

In [30], it was argued that operators which are composed of multiple Laplace–Beltrami $\Delta_{\mathcal{S}}$ operators (or other high-order derivative terms) could be implemented using multiple extension steps. Recalling that $\nabla_{\mathcal{S}}^4 u = \Delta_{\mathcal{S}}(\Delta_{\mathcal{S}} u)$ and applying the principles in section 1,

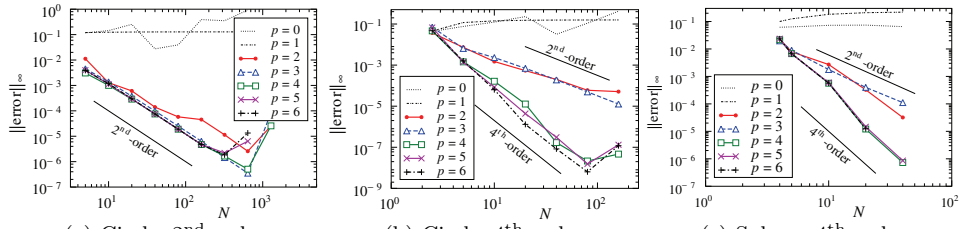
$$\nabla_{\mathcal{S}}^4 u(\boldsymbol{x}) = \Delta_{\mathcal{S}}\big(\underbrace{\Delta\,(u(\mathrm{cp}(\boldsymbol{x})))}_{v}\big) = \Delta\big(v(\mathrm{cp}(\boldsymbol{x}))\big) = \Delta\big(\underbrace{\big[\Delta\,(u(\mathrm{cp}(\boldsymbol{x})))\,\big]}_{v}(\mathrm{cp}(\boldsymbol{x}))\big)$$

for points $\boldsymbol{x}$ on the surface; that is, a closest point extension of the inner Laplacian v is performed before computing the outer Laplacian. With $\boldsymbol{\Delta}_h$ as the discrete diffusion

[3] We rescale the equations to the size of the bunny which accounts for the additional factors of $\frac{1}{900}$ which do not appear in [43].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php



(a) Circle, 2^{nd}-order (b) Circle, 4^{th}-order (c) Sphere, 4^{th}-order

FIG. 5.1. *Biharmonic numerical convergence studies using the operator* M_{bi} *with second-order finite differences (and BDF2 time stepping) and fourth-order finite differences (and BDF4 time stepping). The calculations on circles use* $\Delta t = \frac{1}{4}\Delta x$*, and the calculations on the sphere use* $\Delta t = \frac{1}{8}\Delta x$*. Note* $N = \frac{1}{\Delta x}$.

operator from section 2.2.2, we could consistently approximate the previous expression using the matrix

$$\widetilde{\mathsf{M}_{\text{bi}}} = \boldsymbol{\Delta}_h \mathsf{E} \boldsymbol{\Delta}_h \mathsf{E}.$$

Similar to the case of the Laplace–Beltrami operator, the matrix $-\widetilde{\mathsf{M}_{\text{bi}}}$ has eigenvalues with positive real components, leading to instability. A stable variant arises naturally by using the procedure from section 2.2.3. Specifically, this leads to the matrix

$$\mathsf{M}_{\text{bi}} = \text{stab}(\boldsymbol{\Delta}_h, \mathsf{E})\,\text{stab}(\boldsymbol{\Delta}_h, \mathsf{E}) = \mathsf{M}\mathsf{M},$$

which in turn gives rise to the ODE system

$$\frac{\partial}{\partial t}\boldsymbol{u} = -\mathsf{M}_{\text{bi}}\boldsymbol{u}. \tag{5.2}$$

Note that this is a "one-line" change in computer code for the in-surface heat equation, requiring only the squaring of the M matrix.

5.1. Numerical convergence studies. The convergence of the implicit closest point method for biharmonic problems is tested on the two problems of section 4.1, where here the exact solutions are $u(t,\theta) = e^{-t}\cos(\theta) + e^{-81t}\cos(3\theta)$ and $u(t,\theta,\phi) = \frac{20}{3\pi}\sum_{l=1}^{\infty} e^{-l^2/9} e^{-tl^2(l+1)^2} Y_{ll}(\theta,\phi)$. The two-dimensional problem is solved until $t = 0.5$, and the three-dimensional problem is solved until $t = 0.15$. Figure 5.1 demonstrates that the closest point method achieves the expected accuracy using both second-order and fourth-order finite differences, although the error appears to stagnate at around 10^{-7}. Note that the entries in the M_{bi} matrix are each multiplied by $1/\Delta x^4$. Heuristically, the stagnating error appears to be consistent with rounding errors that are magnified by about $1/\Delta x^4$ with $\Delta x \approx 10^{-2}$.

Our experiments demonstrate that the implicit closest point method gives a viable and extremely simple way to treat problems with biharmonic terms, although for highly accurate results some attention should be paid to the control of roundoff error. In the following section, we apply the technique to a fourth-order problem of interface motion.

5.2. "Surface diffusion" on a surface. In the "surface diffusion" problem, "atoms [on an interface] diffuse from areas of high mean curvature to low mean curvature" [34]. In the standard settings of $\mathbb{R}^2$ and $\mathbb{R}^3$, the interface moves at a speed proportional to the intrinsic Laplacian of its curvature. Here we are interested in the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

corresponding motion of an interface (defined by the zero contour of the function u) on a curved surface $\mathcal{S}$: the term "surface diffusion" is somewhat misleading in this case because the diffusion is occurring on the interface (which in turn lies on the surface $\mathcal{S}$).

A level set formulation of this problem is given by the fourth-order nonlinear surface PDE

$$u_t = -K_{\mathcal{S}}(u) = -\Delta_I \kappa |\nabla_{\mathcal{S}} u|,$$

where $\kappa = \nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} u / |\nabla_{\mathcal{S}} u|)$ is the in-surface curvature of the local level set of u. The Laplacian Δ_I operator is the in-surface analogue of the "surface Laplacian" operator in [34]. Smereka [34] presented a simple and effective splitting

$$u_t = -\beta \nabla_{\mathcal{S}}^4 u + \beta \nabla_{\mathcal{S}}^4 u - K_{\mathcal{S}}(u),$$

from which a semidiscretization in time using IMEX Euler is

$$\frac{u^{n+1} - u^n}{\Delta t} = -\beta \nabla_{\mathcal{S}}^4 u^{n+1} + \beta \nabla_{\mathcal{S}}^4 u^n - K_{\mathcal{S}}(u^n) + O(\Delta t).$$

Notice this is linear in u^{n+1} because the fourth-order $K_{\mathcal{S}}(u)$ is treated explicitly. A full discretization follows using the discrete M_{bi} operator

$$\left[\mathsf{I} + \Delta t \beta \mathsf{M}_{\text{bi}}\right] \left(\boldsymbol{u}^{n+1} - \boldsymbol{u}^n\right) = -\Delta t K^h(\boldsymbol{u}^n),$$

where K^h is a discrete form of $K_{\mathcal{S}}$ obtained using the explicit closest point method. That is, the term $K^h(\boldsymbol{u}^n)$ is formed by first computing κ, the mean curvature of $\boldsymbol{u}^n$, in the embedding space $\mathbb{R}^d$. This computation is followed by a closest point extension of κ. Finally, $\Delta_I \kappa$ is computed by following the procedure in [34, 32]. This explicit computation is performed only locally near the interface. We use $\beta = \frac{1}{2}$ as is recommended in [34].

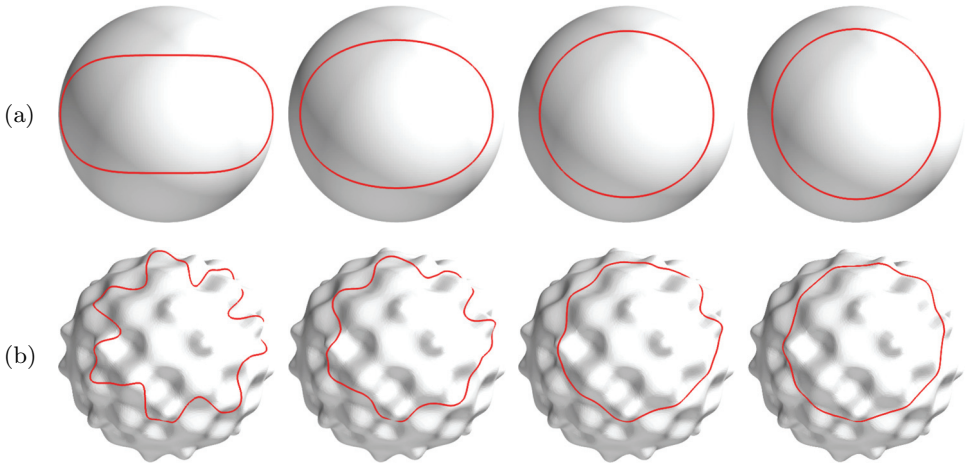


FIG. 5.2. *"Surface diffusion" of interfaces on surfaces. Top row* (a)*: an initial interface on a unit sphere evolves toward a circular steady state (from left to right:* $t = 0$*,* $t = 3\Delta t$*,* $t = 9\Delta t$*, and* $t = 18\Delta t$ *with* $p = 3$*,* $\Delta x = 0.1$*, and* $\Delta t = 100\Delta x^4$*). Bottom row* (b)*: interface evolving on a bumpy sphere* [1] *(left to right:* $t = 0$*,* $t = 3\Delta t$*,* $t = 6\Delta t$*, and* $t = 50\Delta t$ *with* $p = 3$*,* $\Delta x = 0.05$*, and* $\Delta t = 50\Delta x^4$*).*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Figure 5.2 shows two example computations. Our goal here is to demonstrate the effectiveness of the implicit closest point method on this problem, and we have not implemented the full algorithm of [34]. Specifically, some care must be taken when dealing with merging of interfaces. Finally, for simplicity, we have not implemented the local (to the interface) level set approach of [32].

6. Odd-order terms. So far, this paper has concentrated on Laplacian–Beltrami and in-surface biharmonic operators. However, the closest point method can also be applied to odd-order terms. We outline a brief description of a possible approach, based on discretizing the surface divergence of a vector field intrinsic to the surface.

6.1. Surface divergence terms. Consider a vector field $\boldsymbol{w}(\boldsymbol{x}) : \mathcal{S} \to \mathbb{R}^d$ intrinsic to the surface $\mathcal{S}$; that is, any vector in $\boldsymbol{w}$ is tangent to $\mathcal{S}$. Now define the vector field $\boldsymbol{v} : \mathbb{R}^d \to \mathbb{R}^d$ as the closest point extension of the vector field $\boldsymbol{w}$, i.e., $\boldsymbol{v}(\boldsymbol{x}) := \boldsymbol{w}(\text{cp}(\boldsymbol{x}))$. It follows that $\boldsymbol{v}(\boldsymbol{x})$ satisfies the requirements of Principle 2 and thus

$$\nabla_{\mathcal{S}} \cdot \boldsymbol{w} = \nabla \cdot [\boldsymbol{w}(\text{cp}(\boldsymbol{x}))] \quad \text{for points } \boldsymbol{x} \text{ on the surface } \mathcal{S}.$$

We discretize in a similar fashion to section 2.2. Let $\boldsymbol{w_i}$ be a list of length m consisting of the *ith* component of the vector field $\boldsymbol{w}$ evaluated at each of the grid points in the list L. The closest point extension of each component of the vector field can then be approximated using the extension matrix E. Combining this with a discretization of the divergence operator gives

$$\nabla_{\mathcal{S}} \cdot (\boldsymbol{w}) \approx \mathsf{D}_{x_1}\mathsf{E}\boldsymbol{w_1} + \mathsf{D}_{x_2}\mathsf{E}\boldsymbol{w_2} + \cdots + \mathsf{D}_{x_d}\mathsf{E}\boldsymbol{w_d}, \tag{6.1}$$

where D_{x_i} is an $m \times (m + m_g)$ matrix representing a finite difference approximation to the derivative in the x_i direction (the matrices D_{x_i} are analogous to the matrix $\boldsymbol{\Delta}_h$ in section 2.2.2).

Another consistent (and potentially more stable) discretization is obtained using the splitting function $\text{stab}(\cdot,\cdot)$ given in section 2.2.3. This yields

$$\begin{aligned} \nabla_{\mathcal{S}} \cdot \boldsymbol{w} &\approx \text{stab}(\mathsf{D}_{x_1}, \mathsf{E})\boldsymbol{w_1} + \text{stab}(\mathsf{D}_{x_2}, \mathsf{E})\boldsymbol{w_2} + \cdots + \text{stab}(\mathsf{D}_{x_d}, \mathsf{E})\boldsymbol{w_d} \\ &= \mathsf{M}_{x_1}\boldsymbol{w_1} + \mathsf{M}_{x_2}\boldsymbol{w_2} + \cdots + \mathsf{M}_{x_d}\boldsymbol{w_d}, \end{aligned} \tag{6.2}$$

where the $m \times m$ matrices $\mathsf{M}_{x_i} = \text{stab}(\mathsf{D}_{x_1}, \mathsf{E})$ approximate the differential terms of the surface divergence operator. Note that the forms (6.1) and (6.2) are identical if a central difference discretization is chosen since D_{x_i} has zeros along the main diagonal.

6.2. Third-order terms. For a given vector field $\boldsymbol{w}$, suppose we want to form

$$\Delta_{\mathcal{S}}(\nabla_{\mathcal{S}} \cdot \boldsymbol{w}),$$

where $\boldsymbol{w} : \mathcal{S} \to \mathbb{R}^d$ is a vector field defined on the surface and tangent to the surface. As in the biharmonic case in section 5, we perform two closest point extensions. However, here we make use of Principles 1 and 2 to obtain

$$\Delta_{\mathcal{S}}(\nabla_{\mathcal{S}} \cdot \boldsymbol{w}) = \Delta_{\mathcal{S}}\big(\underbrace{\nabla \cdot (\boldsymbol{w}(\text{cp}(\boldsymbol{x})))}_{u}\big) = \Delta\big(u(\text{cp}(\boldsymbol{x}))\big) = \Delta\big(\underbrace{[\nabla \cdot (\boldsymbol{w}(\text{cp}(\boldsymbol{x})))]}_{u}(\text{cp}(\boldsymbol{x}))\big)$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

for points $\boldsymbol{x}$ on the surface $\mathcal{S}$. Applying the discretizations (2.8) and (6.2) yields

$$\begin{aligned}\Delta_{\mathcal{S}}(\nabla_{\mathcal{S}} \cdot \boldsymbol{w}) &\approx \operatorname{stab}(\boldsymbol{\Delta}_h, \mathsf{E})\Big(\operatorname{stab}(\mathsf{D}_{x_1}, \mathsf{E})\boldsymbol{w_1} + \cdots + \operatorname{stab}(\mathsf{D}_{x_d}, \mathsf{E})\boldsymbol{w_d}\Big) \\ &= (\mathsf{M}\,\mathsf{M}_{x_1})\,\boldsymbol{w_1} + (\mathsf{M}\,\mathsf{M}_{x_2})\,\boldsymbol{w_2} + \cdots + (\mathsf{M}\,\mathsf{M}_{x_d})\,\boldsymbol{w_d}. \end{aligned} \tag{6.3}$$

Preliminary tests on the linear advection equation show the approach (6.2) is both consistent and stable. We leave to future work the testing and application of the method to third-order surface PDEs.

7. Conclusions and future work. This paper presents an implicit closest point method for solving surface PDEs. The method introduces one or more matrices into the discretization to carry out the interpolation steps that are central to the closest point method. For the important Laplace–Beltrami and in-surface biharmonic operators we have shown that a stable variant of the algorithm is achieved by introducing the closest point operator into off-diagonal terms only. The corresponding discretization is simple and has the property that computations can be carried out in a small band near the surface with no loss of accuracy whatsoever. The discretization of the in-surface biharmonic operator is obtained trivially by squaring the discretization of the Laplace–Beltrami operator. The power and flexibility of the implicit closest point method are illustrated for a number of examples including in-surface heat flow, reaction-diffusion equations, and higher-order motions. Our experiments are carried out on a variety of interesting surfaces, including triangulated surfaces and surfaces of mixed codimension.

Our approach generalizes in a straightforward fashion to other operators, including odd-order operators and nonlinear operators [22], and we intend to investigate this further. Efficient methods for iteratively solving the implicit equations will be another focus of our studies.

Appendix A. Constructing the computational band. This appendix gives one possible algorithm to construct the band of grid points L enveloping the surface $\mathcal{S}$ and the band of ghost points G. The algorithm is related to the "stencil set approach" in [23] and differs mainly in that for the implicit closest point method we need to store the indices and weights associated with differentiation and extension in order to build the matrices $\boldsymbol{\Delta}_h$ and E.

The algorithm proceeds in two passes. In the first pass, we build the list L of all grid points needed for the closest point extension; these are exactly the points on which the approximate solution will be propagated. This pass also identifies special *base points* $\boldsymbol{b} \in L$ which are grid points that appear in the lower-left corner of an interpolation stencil (e.g., the lower-left grid points in the shaded regions in Figure 2.1). Each of these base points $\boldsymbol{b}$ stores the corresponding interpolation stencil as a list $S_{\text{interp}(\boldsymbol{b})}$ of indices pointing into the list L.

The second pass adds the list G of ghost points; these grid points appear in a differentiation stencil but not in any of the interpolation stencils. Each point $\boldsymbol{x}$ in $L \cup G$ stores $\text{cp}(\boldsymbol{x})$, its closest point on $\mathcal{S}$, and the index (pointing into L) of the base point $\boldsymbol{b}$ of its interpolation stencil. Points $\boldsymbol{x}$ in L also store their differentiation stencil as a list $S_{\text{diff}(\boldsymbol{x})}$ of indices pointing into L and G.

Algorithm 1 implements the two passes. Given the computational band, Algorithm 2 constructs the $\boldsymbol{\Delta}_h$ and E matrices.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 1: Building L and G

```
// First Pass
Add any one grid point near S to L;
repeat
    Get next element x from L;
    foreach point y in the differentiation stencil for the point x do
        Calculate cp(y);
        Find b the base point of the interpolation stencil for cp(y);
        if b not yet processed then
            Let S_interp(b) = ∅;
            foreach z in the interpolation stencil for cp(y) do
                Look for z in L, note its index if found;
                If z is not found, add it to L and note its index;
                Add the index of z to S_interp(b);
until no more elements in L ;
// Second Pass
foreach x in L do
    Set S_diff(x) = ∅;
    foreach y in the differentiation stencil for the point x do
        Find the index in L of the base point b corresponding to cp(y). Associate this
        index and the value of cp(y) with y;
        Look for y in L and G, note its index if found;
        If not found, add it to G and note its index;
        Add the index of y (and which band it is in) to S_diff(x);
```

Algorithm 2: Forming the $\boldsymbol{\Delta}_h$ and E matrices

```
foreach index i where x_i ∈ L do
    foreach j ∈ S_diff(x_i), the differentiation stencil for x_i do
        Set [Δ_h]_{i,j} to the weight for the jth component of the differentiation stencil;
foreach index i where x_i ∈ L ∪ G do
    Retrieve the base point b for x_i, and cp(x_i);
    Compute weights w^{cp(x_i)} with barycentric Lagrange interpolation;
    foreach index j in the interpolation S_interp(b) do
        Set [E]_{i,j} = w_j^{cp(x_i)} (the weight corresponding to the jth component of the
        interpolation stencil S_interp(b));
```

REFERENCES

[1] *Bumpy Sphere, the AIM@SHAPE Shape Repository*, http://shapes.aimatshape.net, 2008.

[2] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit Runge–Kutta methods for time-dependent partial differential equations*, Appl. Numer. Math., 25 (1997), pp. 151–167.

[3] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton, *Implicit-explicit methods for time-dependent partial differential equations*, SIAM J. Numer. Anal., 32 (1995), pp. 797–823.

[4] J.-P. Berrut and L. N. Trefethen, *Barycentric Lagrange interpolation*, SIAM Rev., 46 (2004), pp. 501–517.

[5] M. Bertalmío, A. Bertozzi, and G. Sapiro, *Navier–Stokes, fluid dynamics, and image and video inpainting*, in Proceedings of IEEE-CVPR, 2001, pp. 355–362.

[6] M. Bertalmío, L.-T. Cheng, S. Osher, and G. Sapiro, *Variational problems and partial differential equations on implicit surfaces*, J. Comput. Phys., 174 (2001), pp. 759–780.

[7] M. Burger, *Finite element approximation of elliptic partial differential equations on implicit surfaces*, Comput. Vis. Sci., 12 (2009), pp. 87–100.

[8] L.-T. Cheng, P. Burchard, B. Merriman, and S. Osher, *Motion of curves constrained on surfaces using a level-set approach*, J. Comput. Phys., 175 (2002), pp. 604–644.

[9] G. Dziuk and C. Elliott, *Surface finite elements for parabolic equations*, J. Comput. Math., 25 (2007), pp. 385–407.

[10] G. Dziuk and C. Elliott, *Eulerian finite element method for parabolic PDEs on implicit surfaces*, Interfaces Free Bound., 10 (2008), pp. 119–138.

[11] M. Floater and K. Hormann, *Surface parameterization: A tutorial and survey*, in Advances in Multiresolution for Geometric Modelling, Springer, Berlin, 2005, pp. 157–186.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[12] C. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1971.
[13] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1996.
[14] J. B. Greer, *An improvement of a recent Eulerian method for solving PDEs on general geometries*, J. Sci. Comput., 29 (2006), pp. 321–352.
[15] J. B. Greer, A. L. Bertozzi, and G. Sapiro, *Fourth order partial differential equations on general geometries*, J. Comput. Phys., 216 (2006), pp. 216–246.
[16] X. Gu, Y. Wang, T. Chan, P. Thompson, and S. Yau, *Genus zero surface conformal mapping and its application to brain surface mapping*, IEEE Trans. Med. Imaging, 23 (2004), pp. 949–958.
[17] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations,* II*: Stiff and Differential-Algebraic Problems*, 2nd ed., Springer, Berlin, 1996.
[18] M. Holst, *Adaptive numerical treatment of elliptic systems on manifolds*, Adv. Comput. Math., 15 (2001), pp. 139–191.
[19] A. Hui, *Annie Hui's pig, the AIM@SHAPE Shape Repository*, http://shapes.aimatshape.net, 2008.
[20] E. Jones, T. Oliphant, and P. Peterson, *SciPy: Open Source Scientific Tools for Python*, http://www.scipy.org/Citing_SciPy, 2001.
[21] D. I. Ketcheson, C. B. Macdonald, and S. Gottlieb, *Optimal implicit strong stability preserving Runge–Kutta methods*, Appl. Numer. Math., 59 (2009), pp. 373–392.
[22] C. B. Macdonald, *The Closest Point Method for Time-dependent Processes on Surfaces*, Ph.D. thesis, Simon Fraser University, Burnaby, Canada, 2008.
[23] C. B. Macdonald and S. J. Ruuth, *Level set equations on surfaces via the closest point method*, J. Sci. Comput., 35 (2008), pp. 219–240.
[24] J. Murray, *Mathematical Biology* II*: Spatial Models and Biomedical Applications*, 3rd ed., Springer, New York, 2003.
[25] O. Nemitz, M. B. Nielsen, M. Rumpf, and R. Whitaker, *Finite element methods on very large, dynamic tubular grid encoded implicit surfaces*, SIAM J. Sci. Comput., 31 (2009), pp. 2258–2281.
[26] T. E. Oliphant, *Python for scientific computing*, Comput. Sci. Eng., 9 (2007), pp. 10–20.
[27] L. Olsen, P. Maini, and J. Sherratt, *Spatially varying equilibria of mechanical models: Application to dermal wound contraction*, Math. Biosci., 147 (1998), pp. 113–129.
[28] I. Prigogine and R. Lefever, *Symmetry breaking instabilities in dissipative systems.* II, J. Chem. Phys., 48 (1968), pp. 1695–1700.
[29] S. J. Ruuth, *Implicit-explicit methods for reaction-diffusion problems in pattern formation*, J. Math. Biol., 34 (1995), pp. 148–176.
[30] S. J. Ruuth and B. Merriman, *A simple embedding method for solving partial differential equations on surfaces*, J. Comput. Phys., 227 (2008), pp. 1943–1961.
[31] L. Saboret, M. Attene, and P. Alliez, *Laurent's Hand, the AIM@SHAPE Shape Repository*, http://shapes.aimatshape.net, 2007.
[32] D. Salac and W. Lu, *A local semi-implicit level-set method for interface motion*, J. Sci. Comput., 35 (2008), pp. 330–349.
[33] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*, Prentice–Hall, Englewood Cliffs, NJ, 1998.
[34] P. Smereka, *Semi-implicit level set methods for curvature and surface diffusion motion*, J. Sci. Comput., 19 (2003), pp. 439–455.
[35] J. Stam, *Flows on surfaces of arbitrary topology*, ACM Trans. Graph., 22 (2003), pp. 724–731.
[36] P. Tang, F. Qiu, H. Zhang, and Y. Yang, *Phase separation patterns for diblock copolymers on spherical surfaces: A finite volume method*, Phys. Rev. E (3), 72 (2005), 016710.
[37] L. Tian, C. B. Macdonald, and S. J. Ruuth, *Segmentation on surfaces with the closest point method*, in Proceedings of the ICIP09, International Conference on Image Processing, Cairo, Egypt, IEEE, 2009, to appear.
[38] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[39] A. Turing, *The chemical basis of morphogenesis*, Philos. Trans. R. Soc. Lond., B237 (1952), pp. 37–72.
[40] G. Turk, *Generating textures on arbitrary surfaces using reaction-diffusion*, Comput. Graph., 25 (1991), pp. 289–298.
[41] G. Turk and M. Levoy, *The Stanford Bunny, the Stanford 3D Scanning Repository*, http://www-graphics.stanford.edu/data/3Dscanrep, 1994.
[42] G. van Rossum et al., *The Python Programming Language*, http://www.python.org, 1991.
[43] L. Yang, A. M. Zhabotinsky, and I. R. Epstein, *Stable squares and other oscillatory Turing patterns in a reaction-diffusion model*, Phys. Rev. Lett., 92 (2004), pp. 198–303.

Downloaded 11/04/13 to 142.58.129.109. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.