Closest Point Geometry Processing:
Extensions and Applications of the Closest Point Method for Geometric Problems in Computer Graphics

by

Nathan David King

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Misha Kazhdan Professor, Department of Computer Science Johns Hopkins University
Supervisors:	Christopher Batty Professor, School of Computer Science University of Waterloo
	Steven Ruuth Professor, Department of Mathematics Simon Fraser University
Internal Members:	Toshiya Hachisuka Associate Professor, School of Computer Science University of Waterloo
	Justin Wan Professor, School of Computer Science University of Waterloo
Internal-External Member:	Sander Rhebergen Associate Professor, Department of Applied Mathematics University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis consists of research material written for publication where Nathan King was the lead author (except second author for Chapter 5) under the supervision of Dr. Christopher Batty and Dr. Steven Ruuth and in collaboration with Dr. Haozhe Su, Dr. Mridul Aanjaneya, Ryusuke Sugimoto, and Dr. Toshiya Hachisuka.

Research presented in Chapters 3–4 (excluding Section 3.4) largely consists of contributions and results sections adapted from [71], as well as sole-authored content by Nathan King. The core ideas to treat interior boundary conditions were developed and implemented by Nathan King under the supervision of Dr. Christopher Batty and Dr. Steven Ruuth. The ideas of using a sparse grid data structure and not building the full matrix system in our linear system solver were proposed by Dr. Christopher Batty and Nathan King, respectively, but implemented by Dr. Haozhe Su under the supervision of Dr. Mridul Aanjaneya. Section 3.4 consists of sole-authored content by Nathan King under the supervision of Dr. Christopher Batty and Dr. Steven Ruuth.

Chapter 5 largely consists of contributions and results sections adapted from [164] by Ryusuke Sugimoto and Nathan King under the supervision of Dr. Christopher Batty and Dr. Toshiya Hachisuka. The idea of combining Monte Carlo methods and the closest point method was proposed by Nathan King. Ryusuke Sugimoto proposed and implemented the core method in Algorithm 2, while Nathan King developed its theoretical understanding presented in Section 5.1.2. Both Nathan King and Ryusuke Sugimoto worked collaboratively on the local feature size estimation presented in Section 5.2.1.

Research presented in Chapter 6 largely consists of work adapted from [70] and extended sole-authored content by Nathan King under the supervision of Dr. Christopher Batty and Dr. Steven Ruuth. Section 6.1 consists of content adapted from [72] by Nathan King under the supervision of Dr. Steven Ruuth. Chapters 1, 2, and 7 are a combination of sole-authored content by Nathan King and material from [71] adapted to the structure of a PhD thesis.

Abstract

This thesis develops theoretical aspects and numerical methods for solving partial differential equations (PDEs) posed on any object for which closest point queries can be evaluated. In geometry processing (and computer graphics in general), objects are represented on a computer in many different ways. Requiring only closest point queries allows the methods we develop to be used with nearly any representation. Objects can be manifold or nonmanifold, open or closed, orientable or not, and of any codimension or even mixed codimension. Our work focuses on solving PDEs on manifolds using the *closest point method* (CPM), although some nonmanifold examples are also included.

We develop fundamental extensions of CPM to enable its use for the first time with many applications in geometry processing. Two major impediments stood in our way: the complexity of manifolds commonly found in geometry processing and the inability to impose *interior boundary conditions* (IBCs) with CPM. We first develop a runtime and memory-efficient implementation of the grid-based CPM that allows the treatment of highly complex manifolds (involving tens of millions of degrees of freedom) and avoids the need for GPU or distributed memory hardware. We develop a linear system solver that can improve both memory and runtime efficiency by up to 2× and 41×, respectively. We further improve runtime by up to 17× with a novel spatial adaptivity framework. We then develop a general framework for IBC enforcement that also only requires closest point queries, which finally allows for many geometry processing applications to be performed with CPM. We implicitly partition the embedding space across (extended) interior boundaries. CPM's finite difference and interpolation stencils are adapted to respect this partition while preserving up to second-order accuracy. We show that our IBC treatment provides superior accuracy and handles more general BCs than the only existing method [7].

We deviate from the common grid-based CPM and further develop a discretization-free CPM by extending a Monte Carlo method to surface PDEs. This enables CPM to enjoy common benefits of Monte Carlo methods, e.g., localized solutions, which are useful for view-dependent applications. Finally, we introduce an algorithm to compute geodesic paths that does not even require a manifold PDE; only heat flow on a 1D line and closest point queries are required. Our method is more general, robust, and always faster (up to 1000×) than the state-of-the-art for general representations [193]. Our method can be up to 100,000× faster (with high-resolution meshes) or slower (with low-resolution meshes) than the state-of-the-art in terms of runtime [152]. Convergence studies on example PDEs with analytical solutions are given throughout. We further demonstrate the effectiveness of our work for applications from geometry processing, including diffusion curves, vector field design, geodesic distance and paths, harmonic maps, and reaction-diffusion textures.

Acknowledgements

This work would not have been possible without the support of many people in both my professional and personal life. First and foremost, I am grateful for my journey, which was a privilege that is not afforded to many in this world.

My supervisors, Christopher Batty and Steve Ruuth, undoubtedly provided a terrific environment for my success with their encouragement, shared enthusiasm, and knowledge. Thank you both for allowing space for my curiosity. Thank you, Christopher, for your vast knowledge of the field of computer graphics, your helpful ideas, and for seeing the potential in my proposed closest point method research from the beginning. Thank you, Steve, for creating the closest point method, for your helpful ideas, and for helping keep my mathematics formal. Moreover, I am sincerely grateful to both of you for your understanding and support during personal difficulties. I also thank my undergraduate supervisor, Ronald Haynes, for sparking my interest in numerical methods and Joseph Teran for inspiring me to pursue my PhD in computer graphics with his talk on snow simulation.

I am grateful for the amazing collaborators I was fortunate enough to work with during my PhD. Thank you Haozhe Su, Ryusuke Sugimoto, Mridul Aanjaneya, and Toshiya Hachisuka, for your enthusiasm, for being great team members, and for investing your time and energy.

Thank you to my colleagues and friends in the Computational Motion Group who have made my PhD experience more enjoyable: JC Chang, Brooke Dolny, Ryan Goldade, Yu Gu, Rikin Gurditta, Michael Honke, Clara Kim, Jade Marcoux-Ouellet, Sina Nabizadeh, Tümay Özdemir, Jonathan Panuelos, Ryusuke Sugimoto, Joel Wretborn, and many more.

I am thankful for the real-world experience I gained through my internships at Meta. I was able to work with and alongside many brilliant researchers on intriguing projects for virtual reality, from which I learned a lot. Thank you, Ryan Goldade and Hsaio-yu Chen, for your mentorship, as well as Breannan Smith, Takaaki Shiratori, Tuur Stuyck, and Doug Roble, for being great collaborators. Thank you to all the others who made the internships fun, including Wesley Chang, Egor Larionov, Sunmin Lee, Yifei Li, and Joy Zhang.

Thank you to the friends I made at conferences for the interesting discussions and amazing experiences. These conferences recharged me when lost in the depth of my own research by talking with other researchers with such enthusiasm for their work and computer graphics in general. Thank you Noam Aigerman, Otman Benchekroun, Honglin Chen, Dave Levin (specifically for karaoke in Japan!), Derek Liu, Sina Nabizadeh, Yuta

Noma, Silvia Sellán, Nick Sharp, Rohan Sawhney, and others. A special thanks to Ryusuke Sugimoto for the long discussions we had way past my usual bedtime while sharing hotels.

Thanks to those who helped during the personal difficulties that I faced during my PhD: Christopher Batty, Rook Bridson, Wesley Chang, Hsaio-yu Chen, Ryan Goldade, Yifei Li, Doug Roble, Steve Ruuth, Tuur Stuyck, the medical teams, my personal trainer Sheldon, my recreational hockey friends, and my friends and family. My friends and family have supported me tremendously through the ups and downs of the overall PhD, even though they did not always understand what I was doing. A big thank you to my friends Kristian, Josh, Adam, Don, Ray, Levi, and Rey, as well as my parents, Rupert and Donna, and brothers Braderick and Matthew.

To Jillian, thank you for the support you were able to provide for many years.

Dedication

To those who take the path less travelled,
the ones with the determination to endure hardships,
the ones with the passion to persevere,
the ones who strive to make a difference in the world.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BC Boundary condition. v, xiii, xiv, 9, 26, 27, 29, 30, 34, 36, 38, 42, 47, 49–55, 57–60, 62, 63, 71, 75, 76, 80, 88, 118, 135, 138–140

BFS Breadth first search. 23, 24, 45

BiCGSTAB Biconjugate gradient stabilized method. xx, 6, 17, 25–28, 34, 38, 45, 76

CNC Computer numerical control. 114

CP Closest point. xiii, 10–19, 21, 25, 29–32, 43, 50–53, 57, 58, 61, 69, 76, 80, 92–94, 96, 102, 112, 114, 118, 138, 139

CPM Closest point method. v, x–xiii, xv, xvi, xviii, 5–14, 17, 18, 20, 21, 23, 25, 28–32, 34–36, 38, 43, 45, 46, 48–52, 55, 56, 62–78, 80, 81, 83, 85–88, 92, 93, 97, 102, 108, 110, 111, 116–118, 128, 135, 136, 138–140, 142

CPU Central processing unit. 23, 126

DOF Degrees of freedom. x, xiv, 17, 21, 24, 28, 31, 34–36, 38, 42, 45–49, 52–55, 57–60, 62, 69, 75, 76, 88, 135, 138, 139

FD Finite difference. xiii, 18, 20, 21, 24, 32, 50–53, 55–58, 60, 61

FEM Finite element method. 17, 138

GPU Graphics processing unit. 9, 23, 45, 126

IBC Interior boundary condition. v, xi, xiv–xvi, 6, 27, 28, 47–49, 52, 53, 55–60, 62–69, 71–76, 79, 80, 82–88, 138–140

LBFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm. 125, 163

LiDAR Light detection and ranging. 1

PDE Partial differential equation. v, xi, xii, 3–5, 8, 9, 11–19, 21, 23, 25, 28, 29, 31, 32, 47–49, 53, 57–59, 64, 69, 72, 75, 76, 83, 85, 87–90, 92–94, 96, 110, 112, 117, 135, 136, 138–140, 142

PWoS Projected walk-on-spheres. xvi, xvii, 89–91, 93, 96, 97, 102, 108–110

RRT Rapidly-exploring random trees. xviii, xx, 120–122, 124, 125, 130, 132, 134

SAT Scale axis transform. 100–103

SDF Signed distance fields. 140, 141

SNS Spherical neural surfaces. 140, 141

SOTA State-of-the-art. 116, 132, 134

WENO Weighted essentially non-oscillatory. 139

WoS Walk-on-spheres. 89, 90, 92–94, 96, 97, 112

List of Symbols

$\mathcal{C}$ Subset of $\mathcal{S}$ where interior boundary condition is enforced. x, xiv–xvii, xix, xxiii, xxiv, 47–49, 52–69, 71–76, 78–80, 83, 85–88, 93–97, 102, 104, 110, 137, 140

Δt Time-step size. 25, 63, 76, 80, 85, 87, 118–120

$\Delta_{\mathcal{S}}$ Laplace-Beltrami operator. 12, 13, 15, 18, 19, 25, 33, 35, 38, 43, 63, 64, 71, 75, 79, 87, 92, 108

Δ Cartesian Laplacian operator. 13–16, 18, 90, 92–94, 117, 118

$\mathcal{S}_{\perp}$ Manifold orthogonal to $\mathcal{S}$ along $\partial\mathcal{S}$ or $\mathcal{C}$ (defined in (4.2)). x, xiv, 48–50, 52–61, 76

$\mathcal{S}$ Manifold. xiii–xvi, xviii, xix, xxiii–xxv, 4, 6, 7, 10–25, 28–36, 38, 42–67, 69, 71–76, 78–80, 83, 85–88, 92–100, 102, 104, 107, 108, 110, 114, 116–126, 128, 130, 135–139, 142, 163, 164

$\mathbf{b}_{\mathcal{C}}$ Unit binormal vector along $\mathcal{C}$. 63, 64, 71, 79

$\Omega(\mathcal{C})$ Interior boundary subset of $\Omega(\mathcal{S})$. xiv, xxiii, 52–62, 66, 69

$\Omega(\partial\mathcal{C})$ Boundary subset of interior boundary subset $\Omega(\mathcal{C})$. 59

$\Omega(\partial\mathcal{S})$ Exterior boundary subset of $\Omega(\mathcal{S})$. xiv, 29, 42, 49–53, 55, 59

$r_{\Omega(\mathcal{S})}$ Computational tube radius of Ω defined in (3.4). xix, 20–24, 35, 36, 38, 49, 53, 60, 69, 73, 76, 136, 138, 139

$\Omega(\mathcal{S})$ Computation tube of $\mathcal{S}$ defined in (2.1). xiii, xiv, xix, xxiii, 14, 17–24, 31–36, 38, 43–46, 48–51, 53–55, 57–62, 66, 69, 73, 76, 85, 118, 120, 121, 135, 136, 138, 139

$\mathbf{n}_{\partial\mathcal{S}}$ Unit conormal vector along $\partial\mathcal{S}$. 50, 63

$\overline{\text{cp}}$ Closest point of the point "reflected" through ∂S defined in (4.3). This notation is also used for the closest point of the point reflected through $\mathcal{C}$ defined in (4.11). 50, 51, 58

$\text{cp}_{\mathcal{S}-\mathcal{C}}$ Difference between closest points to $\mathcal{S}$ and $\mathcal{C}$, i.e., $\text{cp}_{\mathcal{S}} - \text{cp}_{\mathcal{C}}$. 55–61

E Closest point extension operator. 13–16, 18, 30, 31, 92, 93

$\text{cp}_{\mathcal{S}}$ Closest point function or query for $\mathcal{S}$. xiii–xvi, xviii, xix, xxiv, 10–14, 16–21, 23, 24, 29–32, 43, 49–53, 55–61, 65–67, 72–74, 76, 80, 85–88, 92–97, 99, 104, 117–125, 130, 136–138, 142, 163, 164

$\dim(\mathcal{S})$ Dimension of manifold $\mathcal{S}$. 10, 17, 21–23, 73, 94, 97, 104, 135

d Dimension of embedding space surrounding $\mathcal{S}$. xxv, 10, 17–23, 62, 90, 92, 97, 135

$\nabla_{\mathcal{S}} \cdot$ Manifold intrinsic divergence operator. 11, 12, 75

$\nabla \cdot$ Cartesian divergence operator. 11, 12

q Number of grid points from the centre of hyper-cross FD stencil. 20, 21, 31, 34, 136

$\nabla_{\mathcal{S}}$ Manifold intrinsic gradient operator. 11, 12, 50, 63, 64, 71, 75, 79, 87, 114

∇ Cartesian gradient operator. 11, 12

p Degree of polynomial interpolant. 18, 20, 21, 31, 34, 61, 69, 71, 136

$\mathcal{I}_i$ Indices of grid points in interpolation stencil of $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$. 18, 20, 21, 51, 53, 57, 58

$\text{LFS}(\mathbf{y})$ Local feature size at $\mathbf{y} \in \mathcal{S}$. 10, 35, 36, 42, 92, 93, 138

$\text{med}(\mathcal{S})$ Medial axis of $\mathcal{S}$. xxiv, 10, 11, 92, 96, 138

$\mathbf{n}_{\mathcal{S}}$ Unit manifold normal vector. 49, 50, 60, 61, 63, 80

$\partial\mathcal{S}$ Exterior boundary of manifold §. xiii, xiv, xxiii, 6, 29–32, 42, 46, 47, 49–53, 55, 59, 63, 73, 75, 88, 117

$\text{reach}(\mathcal{S})$ Reach of $\mathcal{S}$, i.e., the minimum distance from $\mathcal{S}$ to $\text{med}(\mathcal{S})$. xix, 10, 35, 36, 38, 136, 138, 139

$u_{\mathcal{S}}$ Scalar manifold intrinsic function. xiii, 11–13, 15, 18, 19, 25, 30, 31, 33, 35, 38, 42, 43, 50, 51, 53, 58, 62, 92, 93, 107, 108

Chapter 1

Introduction

1.1 Motivation

In the field of geometry processing, the lifecycle of discrete objects (a.k.a. digital objects) starts with their creation, then they are analysed and/or manipulated in various ways, and finally ingested by us through visualization and/or input into downstream applications.

Discrete objects with astonishing realism have been created in numerous ways. Data from real-world objects can be captured using scanning equipment such as LiDAR sensors, which are common in self-driving cars for mapping their environment [195]. Real-world objects can also be captured using images from multiple views to bring them into virtual worlds (such as video games, movies, and the metaverse) with just a smartphone camera [73]. Discrete objects can be created manually by researchers, engineers, and artists. Manual creation can be tedious and time-consuming; however, many software programs have been developed to aid with this endeavour (e.g., AutoCAD [9], nTop [121], Houdini [161], Blender [48], Maya [10]). With recent developments in generative AI, many objects can be created from just text descriptions [133].

With the vast number of ways in which a discrete object can be created comes a vast number of different discrete representations. See Figure 1.1 for some example discrete representations. A commonly used representation is a mesh, which stores a list of vertices and a list of faces formed by connecting those vertices. Point clouds are obtained from LiDAR scanners and stored using the coordinate values of each point in the cloud. Objects can also be represented in functional form, e.g., parameterizations and level sets. Neural representations of objects are becoming increasingly common, such as neural signed distance fields [168] and neural radiance fields [101].

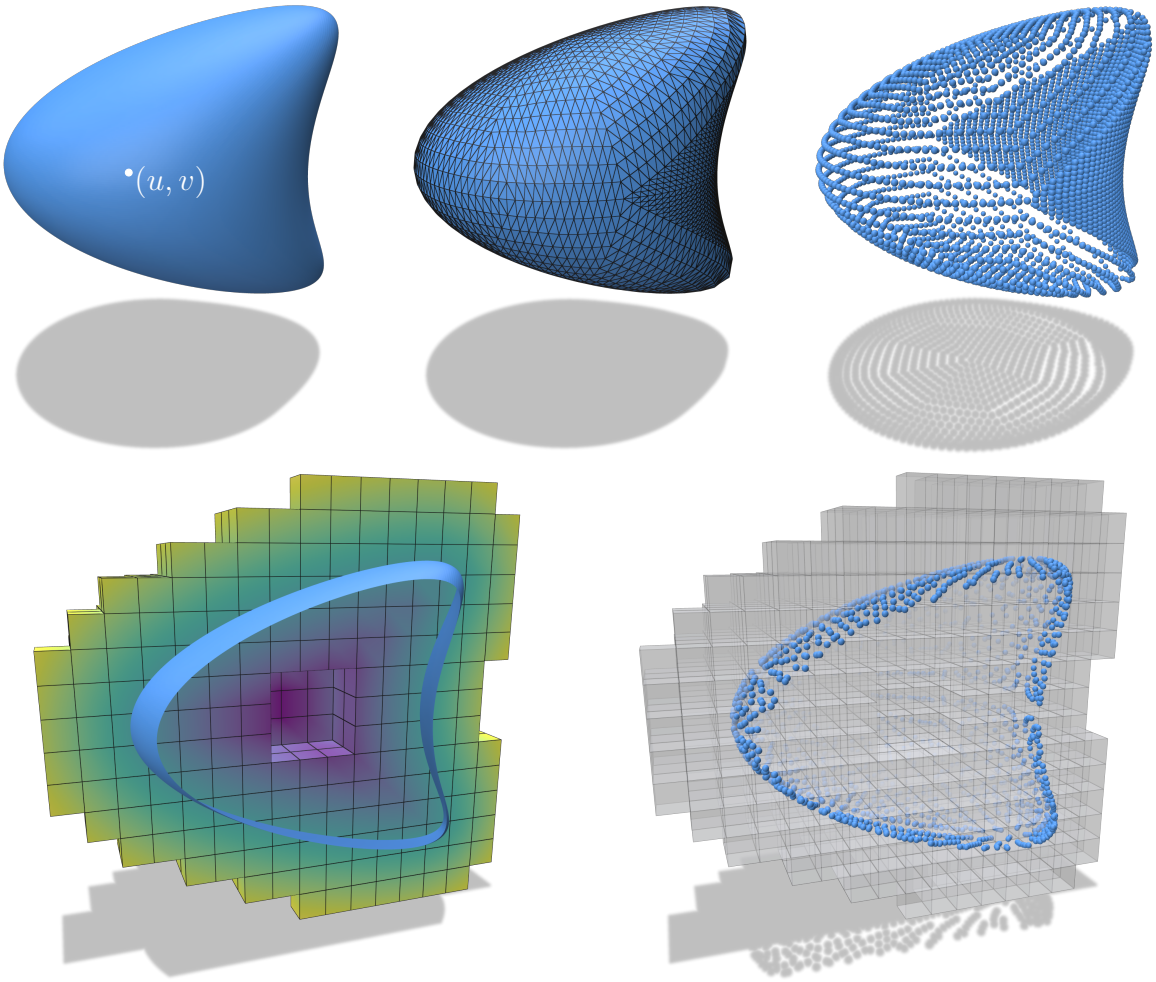


Figure 1.1: Discrete object representations that support closest point queries (bottom right) include parametrizations (top left), meshes (top middle), point clouds (top right), and level sets (bottom left).

The next step in the discrete object's lifecycle is for it to be analysed and/or manipulated by algorithms that researchers and engineers create. An unfortunate consequence of having many representations is that hours, days, weeks, or even months of research and development time are spent on algorithms that accomplish the same task, but for different discrete representations. For example, the method for computing geodesic distance by Crane et al. [33] was initially demonstrated on meshes and point clouds, but has since been implemented for subdivision surfaces, spline surfaces, and voxel surfaces [30]. The research and development time can be orders of magnitude larger than the runtime of an algorithm (usually on the order of milliseconds to days). Therefore, the existence of many representations causes wasted research and development time that could be spent more efficiently accomplishing other important tasks/applications.

Applications of geometry processing include computing properties of the shape, such as normals, Gaussian and mean curvatures, and geodesic distance and paths [34]. Other applications analyse geometry through operators such as the Laplace-Beltrami and biharmonic operators and their eigen-decompositions [5, 137]. Another common task in geometry processing is reconstructing the discrete object itself from data captured of an object in the real-world [148].

There are also segmentation tasks similar to image segmentation. For example, segmenting the inside and outside of a surface [13], cutting the surface into smaller pieces [151], or segmenting data on surfaces [172]. Surface mappings such as conformal [29], harmonic [72], and others are useful for understanding differences, to transfer data, or interpolate between the two shapes. Other applications include surface smoothing, deformation of shapes for character skinning, parameterizing surfaces for texture mapping, and remeshing.

The above geometry processing applications are implemented in libraries such as libigl [62], Geometry Central [154], and CGAL [170]. Most work in geometry processing has been done with a mesh representation of an object. The website for libigl states "Most are tailored to operate on a generic triangle mesh..." and the Geometry Central website states "...with a particular focus on meshes."

However, constructing clean meshes is a difficult problem and most meshes constructed "in the wild" consist of problematic elements (e.g., skinny triangles, self-intersections, and spurious holes) for downstream algorithms. It is possible to improve robustness using intrinsic Delaunay triangulations [156]. However, meshes usually only consist of planar elements, which restricts the order of accuracy. Higher-order patches are possible, e.g., [116], but complicate algorithms.

We focus on geometry processing algorithms that use *manifold partial differential equations*, which are partial differential equations (PDEs) whose solutions are restricted to lie

on a manifold $\mathcal{S}$. In geometry processing, the numerical solution to a manifold PDE is typically sought by approximating the manifold as a mesh and discretizing the PDE using finite element or discrete exterior calculus techniques. However, one must perform mesh generation if the input manifold is not given as a mesh. The mesh quality also strongly influences the resulting solution and therefore remeshing is required if the input mesh is of low quality or inappropriate resolution. Both mesh generation and remeshing are nontrivial tasks. Furthermore, depending on the chosen numerical method, the discretization of a particular manifold PDE can differ significantly from the corresponding discretized PDE on Cartesian domains; further analysis can be needed to derive an appropriate convergent scheme for the manifold case.

1.2 Approach

A single discrete representation would be ideal to allow for research and development on one unified framework that handles any object and any application. We could then finally create, analyse/manipulate, and ingest discrete objects without wasted effort. In this thesis, we begin to develop this unified framework using the *closest point representation* for specific applications/algorithms from geometry processing. Even though our emphasis is on geometry processing, our work extends to other areas of computer graphics and even other fields since manifold PDEs also arise naturally in applied mathematics, mathematical physics, image processing, computer vision, and fluid dynamics.

A closest point representation stores the closest point on the object (in Euclidean distance) of a query point in the space surrounding the object. Closest points benefit from their ability to represent very general objects. Both manifolds (spaces that resemble Euclidean space locally at every point) and nonmanifold objects can be represented using closest points. Input objects can be open or closed, orientable or not, and of any codimension or even mixed codimension. The complete list of the types of objects a closest point representation can handle is an open question. Therefore, even though the most common objects in geometry processing are surfaces (2D manifolds), much of the work in this thesis is applicable to more than just surfaces. Our work primarily uses 1D curves and 2D surfaces embedded in $\mathbb{R}^2$ and $\mathbb{R}^3$ that are manifold, but some examples use nonmanifold objects.

A closest point representation also has a powerful property that makes it versatile for applications: It is simultaneously an implicit and an explicit representation. To our knowledge, no other single representation possesses this property. Implicit and explicit

representations have their own benefits and limitations for different applications. For example, the (implicit) level set method for surface evolution [123] automatically handles topology changes; it does not require the complexities of manually altering the explicit representation for topology changes (see, e.g., [35]). Heiss-Synak et al. [55] recently introduced a method for surface tracking that combines a local implicit surface with an explicit mesh to avoid handling all possible types of mesh intersections while better preserving surface detail. Furthermore, Poursaeed et al. [134] developed a neural representation that simultaneously outputs an implicit and explicit surface representation. Their combined neural representation provided more accurate implicit occupancy functions and smoother explicit surfaces with more accurate normals.

Closest point queries are available for many common discrete manifold representations, as highlighted by Sawhney and Crane [140]. Therefore, our algorithms can be applied to meshes, level sets, point clouds, parametric manifolds, constructive solid geometry, neural implicit surfaces, and many more. Such generality is appealing given the increasing demand for algorithms that can ingest general "in-the-wild" and high-order geometries [60, 13, 140, 93].

Another benefit of the closest point representation is that manifold PDEs can be solved using the *closest point method* (CPM) [138]. CPM is an embedding technique that solves the manifold problem by embedding it into a surrounding Cartesian space. Other embedding methods exist, but CPM is an especially attractive instance of this strategy, as it offers a remarkable combination of simplicity, accuracy, robustness, and generality. Its simplicity, accuracy, and robustness come from its ability to leverage standard Cartesian numerical methods in the embedding space to solve the desired manifold problem, given only closest point queries to the manifold (i.e., through the closest point representation introduced above). Its generality lies in its support for diverse manifold characteristics, manifold representations, and manifold PDEs.

Moreover, the *embedding PDE* solved on the Cartesian domain is often simply the Cartesian analogue of the desired manifold PDE. Thus, CPM has been applied to the heat equation, Poisson and screened-Poisson equations, Laplace-Beltrami eigenproblem, biharmonic equation, advection-diffusion and reaction-diffusion equations, Hamilton-Jacobi equation, Navier-Stokes equation, Cahn-Hilliard equation, computation of (p-)harmonic maps, and more.

Yet, despite the desirable properties of the closest point representation and CPM, computer graphics researchers have only used them infrequently, and almost exclusively for fluid animation [57, 7, 6, 68, 108]. However, some prior work has focused on problems of relevance to geometry processing, but appears in the applied mathematics literature. For

example, Macdonald et al. [89] computed eigenvalues and eigenfunctions of the Laplace-Beltrami operator via CPM. The resulting eigenvalues of surfaces were used by Arteaga and Ruuth [5] to compute the 'Shape-DNA' [137] for clustering similar surfaces into groups. Segmentation of data on surfaces was demonstrated by Tian et al. [172] who adapted the Chan-Vese algorithm common in image processing. Different approaches to compute normals and curvatures were discussed in the original CPM paper [138, Appendix A].

1.3 Contributions

In the present work, we demonstrate CPM's wider potential value for geometry processing by extending CPM to handle several applications: diffusion curves, geodesic distance and paths, vector field design, and reaction-diffusion textures. However, crucial limitations of the existing CPM stand in the way of this objective.

To scale up to manifolds with finer details that are common in geometry processing, we develop a tailored numerical framework and solver discussed in Chapter 3. The computational domain for CPM is only required near the manifold $\mathcal{S}$, so we use a sparse grid structure to improve memory and runtime efficiency. We also develop a custom preconditioned BiCGSTAB solver to improve runtime when solving the linear system that also better utilizes memory. The combination of the sparse grid structure near $\mathcal{S}$ and the custom solver allows us to efficiently scale to tens of millions of degrees of freedom. We also develop an approach to support spatial adaptivity of the computational domain to reduce the number of degrees of freedom in regions where they are unnecessary.

CPM previously supported standard boundary conditions on the geometric (exterior) boundary of an open manifold, $\partial\mathcal{S}$, but it did not yet support accurate *interior boundary conditions* (IBCs), i.e., boundary conditions at manifold points or curves away from $\partial\mathcal{S}$. CPM's use of the embedding space makes enforcing IBCs nontrivial, but they are vital for some of the geometry processing applications above. For example, the curves in diffusion curves or the source points for geodesic distance computation generally lie on the interior of $\mathcal{S}$. Therefore, in Chapter 4, we propose a novel mechanism that enables accurate IBC enforcement for CPM in $\mathbb{R}^2$ and $\mathbb{R}^3$, while retaining its simplicity, accuracy, robustness, and generality.

With these improvements to CPM we are able to effectively handle the applications above. All applications benefit from the scalable implementation discussed in Chapter 3. Examples for diffusion curves, geodesic distance, vector field design, reaction-diffusion textures, and harmonic maps with feature curves and points are shown in Chapter 4 since they require IBCs.

Although it is most common for CPM to use standard numerical methods on a Cartesian grid in the embedding space surrounding $\mathcal{S}$, as in Chapters 3 and 4, we further explore two extensions of CPM that do not. Chapter 5 explores a discretization-free implementation of CPM using a Monte Carlo technique, while Chapter 6 introduces an algorithm for geodesic paths. We conclude with an in-depth discussion of future directions for CPM research in Chapter 7 but start with a review of the grid-based CPM in Chapter 2.

In summary, the key contributions of our work are to:

- employ a sparse grid structure and develop a custom solver for runtime and memory efficiency, which enables scaling to tens of millions of degrees of freedom;
- provide a spatial adaptivity framework for the grid-based CPM;
- introduce a framework for interior boundary conditions for the grid-based CPM with up to second-order accuracy;
- provide a discretization-free implementation of CPM by hybridizing it with the walk-on-spheres Monte Carlo method;
- introduce a novel approach to compute geodesic paths using harmonic maps;
- demonstrate the effectiveness of our work for several geometry processing tasks; and
- provide insights into promising future directions for CPM research.

Chapter 2

Closest Point Method Review

CPM was introduced by Ruuth and Merriman [138], who applied it to diffusion, advection, advection-diffusion, mean curvature flow of curves on surfaces, and reaction-diffusion. They drew inspiration from earlier embedding methods based on level sets [18, 53], while eliminating the restriction to closed manifolds, supporting more general PDEs, and allowing for narrow-banding without loss of convergence order. This chapter provides a review of the basics of CPM and the variants that exist. We briefly review the literature on CPM and related methods first, but delay the review of related work specific to our contributions to the chapters that follow. Note that abbreviations and most mathematical symbols throughout the thesis have hyperlinks to a glossary to assist the reader.

CPM has been shown to be effective for a wide range of PDEs in addition to those used by Ruuth and Merriman [138]. These include the screened-Poisson (a.k.a. positive-Helmholtz) equation [24, 98], Hamilton-Jacobi equations/level-set equations [87], biharmonic equations [88], Cahn-Hilliard equation [150, 49], Navier-Stokes equation [7, 191], construction of (p-)harmonic maps [72], and more. Although originally designed for manifold PDEs, CPM can also be applied to volumetric (codimension-0) problems and surface-to-bulk coupling scenarios [90]. Related closest point mapping approaches have also been used to handle integral equations [76, 75, 23, 26].

CPM has mostly been used on static manifolds with a uniform grid in the embedding space as the computational domain. However, Petras and Ruuth [127] combined CPM with a grid-based particle method to solve PDEs on moving surfaces. A mesh-free CPM approach was investigated in [132, 25, 128, 129, 130] using radial-basis functions. In Chapter 5, we introduce a method that is completely discretization-free, but much of the work in this thesis involves the grid-based CPM.

The CutFEM family of methods [20] represents another embedding approach. They use finite elements (rather than finite differences) on a non-conforming simplicial embedding mesh. They have been used to solve various manifold PDEs (e.g., Laplace-Beltrami [21], convection [22]). However, cutting the mesh can result in small intersections with the manifold and lead to poorly conditioned matrices or even unstable discretizations [20]. Penalty terms are added to alleviate these issues; however, they must be carefully constructed for each type of PDE.

CPM in Computer Graphics

In the computer graphics community, perhaps the most closely related work is the embedding method of Chuang et al. [28]. They solve Poisson problems using the finite element method on a 3D grid surrounding a mesh with B-spline basis functions that are subsequently restricted to the object's surface. They demonstrated geometry processing applications such as texture back-projection and curvature estimation. They also showed that the observed eigenspectra are much less dependent on the surface triangulation than with standard mesh-based methods. While their approach has some conceptual connections to CPM, it does not possess the same degree of simplicity or generality as CPM (e.g., it only applies to mesh representations). The thesis by Chuang [27] further demonstrates an extension of this approach to use locally nonmanifold grids to address narrow bottlenecks, where two pieces of a surface are close in Euclidean distance but far apart in geodesic distance. Our spatial adaptivity framework in Section 3.4 helps address the bottleneck issue without the use of nonmanifold grids. Chapter 4 does introduce a nonmanifold grid structure but with the distinct aim of handling interior BCs.

CPM itself has been applied in computer graphics, primarily for fluid animation. Hong et al. [57] used a modified CPM to evolve and control the motion of flame fronts restricted to surfaces. The work of Kim et al. [68] increased the apparent spatial resolution of an existing volumetric liquid simulation by solving a wave simulation on the liquid surface. The surface wave equation and Navier-Stokes equations were solved by Auer et al. [7] with a real-time implementation on the GPU. Auer and Westermann [6] subsequently extended this work to support deforming surfaces given by a sequence of time-varying triangle meshes (predating the moving surface work of Petras and Ruuth [127] in computational physics). Morgenroth et al. [108] employed CPM for one-way coupling between a volumetric fluid simulation and a surface fluid simulation for applications such as oil films spreading on liquid surfaces.

The use of closest points in a grid surrounding a surface has also been used for visualization purposes in computer graphics. Auer and Westermann [8] create triangulations with

a subset of the closest points as vertices. Demir and Westermann [39] visualize the surface directly via ray-casting instead. Kim and Hansen [67] visualize flow fields on surfaces, while Demir et al. [40] visualize the central tendency of a group of surfaces.

2.1 CPM Basics

Consider a manifold $\mathcal{S}$ embedded in $\mathbb{R}^d$, where $d \geq \dim(\mathcal{S})$. The closest point method uses a closest point (CP) representation of $\mathcal{S}$, which is a mapping from points $\mathbf{x} \in \mathbb{R}^d$ to points $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}) \in \mathcal{S}$. The point $\mathrm{cp}_{\mathcal{S}}(\mathbf{x})$ is defined as the closest point on $\mathcal{S}$ to $\mathbf{x}$ in Euclidean distance, i.e.,

$$\mathrm{cp}_{\mathcal{S}}(\mathbf{x}) = \arg\min_{\mathbf{y}\in\mathcal{S}} \|\mathbf{x} - \mathbf{y}\|,$$

where $\|\cdot\|$ denotes the 2-norm throughout. A CP representation can be viewed as providing both implicit and explicit representations. The mapping $\mathrm{cp}_{\mathcal{S}} : \mathbb{R}^d \to \mathcal{S}$ represents $\mathcal{S}$ implicitly: a traditional scalar (though unsigned) implicit manifold can be recovered as the zero-isocontour of the distance function $\|\mathbf{x} - \mathrm{cp}_{\mathcal{S}}(\mathbf{x})\|$. A vector-valued implicit representation of $\mathcal{S}$ is also available since $\mathbf{x} = \mathrm{cp}_{\mathcal{S}}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{S}$. Meanwhile, the closest points themselves give an explicit representation of $\mathcal{S}$, albeit without connectivity (i.e., a point cloud).

CPM embeds the manifold problem into the space surrounding $\mathcal{S}$. Consider a tubular neighbourhood (see Figure 2.1 left) defined as

$$\mathcal{N}(\mathcal{S}) = \left\{ \mathbf{x} \in \mathbb{R}^d \;\middle|\; \|\mathbf{x} - \mathrm{cp}_{\mathcal{S}}(\mathbf{x})\| \leq r_{\mathcal{N}(\mathcal{S})} \right\}, \tag{2.1}$$

where $r_{\mathcal{N}(\mathcal{S})}$ is called the *tube radius*. For general $\mathcal{S}$, the closest point $\mathrm{cp}_{\mathcal{S}}(\mathbf{x})$ is rarely unique for all $\mathbf{x} \in \mathbb{R}^d$. However, for smooth and compact manifolds, $\mathrm{cp}_{\mathcal{S}}(\mathbf{x})$ is unique for $\mathbf{x}$ in a tubular neighbourhood $\mathcal{N}(\mathcal{S})$ with a sufficiently small tube radius $r_{\mathcal{N}(\mathcal{S})}$ [96].

Uniqueness of $\mathrm{cp}_{\mathcal{S}}$ is equivalent to requiring $\mathcal{N}(\mathcal{S}) \cap \mathrm{med}(\mathcal{S}) = \emptyset$, since by definition the medial axis of $\mathcal{S}$, denoted $\mathrm{med}(\mathcal{S})$, is the subset of $\mathbb{R}^d$ that has at least two closest points on $\mathcal{S}$. The $\mathrm{reach}(\mathcal{S})$ is the minimum Euclidean distance from $\mathcal{S}$ to $\mathrm{med}(\mathcal{S})$. Thus, for a tube with constant radius, the tube radius must satisfy $r_{\mathcal{N}(\mathcal{S})} < \mathrm{reach}(\mathcal{S})$ to ensure uniqueness of $\mathrm{cp}_{\mathcal{S}}$. Hence, $\mathcal{N}(\mathcal{S})$ depends on the geometry of $\mathcal{S}$ since $\mathrm{reach}(\mathcal{S})$ depends on curvatures and bottlenecks (thin regions where the Euclidean distance is small between geodesically distant parts of $\mathcal{S}$) of $\mathcal{S}$ (see [1, Section 3]). If $r_{\mathcal{N}(\mathcal{S})}$ is allowed to vary for $\mathbf{y} \in \mathcal{S}$, then the tube radius must instead satisfy $r_{\mathcal{N}(\mathcal{S})}(\mathbf{y}) < \mathrm{LFS}(\mathbf{y})$, where LFS is the local feature size.

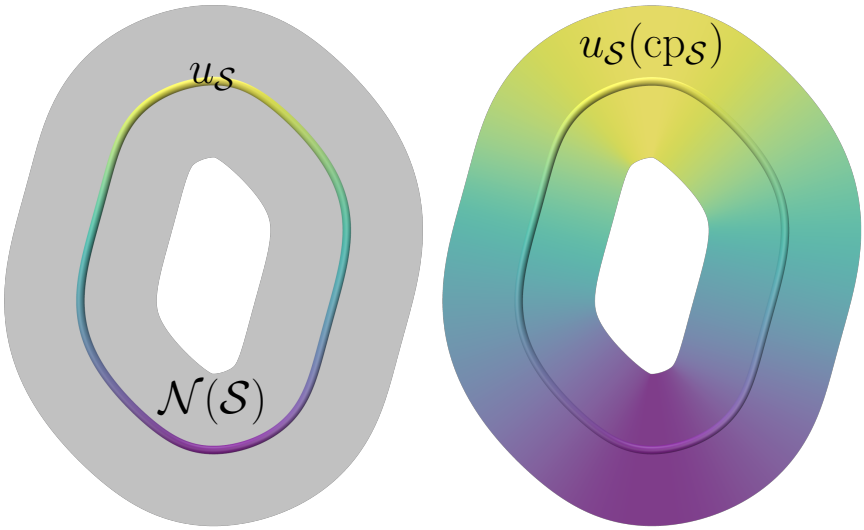


Figure 2.1: Left: A tube $\mathcal{N}(\mathcal{S})$ (grey) around a 1D curve $\mathcal{S}$ (coloured) embedded in $\mathbb{R}^2$. Right: A visualization of $u \in \mathcal{N}(\mathcal{S})$ resulting from the CP extension of $u_{\mathcal{S}} \in \mathcal{S}$.

The local feature size is defined as the minimum Euclidean distance from $\mathbf{y}$ to the medial axis $\mathrm{med}(\mathcal{S})$ [4].

To solve manifold PDEs with CPM an *embedding PDE* is constructed on $\mathcal{N}(\mathcal{S})$, whose solution agrees with the solution of the manifold PDE at points $\mathbf{y} \in \mathcal{S}$. Let $u_{\mathcal{S}}(\mathbf{y})$, for $\mathbf{y} \in \mathcal{S}$, and $u(\mathbf{x})$, for $\mathbf{x} \in \mathcal{N}(\mathcal{S})$, denote the functions defined on the manifold $\mathcal{S}$ and the neighbourhood $\mathcal{N}(\mathcal{S})$, respectively.

CPM relies on two principles to construct an embedding PDE. The manifold gradient $\nabla_{\mathcal{S}}$ and manifold divergence $(\nabla_{\mathcal{S}} \cdot)$ operators are replaced by the standard Cartesian gradient ∇ and divergence $(\nabla \cdot)$ operators via the following principles [138]:

Principle 1 *Let v be any function on $\mathcal{N}(\mathcal{S})$ that is constant along normal directions of $\mathcal{S}$. Then, on $\mathcal{S}$, manifold gradients are equivalent to standard gradients, $\nabla_{\mathcal{S}} v = \nabla v$.*

Principle 2 *Let $\mathbf{v}$ be any vector field on $\mathcal{N}(\mathcal{S})$ that is tangent to $\mathcal{S}$ and tangent to all manifolds displaced by a fixed distance from $\mathcal{S}$ (i.e., level sets of the distance function of $\mathcal{S}$). Then, on $\mathcal{S}$, $\nabla_{\mathcal{S}} \cdot \mathbf{v} = \nabla \cdot \mathbf{v}$.*

For Principle 1, the only changes in v are in the tangential direction of $\mathcal{S}$ since the function v on $\mathcal{N}(\mathcal{S})$ is constant in the normal direction. Hence, Cartesian gradients on $\mathcal{N}(\mathcal{S})$ are

equivalent to manifold gradients for points on the manifold. Similarly for Principle 2, if a vector field is constrained to the tangent space of $\mathcal{S}$ (or the tangent spaces of the level sets of the distance function to $\mathcal{S}$) it can only spread out (or compress) within that tangent space.

Extensions of manifold functions into $\mathcal{N}(\mathcal{S})$ such that the function is constant in the normal direction of $\mathcal{S}$ can be accomplished efficiently using the *closest point extension*. The CP extension is just the composition of the manifold function with the closest point function. That is, $u_{\mathcal{S}}(\text{cp}_{\mathcal{S}}(\mathbf{x}))$ is the CP extension of $u_{\mathcal{S}}$ at the point $\mathbf{x} \in \mathcal{N}(\mathcal{S})$ (see Figure 2.1 right).

Higher order derivatives can be handled by combining Principles 1 and 2. To illustrate this idea, consider the Laplace-Beltrami operator $\Delta_{\mathcal{S}} u_{\mathcal{S}} = \nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} u_{\mathcal{S}})$. If $u_{\mathcal{S}}$ is a function defined on $\mathcal{S}$, then $u_{\mathcal{S}}(\text{cp}_{\mathcal{S}})$ is constant along normal directions of $\mathcal{S}$ and therefore $\nabla_{\mathcal{S}} u_{\mathcal{S}} = \nabla u_{\mathcal{S}}(\text{cp}_{\mathcal{S}})$ on $\mathcal{S}$, by Principle 1. Principle 2 implies that $\nabla_{\mathcal{S}} \cdot (\nabla_{\mathcal{S}} u_{\mathcal{S}}) = \nabla \cdot (\nabla u_{\mathcal{S}}(\text{cp}_{\mathcal{S}}))$ on $\mathcal{S}$, since $\nabla u_{\mathcal{S}}(\text{cp}_{\mathcal{S}})$ is always tangent to the level sets of the distance function of $\mathcal{S}$. In this fashion, an embedding PDE is obtained that involves standard Cartesian derivatives and a closest point function. Some example embedding PDEs are given in Section 2.2. See the work of März and Macdonald [96] for a more thorough theoretical study of the foundations of CPM.

The solution to the embedding PDE is then solved using a chosen numerical method. The majority of this thesis uses a uniform Cartesian grid discretization of $\mathcal{N}(\mathcal{S})$, as discussed in Chapter 3 (before Section 3.4). Finite-difference methods are then used to approximate differential operators and barycentric-Lagrange interpolation is used for the CP extension on the uniform grid. Chapter 5 introduces a discretization-free numerical method for CPM using a Monte Carlo approach instead.

2.2 Variants of the Closest Point Method

In this section, we summarize different variants of CPM for solving manifold PDEs. The foundational work of Ruuth and Merriman [138] introduced CPM with time-dependent problems and explicit time-stepping (Section 2.2.1). The following two variants (Sections 2.2.2 and 2.2.3) are required to address issues that arise due to implicit time-stepping and stationary problems. The fourth and final variant (Section 2.2.4) guarantees that the solution $u \in \mathcal{N}(\mathcal{S})$ is constant in the normal direction of $\mathcal{S}$.

We use the first three variants in Chapters 3 and 4 depending on whether stationary or time-dependent problems are being solved and if explicit or implicit time-stepping is used.

Chapter 5 uses the third variant to explain the theory of our method. Chapter 6 does not use any variant of CPM; it is based on CPM work of King and Ruuth [72] that simplifies to a PDE on a 1D line for geodesic paths, so standard finite-differences in 1D can be used. The fourth variant is included here for completeness.

Section 2.2.2 discusses how the numerical stability issue from implicit time-stepping is addressed by Macdonald and Ruuth [88]. Macdonald, Brandman, and Ruuth [89] were the first to handle a stationary problem, specifically computing the eigenvalues and eigenvectors of the Laplace-Beltrami operator. Section 2.2.3 summarizes how stationary problems using the Ruuth and Merriman [138] approach are not well-posed and how this is addressed by Macdonald, Brandman, and Ruuth [89]. Interestingly, the method of Macdonald, Brandman, and Ruuth [89] is a generalization of the method used to improve the numerical stability by Macdonald and Ruuth [88]. Finally, Section 2.2.4 discusses an approach to guarantee that the solution is constant in the normal direction of the manifold. Chen and Macdonald [24] introduced this approach for stationary problems and von Glehn et al. [177] did the same for time-dependent problems.

2.2.1 Ruuth and Merriman [138] Approach

The original CPM was developed for time-dependent problems using explicit time-stepping. First, the initial manifold data ${u_{\mathcal{S}}}^0$ is extended onto $\mathcal{N}(\mathcal{S})$ using the CP extension. The CP extension operator, denoted E, extends manifold functions onto $\mathcal{N}(\mathcal{S})$ to be constant in the normal direction of $\mathcal{S}$ and is defined as $Eu_{\mathcal{S}}(\mathbf{y}) = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}}(\mathbf{y}))$. For functions $u \in \mathcal{N}(\mathcal{S})$ the extension E acts on the restriction of u to the manifold, i.e., $Eu = E(u|_{\mathcal{S}})$.

We will illustrate this variant of CPM with the heat equation

$$\frac{\partial u_{\mathcal{S}}}{\partial t} = \Delta_{\mathcal{S}} u_{\mathcal{S}}. \tag{2.2}$$

The Laplace-Beltrami operator $\Delta_{\mathcal{S}}$ in (2.2) is equivalent to the following:

$$\Delta_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{y}) = \Delta[Eu_{\mathcal{S}}](\mathbf{y}), \quad \mathbf{y} \in \mathcal{S}, \tag{2.3}$$

where Δ is the Laplace operator on $\mathcal{N}(\mathcal{S})$. The notation $\Delta[Eu_{\mathcal{S}}](\mathbf{y})$ means that we first compute $Eu_{\mathcal{S}}$, then apply the Laplacian Δ and finally evaluate at $\mathbf{y}$. The embedding PDE to be solved becomes

$$\frac{\partial u}{\partial t} = \Delta[Eu] \quad \text{on } \mathcal{N}(\mathcal{S}). \tag{2.4}$$

The following two steps are alternated:

- *Evolution.* The embedding PDE is solved on $\mathcal{N}(\mathcal{S})$ for one time-step with an explicit time-stepping method (or one stage of a Runge-Kutta method is performed).
- *CP extension.* The solution on $\mathcal{S}$ is re-extended onto $\mathcal{N}(\mathcal{S})$ by replacing u with Eu.

2.2.2 Macdonald and Ruuth [88] Approach

The naive approach for implicit time-stepping would be to apply an implicit method to (2.4). However, this can result in a numerically unstable method when discretized. Consider a uniform grid $\Omega(\mathcal{S})$ with spacing h discretizing $\mathcal{N}(\mathcal{S})$ (see Chapter 3 for details). The naive approach is unstable when the time-step is $O(h)$ because some of the eigenvalues of $\widetilde{\mathbf{M}} = \mathbf{LE}$ are in the right half of the real-complex plane (see [88, Figure 2.2]), where $\mathbf{L}$ is a finite-difference approximation of Δ and $\mathbf{E}$ is a Lagrange interpolation approximation of E on $\Omega(\mathcal{S})$.

Macdonald and Ruuth [88] realized that the CP extension was redundant for diagonal terms in $\mathbf{L}$ since $u(\mathbf{x}) = u(\text{cp}_{\mathcal{S}}(\mathbf{x}))$. With the aim of improving stability, they removed redundant CP extension operations to increase diagonal dominance. Specifically, $\widetilde{\mathbf{M}}$ was replaced by $\mathbf{M} = \mathbf{LE} - \frac{2d}{h^2}(\mathbf{I} - \mathbf{E})$. An implicit time-stepping method is then applied to

$$\frac{\partial u}{\partial t} = \mathbf{M}u \quad \text{on } \Omega(\mathcal{S}). \tag{2.5}$$

They applied their approach to problems to confirm that it was stable in practice for any time-step size.

2.2.3 Macdonald, Brandman, and Ruuth [89] Approach

The work of Macdonald, Brandman, and Ruuth [89] studied eigenproblems with CPM. The null-eigenspace of the embedding PDE is infinite-dimensional (see [89, Problem 2]) if the approach of Ruuth and Merriman [138] is used (i.e., replacing the Laplace-Beltrami operator with (2.3)), making the problem ill-posed. Therefore, a penalty method (penalizing solutions far from constant normal extensions) was introduced to restrict the null-eigenspace and provide a one-to-one correspondence between the manifold PDE eigenproblem and the embedding PDE eigenproblem. Specifically, $\Delta[Eu](\mathbf{x})$ was replaced by

$$\Delta_\gamma u(\mathbf{x}) = \Delta[Eu](\mathbf{x}) - \gamma\Big(u(\mathbf{x}) - [Eu](\mathbf{x})\Big). \tag{2.6}$$

The scalar $\gamma = \frac{2d}{\epsilon^2}$ is taken in [89] which allows large changes in the normal direction to be penalized since $\Delta_\gamma u$ will be large if $|u - Eu|$ is not $O(\epsilon^2)$.

This idea can be extended to other manifold PDEs to avoid ill-posedness of problems. Consider the Poisson equation

$$\Delta_{\mathcal{S}} u_{\mathcal{S}} = f_{\mathcal{S}}. \tag{2.7}$$

To define the embedding PDE on $\mathcal{N}(\mathcal{S})$, we also extend $f_{\mathcal{S}}$ as $f(\mathbf{x}) = [Ef_{\mathcal{S}}](\mathbf{x})$. The equation $\Delta[Eu_{\mathcal{S}}](\mathbf{x}) = f(\mathbf{x})$, for $\mathbf{x} \in \mathcal{N}(\mathcal{S})$, is ill-posed because f is constant in the normal direction of $\mathcal{S}$ but $\Delta[Eu_{\mathcal{S}}]$ is not guaranteed to be. Therefore, the embedding PDE for (2.7) becomes

$$\Delta[Eu_{\mathcal{S}}](\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x}), \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}), \tag{2.8}$$

where $g(\mathbf{x})$ is a function that compensates for $\Delta[Eu_{\mathcal{S}}]$ not being constant in the normal direction of $\mathcal{S}$. The function $g(\mathbf{x})$ is nonzero for $\mathbf{x} \in \mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$ and $g|_{\mathcal{S}} = 0$ to ensure (2.8) is consistent with the manifold PDE (2.7) on $\mathcal{S}$. Any function g with these conditions has the form $g(\mathbf{x}) = \gamma(v(\mathbf{x}) - Ev(\mathbf{x}))$, where $\gamma \in \mathbb{R}$ and $\gamma \neq 0$.

The Macdonald-Brandman-Ruuth approach (see [24, Section 2.3]) takes $v|_{\mathcal{S}} = u_{\mathcal{S}}$ to allow (2.8) to be written as an equation in one unknown, $v(\mathbf{x})$, since $Eu_{\mathcal{S}} = Ev$ (but importantly $v \neq Ev$ except on $\mathcal{S}$). Overall, the embedding PDE becomes

$$\Delta[Ev](\mathbf{x}) - \gamma\Big(v(\mathbf{x}) - [Ev](\mathbf{x})\Big) = f(\mathbf{x}), \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}). \tag{2.9}$$

2.2.4 Guaranteeing Solutions Constant in the Normal Direction

The Macdonald and Ruuth [88] and Macdonald, Brandman, and Ruuth [89] (Sections 2.2.2 and 2.2.3) approaches do not produce solutions on $\mathcal{N}(\mathcal{S})$ that are constant in the normal direction of $\mathcal{S}$. For example, rearranging (2.9) for the solution $v(\mathbf{x})$ we have

$$v(\mathbf{x}) = \frac{1}{\gamma}\Big(\Delta[Ev](\mathbf{x}) + \gamma[Ev](\mathbf{x}) + f(\mathbf{x})\Big), \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}). \tag{2.10}$$

Since $\Delta[Ev](\mathbf{x})$ is not constant in the normal direction of $\mathcal{S}$, the solution $v(\mathbf{x})$ is not (even though $[Ev](\mathbf{x})$ and $f(\mathbf{x})$ are). The Ruuth and Merriman [138] approach does result in a solution that is constant in the normal direction since a CP extension is applied in the final step. However, a CP extension can always be applied as a final step in any variant to obtain a solution constant in the normal direction on $\mathcal{N}(\mathcal{S})$.

Chen and Macdonald [24] introduce an alternative approach that directly enforces the constraint $u(\mathbf{x}) = [Eu](\mathbf{x})$, for $\mathbf{x} \in \mathcal{N}(\mathcal{S})$. Ultimately, they just need to swap the order of

the Laplacian and CP extension operators, i.e., they use $E[\Delta v]$ instead of $\Delta[Ev]$, which is justified by the following. The constraint $u(\mathbf{x}) = [Eu](\mathbf{x})$ being imposed means that Δu can replace the Laplace-Beltrami operator on $\mathcal{S}$ in the manifold PDE (2.7), thus

$$\Delta u(\mathbf{y}) = f(\mathbf{y}), \quad \mathbf{y} \in \mathcal{S}. \tag{2.11}$$

The point $\text{cp}_{\mathcal{S}}(\mathbf{x}) \in \mathcal{S}$ so we can replace $\mathbf{y}$ with $\text{cp}_{\mathcal{S}}(\mathbf{x})$ to obtain the embedding PDE on $\mathcal{N}(\mathcal{S})$ as

$$[\Delta u](\text{cp}_{\mathcal{S}}(\mathbf{x})) = f(\text{cp}_{\mathcal{S}}(\mathbf{x})), \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}),$$

or equivalently using our CP extension operator notation

$$E[\Delta u] = f, \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}), \tag{2.12}$$

where $Ef = f$ due to the idempotence of the CP extension.

The full problem then becomes

$$\begin{aligned} &E[\Delta u] = f, \\ \text{subject to} \quad &u = Eu, \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}). \end{aligned} \tag{2.13}$$

Combining the two equations results in a single embedding PDE

$$E[\Delta u] - \gamma\Big(u - Eu\Big) = f, \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}). \tag{2.14}$$

The benefit of this approach is not obvious in the continuous setting since one can easily perform a final CP extension to obtain a solution that is constant in the normal direction of $\mathcal{S}$ instead of requiring this a priori. However, Chen and Macdonald [24] note that different interpolants can be used for the CP extension E of the $E[\Delta u]$ and Eu terms. Specifically, linear and cubic interpolation can be used for the $E[\Delta u]$ and Eu terms, respectively. This gives a discretization with smaller stencils and, therefore, lower computational cost.

von Glehn et al. [177] provides an approach for time-dependent PDEs based on the same ideas. The solution being constant in the normal direction of $\mathcal{S}$ provides more benefit in this scenario since the Ruuth and Merriman [138] approach requires CP extensions after every time step (or each Runge-Kutta stage). von Glehn et al. [177] are therefore able to work within a method-of-lines formulation. In contrast, the Ruuth and Merriman [138] approach cannot be formulated as a method-of-lines technique (although the Macdonald and Ruuth [88] approach can).

Chapter 3

Scalability and Spatial Adaptivity

One may mistakenly reason that CPM is necessarily more computationally expensive in memory and runtime than methods applied directly on $\mathcal{S}$ (e.g., FEM on a triangle mesh) since CPM uses the higher-dimensional embedding space $\mathcal{N}(\mathcal{S}) \subseteq \mathbb{R}^d$. However, careful consideration of CPM's scaling behaviour shows that this is not the case. The computational domain (for the grid-based CPM) is a collection of Cartesian grid points $\Omega(\mathcal{S}) \subset \mathcal{N}(\mathcal{S})$ with uniform spacing h. We show theoretically in Section 3.2 that CPM scales with $\dim(\mathcal{S})$ rather than the dimension d of the embedding space because $\Omega(\mathcal{S})$ is a subset of the tubular neighbourhood $\mathcal{N}(\mathcal{S})$, which has finite tube-radius. In practice, naive implementations for the construction of $\Omega(\mathcal{S})$ can lead to undesirable scaling with d. Therefore, we develop a memory and runtime efficient implementation to construct $\Omega(\mathcal{S})$ that scales with $\dim(\mathcal{S})$. To further improve memory and runtime efficiency, we create a custom BiCGSTAB solver in Section 3.3 to solve linear systems resulting from the discretization of CPM. Finally, Section 3.4 introduces a spatial adaptivity framework that allows fewer DOFs to be used overall by restricting the use of high-resolution grids only to regions of $\mathcal{N}(\mathcal{S})$ where it is required. We start with a review of the commonly used discretization of CPM on the Cartesian grid $\Omega(\mathcal{S})$.

3.1 Discrete Setting of the Grid-Based CPM

In the discrete setting, the closest point $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ to each grid point $\mathbf{x}_i \in \Omega(\mathcal{S})$ is computed and stored. Discrete approximations of the CP extension and differential operators are needed to solve the embedding PDE. We illustrate how these operators are discretized for

the Laplace-Beltrami operator $\Delta[Eu_{\mathcal{S}}](\mathbf{x})$. Both the CP extension $[Eu_{\mathcal{S}}](\mathbf{x})$ and the Laplacian Δ need to be approximated. Interpolation is used to approximate the CP extension and finite differences (FDs) are used for differential operators.

The CP extension requires interpolation since $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$ is generally not a grid point in $\Omega(\mathcal{S})$. Thus, the manifold value $[Eu_{\mathcal{S}}](\mathbf{x}_i) = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i))$ is approximated by interpolating from discrete values $u_i \approx u(\mathbf{x}_i)$ stored at grid points $\mathbf{x}_i \in \Omega(\mathcal{S})$ surrounding $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$. The interpolation degree should be chosen such that interpolation error does not dominate the solution. Furthermore, the interpolation degree should be high enough to ensure a consistent approximation for the PDE, i.e., at least quadratic interpolation should be used for PDEs involving second-order differential operators like the Laplacian. In practice, cubic interpolation is commonly used for this scenario to ensure interpolation errors do not dominate. Throughout, we use barycentric-Lagrange interpolation [17] with polynomial degree p. This is an efficient and robust form of Lagrange interpolation for CPM [138, Section 2.5].

For a given grid point $\mathbf{x}_k \in \Omega(\mathcal{S})$, we have the following approximation of the closest point extension:

$$u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_k)) \approx \sum_{j \in \mathcal{I}_k} w_j^k u_j, \tag{3.1}$$

where $\mathcal{I}_k$ denotes the set of indices corresponding to grid points in the interpolation stencil for the query point $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_k)$ and w_j^k are the barycentric-Lagrange interpolation weights corresponding to each grid point in $\mathcal{I}_k$.

FD discretizations on $\Omega(\mathcal{S})$ are used to approximate a Cartesian differential operator $\mathcal{L}$ as

$$\mathcal{L}u(\mathbf{x}_i) \approx \sum_{k \in \mathcal{D}_i} l_k^i u_k, \tag{3.2}$$

where $\mathcal{D}_i$ denotes the set of indices corresponding to grid points in the FD stencil centred at the grid point $\mathbf{x}_i$. The FD weights are denoted l_k^i for each $\mathbf{x}_k$ with $k \in \mathcal{D}_i$. For example, the common second-order centred-difference for the discrete Laplacian has weights $1/h^2$ if $k \neq i$ and $-2d/h^2$ if $k = i$.

With these CP extension and differential operator approximations, the Laplace-Beltrami operator $\Delta_{\mathcal{S}} u_{\mathcal{S}}$ is approximated on $\Omega(\mathcal{S})$ as

$$\Delta u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)) \approx \sum_{k \in \mathcal{D}_i} l_k^i \left(\sum_{j \in \mathcal{I}_k} w_j^k u_j \right). \tag{3.3}$$

Matrices $\mathbf{E}$ and $\mathbf{L}$ can be constructed for the CP extension and discrete Laplacian, respectively. The standard 7-point discrete Laplacian in $\mathbb{R}^3$ (5-point in $\mathbb{R}^2$) is used. Constructing the (sparse) matrices amounts to storing stencil weights for the i-th unknown in the columns of row i. The discretization in (3.3) is equivalent to applying $\widetilde{\mathbf{M}} = \mathbf{LE}$ (introduced in Section 2.2.2) to the vector $\mathbf{u} = [u_0, u_1, \ldots, u_N]^T$. Throughout this thesis, $\mathbf{L}$ and $\mathbf{E}$ are applied directly when explicit time-stepping is used (see Section 2.2.1). For implicit time-stepping and stationary PDEs we instead use the method discussed in Sections 2.2.2 and 2.2.3 with $\gamma = 2d/h^2$. This choice of γ results in a numerically stable scheme for implicit time-stepping [88] and improved diagonal dominance for the shifted Poisson equation [24]. The matrix approximation of the Laplace-Beltrami operator becomes

$$\mathbf{M} = \text{diag}(\mathbf{L}) + (\mathbf{L} - \text{diag}(\mathbf{L}))\mathbf{E}.$$

For example, to solve the discrete embedding PDE for $\Delta_{\mathcal{S}} u_{\mathcal{S}} = f_{\mathcal{S}}$ we solve the linear system

$$\mathbf{Mu} = \mathbf{f},$$

for unknowns u_i at grid points $\mathbf{x}_i \in \Omega(\mathcal{S})$ and where the components of the vector $\mathbf{f}$ are $f_{\mathcal{S}}(\text{cp}_{\mathcal{S}}(\mathbf{x}_i))$.

Note that $f_{\mathcal{S}}(\text{cp}_{\mathcal{S}}(\mathbf{x}_i))$ is not computed using $\mathbf{E}$ since $f_{\mathcal{S}}$ will only be specified on the manifold (and sometimes only on a subset of $\mathcal{S}$). In other words, $\mathbf{E}$ is an operator that can only be applied to functions already on the grid $\Omega(\mathcal{S})$. In general, manifold PDEs involve some given data on the manifold, which must first be extended onto $\Omega(\mathcal{S})$. Examples include known functions like the source $f_{\mathcal{S}}$, initial conditions $u_{\mathcal{S}}(t = 0)$ for time-dependent problems, or boundary conditions on $\mathcal{S}$ (or interior boundary conditions discussed in Chapter 4). The necessary extension procedure depends on the specific representation of the manifold and the data. For example, if $f_{\mathcal{S}}(s)$ is given as a function on a parameteric manifold $\mathbf{r}(s)$, the inverse of the parameterization $\mathbf{r}^{-1}(s)$ could be used to obtain the parameter s_i corresponding to $\text{cp}(\mathbf{x}_i)$ so that $f_{\mathcal{S}}(s_i)$ can be assigned at $\mathbf{x}_i$. Alternatively, if the data is given discretely at the vertices of a triangulation, linear interpolation using the barycentric coordinates of the triangle containing $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ can be used to obtain the value at $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$. The important point is that the extension must still be a CP extension: data at $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ is assigned to $\mathbf{x}_i \in \Omega(\mathcal{S})$.

However, the solution to the original manifold PDE can be recovered from the solution on $\Omega(\mathcal{S})$ at any $\mathbf{y}_j \in \mathcal{S}$, $j = 1, 2, \ldots, J$, using a similar interpolation matrix to $\mathbf{E}$. The only necessary change is to construct the interpolation matrix for the new locations $\mathbf{y}_j$ instead of $\text{cp}(\mathbf{x}_i)$. This interpolation allows the solution to be transferred to any explicit representation, e.g., triangle mesh or point cloud, that may be needed for downstream

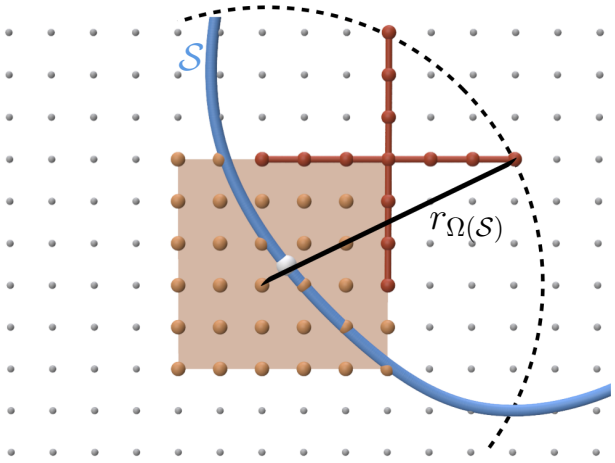


Figure 3.1: The tube radius required for CPM calculated using the interpolation stencil (orange) for $\text{cp}_{\mathcal{S}}(\mathbf{x}_k)$ (white) and FD stencil (red).

tasks. For example, many of our results are visualized interpolating onto the vertices of a triangulation. If the given manifold $\mathcal{S}$ is provided as a triangulation we use it; if a manifold can be described by a parameterization, we connect evenly spaced points in the parameter space to create a triangulation. The solution can be visualized in other ways also. Demir and Westermann [39] proposed a direct raycasting approach based on the closest points $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ for $\mathbf{x}_i \in \Omega(\mathcal{S})$. The set of $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ can also be considered a point cloud and visualized as such. Both point clouds and triangulations are visualized using `polyscope` [157] in our work.

3.2 Scalability

One could use a grid $\Omega(\mathcal{S})$ that fills a bounding box in $\mathbb{R}^d$, but this choice is inefficient since only a subset of those points (i.e., those near $\mathcal{S}$) affect the numerical solution on the manifold. It is only required that all grid points $\mathbf{x}_j$ within the interpolation stencil $\mathcal{I}_k$ of any closest point $\text{cp}_{\mathcal{S}}(\mathbf{x}_k)$ have accurate approximations of the differential operators. Barycentric-Lagrange interpolation uses a hypercube stencil of $p+1$ grid points in each dimension, where p is the polynomial degree. Consider a hyper-cross FD stencil that uses q grid points from the centre of the stencil in each dimension. Figure 3.1 shows an example interpolation stencil with $p = 5$ (orange) and a FD stencil with $q = 3$ (red). These large stencils are only used for illustration purposes; all our examples use $p = 2$ or 3 and $q = 1$.

To obtain accurate approximations of differential operators at all $\mathbf{x}_j$ with $j \in \mathcal{I}_k$, we have to extend data not only to all $\mathbf{x}_j$, but also to all $\mathbf{x}_i$ with $i \in \mathcal{D}_j$ for $j \in \mathcal{I}_k$. This mathematical description is rather terse, but one can think of sliding the centre of the red FD stencil in Figure 3.1 over all orange grid points of the interpolation stencil. The union of all grid points touched by both the interpolation and sliding FD stencils gives the required grid points. An upper bound for the computational tube-radius, $r_{\Omega(\mathcal{S})}$, can therefore be computed as follows. The interpolation stencil $\mathcal{I}_k$ surrounds $\text{cp}_{\mathcal{S}}(\mathbf{x}_k)$ in the most symmetric way, i.e., $\text{cp}_{\mathcal{S}}(\mathbf{x}_k)$ is in the centre cell for odd p, and nearest to the centre grid point for even p. Consider the case when p is odd and place an FD stencil centre at one of the corners of the interpolation hypercube. Then, calculate the distance between the tip of the FD stencil and the centre cell's grid point that is farthest away (see the two grid points joined by the black line in Figure 3.1). The upper bound for $r_{\Omega(\mathcal{S})}$ is therefore [138, 87]

$$r_{\Omega(\mathcal{S})} = h\sqrt{(d-1)\left(\frac{p+1}{2}\right)^2 + \left(q + \frac{p+1}{2}\right)^2}. \tag{3.4}$$

When p is even, (3.4) also gives the correct upper bound on $r_{\Omega(\mathcal{S})}$. A similar argument applies, but uses the fact that $\text{cp}_{\mathcal{S}}(\mathbf{x}_k)$ is at most $h/2$ away from the centre grid-point of the interpolation stencil.

The fact that the CP extension is done for all grid points within a distance $r_{\Omega(\mathcal{S})}$ to $\mathcal{S}$ is the reason CPM does not need artificial boundary conditions on the edge of the computational tube $\Omega(\mathcal{S})$. This property is in contrast to the level-set method for manifold PDEs [18] where restricting $\Omega(\mathcal{S})$ for efficiency requires artificial boundary conditions, which degrade the accuracy and convergence order of the method.

With the result in (3.4), our computational domain $\Omega(\mathcal{S})$ becomes all grid points $\mathbf{x}_i$ satisfying $\|\mathbf{x}_i - \text{cp}_{\mathcal{S}}(\mathbf{x}_i)\| \leq r_{\Omega(\mathcal{S})}$. Linear systems resulting from the embedding PDE discretization are then solved on this tubular $\Omega(\mathcal{S})$. For large systems (usually resulting from problems with $d \geq 3$) memory consumption is dominated by the storage of $\Omega(\mathcal{S})$. However, computation time is dominated by the linear system solve.

A misconception of CPM is that it is always more computationally expensive (in memory and computation time) than methods that solve the manifold PDE directly on $\mathcal{S}$ because it is an embedding method. Even though the problem is solved on a higher-dimensional space, it still scales with $\dim(\mathcal{S})$ rather than d because $\Omega(\mathcal{S})$ is within a tube $\mathcal{N}(\mathcal{S})$ of radius $r_{\Omega(\mathcal{S})}$. The scaling of CPM depends on the number of unknowns (a.k.a. DOFs), which corresponds to the number of grid points in $\Omega(\mathcal{S})$.

The number of grid points N can be approximated by

$$N \approx \frac{\text{vol}(\mathcal{N}(\mathcal{S}))}{h^d}, \tag{3.5}$$

where $\mathcal{N}(\mathcal{S})$ has a tube radius $r_{\mathcal{N}(\mathcal{S})} = r_{\Omega(\mathcal{S})}$. Denoting $s = \dim(\mathcal{S})$, the volume of $\mathcal{N}(\mathcal{S})$ can be computed using Weyl's tube formula [182, 52]:

$$\text{vol}(\mathcal{N}(\mathcal{S})) = \frac{(\pi r_{\Omega(\mathcal{S})}^2)^{(d-s)/2}}{(\frac{1}{2}(d-s))!} \sum_{c=0}^{[s/2]} \frac{k_{2c}(\mathcal{S}) r_{\Omega(\mathcal{S})}^{2c}}{(d-s+2)(d-s+4)\cdots(d-s+2c)}. \tag{3.6}$$

where $k_{2c}(\mathcal{S})$ are integrals of complicated curvature functions, but importantly they are independent of how $\mathcal{S}$ is embedded in $\mathbb{R}^d$. For example, for a 1D curve embedded in 2D

$$\text{area}(\mathcal{N}(\mathcal{S})) = 2 r_{\Omega(\mathcal{S})} \text{length}(\mathcal{S}). \tag{3.7}$$

For a 1D curve embedded in 3D

$$\text{vol}(\mathcal{N}(\mathcal{S})) = \pi r_{\Omega(\mathcal{S})}^2 \text{length}(\mathcal{S}). \tag{3.8}$$

For a 2D surface embedded in 3D

$$\text{vol}(\mathcal{N}(\mathcal{S})) = 2 r_{\Omega(\mathcal{S})} \text{area}(\mathcal{S}) + \frac{2 r_{\Omega(\mathcal{S})}^3}{3} \int_{\mathcal{S}} K d\mathcal{S}, \tag{3.9}$$

where K is the Gaussian curvature of $\mathcal{S}$.

The only term that depends on the grid resolution in (3.6) is $r_{\Omega(\mathcal{S})}$, which is proportional to h (see (3.4)). Simplifying (3.6) by lumping constants independent of h, we have

$$\text{vol}(\mathcal{N}(\mathcal{S})) = \sum_{c=0}^{[s/2]} K_{2c} r_{\Omega(\mathcal{S})}^{d-s+2c}, \tag{3.10}$$

Since $r_{\Omega(\mathcal{S})} = h\lambda$ (where λ is the square-root term in (3.4)) the approximate number of grid points is

$$N \approx \frac{\sum_{c=0}^{[s/2]} K_{2c} (h\lambda)^{d-s+2c}}{h^d} = \sum_{c=0}^{[s/2]} K_{2c} (h\lambda)^{-s+2c}. \tag{3.11}$$

Thus, $N = O(h^{-\dim(\mathcal{S})})$ since as $h \to 0$ the leading-order term is when $c = 0$. In other words, N scales exponentially with exponent $\dim(\mathcal{S})$ as the grid spacing h shrinks. This

result was known intuitively in previous CPM work, but the above gives a theoretical justification.

Ruuth and Merriman [138] used a simple procedure to construct $\Omega(\mathcal{S})$ that involved storing a uniform grid in a bounding box of $\mathcal{S}$ and computing the closest point for every grid point in the bounding box. Finally, an indexing array was used to label which grid points are within a distance $r_{\Omega(\mathcal{S})}$ of $\mathcal{S}$. The procedure of Ruuth and Merriman [138] gives linear systems that scale with $\dim(\mathcal{S})$, but memory usage and closest point computation still scale with d.

Macdonald and Ruuth [88] used a breadth-first-search (BFS), starting at a grid point near $\mathcal{S}$, that allows the number of closest points computed to scale with $\dim(\mathcal{S})$. We use a similar BFS when constructing $\Omega(\mathcal{S})$. However, Macdonald and Ruuth [88] still required storing the grid in the bounding box of $\mathcal{S}$, while we adopt sparse grid structures which achieve efficient memory use by allocating only grid points of interest instead of the full grid.

May et al. [98] overcame memory restrictions arising from storing the full bounding-box grid by using domain decomposition to solve the PDE with distributed memory parallelism. The code detailed by May et al. [99] is publicly available but requires specialized hardware to exploit distributed memory parallelism.

Auer et al. [7] also used specialized hardware, i.e., their CPM-based fluid simulator was implemented on a GPU. However, they employed a two-level sparse block structure for memory-efficient construction of $\Omega(\mathcal{S})$ that is also suitable for the CPU. A coarse-level grid in the bounding box of $\mathcal{S}$ is used to find blocks of the fine-level grid (used to solve the PDE) that intersect $\mathcal{S}$. Thus, the memory usage to construct the fine-level grid $\Omega(\mathcal{S})$ scales with $\dim(\mathcal{S})$, as desired. The coarse-level grid still scales with d, but does not cause memory issues because its resolution is much lower than the fine-level one. We adopt a similar approach for constructing $\Omega(\mathcal{S})$, although our implementation is purely CPU-based.

We use a BFS procedure to only compute $\text{cp}_{\mathcal{S}}$ near $\Omega(\mathcal{S})$. We adopt a sparse-grid data structure [149] and allocate memory for it only as needed during the BFS. We took inspiration from the development of level-set methods for surface evolution [123], which followed a similar path to an efficient implementation (see, for example, [2, 126, 183, 120, 58, 111, 149]). The BFS can be started from any grid point $\mathbf{x}_0$ within $r_{\Omega(\mathcal{S})}$ distance to the manifold. The BFS for $\Omega(\mathcal{S})$ construction is detailed in Algorithm 1. The use of a BFS could fail if $\mathcal{S}$ is composed of disjoint pieces. However, PDEs are only solved on a single, connected manifold throughout this thesis.

The computational tube-radius $r_{\Omega(\mathcal{S})}$ given by (3.4) is an upper bound on the grid points needed in $\Omega(\mathcal{S})$. The *stencil set* approach to construct $\Omega(\mathcal{S})$ given by Macdonald

Algorithm 1: BFS to construct $\Omega(\mathcal{S})$

Given $\mathbf{x}_0$ near $\mathcal{S}$, i.e., with $\|\mathbf{x}_0 - \text{cp}_\mathcal{S}(\mathbf{x}_0)\| \leq r_{\Omega(\mathcal{S})}$
Add $\mathbf{x}_0$ to $\Omega(\mathcal{S})$ and store $\text{cp}_\mathcal{S}(\mathbf{x}_0)$
Add $\mathbf{x}_0$ to the queue Q
while $Q \neq \emptyset$ **do**
  Set $\mathbf{x}_{\text{current}} \leftarrow Q$.front()
  for *each neighbour* $\mathbf{x}_{\text{nbr}}$ *of* $\mathbf{x}_{\text{current}}$ **do**
    if $\mathbf{x}_{\text{nbr}}$ *has not been visited* **then**
      Compute $\text{cp}_\mathcal{S}(\mathbf{x}_{\text{nbr}})$
      if $\|\mathbf{x}_{\text{nbr}} - \text{cp}_\mathcal{S}(\mathbf{x}_{\text{nbr}})\| \leq r_{\Omega(\mathcal{S})}$ **then**
        Add $\mathbf{x}_{\text{nbr}}$ to $\Omega(\mathcal{S})$ and store $\text{cp}_\mathcal{S}(\mathbf{x}_{\text{nbr}})$
        Add $\mathbf{x}_{\text{nbr}}$ to Q
      end
    end
  end
  Pop front of Q
end

and Ruuth [87, 88] can reduce the number of DOFs by including only the strictly necessary grid points for interpolation and FD stencils. It was shown by Macdonald and Ruuth [87] that the reduction in the number of DOFs is between 6-15% for $\mathcal{S}$ as the unit sphere. We opted for implementation simplicity over using the stencil set approach due to this low reduction in the number of DOFs. Furthermore, when the final interpolation of the solution is performed for any $\mathbf{y} \in \mathcal{S}$ (as discussed just above Section 3.2), the stencil set approach does not always provide all the necessary grid points. Only the grid points needed for the closest points to grid points, $\text{cp}_\mathcal{S}(\mathbf{x}_i)$, are guaranteed.

Note that the method of computing closest points $\text{cp}_\mathcal{S}$, and its cost, will depend on the underlying manifold representation. In Appendix A, we discuss the computation of closest points for some popular representations, including parameterized manifolds, triangulated surfaces, point clouds, signed-distance functions, and more general level-set functions (i.e., implicit manifolds). It is important that the closest point computation be accurate. For example, nearest-neighbours can only be used for point clouds if the point density is high enough (however, using the Fourier transform approach by [90] can soften the accuracy requirement).

3.3 Linear System Solver

One approach to improve the computational efficiency of CPM so that it scales well for large problems is to improve the linear system solver. The linear system resulting from CPM could be solved with direct solvers, e.g., Eigen's SparseLU is used in Section 3.4.3 for examples on 1D curves, but they are only appropriate for smaller linear systems (usually obtained with 2D embedding spaces). Iterative solvers are preferred for larger linear systems (as noted in [88, 24]), particularly for problems involving 2D surfaces embedded in $\mathbb{R}^3$ or higher.

Chen and Macdonald [24] developed a geometric multigrid solver for the manifold screened-Poisson equation. May et al. [98, 99] proposed Schwarz-based domain decomposition solvers and preconditioners for elliptic and parabolic manifold PDEs. The linear system from CPM is non-symmetric due to the CP extension, therefore Eigen's BiCGSTAB is also an option for larger systems. However, we show below that using direct solvers or Eigen's BiCGSTAB with the construction of the full matrix system can be too computationally intensive.

We implement a custom BiCGSTAB solver (with OpenMP parallelism) that avoids explicit construction of the full linear system. Our solver is more efficient, with respect to memory and computation time, compared to Eigen's SparseLU and BiCGSTAB implementations [54], as well as Intel MKL PARDISO [61]. Moreover, it circumvents the intricacies associated with implementing multigrid or domain decomposition techniques.

Our BiCGSTAB implementation closely follows Eigen's BiCGSTAB solver[1], with key differences for memory-efficiency and parallelization. This is achieved by exploiting a key property of iterative Krylov solvers: explicit construction of the system matrix is not required (in contrast to direct solvers). For iterative Krylov solvers, only the *action* of the matrix on a given input vector is required (i.e., the matrix-vector product).

Specifically, we implemented our solver with the goal of solving linear systems $\mathbf{Au} = \mathbf{f}$ with

$$\mathbf{A} = m\mathbf{I} + n\left[\text{diag}(\mathbf{L}) + (\mathbf{L} - \text{diag}(\mathbf{L}))\,\mathbf{E}\right],$$

where $m, n \in \mathbb{R}$ and common choices are $m \in \{0, 1\}$ and $n \in \{1, -\Delta t, -\Delta t/2\}$ (Δt denoting a time-step size). This generalized form for $\mathbf{A}$ supports the applications described throughout this thesis. For example, setting $m = n = 1$ results in the linear system for the screened-Poisson equation $u_{\mathcal{S}} + \Delta_{\mathcal{S}} u_{\mathcal{S}} = f_{\mathcal{S}}$. The matrices $\mathbf{E}$ and $\mathbf{L}$ are stored explicitly and the matrix-vector product $\mathbf{Au}$ is computed as follows:

[1] https://eigen.tuxfamily.org/dox/BiCGSTAB_8h_source.html

1. Compute $\mathbf{a} = \mathbf{E}\mathbf{u}$.
2. Compute $\mathbf{b} = (\mathbf{L} - \text{diag}(\mathbf{L}))\mathbf{a}$.
3. Compute $\mathbf{a} = \text{diag}(\mathbf{L})\mathbf{u}$.
4. Return $\mathbf{v} = m\mathbf{u} + n\mathbf{a} + n\mathbf{b}$.

OpenMP is used for parallelizing each of the steps over the number of grid points N.

In addition, iterative Krylov solvers allow for a *preconditioner* (i.e., approximate inverse operator) for improving convergence of the linear solver. The preconditioner step requires solving the equation $\mathbf{B}\mathbf{z} = \mathbf{r}$, where $\mathbf{B}$ is an approximation to $\mathbf{A}$ and $\mathbf{r}$ is the residual vector. Depending on the particular problem, we either use a diagonal preconditioner or a damped-Jacobi preconditioner. Computing the diagonal entries of $\mathbf{A}$ would require extra computations since the full matrix is not constructed. In practice, however, we found that the diagonal values of $m\mathbf{I} + n\text{diag}(\mathbf{L})$ are a good enough approximation. (In our experiments, we have verified that the infinity norm of the error matches the result produced by Eigen's solver.) For damped-Jacobi preconditioning, the iteration $\mathbf{u} \leftarrow \mathbf{u} + \omega\text{diag}(\mathbf{L})^{-1}\mathbf{r}$ is applied for a fixed number of iterations with $\omega = 2/3$.

Our custom BiCGSTAB solver is faster and more memory efficient than Eigen's SparseLU and BiCGSTAB implementations [54] as well as the Intel MKL PARDISO. An example of the improved efficiency is shown in Figure 3.2 for the heat problem discussed later in Section 4.3.2 with Dirichlet and zero-Neumann interior BCs. Solving the heat equation involves multiple linear system solves (i.e., one for each time step). SparseLU requires the most computation time, even though it prefactors the matrix once and just performs forward/backward solves for each time step. SparseLU also uses the most memory, as expected. PARDISO leverages parallelism during factorization, enhancing the speed of the initialization process compared to Eigen's SparseLU. However, the forward/backward solves are still performed sequentially, limiting the magnitude of the performance improvement.

Table 3.1 gives the max and average computation time speedup, T_{spdup}, and memory reduction, M_{red}, for the results in Figure 3.2. The computation time speedup compared to Eigen's SparseLU (similarly for BiCGSTAB and PARDISO) is computed as

$$T_{\text{spdup}} = T(\text{SparseLU})/T(\text{Ours}),$$

where $T(\text{SparseLU})$ and $T(\text{Ours})$ are the computation times of SparseLU and our solver, respectively. The memory reduction factor is calculated in an analogous manner with

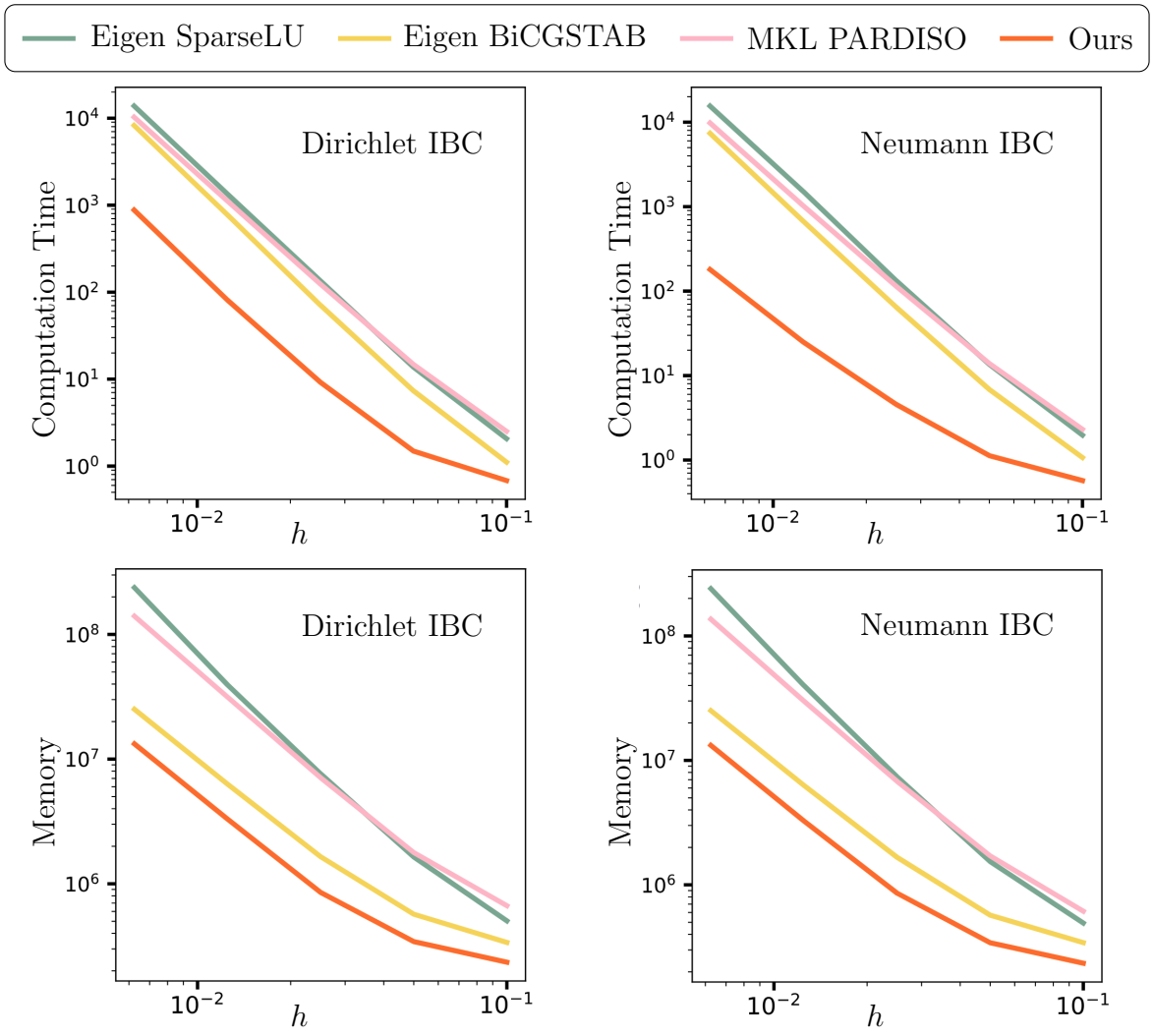


Figure 3.2: Top row: Computation time vs. h plots for the heat equation (4.14) with Dirichlet and zero-Neumann interior BCs with four solver options. Bottom row: Memory vs. h plots for the same problems and solvers. Our solver (orange) achieves the lowest computation time and memory costs.

Table 3.1: Ratios of computation time T_{spdup} and memory usage M_{red} for Eigen's SparseLU and BiCGSTAB as well as PARDISO as compared to our tailored BiCGSTAB solver, for the experiments of Figure 3.2.

Solver	IBC	T_{spdup}		M_{red}	
		Max	Avg.	Max	Avg.
Eigen's SparseLU	Dirichlet	16.6	11.8	17.9	9.1
	Neumann	86.2	38.3	18.1	9.1
Eigen's BiCGSTAB	Dirichlet	9.5	6.6	1.9	1.8
	Neumann	40.9	18.0	1.9	1.8
MKL PARDISO	Dirichlet	13.7	10.5	10.6	7.3
	Neumann	54.2	27.3	10.3	7.0

computation times replaced by memory consumption. The max and average T_{spdup} and M_{red} are computed over all h.

The speedup of our solver is significant compared to Eigen's SparseLU and BiCGSTAB as well as PARDISO. The memory reduction of our method is significant compared to Eigen's SparseLU and PARDISO, but less significant compared to Eigen's BiCGSTAB. The speedup exhibits problem-dependence since T_{spdup} factors in Table 3.1 are larger for the zero-Neumann IBC compared to the Dirichlet IBC. However, as expected, M_{red} is not problem-dependent.

3.4 Spatial Adaptivity

Another approach to reduce the computational cost of a PDE solver is to use a computational domain that adapts spatially depending on where resources are required. Spatial adaptivity allows CPM to scale better for memory usage and runtime by using a higher density of DOFs only in regions of $\mathcal{S}$ where they are needed most. Manteaux et al. [92] survey adaptive methods that are used to improve the efficiency of many problems in computer graphics. Most relevant to our work are spatially adaptive methods on structured grids. For example, quadtrees and octrees have been used to spatially adapt deformable body simulations [41, 147] and liquid simulations [158, 84, 85]. We show that the closest point extension allows us to develop a spatial adaptivity framework that does not require the implementation complexities of quadtrees/octrees. Our work is the first to provide CPM with spatial adaptivity.

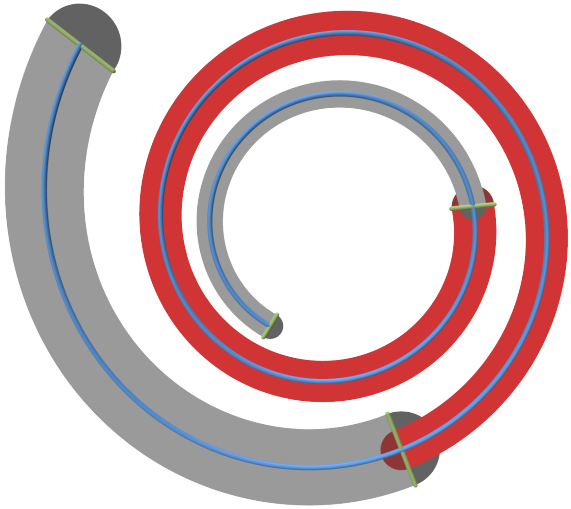


Figure 3.3: Three overlapping adaptive tubes $\mathcal{N}(\mathcal{S}_m)$ (grey and red) for a spiral curve $\mathcal{S}$ (blue). The boundary subsets $\mathcal{N}(\partial\mathcal{S}_m)$ are coloured darker and are the regions past the green lines orthogonal to $\mathcal{S}$ at the boundaries of $\mathcal{S}_m$.

3.4.1 Continuous Setting

To provide spatial adaptivity for CPM, the tube radius $r_{\mathcal{N}(\mathcal{S})}$ must be allowed to vary over different portions of $\mathcal{S}$. We divide $\mathcal{S}$ into M pieces such that $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_M$. Each subset $\mathcal{S}_m$ of $\mathcal{S}$, for $m = 1, 2, \ldots, M$, is endowed with its own tubular neighbourhood $\mathcal{N}(\mathcal{S}_m)$ that has its own tube radius $r_{\mathcal{N}(\mathcal{S}_m)}$ (see Figure 3.3). Note that each subset $\mathcal{S}_m$ is a disjoint piece of $\mathcal{S}$ except at the boundary between two subsets where $\mathcal{S}$ is duplicated.

The definition of $\mathcal{N}(\mathcal{S}_m)$ given in (2.1) includes the half-tubular region surrounding the boundary $\partial\mathcal{S}_m$ (darker regions in Figure 3.3). This half-tubular region, denoted $\mathcal{N}(\partial S_m)$, is the continuous analogue of the boundary subset $\Omega(\partial\mathcal{S})$ defined in (4.1) given by

$$\mathcal{N}(\partial\mathcal{S}_m) = \left\{ \mathbf{x} \in \mathcal{N}(\mathcal{S}_m) \;\middle|\; \mathrm{cp}_{\mathcal{S}_m}(\mathbf{x}) = \mathrm{cp}_{\partial\mathcal{S}_m}(\mathbf{x}) \right\}. \tag{3.12}$$

These boundary subsets $\mathcal{N}(\partial\mathcal{S}_m)$ provide overlaps of the tubular neighbourhoods $\mathcal{N}(\mathcal{S}_m)$ that can be used to globally join the embedding PDE solve on each subset. One could consider joining the problems on each $\mathcal{S}_m$ using a boundary condition on $\partial\mathcal{S}_m$, e.g., similar to how approximate Dirichlet BCs are used in the alternating Schwarz domain decomposition method [173]. However, the CP extension in CPM provides us with a condition that is known to hold exactly.

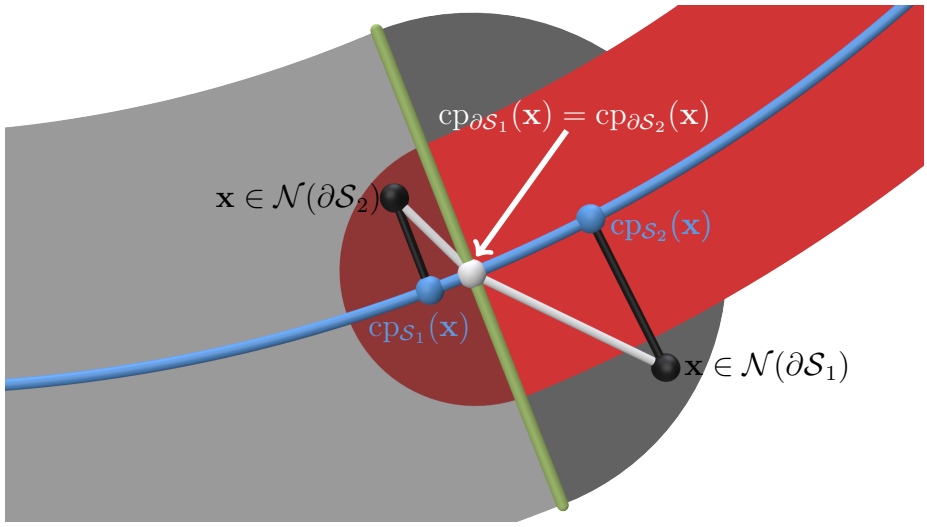


Figure 3.4: Data $u_{\mathcal{S}}$ on the manifold (blue curve) is CP extended to a point $\mathbf{x} \in \mathcal{N}(\mathcal{S}_2)$ (black) by assigning the value of $u_{\mathcal{S}}$ at $\mathrm{cp}_{\mathcal{S}_1}(\mathbf{x})$ (blue point) instead of the value at $\mathrm{cp}_{\partial\mathcal{S}_2}(\mathbf{x})$ (white). Similarly for $\mathbf{x} \in \mathcal{N}(\partial\mathcal{S}_1)$ the value of $u_{\mathcal{S}}$ at $\mathrm{cp}_{\mathcal{S}_2}(\mathbf{x})$ is CP extended to $\mathbf{x}$ instead of the value at $\mathrm{cp}_{\partial\mathcal{S}_1}(\mathbf{x})$.

Consider an example with $M = 2$ so that there is one boundary between $\mathcal{S}_1$ and $\mathcal{S}_2$ given by $\partial\mathcal{S}_1 = \partial\mathcal{S}_2 = \mathcal{S}_1 \cap \mathcal{S}_2$. Denote the CP extension operator for $\mathcal{N}(\mathcal{S})$, $\mathcal{N}(\mathcal{S}_1)$, and $\mathcal{N}(\mathcal{S}_2)$ by E, E_1, and E_2, respectively. Figure 3.4 illustrates the following scenario. The CP extension of manifold data $u_{\mathcal{S}}$ for $\mathbf{x} \in \mathcal{N}(\partial\mathcal{S}_1)$ would be $E_1 u_{\mathcal{S}}(\mathbf{x}) = u_{\mathcal{S}}(\mathrm{cp}_{\partial\mathcal{S}_1}(\mathbf{x}))$. However, this is incorrect for the global problem since globally $Eu_{\mathcal{S}} = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}})$ and $\mathrm{cp}_{\partial\mathcal{S}_1} \neq \mathrm{cp}_{\mathcal{S}}$ on $\mathcal{N}(\partial\mathcal{S}_1)$. Since $\mathrm{cp}_{\mathcal{S}_2} = \mathrm{cp}_{\mathcal{S}}$ on $\mathcal{N}(\partial\mathcal{S}_1)$, we instead perform the CP extension for $\mathbf{x} \in \mathcal{N}(\partial\mathcal{S}_1)$ using $\mathrm{cp}_{\mathcal{S}_2}$:

$$E_1 u_{\mathcal{S}}(\mathbf{x}) \equiv E_2 u_{\mathcal{S}}(\mathbf{x}) = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}_2}(\mathbf{x})), \quad \text{for} \quad \mathbf{x} \in \mathcal{N}(\partial\mathcal{S}_1), \tag{3.13}$$

and vice versa for $\mathbf{x} \in \mathcal{N}(\partial\mathcal{S}_2)$. In short, points in the boundary subset for one region simply use the CP extension for the neighbouring region. CPM can then be applied on $\mathcal{N}(\mathcal{S}_1)$ and $\mathcal{N}(\mathcal{S}_2)$ independently with no other changes to the method, e.g., no artificial or approximate BCs are needed on $\mathcal{N}(\partial\mathcal{S}_1)$ and $\mathcal{N}(\partial\mathcal{S}_2)$ since (3.13) provides a sufficient condition to couple the problem globally.

In general, $\mathcal{S}_m$ can be bordered by more than one subset of $\mathcal{S}$. Therefore, $E_m u_{\mathcal{S}}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{N}(\partial\mathcal{S}_m)$ will be composed of the CP extension from multiple subsets of $\mathcal{S}$. Let $\mathcal{J}_m$ denote the set of indices corresponding to the subsets of $\mathcal{S}$ that border $\mathcal{S}_m$. If $\mathbf{x} \in$

$\mathcal{N}(\partial\mathcal{S}_m) \cap \mathcal{N}(\mathcal{S}_j)$, $j \in \mathcal{J}_m$, then

$$E_m u_{\mathcal{S}}(\mathbf{x}) \equiv E_j u_{\mathcal{S}}(\mathbf{x}). \tag{3.14}$$

3.4.2 Discrete Setting

In the discrete setting, we construct computational tubes $\Omega(\mathcal{S}_m)$ within each tubular neighbourhood $\mathcal{N}(\mathcal{S}_m)$ for $m = 1, 2, \ldots, M$. The computational tubes have different tube-radii, $r_{\Omega(\mathcal{S}_m)}$, given by (3.4). Note that one could choose different grid spacings h, interpolation degree p, or finite difference stencil hyper-cross size q within each $\Omega(\mathcal{S}_m)$. In our numerical examples, however, we only choose a unique grid spacing h_m for the M subsets.

If $\text{cp}_{\mathcal{S}_m}$ can be defined independently for each subset $\mathcal{S}_m$, then the construction of $\Omega_m(\mathcal{S})$ can directly use the breadth-first-search approach given in Algorithm 1. However, this is rarely the case as only $\text{cp}_{\mathcal{S}}$ is usually provided for the entire manifold $\mathcal{S}$. Therefore, our approach to construct $\Omega_m(\mathcal{S})$ first constructs $\Omega(\mathcal{S}_m) \setminus \Omega(\partial\mathcal{S}_m)$, then grows the computational tube by adding grid points for $\Omega(\partial\mathcal{S}_m)$. An added benefit of this approach is that $\text{cp}_{\partial\mathcal{S}_m}$ is not necessary, since the CP extension E_m only involves $\text{cp}_{\mathcal{S}_m}$ or $\text{cp}_{\mathcal{S}_j}$ for $j \in \mathcal{J}_m$. Both $\text{cp}_{\mathcal{S}_m}$ and $\text{cp}_{\mathcal{S}_j}$ are equal to $\text{cp}_{\mathcal{S}}$ for grid points $\mathbf{x}_i \in \Omega(\mathcal{S}_m) \setminus \Omega(\partial\mathcal{S}_m)$ and $\mathbf{x}_i \in \Omega(\mathcal{S}_j) \setminus \Omega(\partial\mathcal{S}_j)$.

Our current implementation constructs the full $\Omega(\mathcal{S})$ for every h_m resolution. Of course, this is an inefficient implementation in both memory and runtime. It is used to illustrate the benefit of our adaptivity framework for the solver runtime improvement. Future work will include constructing only the subsets $\Omega(\mathcal{S}_m)$, which would allow for an understanding of the practical overhead requirements of setting up the adaptive computational tube $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2) \cup \cdots \cup \Omega(\mathcal{S}_M)$.

A sizing function $s(\mathbf{y})$ for $\mathbf{y} \in \mathcal{S}$ is used to determine the subsets $\mathcal{S}_m$, $m = 1, 2, \ldots, M$. The sizing function is problem-dependent and could be determined in many ways. For example, properties of $\mathcal{S}$ could be used to compute $s(\mathbf{y})$, such as the local feature size of $\mathcal{S}$ or the curvature of $\mathcal{S}$. Properties of the manifold PDE to be solved could also be used, e.g., subsets of $\mathcal{S}$ where large gradients in $u_{\mathcal{S}}$ are expected should use smaller h_m. Furthermore, error estimates could be used to construct $s(\mathbf{y})$. Error estimates for CPM have not been studied, but they would be an interesting area of future research. Currently, $s(\mathbf{y})$ is manually constructed to determine $\mathcal{S}_m$. Important future work would involve automatically constructing optimal $s(\mathbf{y})$ to provide the largest reduction in the number of DOFs in the linear system.

In the discrete setting, the value of $s(\mathbf{y})$ is given at a fixed set of points $\mathbf{y}_k \in \mathcal{S}$ for $k = 1, 2, \ldots, K$. These $\mathbf{y}_k$ are points where the final solution is desired, e.g., vertices of a mesh to be used for visualization. For each of the full $\Omega(\mathcal{S})$ with resolutions h_m, $m = 1, 2, \ldots, M$, we find the nearest neighbour in $\mathbf{y}_k$ to each $\text{cp}(\mathbf{x}_i)$ for grid points $\mathbf{x}_i \in \Omega(\mathcal{S})$. The grid point $\mathbf{x}_i$ is then assigned to a subset $\Omega(\mathcal{S}_m)$ based on $s(\mathbf{y}_k^\star)$ of its nearest neighbour $\mathbf{y}_k^\star$. This determines the interior of the subsets $\Omega(\mathcal{S}_m) \setminus \Omega(\partial\mathcal{S}_m)$. We then grow $\Omega(\partial\mathcal{S}_m)$ by adding any grid point that is part of the interpolation or finite-difference stencil of all $\mathbf{x}_i \in \Omega(\mathcal{S}_m) \setminus \Omega(\partial\mathcal{S}_m)$ that are not already part of the subset. We also tag each part of $\Omega(\partial\mathcal{S}_m)$ with the index j of its bordering subsets $\Omega(\mathcal{S}_j)$, $j \in \mathcal{J}_m$, so the CP extension on $\Omega(\partial\mathcal{S}_m)$ can be constructed as in (3.14).

As mentioned above, the only change in CPM necessary for our spatial adaptivity framework is the CP extension on $\Omega(\partial\mathcal{S}_m)$. The FD stencils in each $\Omega(\mathcal{S}_m)$ are the same as if $\Omega(\mathcal{S}_m)$ is an independent computational domain. There is also no theoretical restriction on the resolutions h_m between the bordering subsets. However, it would be interesting to investigate whether restrictions on the change in resolution between bordering subsets affect the accuracy and/or scalability of our spatial adaptivity framework in future work.

3.4.3 Numerical Results

Adaptivity has been used in computer graphics to allow efficient implementations of highly realistic simulations of numerous phenomena. Manteaux et al. [92] provides a review of these methods, highlighting the large number of criteria used to invoke adaptivity. These criteria could be used to construct the sizing function $s(\mathbf{y})$ to be used in our CPM adaptivity framework. In this section, we explore sizing functions based on solution gradients, local feature size, and/or curvatures.

Solution Gradients

Fine grid resolutions are required to capture large variations in the solution of a PDE. When the large variations are localized to a small region of the computational domain, e.g., shocks in fluid simulations [179], spatial adaptivity is an effective approach to improve efficiency.

To study the ability of our approach to improve efficiency for a solution with large,

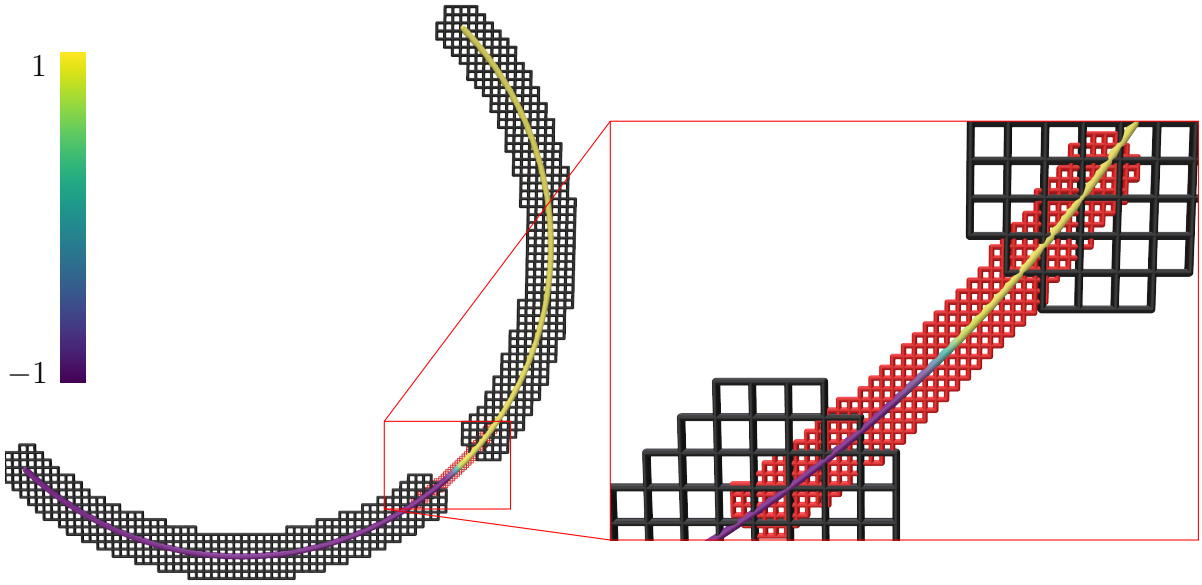


Figure 3.5: Solution of (3.15) on the arc (coloured) computed using the adaptive tube with $h_1 = h_3 = 0.025$ on $\Omega(\mathcal{S}_1)$ and $\Omega(\mathcal{S}_3)$ (black grid) and $h_2 = 0.008$ on $\Omega(\mathcal{S}_2)$ (red grid).

localized solution gradients, we solve the Poisson equation

$$\begin{aligned} \Delta_{\mathcal{S}} u_{\mathcal{S}} &= f_{\mathcal{S}}, \\ u_{\mathcal{S}}\left(-\frac{3\pi}{4}\right) &= -\tanh(100) \approx -1, \\ u_{\mathcal{S}}\left(\frac{\pi}{4}\right) &= \tanh(100) \approx 1, \end{aligned} \tag{3.15}$$

on the arc given by the portion of the unit circle with $\theta \in [-\frac{3\pi}{4}, \frac{\pi}{4}]$. The exact solution is taken as

$$u_{\mathcal{S}}(\theta) = \tanh\left(100\left(\frac{2\theta}{\pi} + \frac{1}{2}\right)\right), \tag{3.16}$$

giving

$$f_{\mathcal{S}}(\theta) = -\frac{80000}{\pi^2} \frac{\tanh\left(100\left(\frac{2\theta}{\pi} + \frac{1}{2}\right)\right)}{\cosh^2\left(100\left(\frac{2\theta}{\pi} + \frac{1}{2}\right)\right)}. \tag{3.17}$$

The solution (3.16) is approximately -1 for $\theta < -\frac{\pi}{4}$ and then rapidly transitions to approximately 1 when $\theta > -\frac{\pi}{4}$ (see Figure 3.5).

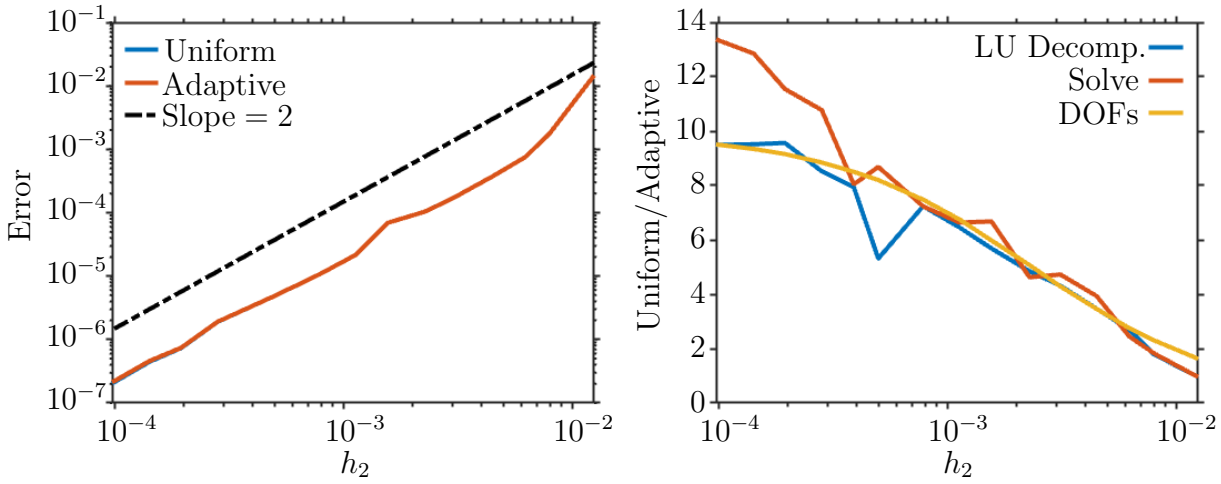


Figure 3.6: Left: Max-norm error as h_2 varies when solving (3.15). The max-norm is computed from 100 equally spaced points in the θ parameter. Right: Improvement factor of the adaptive approach compared to the uniform approaches for LU decomposition time, solve time, and number of DOFs as h_2 varies. Timings are average values from 20 trials.

We divide $\mathcal{S}$ into three subsets $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$ given by $\theta \in [-\frac{3\pi}{4}, -\frac{3\pi}{10}]$, $\theta \in [-\frac{3\pi}{10}, -\frac{2\pi}{10}]$, $\theta \in [-\frac{2\pi}{10}, \frac{\pi}{4}]$, respectively. CPM is used to solve (3.15) with cubic interpolation, second-order centred finite-differences, and second-order BCs (see Section 4.1) on the adaptive computational tube $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2) \cup \Omega(\mathcal{S}_3)$. The tube radii of $\Omega(\mathcal{S}_1)$ and $\Omega(\mathcal{S}_3)$ are held constant with $r_{\Omega(\mathcal{S}_1)} = r_{\Omega(\mathcal{S}_3)}$ computed using (3.4) with $p = 3$, $q = 1$ and $h_1 = h_3 = 0.025$. To investigate the effect of the grading between subsets we vary $r_{\Omega(\mathcal{S}_2)}$ using $h_2 = 0.025 \times 2^{-i}$ for $i = 1, 2, \ldots, 8$ and some arbitrarily chosen values $h_2 = 0.008$, 0.0045, 0.00229, 0.00113, 0.0005, 0.000283, 0.0001426. Note that these arbitrarily chosen h_2 give a $\Omega(\mathcal{S}_2)$ that is not aligned with the other two subsets, as seen in Figure 3.5 where $h_2 = 0.008$.

Eigen's SparseLU is used to solve the linear system since BiCGSTAB is slower for this example embedded in $\mathbb{R}^2$. Figure 3.6 (left) shows a convergence study for the max-norm error with the adaptive computational tube and a uniform computational tube with $h = h_2$. The errors using the adaptive and uniform computational tubes are nearly identical (lying on top of each other in the figure) and show the expected second-order convergence. Note that the error on $\mathcal{S}$ is dominated by the error on $\mathcal{S}_2$, which is why the max-error of the adaptive and uniform solutions match.

Figure 3.6 (right) shows the improvement in efficiency of the adaptive approach by dividing the LU decomposition time, forward/backward solve time, and number of DOFs

of the uniform approach by the corresponding adaptive approach values. When $h = h_2 = 0.0125$, the uniform LU decomposition and linear system solve runtimes are actually 5% faster than the adaptive. However, all other h_2 give an improvement in runtime (by 1.8×–13.4×) for the adaptive method with both the LU decomposition and linear system solve. The improvement factor of the LU decomposition closely follows the factor for the DOFs of the linear system (c.f., Figure 3.6 (right) blue and yellow), which seems to plateau around 10. However, the improvement factor for the solve runtime continues to increase over the factor for the DOFs as $h_2 \to 0$.

Local Feature Size

Manifolds with small local feature size (LFS) can drastically increase CPM's computational cost if a uniform computational tube is used. With a uniform computational tube, the tube radius must satisfy $r_{\Omega(\mathcal{S})} < \text{reach}(\mathcal{S})$ to obtain accurate results. This leads to a large number of unnecessary DOFs for subsets $\mathcal{S}_m$ of $\mathcal{S}$ where LFS $>$ reach($\mathcal{S}$). Therefore, an adaptive computational tube can improve the computational cost of CPM by allowing larger $r_{\Omega(\mathcal{S}_m)}$ for subsets $\mathcal{S}_m$ with LFS $>$ reach($\mathcal{S}$).

Curve in $\mathbb{R}^2$. Consider solving the Poisson equation

$$\begin{aligned} -\Delta_{\mathcal{S}} u_{\mathcal{S}} &= \left(\frac{2\pi}{L}\right)^2 \sin\left(\frac{2\pi s}{L}\right), \\ u_{\mathcal{S}}(0) &= u_{\mathcal{S}}(L) = 0, \end{aligned} \tag{3.18}$$

on the hyperbolic spiral, which is parameterized by

$$\mathbf{r}(\theta) = \begin{bmatrix} \dfrac{\cos\theta}{\theta} \\ \dfrac{\sin\theta}{\theta} \end{bmatrix}, \tag{3.19}$$

and is shown in Figure 3.7 with $\theta \in [0.3, 23]$. The solution to (3.18) is $u_{\mathcal{S}} = \sin(2\pi s/L)$, where L is the total length of the spiral. Note that the arc-length s is used in (3.18), which can be computed from θ via

$$s = \int_{\theta_0}^{\theta} \frac{\sqrt{1+\theta^2}}{\theta^2} d\theta = \left[\ln\left(\theta + \sqrt{1+\theta^2}\right) - \frac{\sqrt{1+\theta^2}}{\theta} \right]_{\theta_0}^{\theta}. \tag{3.20}$$

Table 3.2: Attribute comparison when solving (3.18) on the hyperbolic spiral with uniform computational tubes versus adaptive. Timings are average values over 20 trials. The error is computed at 5000 equally spaced points in θ parameter.

	Uniform $h = 0.08$		Uniform $h = 0.00125$		Adaptive
DOFs	470	(0.1×)	34750	(6.7×)	5128
LU Decomp. Time (s)	1.6×10^{-3}	(0.1×)	8.6×10^{-2}	(5.3×)	1.6×10^{-2}
Solve Time (s)	1.9×10^{-5}	(0.1×)	1.4×10^{-3}	(6.9×)	2.0×10^{-4}
Max Error	43.8	(1626×)	4.4×10^{-4}	(0.02×)	2.7×10^{-2}
Average Error	41.8	(7326×)	4.79×10^{-6}	(0.009×)	5.7×10^{-3}

The LFS of the hyperbolic spiral decreases as θ increases, forcing $r_{\Omega(\mathcal{S})}$ to be small near $\theta = 23$. To obtain an accurate solution, shown in Figure 3.7 (middle row), we take $h = 0.00125$ when using cubic interpolation, second-order centred finite-differences, and second-order BCs so that $r_{\Omega(\mathcal{S})} < \text{reach}(\mathcal{S})$. However, such a fine h is unnecessary for much of $\mathcal{S}$; the $\Omega(\mathcal{S})$ cannot even be seen until the second zoom (blue) in Figure 3.7 (middle row), which highlights the multiresolution nature of the hyperbolic spiral. Unfortunately, using a uniform $\Omega(\mathcal{S})$ with larger h is not a viable option. Figure 3.7 (top row) shows that using $h = 0.08$ gives an extremely inaccurate solution. This inaccurate solution is due to CPM mixing data from parts of $\mathcal{S}$ that are close in Euclidean distance but far in geodesic distance because $h = 0.08$ does not resolve $\mathcal{S}$ well for larger θ.

The adaptive computational tube $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2) \cup \cdots \cup \Omega(\mathcal{S}_7)$ in Figure 3.7 (bottom row) provides an improvement in computational efficiency at the cost of approximately $10\times$ the range of errors using $h = 0.00125$ uniformly. The subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_7$ are divided at $\theta = 1.55, 2.3, 4.2, 6.8, 10, 17$ and h_m is halved between subsets as θ increases starting with $h_1 = 0.08$. Table 3.2 shows that using $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2) \cup \cdots \cup \Omega(\mathcal{S}_7)$ saves almost $7\times$ the number of DOFs and reduces the solve time by a similar factor. The LU decomposition of the linear system requires the most runtime when solving the system, which is about $5\times$ faster with our adaptive approach. The adaptive CPM requires around $10\times$ the computational cost compared to using $\Omega(\mathcal{S})$ with $h = 0.08$, but this $\Omega(\mathcal{S})$ produces an unusable solution. The max and average errors of our solution are about $50\times$ and $100\times$ larger, respectively, than the solution with a uniform $h = 0.00125$, but the solutions are visually indistinguishable.

Surface in $\mathbb{R}^3$. The runtimes for curves embedded in $\mathbb{R}^2$ are small enough that adaptivity may be of little practical importance. However, for surfaces embedded in $\mathbb{R}^3$, adaptivity can provide practical benefits in runtime. Furthermore, the reduction in the number of DOFs

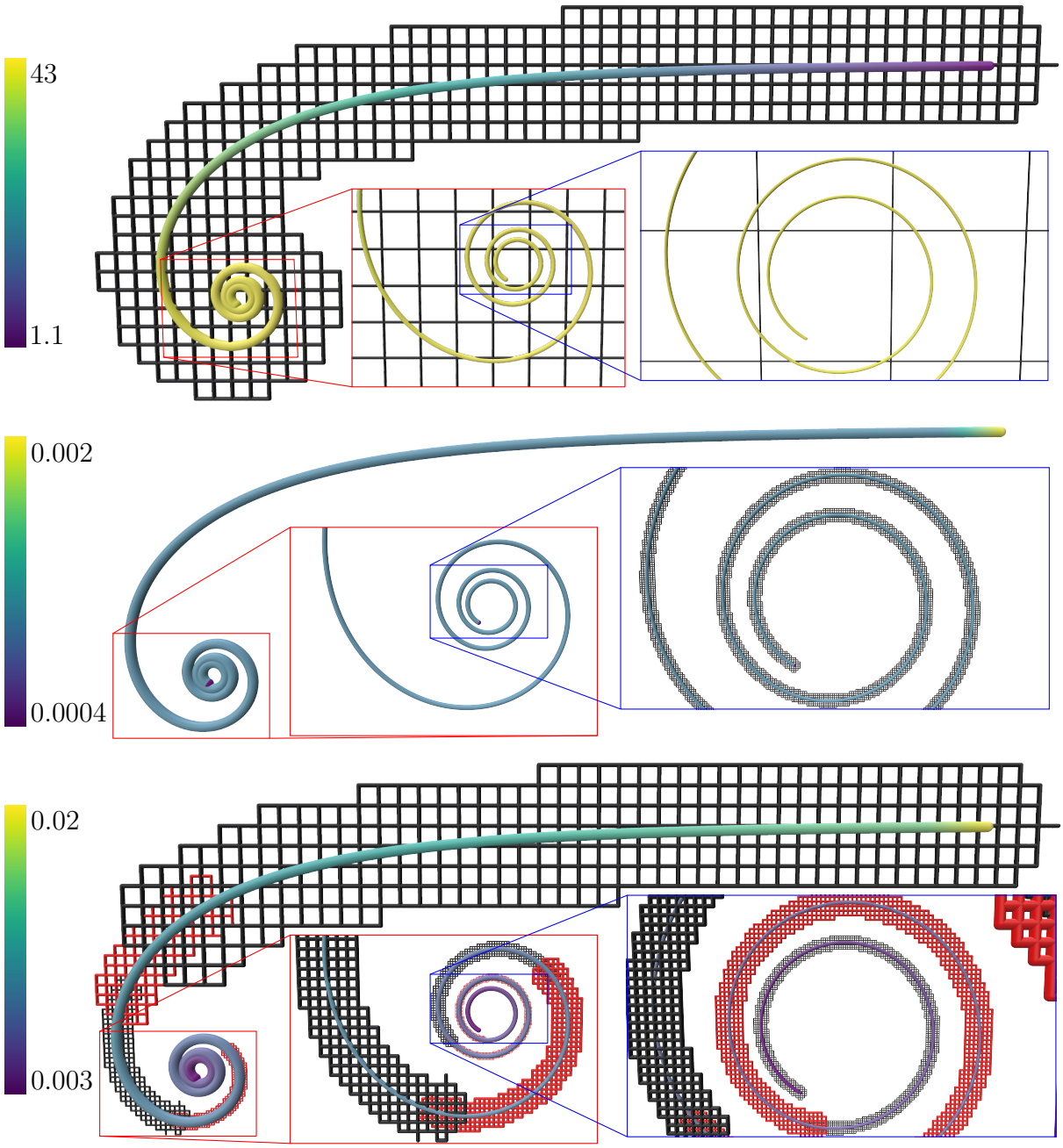


Figure 3.7: Error in the solution of (3.18) on a hyperbolic spiral using a uniform tube with $h = 0.08$ (top row, black) or $h = 0.00125$ (middle row, black) or an adaptive tube with varying h_m (bottom row, black and red).

can mean the difference between being able to store the computational tube in memory or not. In geometry processing and computer graphics in general, the focus is usually on surfaces in 3D, but CPM also extends to problems in any dimension where these runtime and memory savings would be even more advantageous.

Consider a spiral sheet surface constructed by extending the hyperbolic spiral in the z-direction, creating a surface akin to a rolled-up sheet of paper with width W (see Figure 3.9). The example below takes $\theta \in [0.3, 12.5]$ and $z \in [0, 4]$. We solve the screened-Poisson equation

$$-\Delta_{\mathcal{S}} u_{\mathcal{S}} + u_{\mathcal{S}} = f_{\mathcal{S}}, \tag{3.21}$$

on this surface with zero-Neumann boundary conditions and

$$f_{\mathcal{S}}(s, z) = \left(1 + \frac{4\pi^2(L^2 + W^2)}{L^2 W^2}\right) \cos\left(\frac{2\pi s}{L}\right) \cos\left(\frac{2\pi(z - z_0)}{W}\right). \tag{3.22}$$

The exact solution to (3.21) with (3.22) is

$$u_{\mathcal{S}}(s, z) = \cos\left(\frac{2\pi s}{L}\right) \cos\left(\frac{2\pi(z - z_0)}{W}\right). \tag{3.23}$$

Figures 3.8–3.10 show the computational tubes, the solution, and the errors when solving (3.21) on the spiral sheet using a uniform $\Omega(\mathcal{S})$ with $h = 0.064$ (top row) and $h = 0.008$ (middle row) and an adaptive $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2) \cup \cdots \cup \Omega(\mathcal{S}_5)$ (bottom row). The subsets $\mathcal{S}_1, \ldots, \mathcal{S}_5$ are taken to be constant in the z parameter and divided at $\theta = 1.7, 3.1, 4.7, 11$. The resolutions used on $\Omega(\mathcal{S}_1), \ldots, \Omega(\mathcal{S}_5)$ are $h_m = 0.064, 0.032, 0.016, 0.012, 0.008$, respectively. The same interpolation and finite-difference methods that were used for the hyperbolic spiral above are used here, but only first-order BCs are imposed (which are the natural BCs for CPM as explained later in Section 4.1). Our BiCGSTAB solver is used for this problem embedded in $\mathbb{R}^3$ since it is more efficient than SparseLU.

In Figure 3.8 (top row) the $\Omega(\mathcal{S})$ with $h = 0.064$ is unable to resolve the tight roll of the sheet. This results in a solution that is visually incorrect (see Figure 3.9) and quantitatively has large errors (see Figure 3.10). Note that the colour scales in Figures 3.9 and 3.10 are set to the range of values for the adaptive approach. All values outside this range for the uniform solution with $h = 0.064$ are capped at the purple (min) or yellow (max) colours. The actual range of the uniform solution with $h = 0.064$ is -1.5 to 1.4, and the error range is -0.5 to 0.4. The max and average errors are $25\times$ and $178\times$ larger, respectively, for the uniform solution with $h = 0.064$ compared to the adaptive solution (see Table 3.3).

To obtain a uniform $\Omega(\mathcal{S})$ with $r_{\Omega(\mathcal{S})} < \text{reach}(\mathcal{S})$ we must take $h = 0.008$. This results in a $\Omega(\mathcal{S})$ with ~ 2.9 million DOFs and requires about 22 minutes to compute the solution

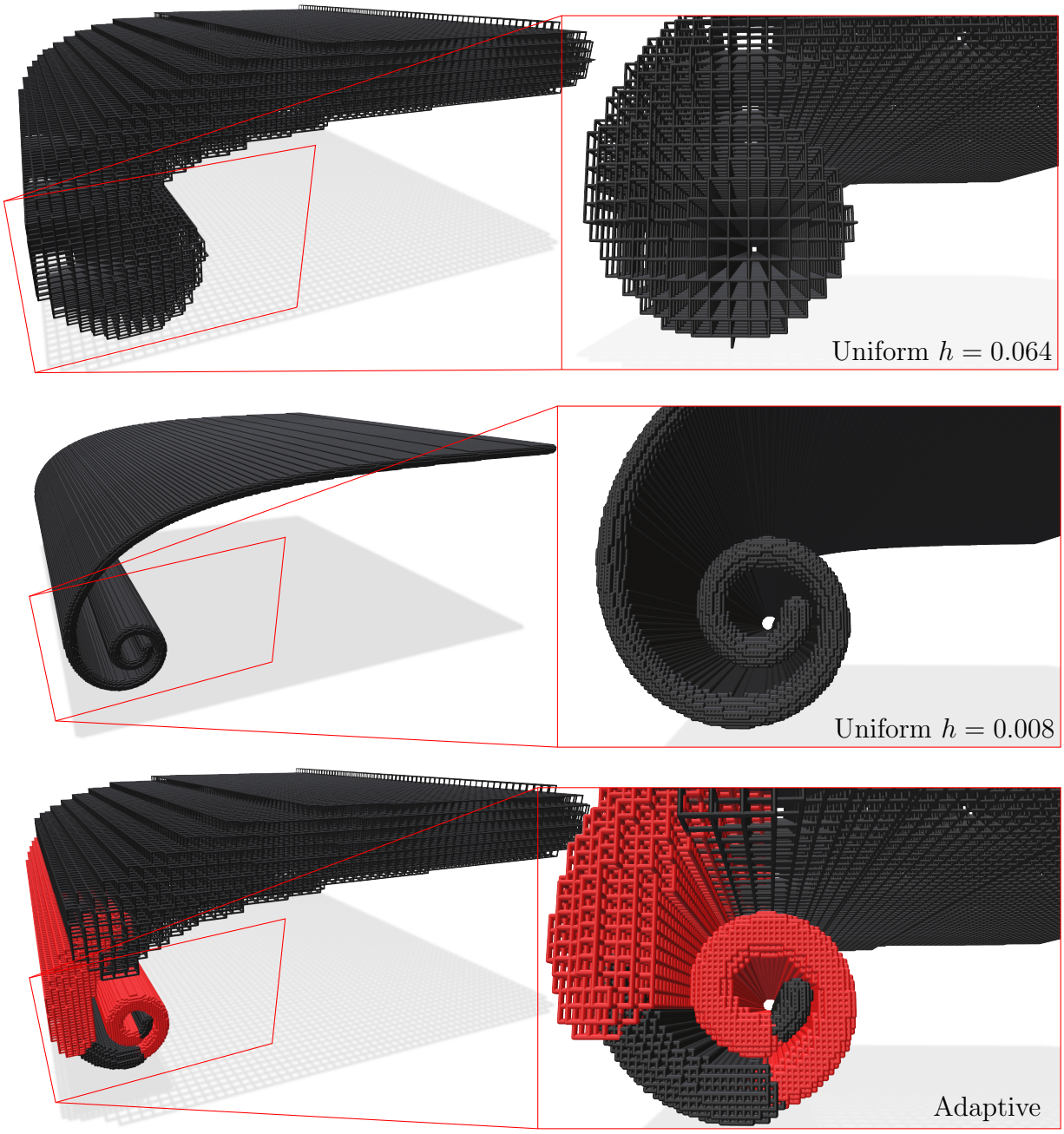


Figure 3.8: Computational tubes used when solving (3.21) on the spiral sheet with the uniform computational tubes and the adaptive tube.

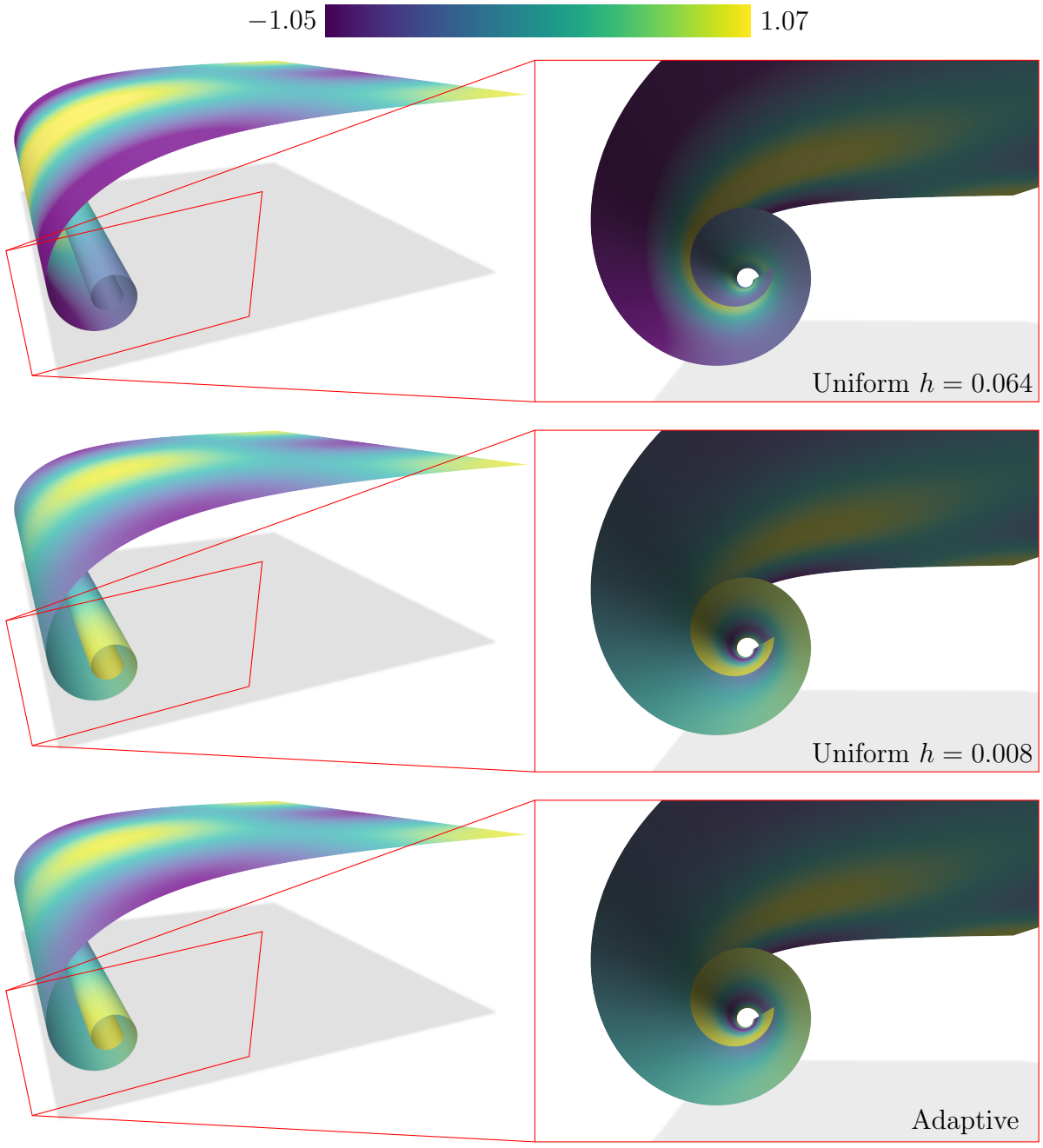


Figure 3.9: Numerical solution when solving (3.21) on the spiral sheet with the uniform computational tubes and the adaptive tube.

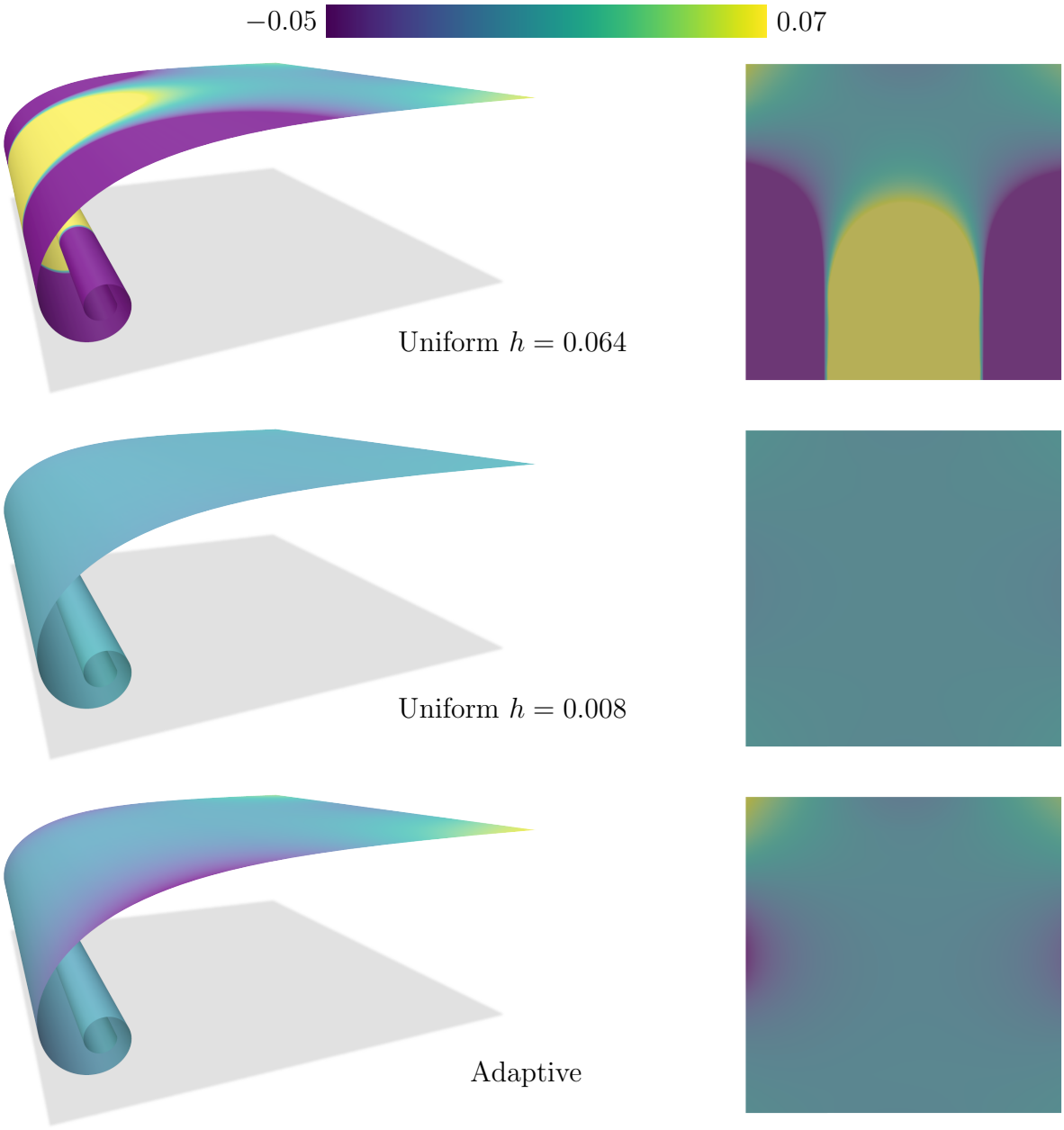


Figure 3.10: Solution errors when solving (3.21) on the spiral sheet with the uniform computational tubes and the adaptive tube. The error is shown on the spiral sheet directly (left column) and with the sheet unrolled (right column) and scaled such that $L = W = 1$.

Table 3.3: Attribute comparison when solving (3.21) on the spiral sheet with uniform computational tubes versus adaptive. The error is computed at 40,000 equally spaced points in θ and z parameters.

	$h = 0.064$		$h = 0.008$		Adaptive
DOFs	45,609	(0.1×)	2,862,171	(6.5×)	440,979
Solver Init. Time (s)	2.3×10^{-2}	(0.08×)	3.3	(11.7×)	2.8×10^{-1}
Solve Time (s)	4.2	(0.06×)	1301	(17.2×)	75.5
Solver Iterations	229	(0.3×)	1643	(1.5×)	1094
Max Error	1.6	(24.6×)	9.2×10^{-3}	(0.14×)	6.8×10^{-2}
Average Error	7.4×10^{-1}	(178×)	1.7×10^{-3}	(0.4×)	4.1×10^{-3}

(see Table 3.3). Figure 3.9 shows that the adaptive computational tube can produce a solution that is visually indistinguishable from the uniform solution with $h = 0.008$ but with 6.5× less DOFs and in only 76 seconds (around 17× faster). The max error is about 7× larger for our adaptive solution but only 2.5× larger on average (see Table 3.3). Figure 3.10 (bottom row) also shows that the error is concentrated near the boundary of $\mathcal{S}$, suggesting that it would be preferable to consider second-order BCs (which basically only adds an extra closest-point evaluation for $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$).

Curvature and Solution Gradients

A final example explores adaptivity with a more general $\mathcal{S}$ that has localized subsets of high curvature relative to the rest of $\mathcal{S}$. Curvature is related to LFS since for regions of $\mathcal{S}$ where there are no bottlenecks (regions of $\mathcal{S}$ that are distant in geodesic distance but close in Euclidean distance), the LFS will be equal to the radius of curvature. Furthermore, this example exhibits both high manifold curvature and relatively large solution gradients occurring within the same regions of $\mathcal{S}$.

The screened-Poisson equation (3.21) is solved on the doughnut mesh shown in Figure 3.11. To illustrate the ability of our adaptivity framework to improve efficiency when there are both large solution gradients and high curvatures, we compute $f_{\mathcal{S}}$ using a manually constructed $u_{\mathcal{S}}$ exhibiting these properties. The solution $u_{\mathcal{S}}$ is constructed as follows. The principal curvatures κ_1 and κ_2 of the doughnut are computed using `geometry-central` [154]. We threshold $\max(|\kappa_1|, |\kappa_2|)$ to determine the vertices of the CPGP letters on the doughnut. Then $u_{\mathcal{S}} = \sin(C\theta)$ is taken, where $\theta \in [0, 2\pi]$ is the angle in the xz-plane (y is the up direction) and $C = 3$ everywhere except for the CPGP letters where $C = 30$.

The screened-Poisson equation is only well-defined if the solution is at least C^2-smooth. However, there is a discontinuity in our constructed $u_{\mathcal{S}}$ around the borders of the CPGP letters due to the change of C. We smooth $u_{\mathcal{S}}$ by CP extending it onto a computational tube $\Omega(\mathcal{S})$ with $h = 0.01$ to give u and then set u at each grid point $\mathbf{x}_i \in \Omega(\mathcal{S})$ to the average value of the six neighbouring grid points of $\mathbf{x}_i$. The CP extension of $u_{\mathcal{S}}$ here is an example where an interpolation scheme specific to the discrete representation of the manifold must be used (as was discussed at the start of this chapter for initial conditions, boundary conditions, and known functions given in a problem definition). The solution $u_{\mathcal{S}}$ is only given at the vertices of the triangle mesh, but we need to know $u_{\mathcal{S}}$ at $\text{cp}(\mathbf{x}_i)$ for all grid points $\mathbf{x}_i \in \Omega(\mathcal{S})$. Linear interpolation using the barycentric coordinates of triangles in the mesh is used to CP extend $u_{\mathcal{S}}$ onto $\Omega(\mathcal{S})$. Cubic interpolation of u in $\Omega(\mathcal{S})$ is used to obtain the smoothed version of $u_{\mathcal{S}}$ back at the vertices of the triangulation.

Due to the discrete construction scheme of $u_{\mathcal{S}}$ discussed above, there is no exact equation for $u_{\mathcal{S}}$. Furthermore, a general triangle mesh does not provide an exact equation for $\Delta_{\mathcal{S}} u_{\mathcal{S}}$ either, so we cannot compute an exact $f_{\mathcal{S}}$ from $u_{\mathcal{S}}$. We can however approximate $f_{\mathcal{S}}$ using CPM's discrete operators. We once again CP extend $u_{\mathcal{S}}$ (now smoothed) onto an $\Omega(\mathcal{S})$ but with $h = 0.004$ to give u. Then, CPM's discrete operators approximating the left-hand-side of (3.21) are applied to u giving f. We finally obtain $f_{\mathcal{S}}$ at vertices of the mesh using cubic interpolation of f on $\Omega(\mathcal{S})$ so it can be used with any resolution $\Omega(\mathcal{S})$.

Figure 3.11 (left column) shows the solution to (3.21) using uniform $\Omega(\mathcal{S})$ with $h = 0.01$ and $h = 0.005$, as well as, an adaptive $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2)$. The subset $\Omega(\mathcal{S}_1)$ away from the CPGP letters uses $h = 0.01$ (black in Figure 3.11 bottom right) and $h = 0.005$ is used for $\Omega(\mathcal{S}_2)$ (red in Figure 3.11 bottom right). We consider $u|_{h=0.005}$ computed on the uniform $\Omega(\mathcal{S})$ with $h = 0.005$ to be a reference solution. The difference between $u|_{h=0.005}$ and $u|_{h=0.01}$ and the solution on the adaptive tube u_{adaptive} shows that $u|_{h=0.01}$ has a larger error around the CPGP letters than u_{adaptive} (see Figure 3.11 right column).

The max and average error in $u|_{h=0.01}$ is about $3\times$ and $2\times$ larger than u_{adaptive}, respectively (see Table 3.4). The runtime of the solver for the adaptive approach is about $11\times$ faster than the uniform approach with $h = 0.005$ and is visually similar, unlike $u|_{h=0.01}$ where visual differences are apparent. The additional runtime of the solver when using the adaptive tube $\Omega(\mathcal{S}_1) \cup \Omega(\mathcal{S}_2)$ to compute a satisfactory solution is only about 8% more than the uniform $\Omega(\mathcal{S})$ with $h = 0.01$, which makes the adaptive approach the obvious choice for this scenario.

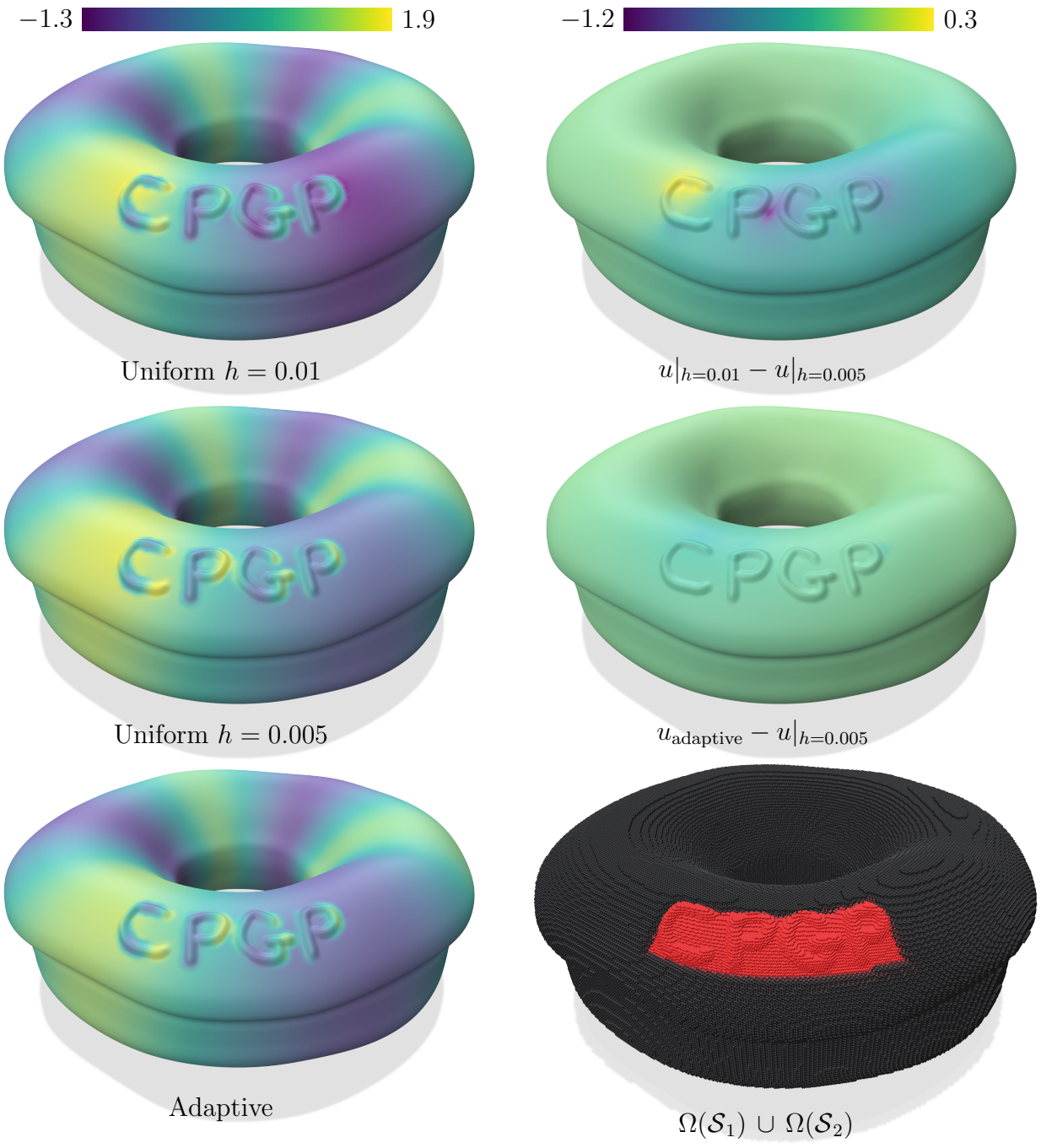


Figure 3.11: Left: Solutions using uniform $\Omega(\mathcal{S})$ with different h and an adaptive tube. Right: Solution differences (top and middle) and the adaptive tube (bottom).

Table 3.4: Attribute comparison when solving the screened-Poisson equation on the doughnut with uniform computational tubes versus adaptive. The error is computed at the 983,040 vertices of the mesh.

	$h = 0.01$		$h = 0.005$		Adaptive
DOFs	737,364	(0.9×)	2,952,748	(3.7×)	804,850
Solver Init. Time (s)	0.7	(3.2×)	11.9	(51.5×)	0.2
Solve Time (s)	137	(0.9×)	1660	(11.1×)	149
Solver Iterations	1152	(1.01×)	3453	(3.04×)	1135
Max Error	1.2	(2.9×)	N/A		0.4
Average Error	0.13	(1.7×)	N/A		0.077

3.5 Summary and Future Work

Important discrete implementation details of the grid-based CPM were explored in this chapter to create an implementation that is memory and runtime efficient. Scaling to problems that require a large number of DOFs (and therefore memory) was accomplished using a BFS construction of the computational tube with a sparse grid data structure. Our custom BiCGSTAB solver improved both memory and runtime efficiency by up to 2× and 41× compared to Eigen's BiCGSTAB implementation, respectively. Spatial adaptivity allowed our linear system to have fewer DOFs, concentrating DOFs only in regions of $\mathcal{S}$ where they are needed, which improved runtime by up to 17×.

We focused on developing a runtime and memory-efficient implementation that allows the treatment of higher-detail manifolds without specialized hardware. However, further improvements could be achieved by incorporating our approaches for the computational tube construction, linear system solver, and spatial adaptivity on specialized hardware such as a GPU or distributed memory machine. NVIDIA Warp [91] would be a good candidate for GPU computing since sparse grid support is already available with NanoVDB [112]. Our spatial adaptivity framework should be easy to incorporate into the domain decomposition method of May et al. [98] since it uses overlapping computational tubes, enabling distributed memory parallelism.

In practice, we have found that the convergence order of our spatial adaptivity framework degrades if there are large solution gradients where the subsets $\Omega(\mathcal{S}_m)$ overlap. More perplexing is that when two subsets $\Omega(\mathcal{S}_1)$ and $\Omega(\mathcal{S}_2)$ overlap but have the same resolution $h_1 = h_2$, the convergence order does not degrade. Understanding the cause of this loss of accuracy when $h_1 \neq h_2$ is interesting future work.

The fact that we use overlapping subsets $\Omega(\mathcal{S}_m)$ also means that extra DOFs are introduced. A nonoverlapping adaptivity framework might be better for computational efficiency. The use of overlapping subsets in our framework benefits from the reuse of code for CPM on uniform $\Omega(\mathcal{S})$ since the same interpolation and finite-difference stencils are used. A nonoverlapping adaptivity framework would require interpolation (with at least quadratic polynomials) and finite-differencing stencils that can handle resolution changes at $\partial\mathcal{S}_m$, such as those used by Min and Gibou [103] or scattered data interpolation [74] and radial-basis-function finite-differences [128].

Chapter 4

Interior Boundary Conditions

Many geometry processing techniques require the solution to PDEs on manifolds embedded in $\mathbb{R}^2$ or $\mathbb{R}^3$, such as curves or surfaces. When the manifold $\mathcal{S}$ is open (i.e., its geometric boundary $\partial\mathcal{S} \neq \emptyset$) some choice of boundary condition (BC) must usually be imposed on $\partial\mathcal{S}$ (e.g., Dirichlet, Neumann, etc.). We will refer to these as *exterior* boundary conditions. In many applications, however, similar types of boundary conditions may be needed at locations on the *interior* of $\mathcal{S}$, irrespective of $\mathcal{S}$ being open or closed. In this case, *interior boundary conditions* (IBCs) should be enforced on a subset $\mathcal{C} \subset \mathcal{S}$, which typically consists of points $\mathcal{C}$ on a 1D curve $\mathcal{S}$, or points and/or curves $\mathcal{C}$ on a 2D surface $\mathcal{S}$.

Input manifolds can take many forms (e.g., triangle meshes, parametrizations, point clouds, implicit functions, etc.). Typically, one must generate a mesh to apply finite element-type techniques or derive specialized discretization procedures for each distinct manifold representation. Existing numerical methods for manifold PDEs support IBCs in various ways depending on the chosen manifold representation and method of discretization.

In the Dirichlet case, the nearest degrees of freedom (DOFs) to the interior boundary can often simply be assigned the desired Dirichlet value. For example, on a point cloud representation, the nearest interior points in the cloud could be set to the Dirichlet value, similar to how *exterior* Dirichlet BCs have been handled in point clouds [80]. With triangle mesh-based discretizations (finite element, discrete exterior calculus, etc.) one can similarly enforce the Dirichlet condition at the nearest surface vertices to the interior boundaries. However, enforcing the IBC at the nearest DOF is inaccurate if the DOF does not lie exactly on the interior boundary $\mathcal{C}$ (i.e., the mesh does not precisely conform to $\mathcal{C}$). Specifically, an error of $O(\|\mathbf{h}\|)$ is introduced where $\|\mathbf{h}\|$ is the distance between the nearest DOF

and $\mathcal{C}$. Moreover, only Dirichlet conditions can be treated in this manner; depending on the chosen manifold representation and/or discretization, it can be nontrivial to enforce Neumann boundary conditions.

With a surface triangulation, a more accurate approach is to remesh the surface with constrained Delaunay refinement (possibly with an intrinsic triangulation) so that vertices or edges of the mesh conform to $\mathcal{C}$, as discussed for example by Sharp and Crane [152]. However, this necessarily introduces remeshing as an extra preprocess. Another mesh-based approach, which avoids remeshing, is the extended finite element method [106, 66], which uses modified basis functions to enforce non-conforming boundaries or discontinuities.

We propose instead to address such problems in a unified manner through a novel extension of CPM to handle IBCs. To enable support for IBCs we derive a method that implicitly partitions the embedding space across interior boundaries. We then adapt CPM's finite difference and interpolation stencils to respect this partition while preserving second-order accuracy.

Most similar to our approach is the method of Shi et al. [159] who enforced Dirichlet IBCs for a manifold PDE method based on level sets. As with CPM, solving surface PDEs with level sets [18] involves extending the problem to the surrounding embedding space. For such embedding methods, it is crucial not only to account for the interior boundary itself but also its influence into the associated embedding space. To do so, the approach of Shi et al. [159] explicitly constructs a triangulation to represent a normal manifold $\mathcal{S}_\perp$ (see (4.2)) extending outwards from the interior boundary curve $\mathcal{C}$ (notably contrasting with the implicit nature of level sets). They then perform geometric tests to determine if stencils intersect $\mathcal{S}_\perp$ and modify the discretization locally. We instead introduce a simple triangulation-free approach to determine if stencils cross $\mathcal{S}_\perp$ that only involves closest points, bypassing explicit construction of $\mathcal{S}_\perp$. Moreover, such level-set approaches necessarily require a well-defined inside and outside, which makes handling open manifolds, nonorientable manifolds, and manifolds of codimension-two or higher impossible with a single level set.

For Dirichlet IBCs in CPM, Auer et al. [7, 6] fixed all the nearest DOFs in the embedding space within a ball centred around $\mathcal{C}$ (considering only the case when $\mathcal{C}$ is a point). This again is only first-order accurate, incurring an $O(h)$ error, where h is the grid spacing of $\Omega(\mathcal{S})$. Enforcing the IBC over a ball effectively inflates the boundary region to a wider area of the surface. That is, a circular region of the surface around the point $\mathcal{C}$ will be fixed with the prescribed condition. We show in Section 4.3 that this approach can also be applied to boundary curves, but the observed error is much larger compared to our proposed method. Moreover, it cannot be applied when Dirichlet values differ on each side

of $\mathcal{C}$.

Our proposed CPM extension overcomes several limitations of the existing CPM (Dirichlet-only) IBC treatment of Auer et al. [2012]. We demonstrate that our method can easily be extended to second-order, for both Dirichlet and zero-Neumann cases. It can also handle jump discontinuities in Dirichlet values across interior boundary curves. Furthermore, our approach supports what we call *mixed* boundary conditions, e.g., Dirichlet on one side and Neumann on the other. Both jump discontinuities and mixed IBCs are useful for various applications, such as diffusion curves [122].

The key attribute of our IBC approach that allows the above flexibility for BC types is the introduction of new DOFs near $\mathcal{C}$. This idea shares conceptual similarities with virtual node algorithms [107], which have been used for codimension-zero problems [15, 56, 11]. It is also similar to the CPM work of Cheung et al. [25], who used new DOFs near sharp features of $\mathcal{S}$ (albeit with the radial-basis function discretization of CPM).

We demonstrate our method's convergence behaviour on selected model PDEs and explore several geometry processing problems: diffusion curves on surfaces, geodesic distance, tangent vector field design, harmonic map construction, and reaction-diffusion textures. Our proposed approach thus offers a powerful and flexible new tool for a range of geometry processing tasks on general manifold representations.

4.1 Exterior Boundary Conditions for Open Manifolds

Our proposed approach for IBCs in Section 4.2 builds on existing CPM techniques for applying exterior BCs at open manifold boundaries, which we review here. A subset $\Omega(\partial\mathcal{S}) \subset \Omega(\mathcal{S})$ of grid points called the boundary subset is used to enforce exterior BCs. It consists of all $\mathbf{x}_i$ satisfying $\text{cp}_{\mathcal{S}}(\mathbf{x}_i) \in \partial\mathcal{S}$, i.e., grid points whose closest manifold point is on the boundary of $\mathcal{S}$. Equivalently,

$$\Omega(\partial\mathcal{S}) = \big\{\mathbf{x}_i \in \Omega(\mathcal{S}) \;\big|\; \text{cp}_{\mathcal{S}}(\mathbf{x}_i) = \text{cp}_{\partial\mathcal{S}}(\mathbf{x}_i)\big\}, \tag{4.1}$$

where $\text{cp}_{\partial\mathcal{S}}$ is the closest point function to $\partial\mathcal{S}$. Geometrically, $\Omega(\partial\mathcal{S})$ is a half-tubular region of grid points past $\partial\mathcal{S}$, halved by the manifold orthogonal to $\mathcal{S}$ at $\partial\mathcal{S}$ defined by

$$\mathcal{S}_\perp = \{\mathbf{x} \in \mathcal{N}(\mathcal{S}) \mid \mathbf{x} = \mathbf{y} + t\,\mathbf{n}_{\mathcal{S}}(\mathbf{y}),\ \mathbf{y} \in \partial\mathcal{S},\ |t| \le r_{\Omega(\mathcal{S})}\}, \tag{4.2}$$

when $\mathcal{S}$ is codimension one. The manifold normal at $\mathbf{y} \in \partial\mathcal{S}$ is defined as the limiting normal $\mathbf{n}_{\mathcal{S}}(\mathbf{y}) = \lim_{\mathbf{z}\to\mathbf{y}} \mathbf{n}_{\mathcal{S}}(\mathbf{z})$, where $\mathbf{z} \in \mathcal{S}$ and $\mathbf{n}_{\mathcal{S}}(\mathbf{z})$ is the unit normal of $\mathcal{S}$ at $\mathbf{z}$.

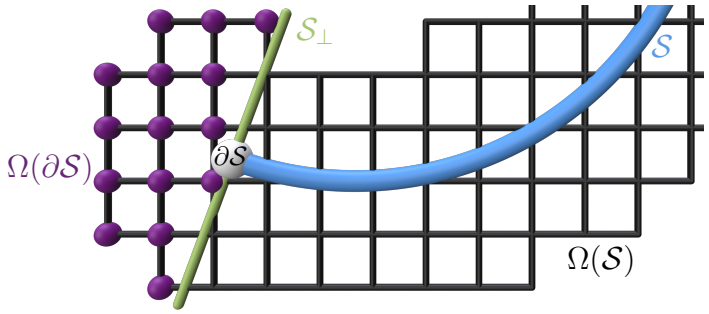


Figure 4.1: The boundary subset $\Omega(\partial\mathcal{S})$ (purple points) for a curve $\mathcal{S}$ (blue) comprises those grid points in $\Omega(\mathcal{S})$ (black grid) whose closest point is on the boundary $\partial\mathcal{S}$ (white point). The points $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$ are those past the normal manifold $\mathcal{S}_\perp$ (green) based at $\partial\mathcal{S}$.

Figure 4.1 illustrates the boundary subset $\Omega(\partial\mathcal{S})$ and $\mathcal{S}_\perp$ for a 1D curve embedded in $\mathbb{R}^2$. Note that (4.2) can easily be generalized for higher-codimensional manifolds, e.g., the normal line $t\,\mathbf{n}_\mathcal{S}(\mathbf{y})$ is replaced by a normal plane for 1D curves embedded in 3D.

The natural boundary conditions of CPM have first-order accuracy and are homogeneous Neumann BCs, $\nabla_\mathcal{S} u_\mathcal{S} \cdot \mathbf{n}_{\partial\mathcal{S}} = 0$, where $\mathbf{n}_{\partial\mathcal{S}}$ is the unit conormal of $\partial\mathcal{S}$. The conormal is a vector normal to $\partial\mathcal{S}$, tangential to $\mathcal{S}$, and oriented outward [44]. Therefore, $\mathbf{n}_{\partial\mathcal{S}}(\mathbf{y}) \neq \mathbf{n}_\mathcal{S}(\mathbf{y})$ for $\mathbf{y} \in \partial\mathcal{S}$, and $\mathbf{n}_{\partial\mathcal{S}}(\mathbf{y})$ is orthogonal to $\mathbf{n}_\mathcal{S}(\mathbf{y})$ since $\mathbf{n}_{\partial\mathcal{S}}(\mathbf{y})$ is in the tangent space of $\mathcal{S}$. The CP extension propagates manifold data constant in both $\mathbf{n}_\mathcal{S}$ and $\mathbf{n}_{\partial\mathcal{S}}$ at $\partial\mathcal{S}$. Hence, finite differencing across the boundary subset $\Omega(\partial\mathcal{S})$ will measure zero conormal derivatives [138] and the discretization of the manifold differential operator can be used without any changes at $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$.

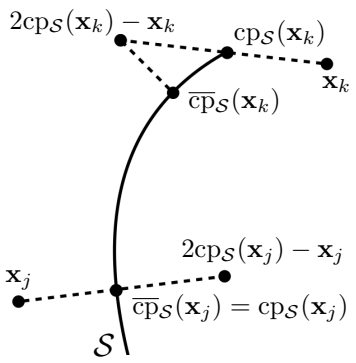
However, to enforce first-order Dirichlet BCs on $\partial\mathcal{S}$, the CP extension step must be changed. The prescribed Dirichlet value at the closest point of $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$ is extended to $\mathbf{x}_i$ (instead of the interpolated value in (3.1)). That is, the CP extension assigns $u_i = u_\mathcal{S}(\mathrm{cp}_{\partial\mathcal{S}}(\mathbf{x}_i))$ for all $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$, where $u_\mathcal{S}(\mathrm{cp}_{\partial\mathcal{S}}(\mathbf{x}_i))$ is the Dirichlet value at $\mathrm{cp}_{\partial\mathcal{S}}(\mathbf{x}_i)$. Only this extension procedure changes; the FD discretization is unchanged for all exterior BC types and orders.

For improved accuracy, second-order Dirichlet and zero-Neumann exterior BCs were introduced by Macdonald et al. [89] using a simple modification to the closest point function. The closest point function is replaced with

$$\overline{\mathrm{cp}}_\mathcal{S}(\mathbf{x}) = \mathrm{cp}_\mathcal{S}(2\mathrm{cp}_\mathcal{S}(\mathbf{x}) - \mathbf{x}). \tag{4.3}$$

Effectively, rather than finding the closest point, this expression determines a "reflected" point, and returns *its* closest point instead.

Observe that $\overline{\text{cp}}_{\mathcal{S}}$ satisfies $\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_j) = \text{cp}_{\mathcal{S}}(\mathbf{x}_j)$ if $\mathbf{x}_j \notin \Omega(\partial\mathcal{S})$ (and $\text{cp}_{\mathcal{S}}(\mathbf{x})$ is unique). Therefore, no change occurs to CPM on the interior of $\mathcal{S}$ (see inset, bottom), so we continue to use $\text{cp}_{\mathcal{S}}(\mathbf{x})$ for $\mathbf{x} \in \Omega(\mathcal{S})\backslash\Omega(\partial\mathcal{S})$. However, for boundary points $\mathbf{x}_k \in \Omega(\partial\mathcal{S})$, we have $\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_k) \neq \text{cp}_{\mathcal{S}}(\mathbf{x}_k)$, since $\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_k)$ is a point on the interior of $\mathcal{S}$ while $\text{cp}_{\mathcal{S}}(\mathbf{x}_k)$ is a point on $\partial\mathcal{S}$ (see inset, top). Hence, for a flat manifold, $u_{\mathcal{S}}(\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_k))$ gives the interior mirror value for $\mathbf{x}_k$. For a general, curved manifold $u_{\mathcal{S}}(\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_k))$ gives an approximate mirror value.



Thus, replacing $\text{cp}_{\mathcal{S}}$ with $\overline{\text{cp}}_{\mathcal{S}}$ will naturally apply second-order homogeneous Neumann exterior BCs: approximate mirror values are extended to $\mathbf{x}_k \in \Omega(\partial\mathcal{S})$, so the effective conormal derivative becomes zero at $\partial\mathcal{S}$. This approach generalizes popular methods for codimension-zero problems with embedded boundaries, where mirror values are also assigned to ghost points (see e.g., [79, Section 2.12]). In practice, the only change required is to replace $\mathcal{I}_k$ and corresponding weights in (3.1) with those for $\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_k)$.

Second-order Dirichlet exterior BCs similarly generalize their codimension-zero counterparts, e.g., the ghost fluid method [50] that fills ghost point values by linear extrapolation. The CP extension at $\mathbf{x}_k \in \Omega(\partial\mathcal{S})$ becomes $u(\mathbf{x}_k) = 2u_{\mathcal{S}}(\text{cp}_{\partial\mathcal{S}}(\mathbf{x}_k)) - u(\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}))$, where $u_{\mathcal{S}}(\text{cp}_{\partial\mathcal{S}}(\mathbf{x}_k))$ is the prescribed Dirichlet value on $\partial\mathcal{S}$. Hence, for $\mathbf{x}_k \in \Omega(\partial\mathcal{S})$ we change (3.1) to

$$u_k = 2u_{\mathcal{S}}(\text{cp}_{\partial\mathcal{S}}(\mathbf{x}_k)) - \sum_{j \in \overline{\mathcal{I}}_k} \overline{w}_j^k u_j, \tag{4.4}$$

where $\overline{\mathcal{I}}_k$ and $\overline{w}_j^k$ are the interpolation stencil indices and weights for $\overline{\text{cp}}_{\mathcal{S}}(\mathbf{x}_k)$, respectively.

Remark that $\mathcal{S}$ can have multiple boundaries, so there may be multiple $\Omega(\partial\mathcal{S})$ regions where this BC treatment must be applied.

4.2 Interior Boundary Conditions

As discussed in Chapter 3, the discrete setting of CPM involves two main operations: interpolation for CP extensions and finite differences (FDs) for differential operators. Exterior BCs are handled by modifying the CP extension interpolation while keeping the

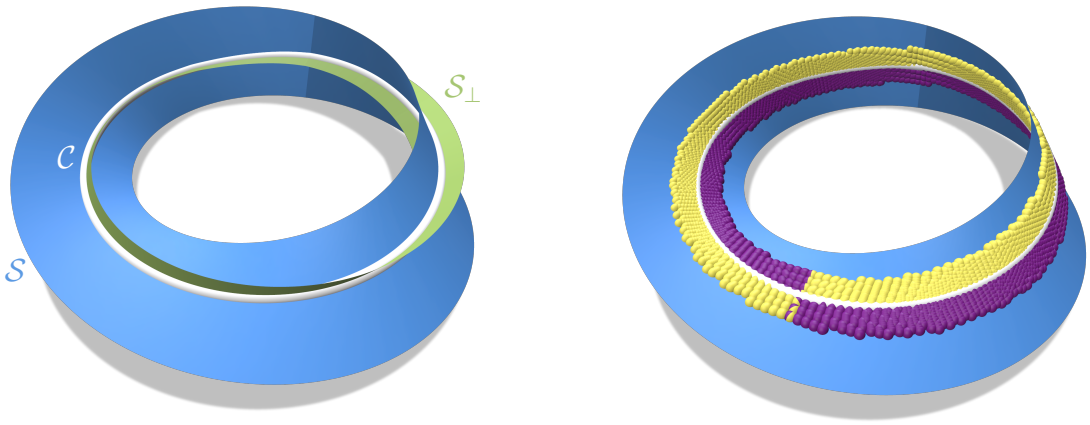


Figure 4.2: On the left, a normal manifold $\mathcal{S}_\perp$ (green) extends perpendicularly outwards from a curve $\mathcal{C}$ (white) where an IBC is to be applied. On the right, closest points $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ for $\mathbf{x}_i \in \Omega(\mathcal{C})$ (yellow and purple) cannot be globally partitioned into two disjoint sets by $\mathcal{C}$ on a nonorientable $\mathcal{S}$ (blue).

finite differencing the same. Below we describe our proposed technique to extend CPM with support for interior BCs, which consists of two key changes: adding new degrees of freedom (DOFs) and carefully altering both the interpolation and FD stencils.

For the rest of this chapter we focus on the cases where the manifold $\mathcal{S}$ is a curve embedded in $\mathbb{R}^2$ or a surface embedded in $\mathbb{R}^3$. Let $\mathcal{C} \subset \mathcal{S}$ denote the interior region where the BC is to be applied, which can be a point (in 2D or 3D) or an open or closed curve (in 3D). Since CPM is an embedding method we must consider the influence of $\mathcal{C}$ on the embedding space $\mathcal{N}(\mathcal{S})$. Let $\mathcal{S}_\perp$ denote a (conceptual) manifold orthogonal to $\mathcal{S}$ along $\mathcal{C}$, i.e., analogous to $\mathcal{S}_\perp$ defined in (4.2) for the exterior boundary case, but with $\partial\mathcal{S}$ replaced by $\mathcal{C}$. See Figure 4.2 (left) for an example curve $\mathcal{C}$ on a surface $\mathcal{S}$ and its normal manifold $\mathcal{S}_\perp$ at $\mathcal{C}$.

4.2.1 Adding Interior Boundary DOFs

Exterior BCs incorporate the BC using grid points $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$ as defined in (4.1). These grid points $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$ are only needed to enforce the exterior BC since they lie on the opposite side of $\mathcal{S}_\perp$ from $\mathcal{S}$. Therefore, CP extension stencils for $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$ can be safely modified to enforce exterior BCs.

For interior BCs, the situation is more challenging. Similar to $\Omega(\partial\mathcal{S})$, a new *interior*

boundary subset $\Omega(\mathcal{C}) \subset \Omega(\mathcal{S})$ is defined as

$$\Omega(\mathcal{C}) = \{\mathbf{x}_i \in \Omega(\mathcal{S}) \mid \|\mathbf{x}_i - \text{cp}_{\mathcal{C}}(\mathbf{x}_i)\| \leq r_{\Omega(\mathcal{S})}\}, \tag{4.5}$$

where $\text{cp}_{\mathcal{C}}$ is the closest point function of $\mathcal{C}$. Comparing with (4.1), the subsets $\Omega(\partial\mathcal{S})$ and $\Omega(\mathcal{C})$ are defined in the same way, except $\Omega(\partial\mathcal{S})$ has the extra property $\text{cp}_{\mathcal{S}}(\mathbf{x}_i) = \text{cp}_{\partial\mathcal{S}}(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \Omega(\partial\mathcal{S})$; i.e., points in the exterior boundary subset have a closest manifold point that is *also* their closest boundary point. Grid points in the interior boundary subset do not: $\mathbf{x}_i \in \Omega(\mathcal{C})$ will in general have $\text{cp}_{\mathcal{S}}(\mathbf{x}_i) \neq \text{cp}_{\mathcal{C}}(\mathbf{x}_i)$ unless the point $\mathbf{x}_i \in \mathcal{S}_\perp$.

Ideally, we would use the grid points $\mathbf{x}_i \in \Omega(\mathcal{C})$ to enforce the IBC, analogous to the exterior case. However, the tubular volume surrounding $\mathcal{C}$, $\{\mathbf{x} \in \mathcal{N}(\mathcal{S}) \mid \|\mathbf{x} - \text{cp}_{\mathcal{C}}(\mathbf{x})\| \leq r_{\Omega(\mathcal{S})}\}$, which contains $\Omega(\mathcal{C})$, also intersects with $\mathcal{S}$. Therefore, we cannot simply repurpose and modify CP extension stencils for $\mathbf{x}_i \in \Omega(\mathcal{C})$, since they are needed to solve the manifold PDE on $\mathcal{S} \setminus \mathcal{C}$.

We propose to add a second set of spatially colocated DOFs, called the *BC DOFs*, at all $\mathbf{x}_i \in \Omega(\mathcal{C})$. The BC DOFs allow us to apply similar techniques for interior BCs as was done for exterior BCs. Specifically, given a computational domain $\Omega(\mathcal{S})$ of $N_{\mathcal{S}}$ grid points and the subset $\Omega(\mathcal{C})$ of $N_{\mathcal{C}}$ grid points, the discrete linear system to be solved will now involve $N_{\mathcal{S}} + N_{\mathcal{C}}$ DOFs. We order the BC DOFs after the original *PDE DOFs*. That is, indices in the set $J_{\mathcal{S}} = \{j \in \mathbb{N} \mid 0 \leq j < N_{\mathcal{S}}\}$ give $\mathbf{x}_j \in \Omega(\mathcal{S})$ while indices in the set $J_{\mathcal{C}} = \{\alpha \in \mathbb{N} \mid N_{\mathcal{S}} \leq \alpha < N_{\mathcal{S}} + N_{\mathcal{C}}\}$ give $\mathbf{x}_\alpha \in \Omega(\mathcal{C})$. Throughout we use Greek letters to denote indices in $J_{\mathcal{C}}$ to clearly distinguish from indices in $J_{\mathcal{S}}$. Note that for every BC DOF $\alpha \in J_{\mathcal{C}}$ there is a corresponding PDE DOF $j \in J_{\mathcal{S}}$ such that $\mathbf{x}_\alpha = \mathbf{x}_j$. The key question then becomes: when do we use PDE DOFs versus BC DOFs?

Intuitively, the answer is simple: interpolation and FD stencils ($\mathcal{I}_i$ and $\mathcal{D}_i$ from (3.1) and (3.2)) must only use manifold data $u_{\mathcal{S}}$ from the same side of $\mathcal{S}_\perp$ that the stencil belongs to. Therefore, if a stencil involves manifold data on the opposite side of $\mathcal{S}_\perp$, the IBC must be applied using the BC DOFs.

Figure 4.3 gives a conceptual illustration of the process for a point $\mathcal{C}$ on a circle $\mathcal{S}$ embedded in $\mathbb{R}^2$. Both BC DOFs and PDE DOFs are present in the region of $\Omega(C)$. The BC DOFs are partitioned into one of two sets depending on which side of $\mathcal{S}_\perp$ the closest point $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ is on. The original grid $\Omega(\mathcal{S})$ and duplicated portion $\Omega(\mathcal{C})$ are cut, and each half of $\Omega(\mathcal{C})$ is joined to the opposing side of $\Omega(\mathcal{S})$.

The same treatment of BCs as in the exterior case is then applied on this nonmanifold grid $\Omega(\mathcal{S}) \cup \Omega(\mathcal{C})$. That is, the required modifications to the CP extension interpolation stencils in Section 4.1 are applied. Unlike the exterior BC case, however, changes to FD

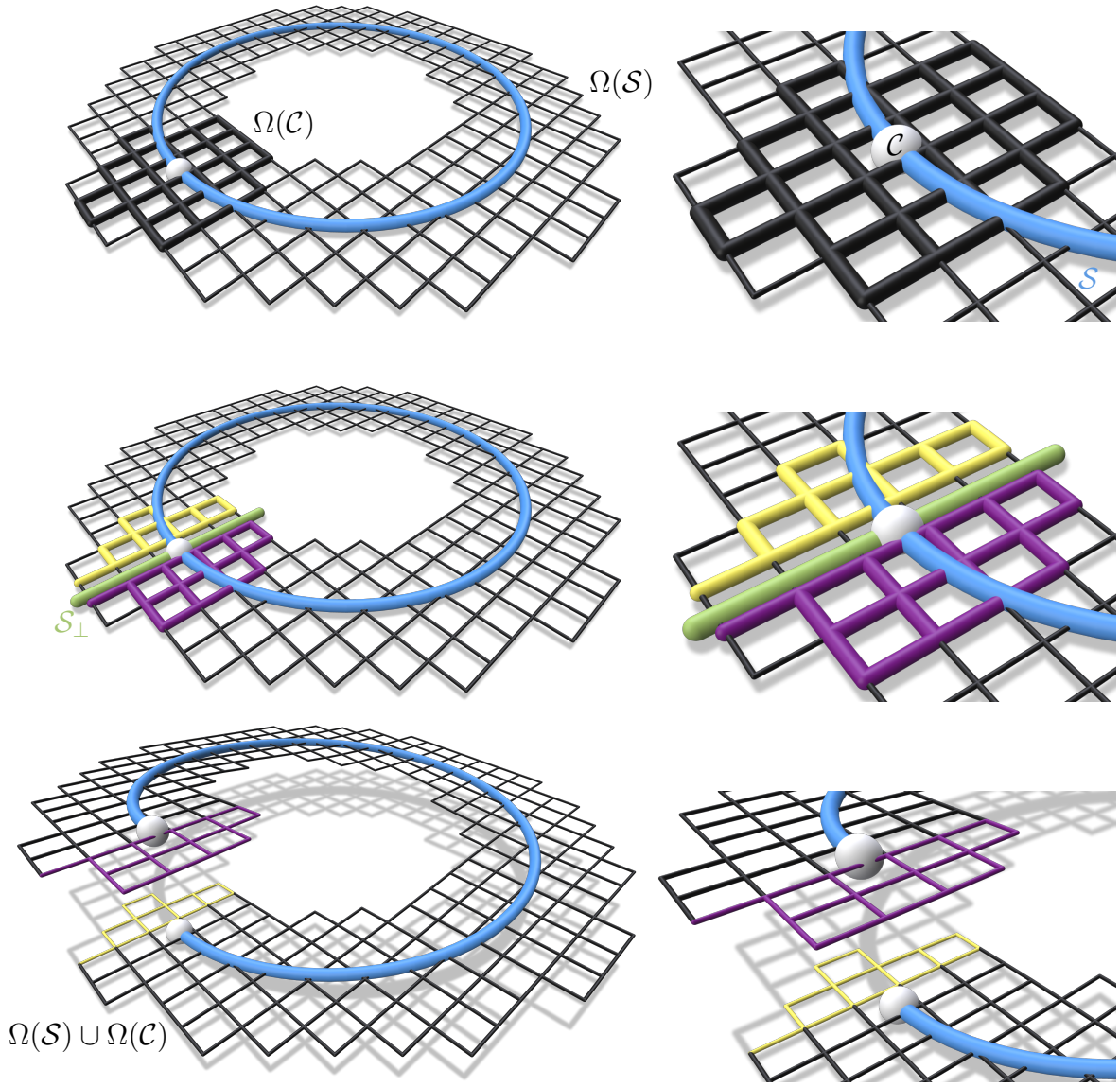


Figure 4.3: A conceptual illustration of our approach to interior boundaries for a point $\mathcal{C}$ (white) on a curve $\mathcal{S}$ (blue) in $\mathbb{R}^2$. Top row: Duplicated BC DOFs are generated in the boundary subset $\Omega(\mathcal{C})$ around $\mathcal{C}$ (thick black grid). Middle row: The normal manifold $\mathcal{S}_\perp$ (green) locally partitions the grid into two sides (yellow, purple). Bottom row: The modified grid connectivity is illustrated by warping it into $\mathbb{R}^3$.

stencils do occur for IBCs since $\Omega(\mathcal{C})$ and $\Omega(\mathcal{S})$ are cut and joined to opposite sides of each other.

If $\mathcal{S}$ is *orientable* then this intuitive picture in Figure 4.3 is an accurate depiction of the necessary grid connectivity. That is, near $\mathcal{C}$ we must duplicate DOFs and cut and join opposite pieces of $\Omega(\mathcal{S})$ and $\Omega(\mathcal{C})$ to produce regions (similar to $\Omega(\partial\mathcal{S})$) where BCs can be imposed. However, if $\mathcal{S}$ is nonorientable the closest points $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ for $\mathbf{x}_i \in \Omega(\mathcal{C})$ cannot be globally partitioned into two sides. For example, on the Möbius strip in Figure 4.2 (right), an apparent flip in the partitioning of $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ is unavoidable as one moves along a curve $\mathcal{C}$ that loops around the whole strip.

Fortunately, IBCs can still be enforced on nonorientable manifolds because the manifold can be oriented *locally*. The interpolation and FD stencils only perform operations in a small local region of $\Omega(\mathcal{S})$, so locally orienting the manifold is sufficient to enforce IBCs.

4.2.2 $\mathcal{S}_\perp$ Crossing Test

We must keep computation local to each stencil to handle nonorientable manifolds. Therefore, first consider testing if any two closest points of $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{N}(\mathcal{S})$ are on opposite sides of $\mathcal{S}_\perp$. A naive approach would be to construct $\mathcal{S}_\perp$ explicitly, e.g., with a surface triangulation (as was done by Shi et al. [159]), and then test if the line segment between $\text{cp}_{\mathcal{S}}(\mathbf{x}_1)$ and $\text{cp}_{\mathcal{S}}(\mathbf{x}_2)$ intersects the triangulation. However, building an explicit surface is counter to the implicit spirit of CPM.

Determining if $\text{cp}_{\mathcal{S}}(\mathbf{x}_1)$ and $\text{cp}_{\mathcal{S}}(\mathbf{x}_2)$ are on opposite sides of $\mathcal{S}_\perp$ can instead be accomplished based on closest points on $\mathcal{C}$. Let $\text{cp}_{\mathcal{C}}(\mathbf{x}_1)$ and $\text{cp}_{\mathcal{C}}(\mathbf{x}_2)$ be the closest points to $\mathbf{x}_1$ and $\mathbf{x}_2$ on $\mathcal{C}$, respectively. Define the vector $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})$ as

$$\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}) \equiv \text{cp}_{\mathcal{S}}(\mathbf{x}) - \text{cp}_{\mathcal{C}}(\mathbf{x}). \tag{4.6}$$

Denote the locally-oriented unit normal to $\mathcal{S}_\perp$ at $\mathbf{y} \in \mathcal{C}$ as $\mathbf{n}_{\mathcal{S}_\perp}(\mathbf{y})$. The function

$$F(\mathbf{x}) \equiv \text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}) \cdot \mathbf{n}_{\mathcal{S}_\perp}(\text{cp}_{\mathcal{C}}(\mathbf{x})) \tag{4.7}$$

will have different signs for $F(\mathbf{x}_1)$ and $F(\mathbf{x}_2)$ if $\text{cp}_{\mathcal{S}}(\mathbf{x}_1)$ and $\text{cp}_{\mathcal{S}}(\mathbf{x}_2)$ are on different sides of $\mathcal{S}_\perp$, or equivalently $F(\mathbf{x}_1)F(\mathbf{x}_2) < 0$. However, this direct test would require computing $\mathbf{n}_{\mathcal{S}_\perp}$ along $\mathcal{C}$ and locally orienting that normal vector.

Instead of checking the directions $\text{cp}_{\mathcal{S}-\mathcal{C}}$ relative to the locally oriented normals $\mathbf{n}_{\mathcal{S}_\perp}$, we can check the directions of $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_1)$ and $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_2)$ relative to each other. As illustrated

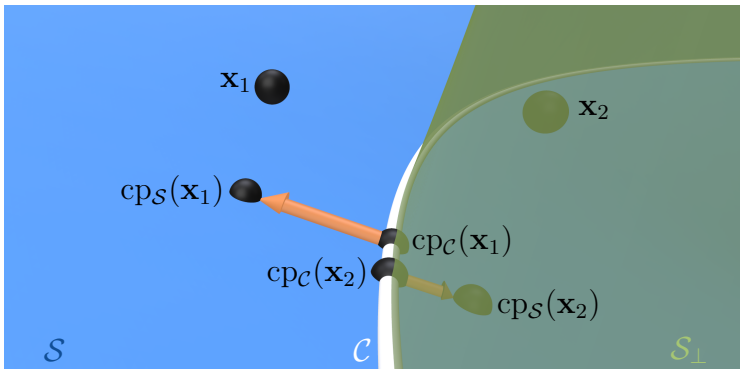


Figure 4.4: For two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{N}(\mathcal{S})$, we can determine if the closest points, $\text{cp}_{\mathcal{S}}(\mathbf{x}_1)$, $\text{cp}_{\mathcal{S}}(\mathbf{x}_2)$, lie on opposite sides of $\mathcal{C}$ based on their orientations relative to the corresponding closest points on $\mathcal{C}$, $\text{cp}_{\mathcal{C}}(\mathbf{x}_1)$, $\text{cp}_{\mathcal{C}}(\mathbf{x}_2)$.

in Figure 4.4, if $\text{cp}_{\mathcal{S}}(\mathbf{x}_1)$ and $\text{cp}_{\mathcal{S}}(\mathbf{x}_2)$ are on opposite sides of $\mathcal{S}_\perp$ the associated $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})$ vectors will point in opposing directions; thus, we can simply check if their dot product is negative:

$$\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_1) \cdot \text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_2) < 0. \tag{4.8}$$

In practice, we find (4.8) sufficient to obtain second-order accuracy in the convergence studies of Section 4.3 on smooth $\mathcal{S}$ and $\mathcal{C}$.

When $\mathbf{x}$ is close to $\mathcal{S}_\perp$ the vector $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}) \approx \mathbf{0}$, which can result in an inaccurate classification of which side $\text{cp}_{\mathcal{S}}(\mathbf{x})$ is on. Therefore, if $\|\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})\| = O(h^2)$ the point $\text{cp}_{\mathcal{S}}(\mathbf{x})$ is considered to lie on $\mathcal{C}$ and can be safely assigned to either side, while maintaining second-order accuracy. In practice, we consider $\text{cp}_{\mathcal{S}}(\mathbf{x})$ to lie on $\mathcal{C}$ if $\|\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})\| < 0.1h^2$.

As we have noted, the locality of this $\mathcal{S}_\perp$ crossing test allows it to handle nonorientable manifolds with CPM and IBCs. However, on orientable manifolds one can still globally orient stencils in $\Omega(\mathcal{C})$ to impose different values or types of IBCs on either side of $\mathcal{C}$. For example, different prescribed Dirichlet values on each side of $\mathcal{C}$ are useful for vector field design. Mixing Dirichlet and Neumann IBCs on $\mathcal{C}$ in this way can also be useful for diffusion curves.

4.2.3 Stencil Modifications

In this section, we describe how to use the $\mathcal{S}_\perp$ crossing test to impose IBCs by altering interpolation and FD stencils. The $\mathcal{S}_\perp$ crossing test (4.8) allows us to determine if any

two points $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{N}(\mathcal{S})$ have closest points $\text{cp}_{\mathcal{S}}(\mathbf{x}_1)$, $\text{cp}_{\mathcal{S}}(\mathbf{x}_2)$ on opposite sides of $\mathcal{S}_\perp$. Ultimately, we employ this test to determine if the closest points $\text{cp}_{\mathcal{S}}(\mathbf{x}_j)$ for $j \in \mathcal{I}_i$ or $\mathcal{D}_i$ are on the opposite side of $\mathcal{S}_\perp$ relative to a stencil for $\mathbf{x}_i$, so the stencil can use the correct PDE vs. BC data.

A stencil is itself assigned to a particular side of $\mathcal{S}_\perp$ based on the location of an associated point on $\mathcal{S}$ that we call the *stencil director*, denoted $\mathbf{y}^\star$. For the FD stencil of $\mathbf{x}_i$ the stencil director is $\mathbf{y}_i^\star = \text{cp}_{\mathcal{S}}(\mathbf{x}_i)$, since grid data at $\mathbf{x}_i$ corresponds to manifold data at $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$. For the interpolation stencil of $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$, the stencil director is the interpolation query point itself, i.e., $\mathbf{y}_i^\star = \text{cp}_{\mathcal{S}}(\mathbf{x}_i)$, which is the same as the FD stencil's director. Each stencil director also has a corresponding *stencil direction* denoted $\mathbf{d}^\star$. For FD and CP extension interpolation stencils $\mathbf{d}_i^\star = \text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_i) = \mathbf{y}_i^\star - \text{cp}_{\mathcal{C}}(\mathbf{x}_i)$.

It is, however, not always the case that $\mathbf{y}_i^\star = \text{cp}_{\mathcal{S}}(\mathbf{x}_i)$. Interpolation of the solution stored on the grid $\Omega(\mathcal{S}) \cup \Omega(\mathcal{C})$ can also be used to obtain the final solution at *any* set of manifold points. For example, if one desires to transfer the solution to a mesh or a point cloud (e.g., for display or downstream processing), interpolation can be used to obtain the solution on vertices of the mesh or points in the cloud. In this case, the stencil director is again the interpolation query point $\mathbf{y}^\star = \mathbf{y}_{\text{query}} \in \mathcal{S}$ and the stencil direction is $\mathbf{d}^\star = \mathbf{y}^\star - \text{cp}_{\mathcal{C}}(\mathbf{y}^\star)$.

PDE DOF Modifications

The first step to incorporate IBCs is to alter the stencils for the PDE DOFs in $J_{\mathcal{S}}$. The computation in both (3.1) and (3.2) for $i \in J_{\mathcal{S}}$ has the form

$$u_i = \sum_{j \in \mathcal{G}_i} c_j^i u_j,$$

where $\mathcal{G}_i \subset J_{\mathcal{S}}$ are indices corresponding to grid points in the stencil for i (i.e., $\mathcal{G}_i = \mathcal{I}_i$ or $\mathcal{G}_i = \mathcal{D}_i$) and c_j^i are corresponding weights.

To incorporate IBCs, the index $j \in \mathcal{G}_i$ is replaced with its corresponding BC DOF index $\alpha \in J_{\mathcal{C}}$ if data at $\mathbf{x}_j$ comes from the opposite side of $\mathcal{S}_\perp$. The corresponding stencil weight c_j^i remains unchanged. Using the $\mathcal{S}_\perp$ crossing test (4.8), for all $j \in \mathcal{G}_i$, we replace $j \in J_{\mathcal{S}}$ with its corresponding $\alpha \in J_{\mathcal{C}}$ if

$$\mathbf{d}_i^\star \cdot \text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_j) < 0. \tag{4.9}$$

If our equations are written in matrix form, these modifications to the PDE DOFs above would change $N_{\mathcal{S}} \times N_{\mathcal{S}}$ matrices to be size $N_{\mathcal{S}} \times (N_{\mathcal{S}} + N_{\mathcal{C}})$. The next step is to add the BC equations for the BC DOFs in $J_{\mathcal{C}}$, resulting in square matrices again of size $(N_{\mathcal{S}} + N_{\mathcal{C}}) \times (N_{\mathcal{S}} + N_{\mathcal{C}})$.

BC DOF Modifications

Finite-difference stencils are added for the BC DOFs with $\alpha \in J_{\mathcal{C}}$ and modified in a similar way to the PDE DOFs above. The same grid connectivity is present in $\Omega(\mathcal{C})$ as the corresponding portion of $\Omega(\mathcal{S})$ (except at the boundary of $\Omega(\mathcal{C})$). Therefore, the same FD stencils on $\Omega(\mathcal{S})$ are used on $\Omega(\mathcal{C})$ except with indices $\beta \in J_{\mathcal{C}}$ (and indices not present in $\Omega(\mathcal{C})$, i.e., grid points in $\Omega(\mathcal{S})$ around the edge of $\Omega(\mathcal{C})$, are removed). Hence, using the $\mathcal{S}_\perp$ crossing test (4.8) for all $\beta \in \mathcal{D}_\alpha$, the index $\beta \in J_{\mathcal{C}}$ is replaced with its corresponding $j \in J_{\mathcal{S}}$ if

$$\mathbf{d}_\alpha^\star \cdot \mathrm{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_\beta) < 0. \tag{4.10}$$

The CP extension BC equations discussed in Section 4.1 for exterior BCs are used on the BC DOFs with $\alpha \in J_{\mathcal{C}}$. However, first-order zero-Neumann IBCs are no longer automatically imposed as in Section 4.1. Instead, for first-order zero-Neumann IBCs, the CP extension extends manifold data $u_{\mathcal{S}}$ at $\mathrm{cp}_{\mathcal{C}}(\mathbf{x}_\alpha)$ for $\mathbf{x}_\alpha \in \Omega(\mathcal{C})$, i.e.,

$$u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{C}}(\mathbf{x}_\alpha)) = u(\mathbf{x}_\alpha) \approx \sum_{\beta \in \mathcal{I}_\alpha} w_\beta^\alpha u_\beta.$$

Once again the $\mathcal{S}_\perp$ crossing test (4.8) is used to ensure DOFs are used from the correct sides of $\mathcal{S}_\perp$. In this case, the stencil director (interpolation query point) is $\mathbf{y}_\alpha^\star = \mathrm{cp}_{\mathcal{C}}(\mathbf{x}_\alpha)$, which gives $\mathbf{d}_\alpha^\star = \mathbf{0}$ since $\mathbf{y}_\alpha^\star$ is on both $\mathcal{C}$ and $\mathcal{S}$. However, the vector $\mathbf{d}_\alpha^\star \equiv \mathrm{cp}_{\mathcal{S}}(\mathbf{x}_\alpha) - \mathrm{cp}_{\mathcal{C}}(\mathbf{x}_\alpha)$ gives the correct direction to define which side of $\mathcal{S}_\perp$ the interpolation stencil belongs to. Then, for all $\beta \in \mathcal{I}_\alpha$, we replace $\beta \in J_{\mathcal{C}}$ with its corresponding $j \in J_{\mathcal{S}}$ if (4.10) holds.

For second-order zero-Neumann IBCs, the only modification required is to replace $\mathrm{cp}_{\mathcal{C}}(\mathbf{x})$ with

$$\overline{\mathrm{cp}}_{\mathcal{C}}(\mathbf{x}) = \mathrm{cp}_{\mathcal{S}}(2\mathrm{cp}_{\mathcal{C}}(\mathbf{x}) - \mathbf{x}). \tag{4.11}$$

Note that (4.11) is different from the form used for exterior BCs in (4.3), as it involves both $\mathrm{cp}_{\mathcal{S}}$ and $\mathrm{cp}_{\mathcal{C}}$. However, the purpose of this modified closest point function (4.11) remains the same, i.e., the point $\overline{\mathrm{cp}}_{\mathcal{C}}(\mathbf{x})$ is an approximate mirror location.

The CP extension equations for BC DOFs, with $\alpha \in J_{\mathcal{C}}$, to enforce Dirichlet IBCs are analogous to Section 4.1. The prescribed Dirichlet value, $u_{\mathcal{S}}$ on $\mathcal{C}$, is extended for first-order Dirichlet IBCs, i.e., $u(\mathbf{x}) = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{C}}(\mathbf{x}))$ or in the discrete setting $u_\alpha = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{C}}(\mathbf{x}_\alpha))$. For second-order Dirichlet IBCs, the extension is $u(\mathbf{x}) = 2u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{C}}(\mathbf{x})) - u(\overline{\mathrm{cp}}_{\mathcal{C}}(\mathbf{x}))$, which becomes analogous to (4.4) in the discrete setting.

4.2.4 Open Curves $\mathcal{C}$ in $\mathbb{R}^3$

Past the endpoints of an open curve $\mathcal{C}$ the PDE should be solved without the IBC being enforced. However, the set $\Omega(\mathcal{C})$ includes half-spherical regions of grid points past the boundary point $\partial\mathcal{C}$. These half-spherical regions are analogous to the exterior boundary subsets $\Omega(\partial\mathcal{S})$ in Section 4.1 and are defined as

$$\Omega(\partial\mathcal{C}) = \{\mathbf{x}_\alpha \in \Omega(\mathcal{C}) \mid \mathrm{cp}_{\mathcal{C}}(\mathbf{x}_\alpha) = \mathrm{cp}_{\partial\mathcal{C}}(\mathbf{x}_\alpha)\}. \tag{4.12}$$

We do not perform the modifications of Section 4.2.3 for points $\mathbf{x}_\alpha \in \Omega(\partial\mathcal{C})$ since this would enforce the IBC where only the PDE should be solved. In other words, the BC DOFs in $\Omega(\partial\mathcal{C})$ are not added to the linear system.

4.2.5 Points $\mathcal{C}$ in $\mathbb{R}^3$

Remarkably, and unlike for open curves, when $\mathcal{C}$ is a point on $\mathcal{S}$ embedded in $\mathbb{R}^3$ no change to the stencil modification procedure in Section 4.2.3 is needed. To understand why, consider two simpler options. First, without *any* boundary treatment whatsoever near $\mathcal{C}$ the PDE is solved but the IBC is ignored. Second, a naive first-order treatment simply sets either the nearest grid point or a ball of grid points around $\mathcal{C}$ to the Dirichlet value; however, at those grid points the PDE is now ignored. Instead, the grid points near $\mathcal{C}$ should be influenced by the IBC at $\mathcal{C}$, while also satisfying the PDE.

Under the procedure of Section 4.2.3, the $\mathrm{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_j)$ and $\mathbf{d}_i^\star$ vectors will point radially outward from the point $\mathcal{C}$ (approximately in the tangent space of $\mathcal{S}$ at $\mathcal{C}$). The $\mathcal{S}_\perp$ crossing test (4.8) becomes a half-space test, where the plane P partitioning the space goes through $\mathcal{C}$ with its normal given by the stencil direction vector, $\mathbf{d}_i^\star$. In the stencil for $\mathbf{y}_i^\star$, points on the same side of P as $\mathbf{y}_i^\star$ are treated as PDE DOFs, while points on the opposite side receive the IBC treatment (either first or second-order as desired). However, the direction of $\mathbf{d}_i^\star$, and hence the half-space, changes for each grid point's stencil (radially around $\mathcal{C}$). The $\mathbf{d}_i^\star$ changes because the location of $\mathbf{y}_i^\star$ changes for each i with $\mathrm{cp}_{\mathcal{C}}(\mathbf{x}_i)$ fixed at $\mathcal{C}$. This spinning of P radially around $\mathcal{C}$ allows the PDE and the IBC to be enforced simultaneously since both PDE and IBC equations are added to the linear system for all points $\mathbf{x}_i \in \Omega(\mathcal{C})$.

Therefore, for a point $\mathcal{C} \in \mathcal{S} \subset \mathbb{R}^3$, our first-order Dirichlet IBC method acts as an improvement of the approach of Auer et al. [7], where only points $\mathbf{x}_j \in \Omega(\mathcal{C})$ on one side of P (which revolves around $\mathcal{C}$) are fixed with the prescribed Dirichlet value. We observe that this reduces the error constant compared to Auer et al. [7] in convergence studies in Section 4.3. Furthermore, our approach in Section 4.2.3 allows us to achieve

second-order accuracy, whereas the method of Auer et al. [7] is restricted to first-order accuracy. Neumann IBCs at a point $\mathcal{C}$ are not well-defined since there is no preferred direction conormal to $\mathcal{C}$.

4.2.6 Localizing Computation Near $\mathcal{C}$

Computation to enforce IBCs should only be performed locally around $\mathcal{C}$ for efficiency. The new BC DOFs satisfy this requirement since they are only added at grid points $\mathbf{x}_i$ within a distance $r_{\Omega(\mathcal{S})}$ of $\mathcal{C}$. This banding of $\Omega(\mathcal{C})$ is possible for the same reason it is possible to band $\Omega(\mathcal{S})$ (see Section 3.2): grid points are only needed near $\mathcal{S}$ and $\mathcal{C}$ because accurate approximations of differential operators are only needed at grid points within interpolation stencils.

The use of the $\mathcal{S}_\perp$ crossing test (4.8) has been discussed in terms of checking all interpolation and FD stencils in $\Omega(\mathcal{S})$ and $\Omega(\mathcal{C})$ above. For efficiency, we would rather only check if $\text{cp}_\mathcal{S}(\mathbf{x}_1)$ and $\text{cp}_\mathcal{S}(\mathbf{x}_2)$ are on different sides of $\mathcal{S}_\perp$ if $\mathbf{x}_1$ and $\mathbf{x}_2$ are near $\mathcal{C}$. However, depending on the geometry of $\mathcal{S}$ and $\mathcal{C}$, points $\mathbf{x}_i \notin \Omega(\mathcal{C})$ can have stencils for interpolating at $\text{cp}_\mathcal{S}(\mathbf{x}_i)$ that cross $\mathcal{S}_\perp$, so testing only points $\mathbf{x}_i \in \Omega(\mathcal{C})$ does not suffice.

We therefore check stencils that include grid points $\mathbf{x}_i \in \Omega(\mathcal{S})$ with $\|\mathbf{x}_i - \text{cp}_\mathcal{C}(\mathbf{x}_i)\| < 2r_{\Omega(\mathcal{S})}$ for all the examples in this chapter. The closest points $\text{cp}_\mathcal{C}(\mathbf{x}_i)$ are needed to compute $\|\mathbf{x}_i - \text{cp}_\mathcal{C}(\mathbf{x}_i)\|$. Computation of $\text{cp}_\mathcal{C}$ for all $\mathbf{x}_i \in \Omega(\mathcal{S})$ is avoided using a similar breadth-first search to the one used in the construction of $\Omega(\mathcal{S})$ (see Algorithm 1).

4.2.7 Improving Robustness of $\mathcal{S}_\perp$ Crossing Test

In practice, manifolds with small bumps of high curvature relative to the grid resolution can cause the $\mathcal{S}_\perp$ crossing test (4.8) to be inaccurate. For example, the headdress of the Nefertiti mesh in Figure 4.9 has many small bumps, which causes the $\text{cp}_{\mathcal{S}-\mathcal{C}}$ and $\mathbf{d}^\star$ vectors to be far from orthogonal to $\mathcal{S}_\perp$ and $\mathcal{C}$. The closest points near $\mathcal{C}$ are then misclassified as being on the wrong side of $\mathcal{S}_\perp$.

To make (4.8) more robust, we modify the $\text{cp}_{\mathcal{S}-\mathcal{C}}$ and $\mathbf{d}^\star$ vectors to be orthogonal to $\mathcal{S}_\perp$ and $\mathcal{C}$ before computing the dot product. We illustrate this for a surface (2D manifold) embedded in $\mathbb{R}^3$ throughout this section. For this case, (4.8) is used with $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})$ replaced by

$$\text{cp}^\perp_{\mathcal{S}-\mathcal{C}}(\mathbf{x}) = \left(\mathbf{I} - \mathbf{n}_\mathcal{S}\mathbf{n}_\mathcal{S}{}^T - \mathbf{t}_\mathcal{C}\mathbf{t}_\mathcal{C}^T\right) \text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}), \tag{4.13}$$

(and similarly for $\mathbf{d}^\star$) where $\mathbf{I}$ is the identity matrix and $\mathbf{t}_\mathcal{C}$ is the unit tangent vector along $\mathcal{C}$. The manifold normal $\mathbf{n}_\mathcal{S}$ and tangent $\mathbf{t}_\mathcal{C}$ are evaluated at $\text{cp}_\mathcal{C}(\mathbf{x})$. Projecting out the $\mathbf{n}_\mathcal{S}$ and $\mathbf{t}_\mathcal{C}$ components is equivalent to projecting $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})$ onto $\mathbf{n}_{\mathcal{S}_\perp}(\text{cp}_\mathcal{C}(\mathbf{x}))$. Therefore, the $\mathcal{S}_\perp$ crossing test (4.8) becomes equivalent to the direct test that checks if $F(\mathbf{x}_1)F(\mathbf{x}_2) < 0$ (see Section 4.2.2), but without needing to orient $\mathbf{n}_{\mathcal{S}_\perp}$. The vectors $\mathbf{n}_\mathcal{S}$ and $\mathbf{t}_\mathcal{C}$ must be evaluated at $\text{cp}_\mathcal{C}(\mathbf{x})$ since the vector $\text{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x})$ starts at $\text{cp}_\mathcal{C}(\mathbf{x})$ (and goes to $\text{cp}_\mathcal{S}(\mathbf{x})$). When $\mathcal{C}$ is a single point the tangent direction is undefined, so only the $\mathbf{n}_\mathcal{S}$ component is projected out in this case. Let us now consider how to compute $\mathbf{n}_\mathcal{S}$ and $\mathbf{t}_\mathcal{C}$.

For a codimension-one manifold $\mathcal{S}$ the Jacobian of the closest point function, $\mathbf{J}_{\text{cp}_\mathcal{S}}$, is the projection operator onto the tangent space of $\mathcal{S}$ for points on the manifold [96, 72, 109]. Therefore, for a surface in $\mathbb{R}^3$, the eigenvectors of $\mathbf{J}_{\text{cp}_\mathcal{S}}$ are the manifold normal $\mathbf{n}_\mathcal{S}$ and two tangent vectors. However, two arbitrary tangent vectors of $\mathcal{S}$ will not suffice; we need the tangent $\mathbf{t}_\mathcal{C}$ along $\mathcal{C}$. The curve $\mathcal{C} \in \mathbb{R}^3$ has codimension two. The corresponding Jacobian for $\mathcal{C}$, $\mathbf{J}_{\text{cp}_\mathcal{C}}$, is likewise equivalent to a projection operator onto the tangent space of $\mathcal{C}$ [75]. However, the eigenvectors of $\mathbf{J}_{\text{cp}_\mathcal{C}}$ only provide a unique tangent vector $\mathbf{t}_\mathcal{C}$, since the normal and binormal to $\mathcal{C}$ can freely rotate around $\mathbf{t}_\mathcal{C}$. Hence, we compute the manifold normal $\mathbf{n}_\mathcal{S}$ from the eigendecomposition of $\mathbf{J}_{\text{cp}_\mathcal{S}}$, while $\mathbf{t}_\mathcal{C}$ is computed from the eigendecomposition of $\mathbf{J}_{\text{cp}_\mathcal{C}}$.

Second-order centred FDs in $\Omega(\mathcal{S})$ are used to compute $\mathbf{J}_{\text{cp}_\mathcal{S}}$. The Jacobian $\mathbf{J}_{\text{cp}_\mathcal{S}}$ is only equivalent to the tangent space projection operator at points on $\mathcal{S}$. Therefore, a CP extension must be performed to obtain the projection operator at all points $\mathbf{x}_i \in \Omega(\mathcal{S})$, i.e., $\mathbf{J}_{\text{cp}_\mathcal{S}}(\mathbf{x}_i) = \mathbf{J}_{\text{cp}_\mathcal{S}}(\text{cp}_\mathcal{S}(\mathbf{x}_i))$. In the discrete setting, the CP extension is computed with the same interpolation discussed in Chapter 3. The Jacobian of $\text{cp}_\mathcal{C}$ is computed similarly over $\Omega(\mathcal{C})$.

From the above computation of $\mathbf{J}_{\text{cp}_\mathcal{S}}$ and $\mathbf{J}_{\text{cp}_\mathcal{C}}$, the projection operators are known at points $\text{cp}_\mathcal{S}(\mathbf{x}_i)$ and $\text{cp}_\mathcal{C}(\mathbf{x}_i)$, respectively. However, since the $\mathbf{n}_\mathcal{S}$ vectors are computed from $\mathbf{J}_{\text{cp}_\mathcal{S}}$, they are not yet available at $\text{cp}_\mathcal{C}(\mathbf{x}_i)$ where we need them. The $\mathbf{n}_\mathcal{S}$ vectors are therefore computed at $\text{cp}_\mathcal{C}(\mathbf{x}_i)$ via barycentric-Lagrange interpolation (with the same degree p polynomials as the CP extension). Interpolating $\mathbf{n}_\mathcal{S}$ vectors requires some care since they are *unoriented* manifold normals. We adapt a technique proposed by Auer et al. [7]: when interpolating $\mathbf{n}_\mathcal{S}$, given at points $\mathbf{x}_i \in \Omega(\mathcal{S})$, we locally orient the vectors within each interpolation stencil by negating vectors satisfying

$$\mathbf{n}_\mathcal{S}(\mathbf{x}_i) \cdot \mathbf{n}_\mathcal{S}(\tilde{\mathbf{x}}) < 0,$$

where $\tilde{\mathbf{x}}$ is a single, fixed grid point in the interpolation stencil.

4.2.8 A Nearest Point Approach for Dirichlet IBCs

It is also interesting to consider a *nearest point* approach for handling Dirichlet IBCs at $\mathcal{C}$, similar to techniques discussed at the start of this chapter for other manifold representations. That is, simply fix the grid points $\mathbf{x}_i \in \Omega(\mathcal{S})$ nearest to $\mathcal{C}$ with the prescribed Dirichlet value, and remove them as DOFs. If $\mathcal{C}$ is a point, a single grid point is assigned the Dirichlet value and removed as a DOF. If $\mathcal{C}$ is a curve, a set of nearest grid points is obtained (i.e., a raster representation of $\mathcal{C}$) and removed as DOFs by assigning Dirichlet values. To our knowledge, this approach has not been used with CPM previously.

This nearest point approach is attractive since new BC DOFs are unnecessary, i.e., $\Omega(\mathcal{C})$ is not needed. However, it can only be used for Dirichlet IBCs with the same value on both sides of $\mathcal{C}$. That is, two-sided Dirichlet IBCs cannot be imposed with the nearest point approach, nor can Neumann IBCs. The nearest point approach is also only first-order accurate since the nearest point can be $h\sqrt{d}/2$ away from $\mathcal{C}$. We observe that the nearest point approach has a better error constant than the method of Auer et al. [7], but a similar or worse error constant than our first-order IBC approach above (see Figure 4.7(c)).

4.3 Convergence Studies

We begin our evaluation by verifying that our proposed IBC schemes achieve the expected convergence orders on various analytical problems. We also compare our approach with the existing CPM approach of Auer et al. [7], the nearest point approach, as well as a standard mesh-based method for reference. All error values are computed using the max-norm. Throughout the rest of this chapter, the subscript $\mathcal{S}$ has been removed from the manifold functions (e.g., $u_{\mathcal{S}}$), since it is apparent from the context.

4.3.1 Poisson Equation with Discontinuous Solution

Consider the Poisson equation

$$\begin{aligned} -\frac{\partial^2 u}{\partial \theta^2} &= 2\cos(\theta - \theta_{\mathcal{C}}), \\ u(\theta_{\mathcal{C}}^-) &= 2, \\ u(\theta_{\mathcal{C}}^+) &= 22, \end{aligned}$$

on the unit circle parameterized by θ. The right-hand-side expression is found by differentiating the exact solution

$$u(\theta) = 2\cos(\theta - \theta_{\mathcal{C}}) + \frac{10}{\pi}(\theta - \theta_{\mathcal{C}}),$$

where $\theta_{\mathcal{C}}$ is the location of the Dirichlet IBC. The Dirichlet IBC is two-sided and thus discontinuous at the point $\theta_{\mathcal{C}}$, with $u = 2$ as $\theta \to \theta_{\mathcal{C}}^-$ and $u = 22$ as $\theta \to \theta_{\mathcal{C}}^+$. We use $\theta_{\mathcal{C}} = 1.022\pi$; no grid points coincide with the IBC location.

Eigen's SparseLU is used to solve the linear system for this problem on the circle embedded in $\mathbb{R}^2$. Figure 4.5(a) shows that the first and second-order IBCs discussed in Section 4.2 achieve the expected convergence rates. Neither the nearest point approach (Section 4.2.8) nor the method of Auer et al. [7] can handle discontinuous IBCs.

4.3.2 Heat Equation

CPM can also be applied to time-dependent problems. Consider the heat equation

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u, \quad \text{with} \quad \begin{cases} u = g, \text{ or} \\ \nabla_{\mathcal{S}} u \cdot \mathbf{b}_{\mathcal{C}} = 0, \end{cases} \quad \text{on } \mathcal{C}, \tag{4.14}$$

where $\mathbf{b}_{\mathcal{C}}$ is the binormal direction to $\mathcal{C}$ that is also in the tangent plane of $\mathcal{S}$, i.e., $\mathbf{b}_{\mathcal{C}} = \mathbf{n}_{\mathcal{S}} \times \mathbf{t}_{\mathcal{C}}$ (see Section 4.2.7). The binormal is analogous to the conormal $\mathbf{n}_{\partial\mathcal{S}}$ used for exterior BCs. If imposing the Dirichlet IBC, the exact solution, g, is used as the prescribed function on $\mathcal{C}$. Here we solve the heat equation on the unit sphere with the exact solution

$$g(\theta, \phi, t) = e^{-2t}\cos(\phi),$$

where θ is the azimuthal angle and ϕ is the polar angle. The IBC is imposed with $\mathcal{C}$ as a circle defined by the intersection of a plane with $\mathcal{S}$. The initial condition is taken as $g(\theta, \phi, 0) = \cos(\phi)$.

Crank-Nicolson time-stepping [79] (a.k.a., trapezoidal rule) is used with $\Delta t = 0.1h$ until time $t = 0.1$. Figure 4.5 (b) and (c) show convergence studies for (4.14) with Dirichlet and zero-Neumann IBCs imposed, respectively. The expected order of accuracy for first and second-order IBCs is achieved for both the Dirichlet and zero-Neumann cases. Recall that the nearest point approach and the method of Auer et al. [7] cannot handle Neumann IBCs.

4.3.3 Screened-Poisson Equation

Exact solutions for manifold PDEs can also be derived on more complex manifolds defined as level sets. Consider the screened-Poisson problem in Section 4.6.5 of [24], which was inspired by an example by Dziuk [43]. The surface is defined as $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 \mid (x_1 - x_3^2)^2 + x_2^2 + x_3^2 = 1\}$, which we refer to as the Dziuk surface.

The screened-Poisson equation we solve is

$$\begin{aligned} -\Delta_{\mathcal{S}} u + u &= f, \\ \nabla_{\mathcal{S}} u \cdot \mathbf{b}_{\mathcal{C}} &= 0, \end{aligned} \tag{4.15}$$

with exact solution $u(\mathbf{x}) = x_1 x_2$. Although the solution is simple, the function f is complicated; we derived it by symbolic differentiation using the formulas in [24, 43].

The zero-Neumann IBC of (4.15) is satisfied on the intersection of $\mathcal{S}$ with the x_1x_2-plane. From the definition of $\mathcal{S}$, this intersection is the unit circle in the x_1x_2-plane. Figure 4.5 (d) and (e) show convergence studies imposing the zero-Neumann IBC on the full circle (closed curve) and the arc with $\theta \in [-\frac{3\pi}{4}, \frac{\pi}{4}]$ (open curve), respectively. The expected order of accuracy is observed for the implementations of first and second-order IBCs.

4.3.4 Different CPM approaches vs. a Mesh-Based Method

CPM is principally designed to solve problems on *general* manifolds, given by their closest point functions. The closest point function can be thought of as a black box allowing many manifold representations to be handled in a unified framework. Hence, we emphasize that one should not expect CPM to universally surpass specially tailored, well-studied approaches for particular manifold representations, such as finite elements on (quality) triangle meshes. Nevertheless, mesh-based schemes provide a useful point of reference for our evaluation. CPM also retains some advantages even for triangle meshes, such as mesh-independent behaviour.

With the above caveat in mind, we compare the various CPM approaches to the standard cotangent Laplacian [131, 43] that approximates the Laplace-Beltrami operator on a triangulation of the surface. We use the implementation from `geometry-central` [154], adapted slightly to include IBCs. The Poisson equation $-\Delta_{\mathcal{S}} u = f$ is solved on the Dziuk surface defined in Section 4.3.3. The same exact solution $u(\mathbf{x}) = x_1 x_2$ is used, but Dirichlet IBCs are imposed using this exact solution.

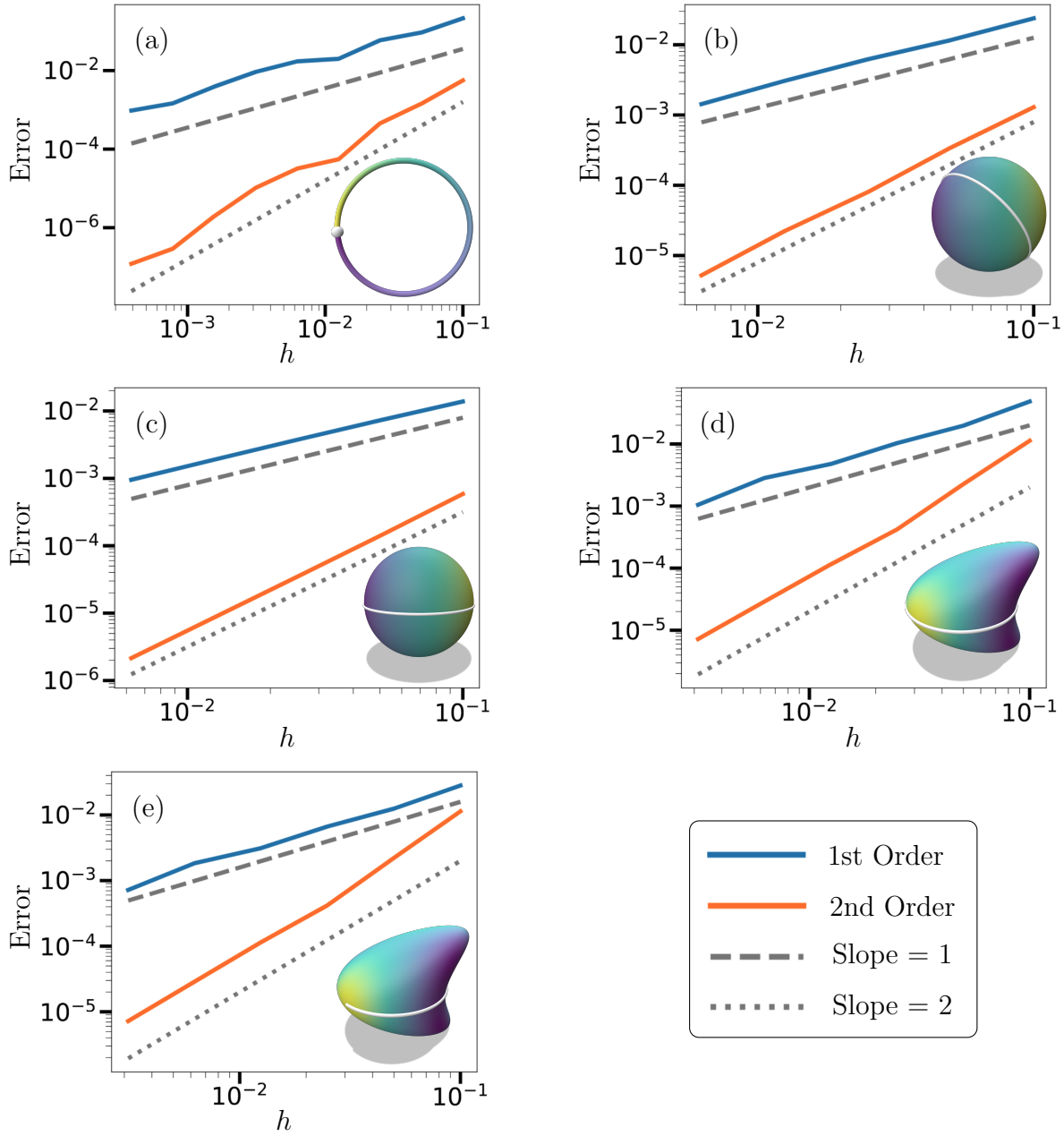


Figure 4.5: Convergence studies and associated geometries for the model problems in Sections 4.3.1-4.3.3. The plots show results for our CPM approach using first (blue) and second (orange) order IBCs, along with lines of slopes 1 (grey, dashed) and 2 (grey, dotted). In (a)-(c) analytical $\text{cp}_{\mathcal{S}}$ are used, while (d) and (e) compute $\text{cp}_{\mathcal{S}}$ from the level-set representation of $\mathcal{S}$. All examples use analytical $\text{cp}_{\mathcal{C}}$.

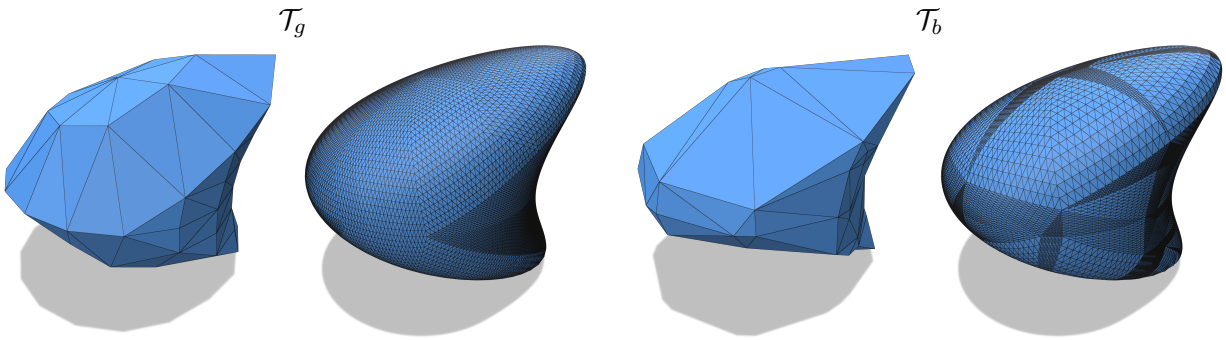


Figure 4.6: Triangulations of the Dziuk surface used for testing. Left: Good-quality triangulation, $\mathcal{T}_g$, at its base resolution (a) and after four rounds of refinement (b). Right: Low-quality triangulation, $\mathcal{T}_b$, also at its base (c) and four times refined (d) resolutions.

"Good" and "bad" triangulations of the Dziuk surface, denoted $\mathcal{T}_g$ and $\mathcal{T}_b$ respectively, are used to illustrate the dependence of the mesh-based method on triangulation quality (Figure 4.6). Both triangulations are constructed starting from six vertices on $\mathcal{S}$ as in [43]. An initial round of 1:4 subdivision is performed by adding new vertices along each edge, at the midpoint for $\mathcal{T}_g$ and at the 20% position for $\mathcal{T}_b$, to induce skinnier triangles in the latter. The new vertices are projected to their closest points on $\mathcal{S}$.

Evaluations under refinement for the mesh-based method are performed starting with the above first-level $\mathcal{T}_g$ and $\mathcal{T}_b$. We refine with uniform 1:4 subdivision, for both $\mathcal{T}_g$ and $\mathcal{T}_b$, by adding new vertices at midpoints of edges and then projecting them onto $\mathcal{S}$ (see Figure 4.6). Delaunay edge flips are also performed to improve the quality of $\mathcal{T}_g$ at each refinement level.

Triangle mesh resolution is measured as the mean edge-length in $\mathcal{T}_g$ or $\mathcal{T}_b$, whereas for CPM resolution is measured as the uniform h used in the computational tube $\Omega(\mathcal{S})$. This core incompatibility makes it inappropriate to use resolution as the independent variable for comparative evaluations of error, computation time, or memory usage. A more equitable comparison is to investigate computation time versus error and memory versus error. Computation times for CPM include the construction of $\Omega(\mathcal{S})$ and $\Omega(\mathcal{C})$ (which involves computing $\text{cp}_{\mathcal{S}}$ and $\text{cp}_{\mathcal{C}}$) and the time for constructing and solving the linear system. Computation times for the mesh-based method include the triangulation refinement and the construction and solution of the linear system. Separate evaluations are performed with $\mathcal{C}$ as a closed curve, an open curve, and a point, since CPM IBC

enforcement is slightly different for each type of $\mathcal{C}$. The $\text{cp}_{\mathcal{S}}$ are computed from a level-set representation, while $\text{cp}_{\mathcal{C}}$ are computed from polyline representations for curves $\mathcal{C}$ and exactly for the point $\mathcal{C}$.

Closed Curve IBC

The boundary curve $\mathcal{C}$ is constructed using the flip geodesics algorithm in `geometry-central` [154]. The resulting $\mathcal{C}$ is represented as a polyline $\mathcal{P}$, which in general does *not* conform to edges or vertices of $\mathcal{T}$. For IBC enforcement, the nearest vertex in the triangulation $\mathcal{T}$ to each vertex in $\mathcal{P}$ is assigned the prescribed Dirichlet value.

This treatment of Dirichlet IBCs for the mesh-based method is first-order accurate in general. More accurate (and involved) Dirichlet IBC approaches could be used as discussed at the beginning of this chapter. However, we set these options aside, as the goal of this comparison is simply to show that CPM with our first and second-order IBC approaches gives comparable results to basic mesh-based methods, that is, mesh-based methods where the representations of $\mathcal{S}$ and $\mathcal{C}$ are held fixed, e.g., no (extrinsic or intrinsic) remeshing is performed.

Figure 4.7 (top row) compares all types of CPM IBC approaches against the mesh-based method on $\mathcal{T}_g$ and $\mathcal{T}_b$ in columns (a) and (b), which show computation time vs. error and memory vs. error, respectively. CPM with second-order IBCs achieves the lowest error for the same computation time and memory usage as other approaches. The mesh-based method with $\mathcal{T}_g$ outperforms the use of $\mathcal{T}_b$, as expected. CPM with first-order IBCs and nearest point approaches are similar and lie between the mesh-based method with $\mathcal{T}_g$ and $\mathcal{T}_b$. The method of Auer et al. [7] has the largest error compared to all others. The expected order of convergence is seen for all CPM IBC approaches in the error versus h plot of Figure 4.7 (top row, (c)).

Open Curve IBC

The open curve $\mathcal{C}$ is also constructed using the flip geodesics algorithm in `geometry-central` [154]. The Dirichlet IBC is enforced in the mesh-based solver in the same way as the closed curve above. Figure 4.7 (middle row) shows the same ranking of the methods as in the closed curve case, except CPM with first-order IBCs now outperforms both triangulations and the nearest point CPM approach. The expected order of convergence is seen for all CPM IBC approaches in Figure 4.7 (middle row, (c)).

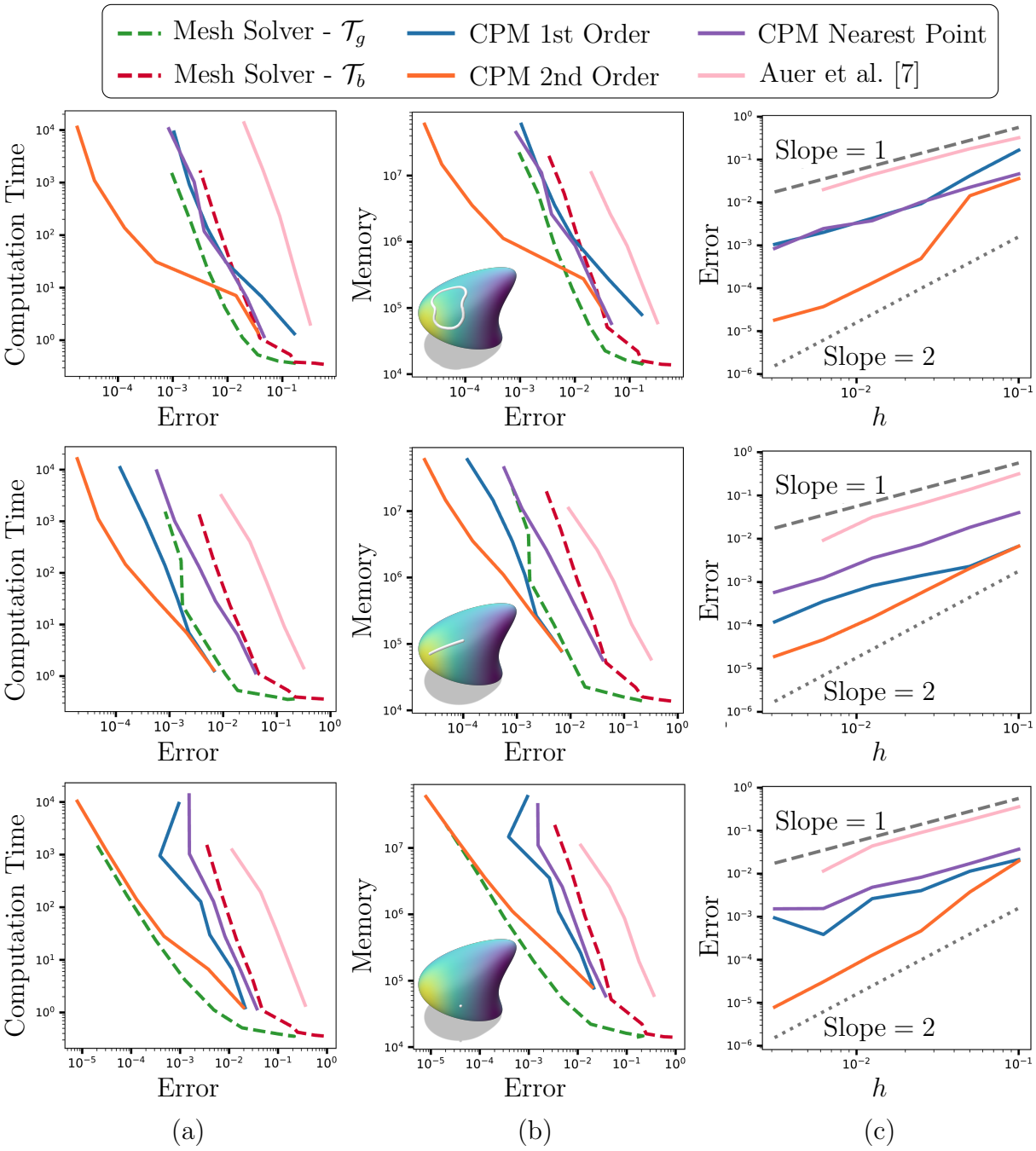


Figure 4.7: A comparison of CPM vs. the mesh-based cotangent Laplacian for the Poisson equation with Dirichlet IBC. Top row: Closed curve $\mathcal{C}$. Middle row: Open curve $\mathcal{C}$. Bottom row: Point $\mathcal{C}$.

Point IBC

The point $\mathcal{C}$ is intentionally chosen as one of the vertices in the base triangulation so that it is present in all refinements of $\mathcal{T}_g$ and $\mathcal{T}_b$. The Dirichlet IBC at $\mathcal{C}$ is imposed by replacing the vertex DOF in $\mathcal{T}$ with the prescribed Dirichlet value. Figure 4.7 (bottom row) shows the results for a point $\mathcal{C}$.

The mesh-based solver on $\mathcal{T}_g$ converges with second-order accuracy (since the IBC is a vertex), but only first-order accuracy on $\mathcal{T}_b$. Therefore, the mesh-based method with $\mathcal{T}_g$ outperforms CPM with second-order IBCs in the larger error regime. In the lower error regime, the latter methods are similar. All other methods show the same ranking as the open curve case.

The expected order of convergence is seen for all CPM IBC approaches in Figure 4.7 (bottom row, (c)). Notably, the second-order IBC version of CPM exhibits slightly higher than expected errors at the finest grid resolution for the closed and open curve IBCs (see Figure 4.7, top and middle rows, (c)). This is caused by the resolution of the polyline representation of $\mathcal{C}$: at fine grid resolutions, the inherent sharp features of the coarse polyline $\mathcal{C}$ begin to be resolved more fully by the discrete CP function. Accordingly, no such reduction in convergence order is seen for the point IBC.

4.4 Applications

We now show the ability of our CPM approach to solve PDEs with IBCs that are common in applications from geometry processing: diffusion curves, geodesic distance, vector field design, harmonic maps, and reaction-diffusion textures.

Quadratic polynomial interpolation, i.e., $p = 2$, is used for all the examples in this section. Current CPM theory suggests that only first-order accuracy can be expected with quadratic polynomial interpolation, but CPM has been observed to give second-order convergence numerically (see [88, Section 4.1.1]). This behaviour is confirmed with IBCs in Figure 4.8.

The main motivation for choosing quadratic interpolation is to obtain smaller computational tube-radii, $r_{\Omega(\mathcal{S})}$, which allows higher curvature $\mathcal{S}$ and $\mathcal{C}$ to be handled with larger h. The resulting $\Omega(\mathcal{S})$ and $\Omega(\mathcal{C})$ contain fewer DOFs and therefore the computation is more efficient. Furthermore, Figure 4.8 shows that, for the same h, quadratic interpolation has lower computation times. Quadratic interpolation is 1.1-2.1 times faster than cubic interpolation in Figure 4.8. We used $p = 3$ in the convergence studies of Section 4.3 because

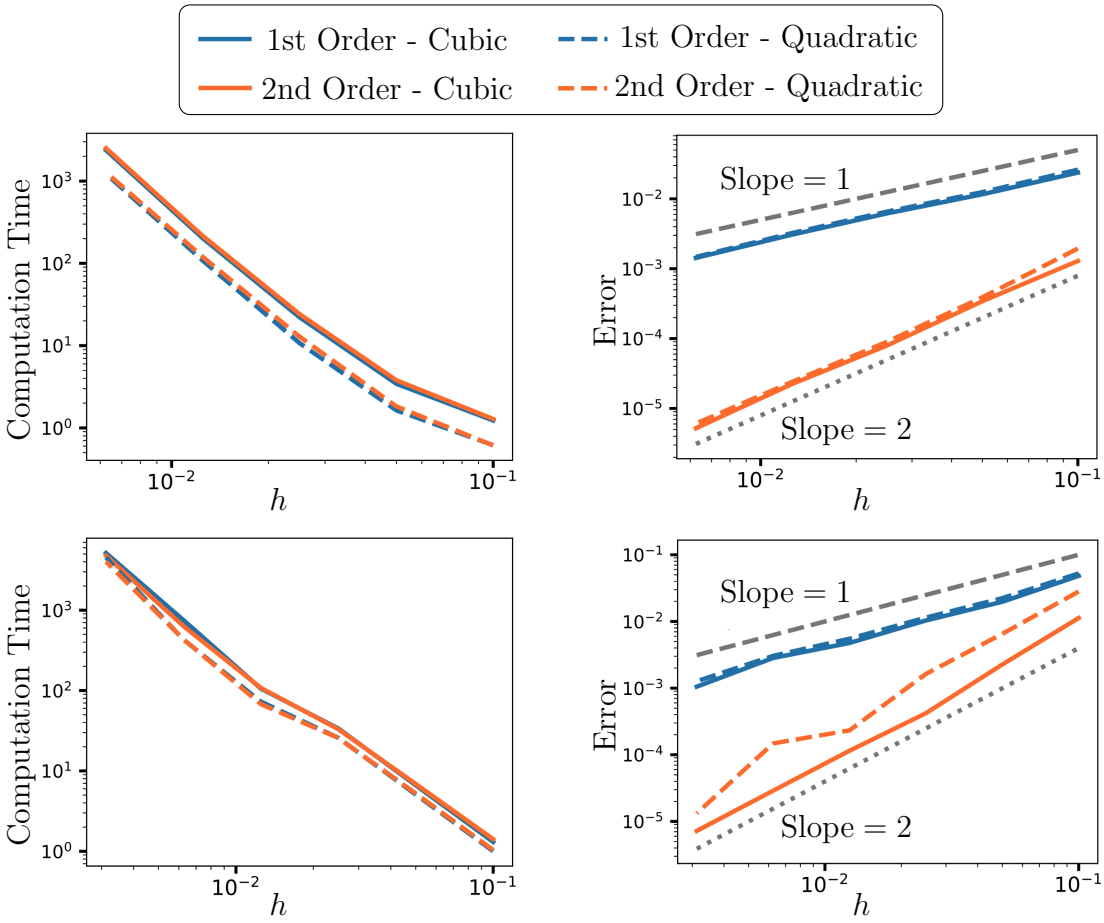


Figure 4.8: A comparison of CPM with quadratic vs. cubic interpolation stencils for the heat (top row) and Poisson (bottom row) problems of Figure 4.5 (b) and (d). Comparable results are achieved, but quadratic is often faster while cubic typically exhibits more regular convergence.

the error for second-order BCs with $p = 2$ can sometimes be less regular (i.e., decreasing unevenly or non-monotonically) than with $p = 3$ (Figure 4.8, bottom right).

CPM with first-order IBCs is used in all the examples in this section. The geodesic distance, vector field design, and harmonic map algorithms used here are themselves all inherently first-order accurate; hence using second-order IBCs would only improve accuracy near $\mathcal{C}$. Second-order IBCs could have been used for diffusion curves and reaction-diffusion textures, but the first-order method was used for consistency. Note also that any surface represented as a mesh is isotropically scaled (with fixed aspect ratio) to fit in $[-1, 1]^3$.

4.4.1 Diffusion Curves

Diffusion curves offer a sparse representation of smoothly varying colours for an image [122] or surface texture [64]. Obtaining colours over all of $\mathcal{S}$ requires solving the Laplace-Beltrami equation with IBCs:

$$\Delta_{\mathcal{S}} u^i = 0, \quad \text{with} \quad \begin{cases} u^i = g^i, \text{ or} \\ \nabla_{\mathcal{S}} u^i \cdot \mathbf{b}_{\mathcal{C}} = 0, \end{cases} \quad \text{on } \mathcal{C}. \tag{4.16}$$

The Laplace-Beltrami equation (4.16) is solved for each colour channel u^i independently with CPM. The colour vector is composed of all the colour channels, e.g., for RGB colours $\mathbf{u} = [u^1, u^2, u^3]^T$. Dirichlet IBCs, $u^i = g^i$ on $\mathcal{C}$, are used to specify the colour values at sparse locations on $\mathcal{S}$. These colours spread over all of $\mathcal{S}$ when the Laplace-Beltrami equation is solved. Zero-Neumann IBCs can be used to treat $\mathcal{C}$ as a passive barrier that colours cannot cross. Two-sided IBCs along $\mathcal{C}$ are also easily handled, and can even be of mixed Dirichlet-Neumann type (not to be confused with Robin BCs).

The surface of the Nefertiti bust [3] is coloured by solving the Laplace-Beltrami equation with CPM with $h = 0.00315$ and IBCs specified by diffusion curves in Figure 4.9. IBC curves are polylines created using the flip geodesics algorithm in `geometry-central` [154]. Most curves are two-sided Dirichlet IBCs (white curves, Figure 4.9 left) that specify a colour for each side of the curve. However, the red and green band on the headdress is created using both two-sided Dirichlet IBCs vertically and two-sided Neumann-Dirichlet IBCs horizontally (black curves, Figure 4.9 left). The (white) vertical Dirichlet IBCs specify green on one side of the curve and red on the other. For the (black) horizontal Neumann-Dirichlet IBCs, the zero-Neumann side allows the different shades of red and green from the vertical Dirichlet IBCs to diffuse together in each block without specifying the colour value horizontally, while the Dirichlet side is specified as the navy colour value.

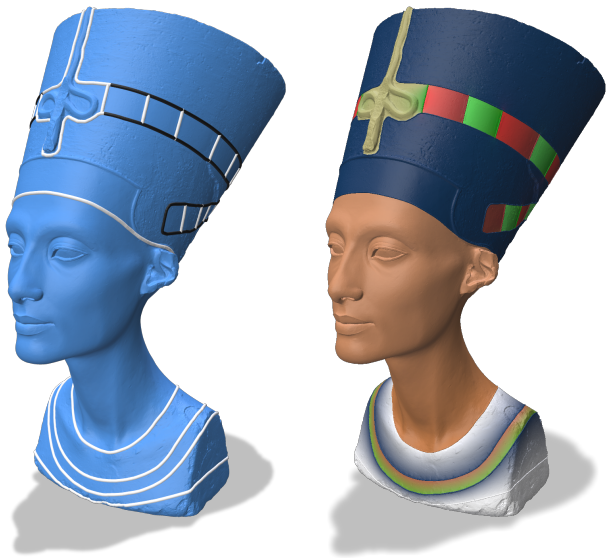


Figure 4.9: Colouring a triangulated surface using diffusion curves.

Mixed-Codimensional Objects

The generality of CPM allows PDEs on mixed-codimensional objects to be solved. The theoretical assumption that $\text{cp}_{\mathcal{S}}$ is unique is violated in this case (near pieces of differing codimension). However, CPM gives the expected result in practice on mixed-codimensional objects (e.g., [88, Figure 4.4]).

Figure 4.10 shows a diffusion curves example (with $h = 0.05$) featuring a mixed 1D and 2D object embedded in $\mathbb{R}^3$. This mixed-codimensional $\mathcal{S}$ is created using analytical closest point functions for the torus, sphere, and line segment. The torus has minor radius $r = 1$ and major radius $R = 3$, while the sphere is of radius 1.25. The closest point to $\mathcal{S}$ is determined by computing the closest point to each of the torus, sphere, and line segments, then taking the closest of all four. The two curves $\mathcal{C}$ in this example are two-sided Dirichlet IBCs. $\mathcal{C}$ on the torus is a torus knot specified by the parametric equation

$$x(s) = v(s)\cos(as), \quad y(s) = v(s)\sin(as), \quad z(s) = \sin(bs), \tag{4.17}$$

with $v(s) = R + \cos(bs)$, $a = 3$, $b = 7$, and $s \in [0, 2\pi]$. Closest points for the torus knot are computed using the optimization problem discussed in Appendix A. $\mathcal{C}$ on the sphere is an analytical closest point function for a circle defined as the intersection of the sphere and a plane. Notice the colour from the torus to the sphere blends across the line segments as expected (see Figure 4.10 zoom).

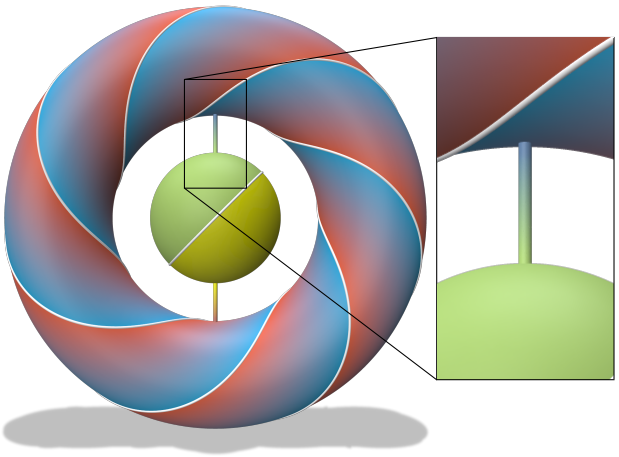


Figure 4.10: Diffusion curves on a nonmanifold object of mixed codimension. Line segments connect the torus to the sphere, which are all represented with analytical $\text{cp}_{\mathcal{S}}$. The $\text{cp}_{\mathcal{C}}$ for the circle on the sphere is computed analytically, while $\text{cp}_{\mathcal{C}}$ for the torus knot is computed from a parametrization.

Codimension-Zero Manifolds

Interestingly, CPM can also be applied with codimension-zero manifolds (see [90, Section 6.2.4]). A codimensional-zero manifold is a solid object that is a subset of $\mathbb{R}^{\dim(\mathcal{S})}$. Consider a codimension-zero $\mathcal{S}$, with a boundary $\partial\mathcal{S}$. The computational domain $\Omega(\mathcal{S})$ consists of all grid points $\mathbf{x}_i \in \mathcal{S}$ (having $\text{cp}_{\mathcal{S}}(\mathbf{x}_i) = \mathbf{x}_i$) plus a layer of grid points outside $\mathcal{S}$ where $\text{cp}_{\mathcal{S}}(\mathbf{x}_i) \in \partial\mathcal{S}$ and $\|\mathbf{x}_i - \text{cp}_{\mathcal{S}}(\mathbf{x}_i)\| \leq r_{\Omega(\mathcal{S})}$.

Figure 4.11 shows an example of applying CPM to the diffusion curves problem with $\mathcal{S}$ as the square $[-1,1]^2$ and $\Omega(\mathcal{S}) \subset \mathbb{R}^2$. A parametric curve on the interior of $\mathcal{S}$ defines a diffusion curve $\mathcal{C}$ as a two-sided Dirichlet IBC, given by

$$x(s) = v(s)\cos(s) + c, \quad y(s) = v(s)\sin(s) + c, \tag{4.18}$$

where

$$v(s) = \frac{\cos(s)\left(\frac{1}{2}(a+b) + \sin(as) + \sin(bs)\right) + \frac{1}{2}(a+b)}{a+b},$$

with $a = 3$, $b = 4$, $c = -\frac{1}{2}$, and $s \in [0, 2\pi]$. Note that the colour varies along $\mathcal{C}$ from red to green inside $\mathcal{C}$ and blue to green outside $\mathcal{C}$. (Such colour variations along boundaries $\mathcal{C}$ can also easily be applied to problems where the embedding domain has higher dimension than

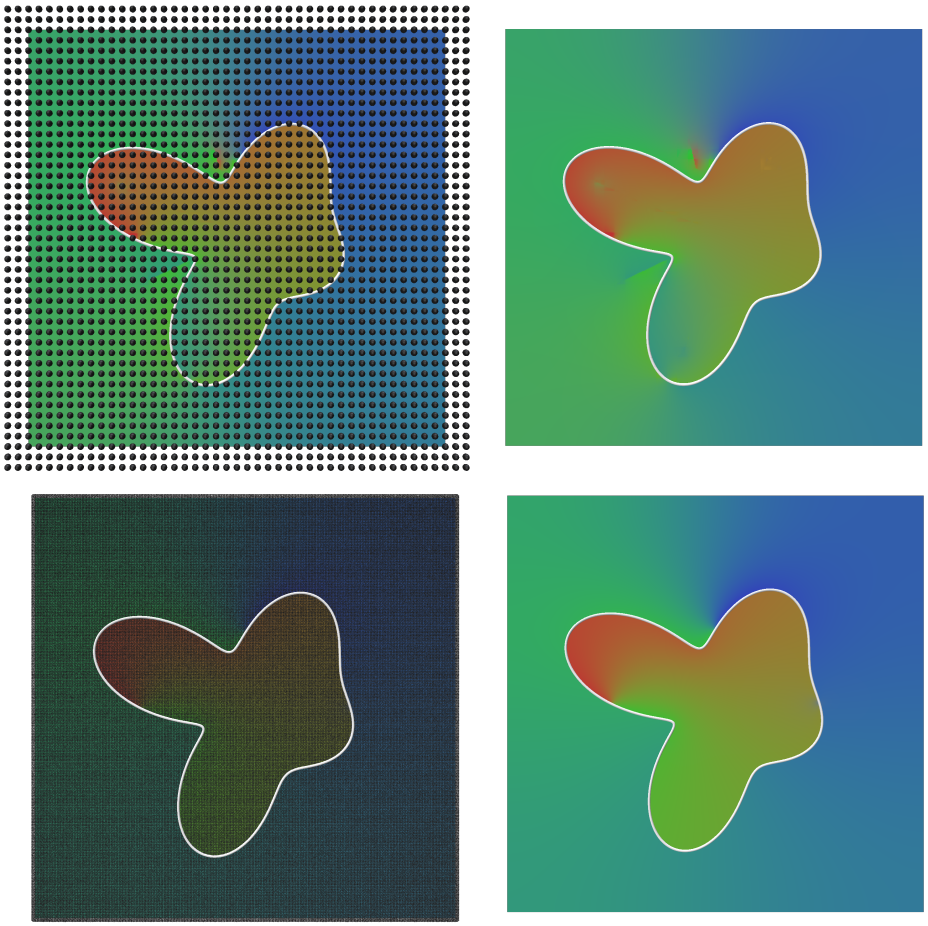


Figure 4.11: CPM applied to a codimension-zero diffusion curve problem, with the Dirichlet colour value varying along the white IBC curve. Top row: At an insufficient grid resolution of $h = 0.05$ (left), high curvature regions exhibit errors near the curve's medial axis (right). Bottom row: A high-resolution grid with $h = 0.005$ (left) resolves the artifacts (right). The $\mathrm{cp}_{\mathcal{S}}$ are computed analytically and $\mathrm{cp}_{\mathcal{C}}$ are computed from a parametric representation.

$\mathcal{S}$.) First-order zero-Neumann exterior BCs are applied on $\partial\mathcal{S}$ naturally by CPM, which enforces no (conormal, i.e., normal to ∂S and in the tangent space of $\mathcal{S}$) colour gradient at $\partial\mathcal{S}$.

The grid spacing h needs to be fine enough near $\mathcal{C}$ to give an accurate solution. Artifacts can occur if stencils undesirably cross the medial axis of $\mathcal{C}$ when h is too large (cf. Figure 4.11 top and bottom rows). It would be beneficial to explore the use of our adaptivity framework in Section 3.4 based on the geometry of $\mathcal{C}$ for this issue. Adaptivity would reduce the total number of DOFs in the linear system and thus improve efficiency while being able to still obtain a satisfactory result.

Applying CPM with $\text{codim}(\mathcal{S}) = 0$ represents an alternative to (or generalization of) various existing embedded boundary methods for irregular domains, e.g., [50, 118, 146]. Advantages and disadvantages of this approach should be explored further in future work. One advantage shown by Macdonald et al. [90] is the ability to couple volumetric and surface PDEs in a unified framework.

4.4.2 Geodesic Distance

The heat method for geodesic distance computation [33] has been implemented on many surface representations, including polygonal surfaces, subdivision surfaces [37], spline surfaces [119], tetrahedral meshes [16], and point clouds [33], with each requiring nonnegligible tailoring and implementation effort. By introducing our Dirichlet IBC treatment for CPM, we enable a single implementation covering all these cases, since closest points can be computed to these and many other manifold representations.

The heat method approximates the geodesic distance ϕ using the following three steps:

1. Solve $\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u$ to give u_t at time t,
2. Evaluate the vector field $\mathbf{X} = -\nabla_{\mathcal{S}} u_t / \|\nabla_{\mathcal{S}} u_t\|$,
3. Solve $\Delta_{\mathcal{S}} \phi = \nabla_{\mathcal{S}} \cdot \mathbf{X}$ for ϕ.

Step (1) uses a Dirac-delta heat source for a point $\mathcal{C}$ or a generalized Dirac distribution over a curve $\mathcal{C}$ as the initial condition. The time discretization of step (1) employs implicit Euler, for one time-step, which is equivalent (up to a multiplicative constant) to solving

$$\begin{aligned} (\mathbf{I} - t\Delta_{\mathcal{S}}) v_t &= 0 \ \text{ on } \ \mathcal{S}\backslash\mathcal{C}, \\ v_t &= 1 \ \text{ on } \ \mathcal{C}. \end{aligned} \tag{4.19}$$

The discrete system for (4.19) can be written as $\mathbf{A}\mathbf{v} = \mathbf{f}$, where $\mathbf{A} \in \mathbb{R}^{(N_\mathcal{S}+N_\mathcal{C})\times(N_\mathcal{S}+N_\mathcal{C})}$ and $\mathbf{v}, \mathbf{f} \in \mathbb{R}^{N_\mathcal{S}+N_\mathcal{C}}$.

Imposing first-order IBCs involves the Heaviside step function for $\mathbf{f}$ since $\mathbf{f}_i = 0$ if i is in the PDE DOF set ($i \in J_\mathcal{S}$) and $\mathbf{f}_i = 1$ if i is in the BC DOF set ($i \in J_\mathcal{C}$). In words, the Heaviside step function jumps from 0 to 1 in the embedding space as you cross $\mathcal{S}_\perp$, i.e., from the PDE DOFs to the BC DOFs. When imposing this IBC in (4.19), CPM can experience Runge's phenomenon due to the polynomial interpolation used for the CP extension. Therefore, we approximate the Heaviside step function with a smooth approximation as

$$\mathbf{f}_i = \frac{1}{2}\tanh\left(-k\|\mathrm{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_i)\|\right) + \frac{1}{2}, \quad \text{with} \quad k = \frac{\operatorname{atanh}(1-\epsilon)}{e}.$$

The parameters e and ϵ correspond to the "extent" $[-e, e]$ and the maximum error of the approximation outside of the extent, respectively. That is, when $\|\mathrm{cp}_{\mathcal{S}-\mathcal{C}}(\mathbf{x}_i)\| = e$, the error in approximating the Heaviside function is ϵ and the error becomes smaller further outside of $[-e, e]$. We choose $e = r_{\Omega(\mathcal{S})}$ and $\epsilon = h$ for our results.

Step (3) of the heat method also involves a Dirichlet IBC, $\phi = 0$ on $\mathcal{C}$, since the geodesic distance is zero for points on $\mathcal{C}$. No special treatment is required for this IBC. To improve accuracy, steps (2) and (3) are applied iteratively as discussed by Belyaev and Fayolle [16]. Two extra iterations of steps (2) and (3) are applied in all our examples of the CPM-based heat method.

We use Eigen's SparseLU to solve (only) the linear systems arising from step (1) of the heat method. Using BiCGSTAB (either Eigen's or our custom solver) results in an incorrect solution despite the iterative solver successfully converging, even under a relative residual tolerance of 10^{-15}. We observed that the small time-step of the heat method, $\Delta t = h^2$, causes difficulties for BiCGSTAB. The reason is that values far from the heat sources are often extremely close to zero. Tiny errors in these values are tolerated by BiCGSTAB, but lead to disastrously inaccurate gradients in step (2), and thus incorrect distances in step (3). Another option is to calculate smoothed distances (see [33, Section 3.3]) using larger time-steps $\Delta t = mh^2$ with $m \geq 100$; in this scenario BiCGSTAB encounters no problems. Our custom BiCGSTAB solver is nevertheless successfully used for step (3) of the heat method.

Figure 4.12 shows the geodesic distance to a single source point on the Dziuk surface, where our CPM-based approach (with $h = 0.0125$) is compared to exact polyhedral geodesics [104] and the mesh-based heat method. Implementations of the latter two methods are drawn from `geometry-central` [154]. All three approaches yield similar results.

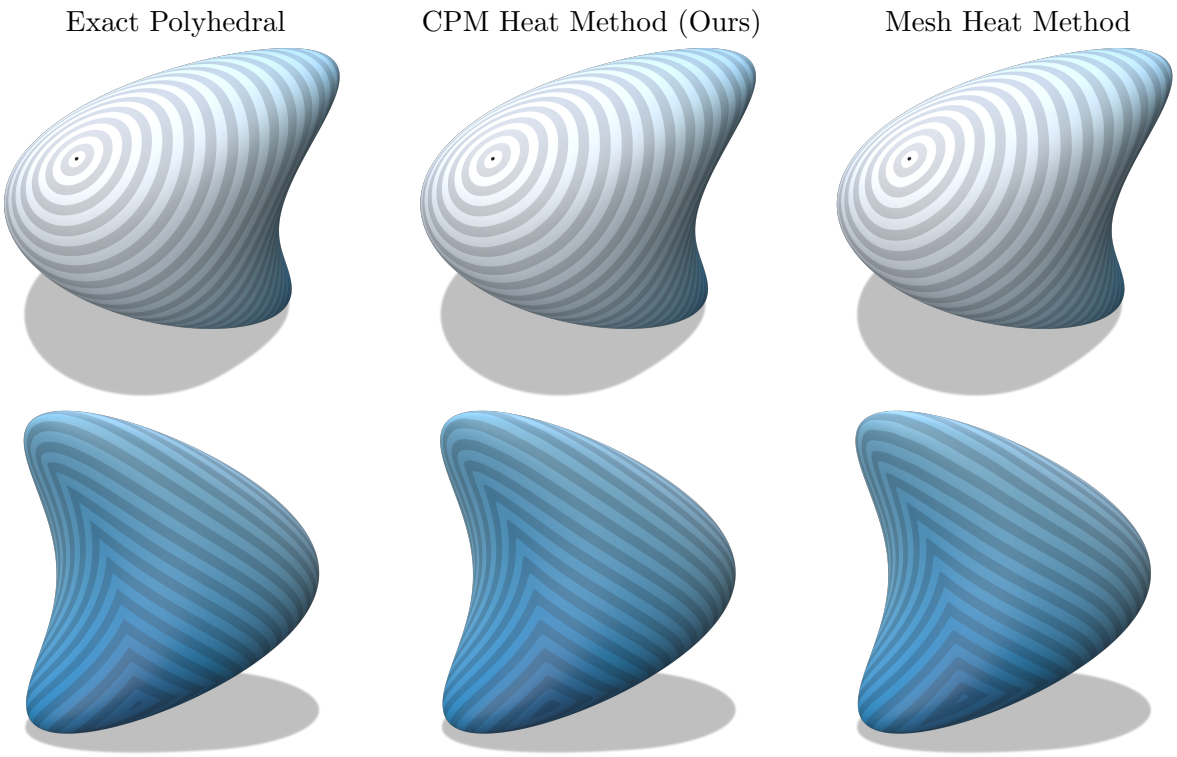


Figure 4.12: CPM vs. mesh-based methods for geodesic distances to a point on a triangulation of the Dziuk surface. The bottom row shows the rear view of the surface. Consistent results are observed.

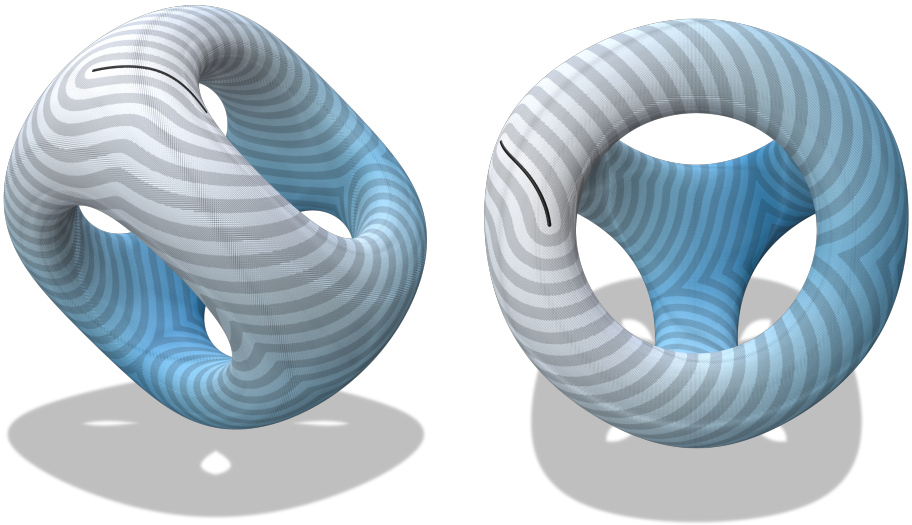


Figure 4.13: Geodesic distance to a polyline curve (black) visualized on the "DecoTetrahedron" level-set surface computed using CPM with $h = 0.025$. The closest points themselves are directly rendered.

For the example in Figure 4.12, closest points are computed from the same triangulation used in the exact polyhedral and mesh-based heat method. However, closest points can also be directly computed from the level-set Dziuk surface (as in Section 4.3.3). To our knowledge, the heat method has not been applied on level-set surfaces before.

We showcase the ability of our CPM to compute geodesic distance on general manifold representations. Figure 4.13 visualizes the geodesic distance to an open curve on the "DecoTetrahedron" [124] level-set surface,

$$\mathcal{S} = \left\{ \mathbf{x} \in \mathbb{R}^3 \;\middle|\; \sum_{i=1}^{3} \left((x_i - 2)^2 (x_i + 2)^2 - 10 x_i^2\right) + 3\left(x_1^2 x_2^2 + x_1^2 x_3^2 + x_2^2 x_3^2\right) + 6 x_1 x_2 x_3 = -22 \right\}.$$

$\mathcal{S}$ and $\mathcal{C}$ can also have mixed representations. For example, Figure 4.14 shows the geodesic distance (using $h = 0.00625$) to the trefoil knot (a.k.a. torus knot with $a = 2$ and $b = 3$, see (4.17)) on a torus with minor and major radii 1 and 2, respectively. The trefoil knot uses a parametric representation while the torus uses an analytical closest point representation.

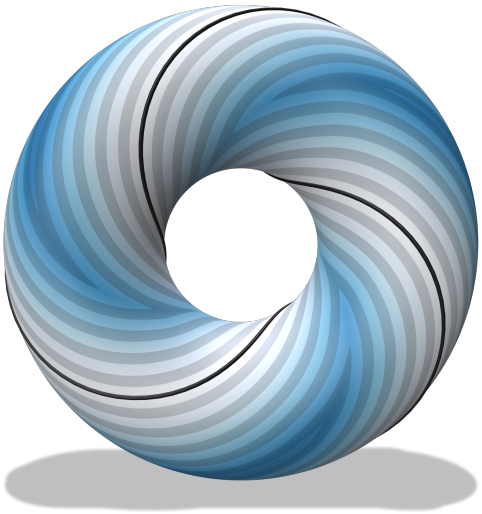


Figure 4.14: Geodesic distance to a parametric curve (black) on an analytical closest point surface.

4.4.3 Vector Field Design

Designing tangent vector fields on surfaces is useful in many applications including texture synthesis, non-photorealistic rendering, quad mesh generation, and fluid animation [36, 194]. One approach for vector field design involves the user specifying desired directions at a sparse set of surface locations, which are then used to construct the field over the entire surface. Adapting ideas from Turk [175] and Wei and Levoy [181], we interpret the user-specified directions as Dirichlet IBCs and use diffusion to obtain the vector field over the whole surface.

We iterate between heat flow of the vector field and projections onto the tangent space to obtain the tangent vector field over all of $\mathcal{S}$. Specifically, each iteration involves the following steps:

1. Perform heat flow independently for each component of $\mathbf{u} = [u^1, u^2, u^3]^T$ according to

$$\frac{\partial u^i}{\partial t} = \Delta_{\mathcal{S}} u^i, \quad \text{with} \quad \begin{cases} u^i = g^i, \text{ or} \\ \nabla_{\mathcal{S}} u^i \cdot \mathbf{b}_{\mathcal{C}} = 0, \end{cases} \quad \text{on } \mathcal{C},$$

   starting from the vector field after the previous iteration.

2. Project $\mathbf{u}(\mathbf{x}_j)$ onto the tangent space of $\mathcal{S}$ using $\mathbf{n}_{\mathcal{S}}$ at $\text{cp}_{\mathcal{S}}(\mathbf{x}_j)$

$$\mathbf{u}(\mathbf{x}_j) = \left(\mathbf{I} - \mathbf{n}_{\mathcal{S}}{\mathbf{n}_{\mathcal{S}}}^T\right)\mathbf{u}(\mathbf{x}_j).$$

One time-step of heat flow is performed on each iteration using implicit Euler with $\Delta t = 0.1h$. A total of 10 iterations are used for all examples. The vector field for the first iteration consists of zero vectors unless the direction is specified by an IBC.

Dirichlet IBCs $\mathbf{g} = [g^1, g^2, g^3]^T$ can be specified at points or curves. For point Dirichlet IBCs the direction of $\mathbf{g}$ is any direction in the tangent space of $\mathcal{S}$. Dirichlet IBCs on curves could also specify any direction in the tangent space of $\mathcal{S}$, but designing vector fields is more intuitive when $\mathbf{g}$ is the unit tangent direction $\mathbf{t}_{\mathcal{C}}$ along $\mathcal{C}$. Zero-Neumann IBCs are also used within our framework to block the vector field from diffusing across $\mathcal{C}$.

Figure 4.15 compares our vector field design approach with the vector heat method [155]. Three directions are specified as Dirichlet IBCs at different locations on the Dziuk surface. Our CPM approach uses $h = 0.025$ and the level-set representation of the Dziuk surface for CP queries. The vector heat method implemented in `geometry-central` [154] is used with a triangulation of the Dziuk surface. Figure 4.15 (right column) shows the dot product of the resultant vectors from our approach and the vector heat method. Both methods produce similar vectors except near the cut locus of the IBC points. Our method provides a smoother transition across the cut locus compared to the vector heat method, which may be beneficial or disadvantageous depending on the application. Sources and sinks can also appear with our method that are not seen with the vector heat method (Figure 4.15, bottom row). The core difference between the methods is that ours applies component-wise diffusion of the vectors, while the vector heat method couples the vector components using the connection Laplacian. Interesting future work would determine how to handle the connection Laplacian with CPM.

Figure 4.16 shows an example of a vector field designed on the Möbius strip using $h = 0.0064$. The Möbius strip is actually a triangulated surface in this example, although its parametric form could be used instead (see [89]). Zero-Neumann exterior BCs are imposed automatically by CPM with first-order accuracy on the geometric boundary. This example shows the ability of our approach to handle open and nonorientable surfaces. There are four points and two curves specifying the IBCs in Figure 4.16. A circular closed curve demonstrates that vortices can be created. The other curve on the Möbius strip enforces a zero-Neumann IBC that blocks direction changes in the vector field (see Figure 4.16 zoom).

Figure 4.17 shows another example on a parametric surface of revolution (with $h = 0.025$), which is constructed by revolving the planar parametric curve (4.18) with $c = \frac{1}{2}$

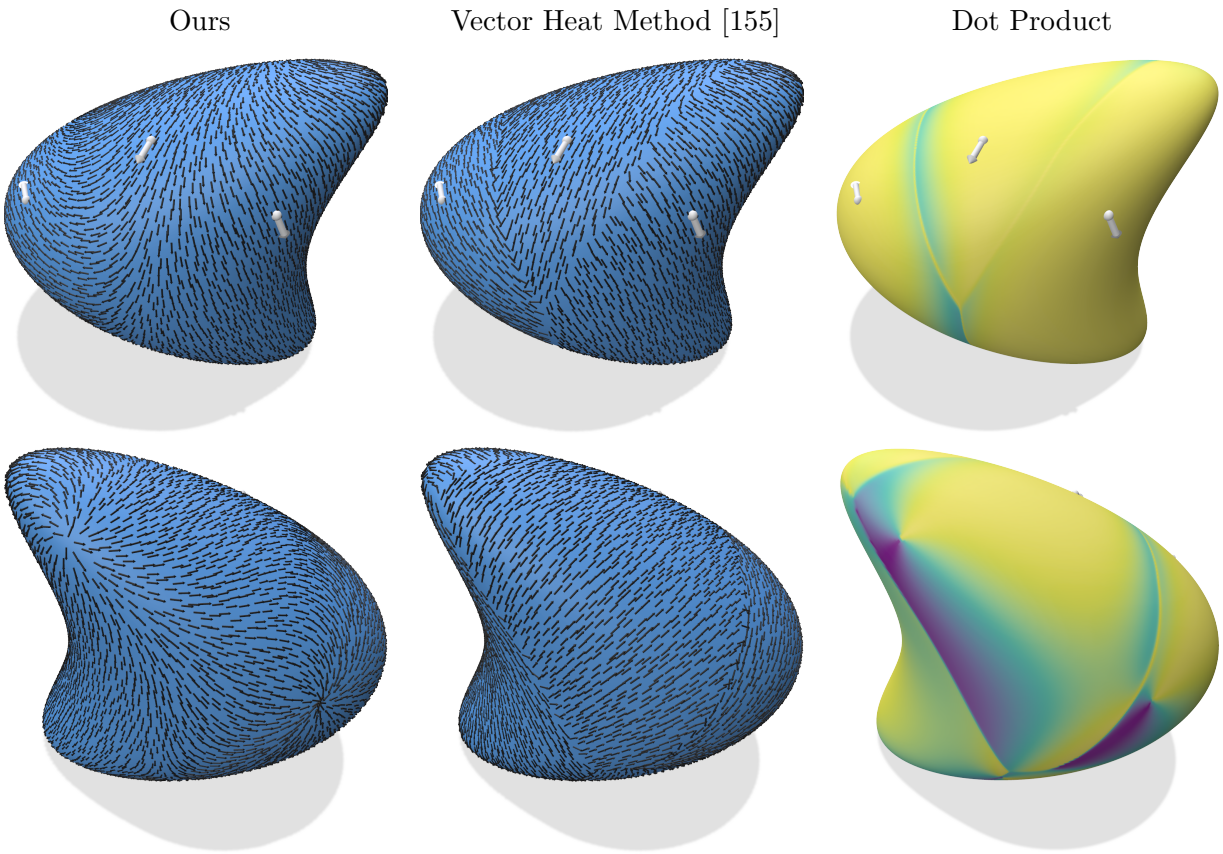


Figure 4.15: Our CPM approach vs. the vector heat method when designing a vector field using three user-prescribed directions (white) on the Dziuk surface. The resultant vector field (black) is consistent away from the cut locus of the IBC points, since the dot product between the vectors produced by each method is close to 1 (colour scale goes from -1 for purple to 1 for yellow).

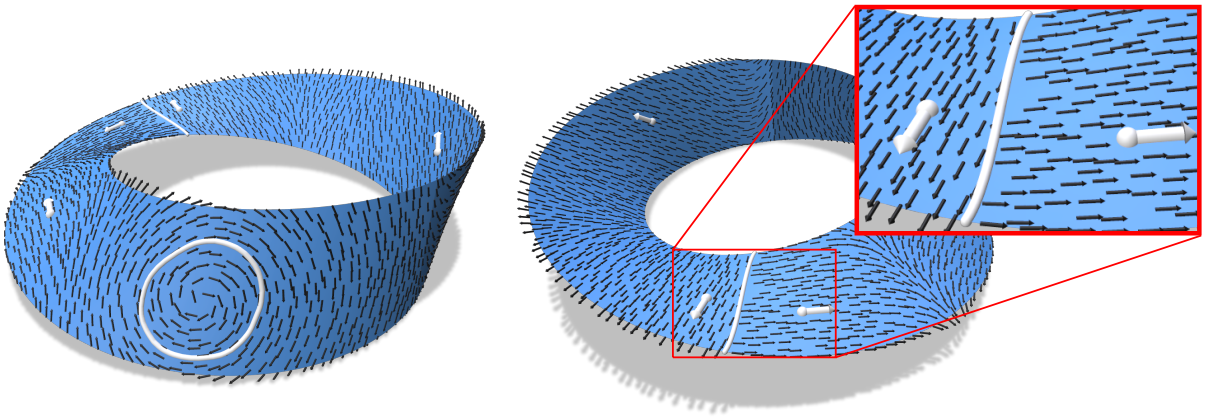


Figure 4.16: Vector field design on a triangulation of a Möbius strip, which is an open and nonorientable surface.

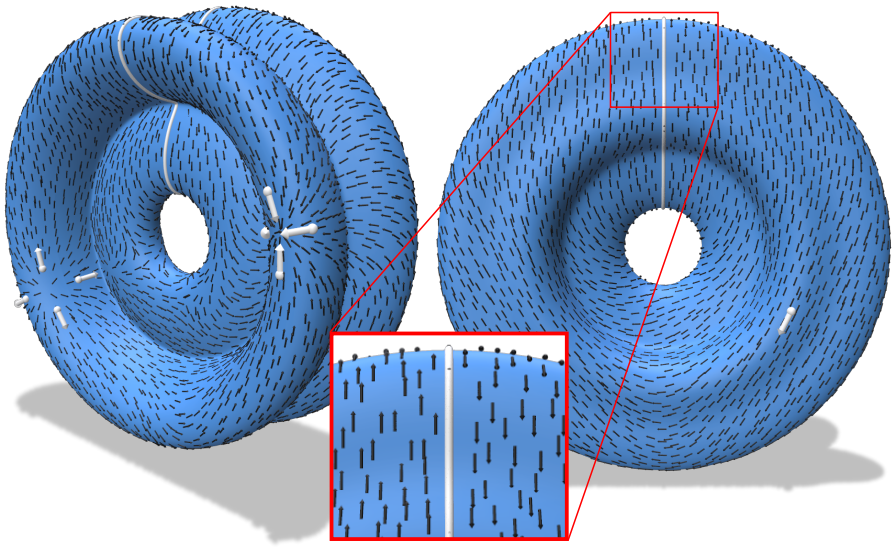


Figure 4.17: Vector field design on a parametric surface of revolution, with Dirichlet IBCs on a parametric curve and points shown in white.

around the z-axis. All IBCs in this example are Dirichlet IBCs. Sinks and sources in the vector field are created with four Dirichlet point IBCs. The curve IBC is a two-sided Dirichlet IBC that flips the direction of the vector field across $\mathcal{C}$ (see Figure 4.17 zoom).

A final vector field design example, on the Lucy surface, is given in Figure 4.18. A point cloud representation of the Lucy surface (vertices of a mesh [171] with $\sim$1 million vertices) is used and the closest point function is defined to return the nearest neighbour; for dense enough point clouds this suffices. For less dense point clouds a smoother closest point function is required, for example using a moving-least-squares based projection method [83, 192]. Nevertheless, the variable point density (i.e., higher density on head, wings, hands, and feet) of the Lucy point cloud in Figure 4.18 (left) does not present any issue in this example.

4.4.4 Harmonic Maps

A map between two manifolds, $\mathcal{S}_1$ and $\mathcal{S}_2$, matches locations on $\mathcal{S}_1$ with locations on $\mathcal{S}_2$. The map can be used to analyse differences between $\mathcal{S}_1$ and $\mathcal{S}_2$ or to transfer data from one manifold to the other. Harmonic maps are a specific type of map that appears in numerous domains, e.g., mathematical physics [14] and medical imaging [159, 160]. In computer graphics, harmonic maps can be used for many applications such as texture transfer, quad mesh transfer, and interpolating intermediate poses from key-frames of a character [46].

King and Ruuth [72] considered applying CPM to compute harmonic maps $\mathbf{u}(\mathbf{y}) : \mathcal{S}_1 \to \mathcal{S}_2$. Adapting their approach, we compute the harmonic map using the gradient descent flow

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= \Pi_{T_{\mathbf{u}}\mathcal{S}_2}(\Delta_{\mathcal{S}_1}\mathbf{u}), \\ \mathbf{u}(\mathbf{y},0) &= \mathbf{f}(\mathbf{y}), \\ \mathbf{u}(\mathbf{y},t) &= \mathbf{g}(\mathbf{y}), \text{ for } \mathbf{y} \in \mathcal{C}_1, \end{aligned} \tag{4.20}$$

where $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$ is the projection operator at the point $\mathbf{u}$ onto the tangent space of $\mathcal{S}_2$. The vector $\Delta_{\mathcal{S}_1}\mathbf{u}$ is defined componentwise, i.e., $\Delta_{\mathcal{S}_1}\mathbf{u} = [\Delta_{\mathcal{S}_1}u^1, \Delta_{\mathcal{S}_1}u^2, \Delta_{\mathcal{S}_1}u^3]^T$. The $\mathbf{f}(\mathbf{y})$ and $\mathbf{g}(\mathbf{y})$ are the initial map (from $\mathcal{S}_1$ to $\mathcal{S}_2$) and the landmark map (from $\mathcal{C}_1 \subset \mathcal{S}_1$ to $\mathcal{C}_2 \subset \mathcal{S}_2$), respectively. The subsets $\mathcal{C}_1$ and $\mathcal{C}_2$ can be landmark points or curves on $\mathcal{S}_1$ and $\mathcal{S}_2$ that are enforced to match using our new Dirichlet IBC treatment; such IBCs were *not* considered by King and Ruuth [72].

An operator splitting approach was used by King and Ruuth [72], which allows (4.20) to be solved with a PDE on $\mathcal{S}_1$ alone. Specifically, one time-step consists of the following:

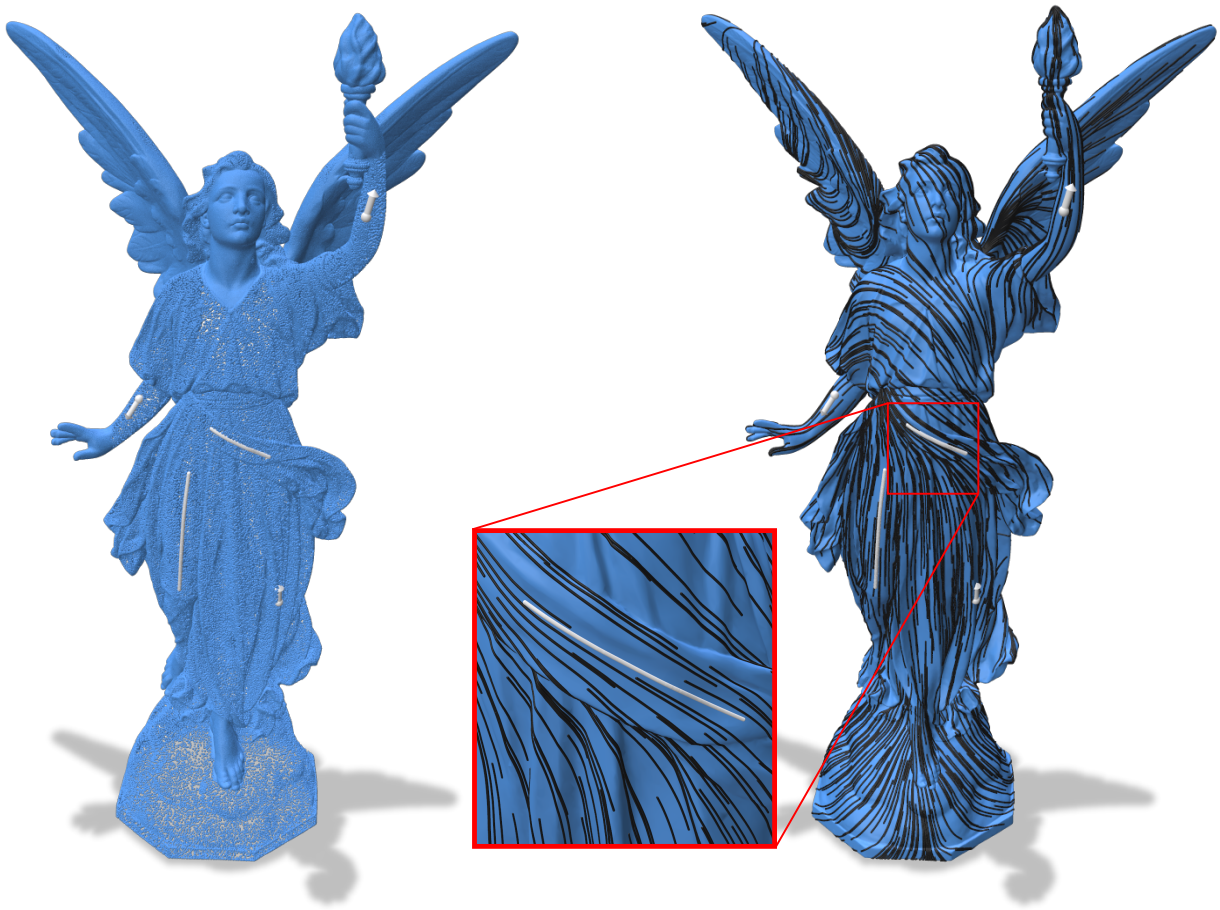


Figure 4.18: Vector field design on a point cloud surface (left), with Dirichlet IBCs on polyline curves and points shown in white. The resulting vector field is visualized with flow lines on a triangulation of the point cloud (right).

1. Solve (4.20) *without the* $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$ *term* using CPM on $\Omega(\mathcal{S}_1)$ with $\Omega(\mathcal{C}_1)$ to enforce the IBC.

2. Project the solution from (1) onto $\mathcal{S}_2$.

Denote the solution from step (1) at $\mathbf{x}_i \in \Omega(\mathcal{S}_1)$ and time-step k by $\mathbf{v}_i^k$. The projection in step (2) simply moves $\mathbf{v}_i^k$ to its closest point on $\mathcal{S}_2$ by setting $\mathbf{u}_i^k = \mathrm{cp}_{\mathcal{S}_2}(\mathbf{v}_i^k)$. One time-step of explicit Euler is used for step (1) with $\Delta t = 0.1h^2$ starting from $\mathbf{u}^{k-1}$. More information on harmonic maps and the algorithm of King and Ruuth [72] is given in Section 6.1.

To perform the above gradient descent flow a valid initial map $\mathbf{u}^0$ is needed to start from. Generating such initial maps in the general case has not yet been addressed for CPM [72]. Approaches based on geodesic distance to landmark curves/points $\mathcal{C}_1$, $\mathcal{C}_2$ could potentially be adapted [46, 159]. However, for our illustrative example of incorporating IBCs while computing harmonic maps, we opt for a simple (but restrictive) initial map construction. The surface $\mathcal{S}_1$ is given by a triangulation and deformed to create $\mathcal{S}_2$ while maintaining the same vertex connectivity. Therefore, the barycentric coordinates of each triangle can be used to initially map any point on $\mathcal{S}_1$ to a point on $\mathcal{S}_2$.

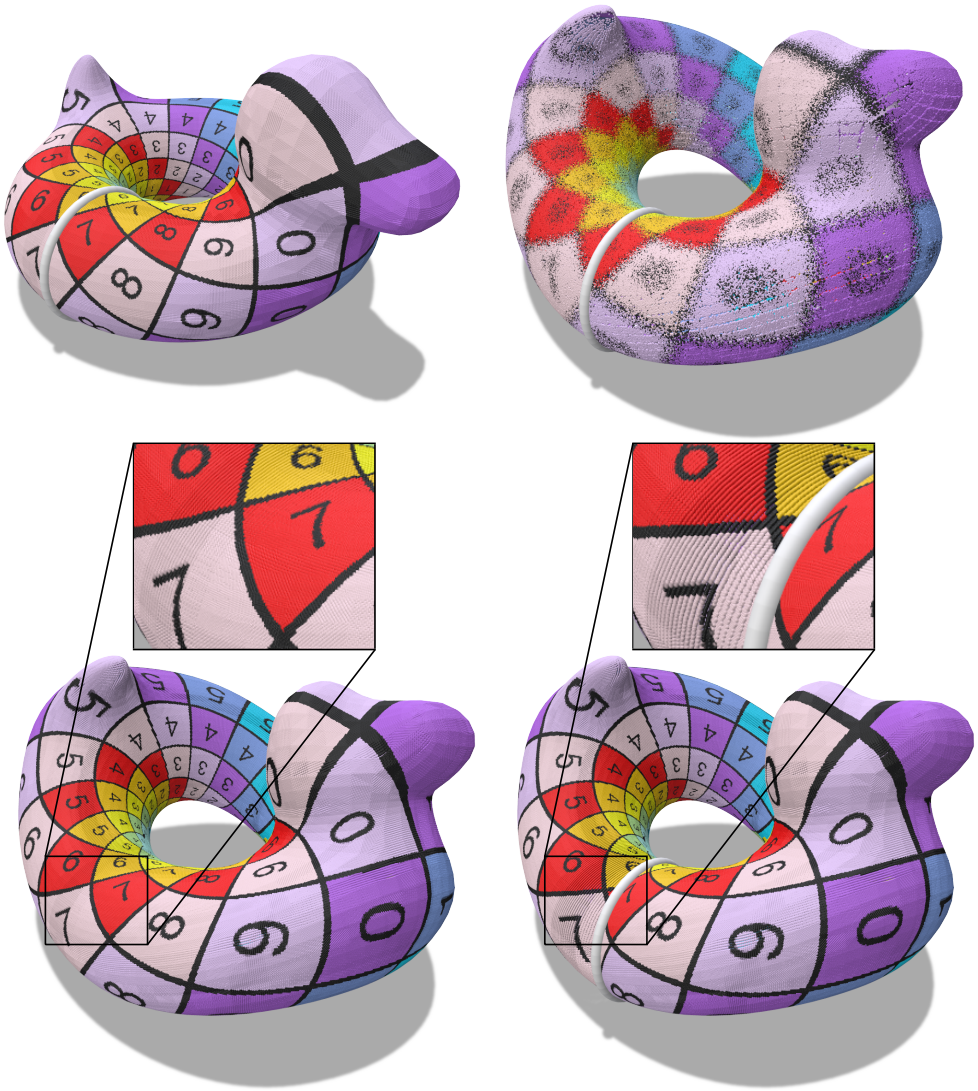
Figure 4.19 shows an example of computing harmonic maps from the Bob [32] surface $\mathcal{S}_1$ to its deformed version $\mathcal{S}_2$. Grid spacing $h = 0.00663$ is used for $\Omega(\mathcal{S}_1)$. The surfaces are visualized as point clouds. $\mathcal{S}_1$ is visualized with the set of closest points of grid points in $\Omega(\mathcal{S}_1)$. Each point in the point cloud for $\mathcal{S}_1$ has a corresponding point location on $\mathcal{S}_2$ given through the mapping $\mathbf{u}$. A texture is added to the surface of $\mathcal{S}_1$ and transferred to $\mathcal{S}_2$ through the mapping $\mathbf{u}$.

To emphasize the effect of computing the harmonic map, noise is added to the initial map (see Figure 4.19 (b)) before performing the gradient descent flow. The gradient descent flow is evolved to steady state using 1500 and 200 time steps with and without the IBC in Figure 4.19 (d) and (c), respectively. The harmonic map with a Dirichlet IBC stretches on one side of $\mathcal{C}_2$ and compresses on the other side to satisfy both the PDE and IBC. Comparing the zoom of Figure 4.19 (c) and (d), the point cloud density in (d) is more sparse on one side of $\mathcal{C}_2$ than in (c) due to the stretching of the map, leaving visual gaps between points in the cloud. The distortion is expected unless the IBC map $\mathbf{g}$ is a harmonic map itself.

4.4.5 Reaction-Diffusion Textures

Much research in geometry processing has focused on Poisson and diffusion problems. There are however applications that require solving more general PDEs, e.g., reaction-

(a) $\mathcal{S}_1$ with Texture (b) Initial Map onto $\mathcal{S}_2$



(c) Harmonic Map onto $\mathcal{S}_2$ w/o IBCs (d) Harmonic Map onto $\mathcal{S}_2$ w/ IBCs

Figure 4.19: Maps from $\mathcal{S}_1$ to $\mathcal{S}_2$ with a texture for visualizing the mapping. Landmark curves (Dirichlet IBCs) $\mathcal{C}_1$ and $\mathcal{C}_2$ are shown in white. (a) $\mathcal{S}_1$ with texture. (b) $\mathcal{S}_2$ with texture from a noisy initial map. (c) $\mathcal{S}_2$ with a CPM harmonic mapped texture without IBCs. (d) $\mathcal{S}_2$ with a harmonic mapped texture using our CPM approach satisfying the IBCs. The surfaces are displayed as point clouds. The $\text{cp}_{\mathcal{S}_1}$ and $\text{cp}_{\mathcal{S}_2}$ are computed from triangulations, while $\text{cp}_{\mathcal{C}_1}$ and $\text{cp}_{\mathcal{C}_2}$ are computed from polylines.

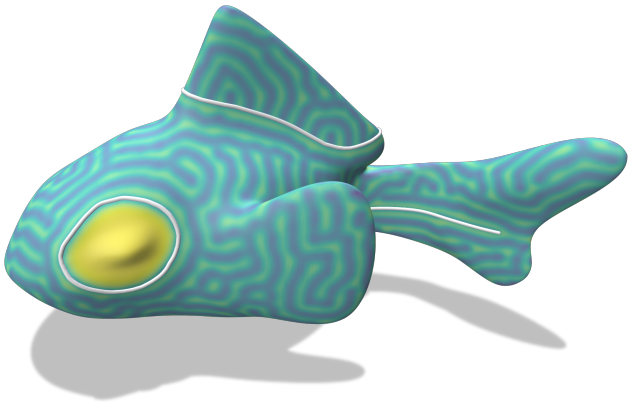


Figure 4.20: Reaction-diffusion texture on a fish surface with zero Dirichlet IBCs around the eye and on the tail. A two-sided zero Dirichlet-Neumann IBC is imposed on the dorsal fin. The surface is coloured yellow for high concentrations of reactant u and purple for low concentrations. The $\text{cp}_{\mathcal{S}}$ are computed from a triangulation, while the $\text{cp}_{\mathcal{C}}$ are computed from polylines.

diffusion textures [174]. Reaction-diffusion textures involve solving coupled equations on surfaces. These PDEs can form patterns from random initial conditions and have been solved on meshes [174], level sets [18], and closest point surfaces [90]. Here we impose IBCs to control regions of the texture, emphasizing the generality of CPM, and our novel boundary condition treatment, with respect to PDE type.

The Gray-Scott model [125]

$$\begin{cases} \frac{\partial u}{\partial t} = \mu_u \Delta_{\mathcal{S}} u - uv^2 + F(1-u), \\ \frac{\partial v}{\partial t} = \mu_v \Delta_{\mathcal{S}} v + uv^2 - (F+k)v, \end{cases} \tag{4.21}$$

with

$$\begin{cases} u = g_u \text{ or } \nabla_{\mathcal{S}} u \cdot \mathbf{b}_{\mathcal{C}} = 0, \\ v = g_v \text{ or } \nabla_{\mathcal{S}} v \cdot \mathbf{b}_{\mathcal{C}} = 0, \end{cases} \quad \text{on } \mathcal{C}, \tag{4.22}$$

is solved with CPM. Figure 4.20 shows u on a fish [31] for a set of IBCs. The constants $\mu_u = 1.11 \times 10^{-5}$, $\mu_v = \mu_u/3$, $F = 0.054$, $k = 0.063$ are used with forward Euler time-stepping until $t = 10,000$ with $\Delta t = 0.9$ and $h = 0.01$. The initial condition is taken as $u = 1 - p$, $v = p/2$ where p is given by small random perturbations around

$$\frac{1}{2} e^{100(z-0.1)^2} + \frac{1}{2}.$$

Zero Dirichlet IBCs allow stripes to be placed around the dorsal fin and tail. The upper side of the dorsal fin IBC is a zero Neumann IBC, which causes the pattern to intersect perpendicular to the IBC curve. A closed (zero Dirichlet) IBC curve allows for control of concentrations of the reactants u and v in the eye.

4.5 Summary and Future Work

This chapter extended CPM to solve manifold PDEs with interior boundary conditions (Dirichlet and zero-Neumann) while obtaining up to second-order accuracy. The ability to enforce IBCs enabled CPM to be applied to many PDE-based geometry processing tasks and applications that were not previously possible. We showed that our first-order version has a lower error constant compared to that of Auer et al. [7]. Furthermore, the method Auer et al. [7] proposed only applies to Dirichlet IBCs, but ours handles both Dirichlet and Neumann. We also introduced the nearest point approach (Section 4.2.8) for Dirichlet IBCs, which is less accurate than our main approach for IBC treatment, but is more accurate than the method of Auer et al. [7]. This nearest point approach may be preferred when accuracy is less important than avoiding the complexities of introducing duplicate DOFs.

CPM work to date has only addressed Dirichlet and zero-Neumann (exterior) BCs. Macdonald et al. [90] solved a surface-to-bulk coupled PDE with Robin BCs on the boundary of the bulk (but $\mathcal{S}$ was closed, i.e., $\partial\mathcal{S} = \emptyset$). Extending CPM to impose inhomogeneous-Neumann, Robin, and other types of BCs is an important area of future work. Fortunately, the interior BC framework developed here directly generalizes existing CPM approaches for exterior BCs; therefore, our work likely makes any future extensions of CPM for other exterior BC types immediately applicable to interior BCs as well.

Third-order and higher (exterior and interior) BCs are also important for higher-order PDE discretizations. CPM itself extends naturally to higher order, but CPM with higher-order exterior BCs has not yet been explored. Macdonald et al. [89] pointed out that a replacement for $\overline{\text{cp}}_{\mathcal{S}}$ is required to incorporate the curvature of $\mathcal{S}$ near $\partial\mathcal{S}$. For higher-order interior BCs an improved $\mathcal{S}_{\perp}$ crossing test (4.8), involving curvatures of $\mathcal{S}$ near $\mathcal{C}$, is likely also needed.

Chapter 5

Monte Carlo Closest Point Method

For *volumetric* PDEs, Monte Carlo methods have recently garnered significant attention in the graphics community due to their unique advantages over traditional discretization-based PDE solvers, including the ability to estimate solution values in a pointwise, spatial discretization-free manner. One such method is the *walk-on-spheres* (WoS) method [110] introduced to graphics by Sawhney and Crane [140]. They primarily focused on the (constant coefficient) Poisson and screened-Poisson equations in a volumetric domain, and follow-up work likewise emphasizes volumetric problems. In this chapter, we consider instead the problem of solving *surface PDEs*.

Sawhney et al. [141] proposed an extension of WoS to support second-order linear elliptic PDEs with spatially varying coefficients, and as one application, they demonstrated a method to solve the Laplace equation on a 2D surface embedded in 3D. However, this approach requires that the conformal parametrization of the surface be readily available, limiting the method's applicability.

We propose a simpler generalization of WoS for surface PDEs, the *projected walk-on-spheres* (PWoS) method, which only assumes the availability of a closest surface point query and an unoriented surface normal direction query. PWoS supports Dirichlet boundary conditions and inherits the advantages of WoS: PWoS is a pointwise, discretization-free Monte Carlo method. Since our method does not require the meshing of the surface, it is particularly advantageous over traditional approaches, such as the finite element method, when the computation can be localized and when the surface is given as an implicit representation, such as a signed distance function. The resulting solution is also free of mesh-dependent discretization artifacts, such as from linear interpolation, as we show in Figure 5.1. Compared to WoS, which performs walks on spheres inside the domain, PWoS

performs walks on spheres inside a Cartesian embedding neighbourhood domain around the surface. After each step of the walk, it *projects* the sampled point on the sphere onto the surface. We motivate this simple modification to the original WoS through its connection to the *closest point method* [138, 96].

To confirm PWoS's accuracy, we perform convergence studies of the method applied to the surface Poisson and screened-Poisson equations. Finally, we demonstrate its use on two representative graphics applications: diffusion curves and geodesic distance computation.

5.1 Background

This section briefly reviews the two core mathematical ideas on which our method is based.

5.1.1 Walk-on-Spheres

WoS solves volumetric PDEs such as the Poisson equation $\Delta u = f$ over a Cartesian domain in $\mathbb{R}^d$, with Dirichlet boundary conditions. Consider a d-ball $B_r(\mathbf{x})$, centred at $\mathbf{x}$ with radius r, fully contained within the domain. The integral equation

$$u(\mathbf{x}) = \frac{1}{|\partial B_r(\mathbf{x})|} \int_{\partial B_r(\mathbf{x})} u(\mathbf{y}) \, \mathrm{d}\mathbf{y} + \int_{B_r(\mathbf{x})} f(\mathbf{z}) G(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z} \tag{5.1}$$

holds for the Poisson equation $\Delta u = f$ in general, where $|\partial B_r(\mathbf{x})|$ denotes the surface area of the sphere that bounds the ball $B_r(\mathbf{x})$ and G denotes the Green's function for the Poisson equation on $B_r(\mathbf{x})$ [140]. On the right-hand side, the first term is a boundary integral over the $(d-1)$-sphere, and the second term is a volume integral over the d-ball. If we perform Monte Carlo integration of the first term by uniformly sampling a point on the sphere and of the second term by sampling N_V points $\mathbf{z}_i$ inside the ball with probability density function (PDF) $p(\mathbf{z}_i)$, we get the recursive relationship used in WoS:

$$\widehat{u}(\mathbf{x}) = \widehat{u}(\mathbf{y}) + \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{G(\mathbf{x}, \mathbf{z}_i) f(\mathbf{z}_i)}{p(\mathbf{z}_i)}, \tag{5.2}$$

where the hat notation indicates that a variable is a Monte Carlo estimate. The first term on the right-hand side is also a Monte Carlo estimate because the solution $u(\mathbf{y})$ is unknown at point $\mathbf{y}$ in general. At each recursion step, WoS applies this recursive relationship to the largest ball inside the domain bounded by Dirichlet boundaries. It terminates the

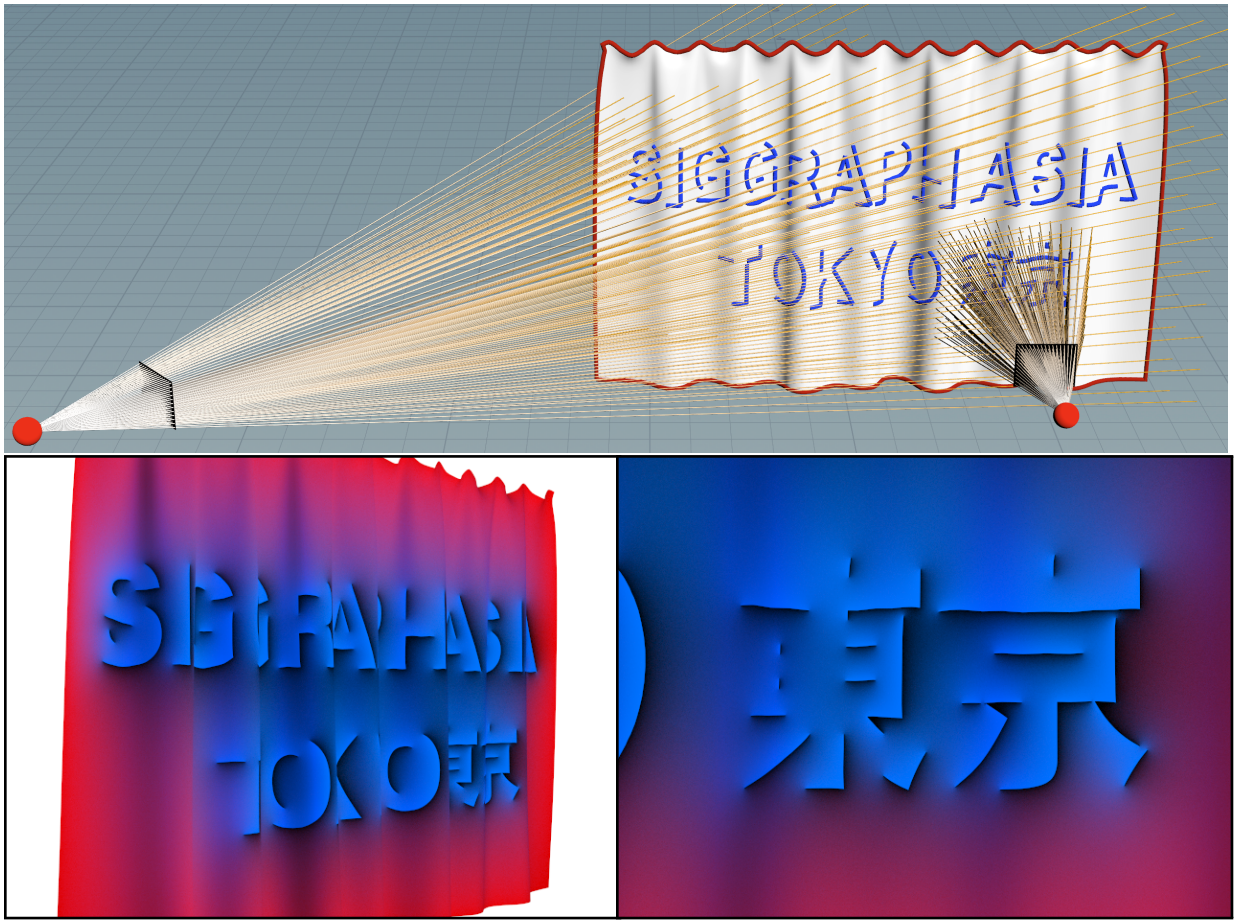


Figure 5.1: View-dependent diffusion curves with PWoS. Using our method, we solve the Laplace equation on a curved surface in a view-dependent manner. The pointwise and discretization-free nature of PWoS allows for the evaluation of the colours only at visible points where the object colour is required by a shading algorithm with stochastic pixel-filtering.

recursion when the sample point $\mathbf{x}$ during the recursion falls within a small distance ϵ of the domain boundary, by using the known solution at the closest boundary point $\overline{\mathbf{x}}$ as the solution estimate: $\widehat{u}(\mathbf{x}) = u(\overline{\mathbf{x}})$. WoS thereby estimates the solution at each evaluation point independently, offering intrinsic parallelism. Our method generalizes WoS, originally proposed for volumetric PDEs, by incorporating the closest point extension theory of CPM.

5.1.2 Surface PDEs and Closest Point Extension

For a full review of CPM for surface PDEs and the CP extension see Chapter 2. In this section, we briefly recall the important CPM theory used for our Monte Carlo approach (for the reader's ease). Consider the Poisson equation defined on a surface $\mathcal{S}$

$$\Delta_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{y}) = f_{\mathcal{S}}(\mathbf{y}), \quad \mathbf{y} \in \mathcal{S}. \tag{5.3}$$

For convenience, we will use the word surface to refer to any nonzero codimension object in $\mathbb{R}^d$, including mixed-codimension objects. In general, the mapping $\text{cp}_{\mathcal{S}}(\mathbf{x})$ may not be unique: there may exist more than one closest point for a given $\mathbf{x}$. The neighbourhood $\mathcal{N}(\mathcal{S})$ where the closest point function is unique is a spatially adaptive tube where $r_{\mathcal{N}(\mathcal{S})}$ is a function of $\mathbf{x}$ such that

$$r_{\mathcal{N}(\mathcal{S})}(\mathbf{x}) < \text{LFS}(\text{cp}_{\mathcal{S}}(\mathbf{x})),$$

where $\text{LFS}(\mathbf{y})$ is the local feature size at point $\mathbf{y} \in \mathcal{S}$. The local feature size is defined as the minimum Euclidean distance from $\mathbf{y}$ to the medial axis $\text{med}(\mathcal{S})$ [4]. The medial axis $\text{med}(\mathcal{S})$ is the set of points in $\mathbb{R}^d$ where there is more than one closest point. Note that when $\mathcal{S}$ is a closed surface the definition of the medial axis that we use contains both the interior part that is bounded by $\mathcal{S}$ and the exterior part that lies outside the bounded domain.

Within $\mathcal{N}(\mathcal{S})$, or a subset of it, surface differential operators can be replaced by Cartesian differential operators with *CP extensions* E. The embedding PDE on $\mathcal{N}(\mathcal{S})$ for (5.3) is

$$\Delta[Eu_{\mathcal{S}}](\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x}), \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}), \tag{5.4}$$

where $f(\mathbf{x}) = Ef_{\mathcal{S}}(\mathbf{x})$. Recall from Section 2.2.3 that $g(\mathbf{x})$ is a function that ensures the embedding PDE is well posed, since $\Delta[Eu_{\mathcal{S}}]$ is not constant in the normal direction of $\mathcal{S}$ but f is. The function $g(\mathbf{x})$ is nonzero for $\mathbf{x} \in \mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$ and $g|_{\mathcal{S}} = 0$ to ensure (5.4) is consistent with the surface PDE (5.3) on $\mathcal{S}$. Any function g with these conditions has the form $g(\mathbf{x}) = \gamma(v(\mathbf{x}) - Ev(\mathbf{x}))$, where $\gamma \in \mathbb{R}$ and $\gamma \neq 0$.

The Macdonald-Brandman-Ruuth [89] approach (see Section 2.2.3) takes $v|_{\mathcal{S}} = u_{\mathcal{S}}$ to allow (5.4) to be written as an equation in one unknown, $v(\mathbf{x})$, since $Eu_{\mathcal{S}} = Ev$ (but importantly $v \neq Ev$ except on $\mathcal{S}$). We instead do not restrict $v|_{\mathcal{S}}$ to be $u_{\mathcal{S}}$ and interpret $g(\mathbf{x})$ as a modification to the source term $f(\mathbf{x})$, then solve for the unknown solution $u(\mathbf{x}) = Eu_{\mathcal{S}}$ in (5.4). As proven by von Glehn et al. [177, Section 3.2], $u(\mathbf{x}) = Eu(\mathbf{x})$ since $u(\mathbf{x})$ is the extension of a surface function. The property that $u(\mathbf{x}) = Eu(x) = u(\mathrm{cp}_{\mathcal{S}}(\mathbf{x}))$ allows our projection step during the walk in PWoS, detailed in Section 5.2.

We show through our numerical examples that taking $g(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{N}(\mathcal{S})$, provides qualitatively correct results for graphics applications and quantitatively convergent results in most examples. However, the choice of $g(\mathbf{x}) = 0$ causes (5.4) to be ill-posed as discussed above, and we observe some bias in some convergence studies in Section 5.3 when f is complex or LFS is small. Interesting future work would involve constructing a more accurate $g(\mathbf{x})$ function to improve convergence.

Chen and Macdonald [24] (Section 2.2.4) show an alternative approach to obtain a solution in the embedding space that satisfies $u = Eu$, but it involves modifying the embedding PDE in a way that is inconsistent with WoS. They extend the Cartesian Laplacian, i.e., they use $E[\Delta v]$ instead of $\Delta[Ev]$ in the embedding PDE. Applying the extension after the Laplacian would invalidate the use of the mean value theorem in WoS since it is no longer the Cartesian Laplacian of a function.

In the traditional CPM, one solves the embedding PDE inside a narrow tubular subset of $\mathcal{N}(\mathcal{S})$ that is within a constant distance to $\mathcal{S}$. For the typical grid-based variant, the tubular subset is spatially discretized with a grid of uniform spacing. Interpolation and finite differences are applied on the grid to approximate the CP extension and the spatial Cartesian differential operators, respectively, and then the resulting linear system is solved. Other variants of CPM [127, 132, 25] also require some discretization within $\mathcal{N}(\mathcal{S})$. Thus, while traditional CPM is agnostic to the specific surface representation, it still discretizes the embedding space and solves a globally coupled system. Moreover, imposing exterior or interior boundary conditions requires tedious grid operations (see Chapter 4). By contrast, we develop a spatial discretization-free, pointwise Monte Carlo estimator for surface PDEs by incorporating the closest point extension concept into WoS.

When there are Dirichlet boundaries $\mathcal{C} \subset \mathcal{S}$, on which the solution $u_{\mathcal{S}}$ is given, one can geometrically extend the boundary itself out into the embedding space in the normal directions, assigning it the same boundary value in accordance with the CP extension. Note that such boundaries need not coincide with the geometric boundaries of the surface itself. In the context of grid-based CPM, Chapter 4 discusses how to impose such boundary conditions by duplicating degrees of freedom near the extended boundary. In our work,

we devise a method that uses only the closest point function $\text{cp}_{\mathcal{C}}(\mathbf{x})$ to the (pre-extension) boundary $\mathcal{C}$, without the need to construct the extended boundary geometry or perform any complex duplication of degrees of freedom.

5.2 Method

While our algorithm is generalizable to other configurations, we describe our method for the case when $\mathcal{S}$ is embedded in $\mathbb{R}^3$ and $\dim(\mathcal{S}) = 1, 2$. Recall that we use the word surface to refer both to "surfaces" with $\dim(\mathcal{S}) = 1$ (curves) as well as $\dim(\mathcal{S}) = 2$ surfaces. We also allow mixed codimension where parts of the surface have $\dim(\mathcal{S}) = 1$ and the rest of the surface has $\dim(\mathcal{S}) = 2$.

We assume that we can query the closest point function $\text{cp}_{\mathcal{S}}(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^3$. Additionally, for surfaces with $\dim(\mathcal{S}) = 2$, we assume that we can query the unoriented normal direction of the surface $\mathbf{n}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{S}$. For surfaces with $\dim(\mathcal{S}) = 1$, we assume that we can query the tangential direction of the surface $\mathbf{t}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{S}$. These assumptions are valid for common surface representations, including, but not limited to, polygonal meshes, oriented point clouds, and implicit functions.

Additionally, we assume the Dirichlet boundaries $\mathcal{C}$ have a lower dimension than the dimension of $\mathcal{S}$ and support the closest point query $\text{cp}_{\mathcal{C}}$. When solving a two-sided boundary value problem for boundaries with $\dim(\mathcal{C}) = 1$, we also assume that we can query the tangent direction of $\mathcal{C}$. The theory discussed in Section 5.1.2 is based on the assumption that $\mathcal{S}$ is smooth; in practice, we observe that applying our technique on discretized surfaces with sharp features behaves well as we show in Section 5.3.

The core idea of our method is to apply the WoS recursive relationship within $\mathcal{N}(\mathcal{S})$ while utilizing the CP extension constraint that $u(\mathbf{x}) = u(\text{cp}_{\mathcal{S}}(\mathbf{x}))$. To do so, we modify the walk process to use spheres contained within $\mathcal{N}(\mathcal{S})$ and to *project* the walk position at each recursion step, as illustrated in Figure 5.2.

The problem we solve is the embedding PDE $\Delta u(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$ within $\mathcal{N}(\mathcal{S})$. The Monte Carlo estimate of (5.2) holds by assuming $g(\mathbf{x}) = 0$ because the embedding PDE is defined with the Cartesian differential operator. To estimate the surface PDE's solution at point $\mathbf{x} \in \mathcal{S}$, we consider a 3D ball centred at $\mathbf{x}$ and fully contained inside $\mathcal{N}(\mathcal{S})$. Theoretically, it should be the largest ball fully contained inside $\mathcal{N}(\mathcal{S})$ that does not cross the extended Dirichlet boundaries $\mathcal{C}$, to minimize the number of steps needed to reach the boundary. We determine the radius of such a ball by taking the minimum of a conservative

Algorithm 2: Projected Walk-on-Spheres

Input: surface $\mathcal{S}$, boundary $\mathcal{C}$, evaluation point $\mathbf{x} \in \mathcal{S}$, sample walk count N_P, volume sample count N_V, tolerance ϵ

Function `EstimateSolution`($\mathcal{S}$, $\mathcal{C}$, N_P, N_V, $\mathbf{x}$, ϵ)**:**

- $\mathcal{M} \leftarrow$ `medialAxisPointCloud`($\mathcal{S}$) `// Section 5.2.1`
- $\widehat{u}_{\text{sum}} \leftarrow 0$
- **for** $n \leftarrow 1$ *to* N_P **do**
  - $\widehat{u} \leftarrow$ `RecursiveEstimate`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, N_V, $\mathbf{x}$, ϵ)
  - $\widehat{u}_{\text{sum}} \leftarrow \widehat{u}_{\text{sum}} + \widehat{u}$
- **end**
- **return** $\widehat{u}_{\text{sum}}/N_P$

Function `RecursiveEstimate`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, N_V, $\mathbf{x}$, ϵ)**:**

- $\delta \leftarrow$ `distanceToBoundary`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, $\mathbf{x}$) `// Section 5.2.2`
- **if** $\delta < \epsilon$ **then**
  - **return** $u(\text{cp}_{\mathcal{C}}(\mathbf{x}))$
- $l \leftarrow$ `localFeatureSize`($\mathcal{S}$, $\mathcal{M}$, $\mathbf{x}$) `// Section 5.2.1`
- $r \leftarrow \min(l, \delta)$
- $\mathbf{y} \leftarrow$ `uniformSphereSample(center=`$\mathbf{x}$ `, radius=`r`)`
- $\widehat{u}_{\text{sphere}} \leftarrow$ `RecursiveEstimate`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, N_V, $\text{cp}_{\mathcal{S}}(\mathbf{y})$, ϵ)
- $\{\mathbf{z}_1, \ldots \mathbf{z}_{N_V}\} \leftarrow$ `ballSample(center=`$\mathbf{x}$ `, radius=`r`)`
- $\widehat{u}_{\text{ball}} \leftarrow \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{G(\mathbf{x},\mathbf{z}_i) f(\text{cp}_{\mathcal{S}}(\mathbf{z}_i))}{p(\mathbf{z}_i)}$
- **return** $\widehat{u}_{\text{sphere}} + \widehat{u}_{\text{ball}}$

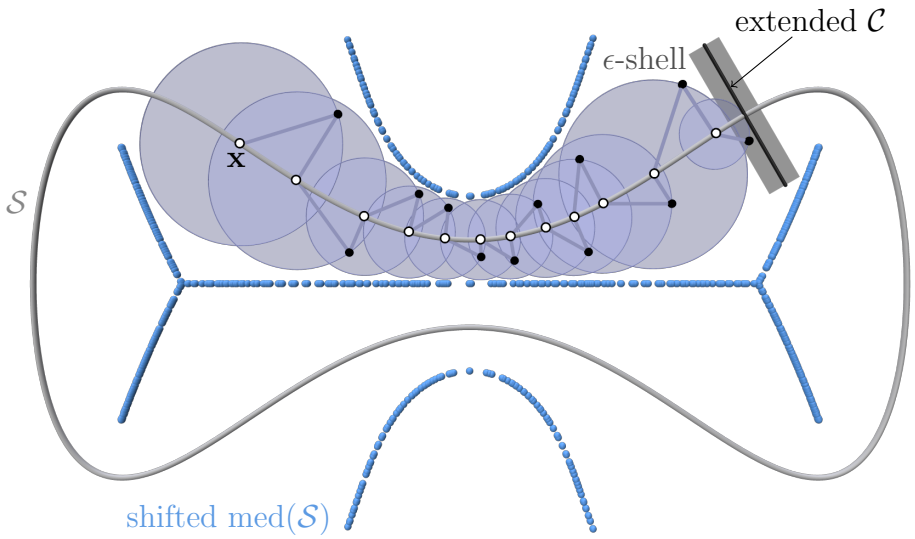


Figure 5.2: A PWoS path for the Laplace equation on a grey 1D (curve) surface embedded in 2D space, starting from $\mathbf{x}$ and terminating at the extended Dirichlet boundary $\mathcal{C}$.

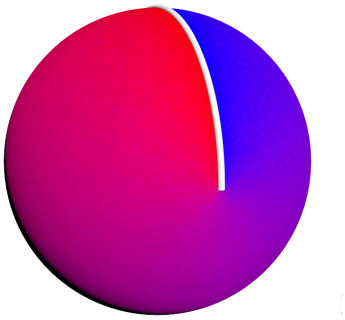
(under-)estimate of the local feature size at point $\mathbf{x}$ (Section 5.2.1) and the distance to the (extended) Dirichlet boundaries (Section 5.2.2).

In (5.2), the Monte Carlo estimate of the solution on the sphere, $\widehat{u}(\mathbf{y})$, needs to be evaluated at point $\mathbf{y}$, which does not lie on $\mathcal{S}$ in general. The CP extension constraint of u provides a convenient relationship here: the embedding PDE's solution at point $\mathbf{y}$ should coincide with the surface PDE's solution at the projected point, $\mathrm{cp}_{\mathcal{S}}(\mathbf{y})$. We can therefore project the point $\mathbf{y}$ onto $\mathcal{S}$ at the end of each recursion step hence $\widehat{u}(\mathbf{y}) = \widehat{u}(\mathrm{cp}_{\mathcal{S}}(\mathbf{y}))$. After this projection at the end of each step, we continue the recursion. The source term similarly uses the closest point projection for the CP extension, replacing the recursive relationship of WoS (5.2) with

$$\widehat{u}(\mathbf{x}) = \widehat{u}(\mathrm{cp}_{\mathcal{S}}(\mathbf{y})) + \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{G(\mathbf{x}, \mathbf{z}_i) f(\mathrm{cp}_{\mathcal{S}}(\mathbf{z}_i))}{p(\mathbf{z}_i)}. \tag{5.5}$$

We choose $p(\mathbf{z}_i) \propto 1/\|\mathbf{x} - \mathbf{z}_i\|_2$ in our implementation.

Analogous to the original WoS method, we terminate the recursion when the point $\mathbf{x}$ falls within a distance ϵ of the (extended) Dirichlet boundary by taking the boundary value $u(\mathrm{cp}_{\mathcal{S}}(\mathbf{x}))$. We provide pseudocode for an instance of our algorithm in Algorithm 2, where



Local Feature Size	Avg. Step Count
0.99	31.1398
0.5	47.84
0.25	128.981
0.125	462.781
0.0625	1818.73

Figure 5.3: Average number of steps required with different conservative local feature size estimates. While any positive value smaller than 1 is valid for this setup, using a local feature size estimate that is too small leads to excessively long walks.

we highlight the difference between our proposed method and WoS. We also provide a visualization of a potential path of our algorithm when $\mathcal{S}$ is embedded in $\mathbb{R}^2$ in Figure 5.2.

Notably, PWoS is a generalization of the WoS algorithm: when $\dim(\mathcal{S}) = d$ (i.e., the codimension-zero case), the local feature size is infinite, the distance to the Dirichlet boundary $\mathcal{C}$ can be computed with the closest point query $\text{cp}_{\mathcal{C}}$, and the last closest point projection of $\mathbf{y}$ has no effect since $\text{cp}_{\mathcal{S}}(\mathbf{y}) = \mathbf{y}$. When $\dim(\mathcal{S}) < d$, in addition to the closest point projection at the end of each recursion step, our algorithm utilizes two nontrivial steps: the local feature size estimation using a medial axis point cloud and the computation of the distance to the (extended) Dirichlet boundary. We discuss these in the following two subsections.

5.2.1 Local Feature Size Estimation

To determine the radius of the sphere centred at $\mathbf{x} \in \mathcal{S}$ that is fully contained inside $\mathcal{N}(\mathcal{S})$ at each recursion step of the walk, we need a conservative lower bound estimate for the local feature size at $\mathbf{x}$. One naive approach would be to use a small enough positive constant value for all $\mathbf{x} \in \mathcal{S}$, similar to the grid-based CPM without spatial adaptivity [138]. This is a valid strategy, but it would often yield a sphere radius smaller than necessary, requiring more recursion steps for the walks to reach a Dirichlet boundary and making the method inefficient. Figure 5.3 illustrates a result of our algorithm on a unit sphere using different (artificial) local feature size estimates. The analytical local feature size of this surface is 1 everywhere. Although using any smaller value would still give consistent results, it significantly increases the average step count for the walks.

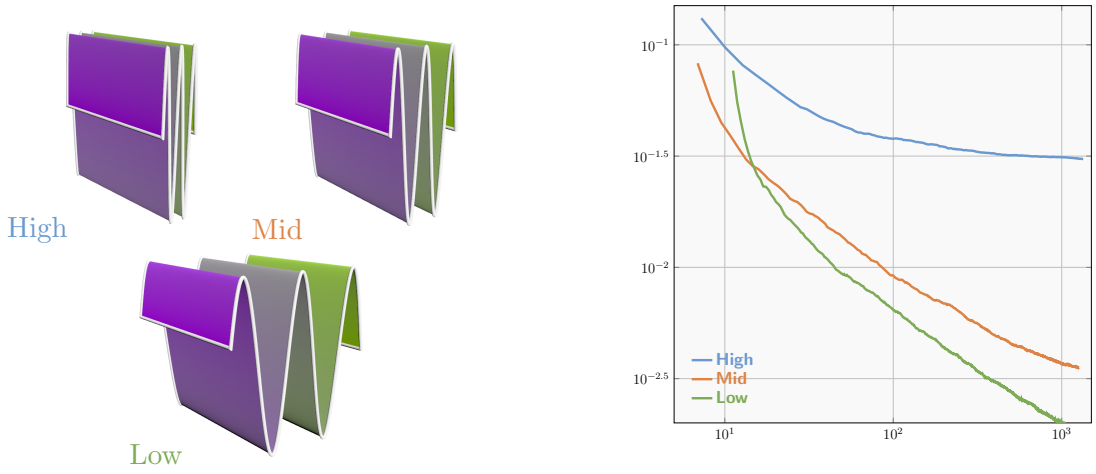


Figure 5.4: Effect of local feature size on convergence speed. The vertical axis of the convergence plot represents the root mean squared error (RMSE), while the horizontal axis shows the time in seconds.

Figure 5.4 further compares the error versus runtime of our method for problems with different local feature sizes. We bend a rectangular strip of size 10 by 2 units along sinusoidal curves with high, medium, and low frequencies (top left, top right, and bottom, respectively, in Figure 5.4) and solve the Laplace equation. The analytical solution is defined as the distance along the longer edge of the strip from one of the shorter edges. We used 1000 sample evaluation points. The runtime in seconds is measured on a MacBook Pro with an M1 Pro chip. The geometry with a larger local feature size allows for faster convergence with lower bias. For more complex shapes, one cannot compute the analytical local feature size in general and using a small constant value in its place is inefficient. This issue motivates our need for a better local feature size estimate.

To estimate the local feature size, we compute a point cloud approximation of the medial axis and estimate the local feature size as the distance from any query point $\mathbf{x} \in \mathcal{S}$ to its nearest point in the medial axis point cloud. One could use any local feature size estimation algorithm and/or medial axis extraction algorithm (see e.g., [167]), such as one that outputs or uses the medial axis' connectivity. Since such methods are often comparatively costly, we employed a simple point-cloud-based method, which we describe below.

Medial Ball Extraction

We first densely scatter points $\mathbf{x}_i$ inside a ball in $\mathbb{R}^3$ having a radius equal to half the length of the diagonal of the bounding box of $\mathcal{S}$, so the entire surface is fully enclosed. For each point $\mathbf{x}_i$, we use the closest point query $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ to compute its two opposing normal directions at $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$. Specifically, we normalize the vector $\mathbf{x}_i - \text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ to get the first direction and invert its direction to get the second one. We then use the method of Ma et al. [86] to extract a point cloud that represents the medial axis, as follows. For each side of the surface (i.e., each normal), we start with a large sphere tangent to $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$, whose centre necessarily lies on the normal ray. The initial radius of the ball is set to the length of the diagonal of the bounding box of $\mathcal{S}$. Then, we iteratively shrink the size of the sphere, moving its centre to maintain tangency at the surface point $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$, until the closest surface point from the centre of the sphere does not change.

Algorithm 3: Medial Ball Centre Extraction

Given: Point $\mathbf{x}$, initial sphere radius r, # max iterations K, tolerance tol

$\mathbf{n} \leftarrow (\mathbf{x} - \text{cp}_{\mathcal{S}}(\mathbf{x})) / \|\mathbf{x} - \text{cp}_{\mathcal{S}}(\mathbf{x})\|$
while $k < K$ **do**
  $k \leftarrow k + 1$
  $\mathbf{p} \leftarrow \text{cp}_{\mathcal{S}}(\text{cp}_{\mathcal{S}}(\mathbf{x}) + r\mathbf{n})$
  $\mathbf{v} \leftarrow \mathbf{p} - \text{cp}_{\mathcal{S}}(\mathbf{x})$
  if $\|\mathbf{v}\| < \text{tol}$ **OR** $\mathbf{v} \cdot \mathbf{n} < \text{tol}$ **then**
    return $\text{cp}_{\mathcal{S}}(\mathbf{x}) + r\mathbf{n}$
  end
  $r_{\text{prev}} \leftarrow r$
  $r \leftarrow \frac{\|\mathbf{v}\|}{\mathbf{v} \cdot \mathbf{n}}$
  if r_{prev} - $r < \text{tol}$ **then**
    return $\text{cp}_{\mathcal{S}}(\mathbf{x}) + r\mathbf{n}$
  end
end

This algorithm gives *medial balls*, i.e., balls with their centres on the medial axis. As the number of initial scattered points increases, the extracted point cloud balls tend to cover the entire medial axis. While Ma et al. [86] assumed that the surface is represented as an oriented point cloud, we observed that the algorithm works well with the closest point surface representation by adjusting its termination criteria. As shown in Algorithm 3, we

terminate if the distance between closest points is small, if the projection onto the normal vector of the difference between closest points is small, or if there is little change in the radius of the medial ball. Figure 5.5 (a) is an example of computing the medial axis point cloud for the Dziuk surface parameterization with tol $= 10^{-3}$.

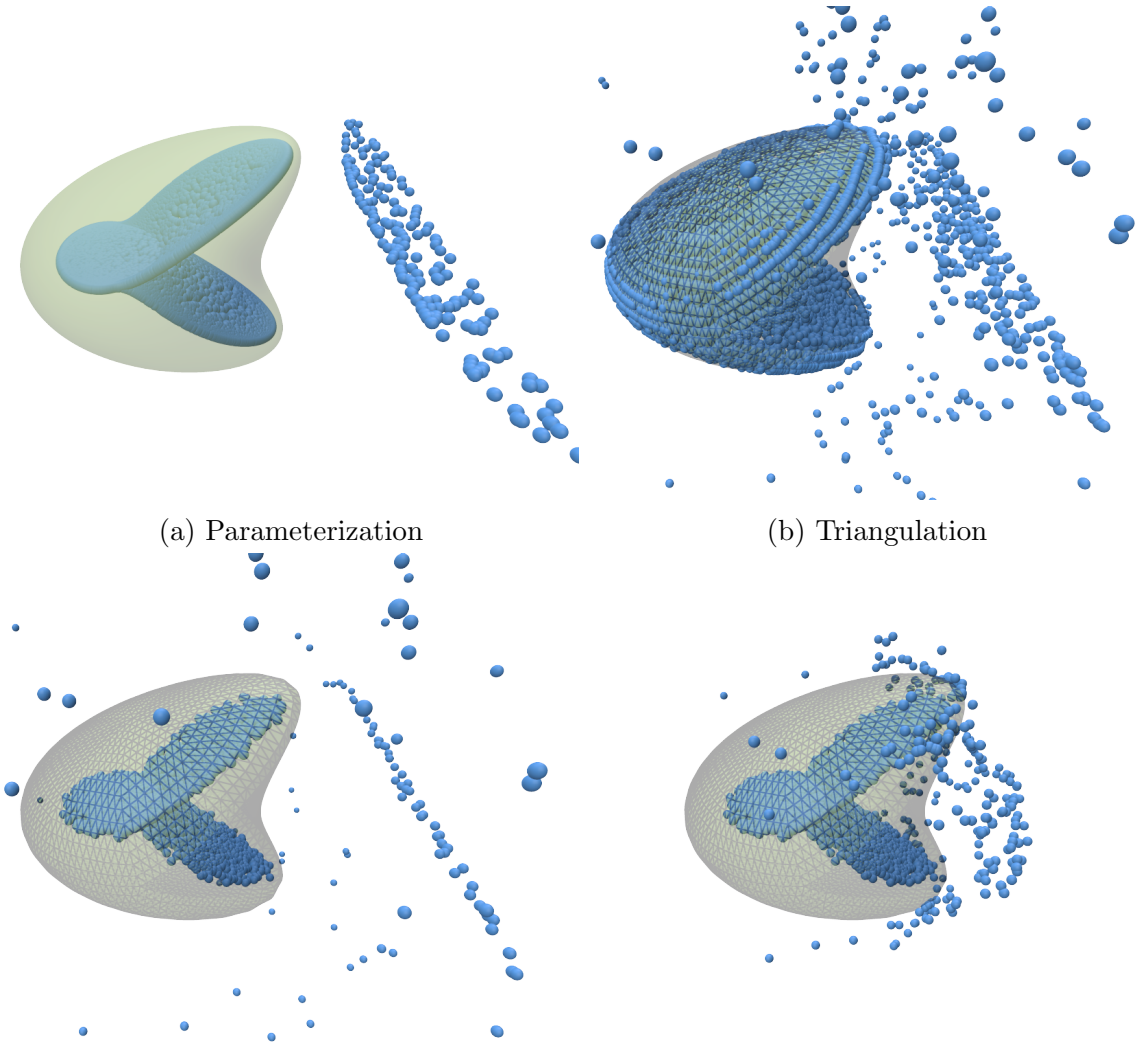
Scale Axis Pruning

Directly using the medial ball centres as the medial axis point cloud does not work well in general when $\mathcal{S}$ contains any noise or artificial sharp corners introduced by the discretization of a smooth surface. Figure 5.5 (b) shows that Algorithm 3 produces many undesirable medial ball centres on $\mathcal{S}$ itself when used on a relatively smooth triangulation of the Dziuk surface. This is expected since the exact medial axis will touch all edges/vertices in a concave region of any triangulation. Also, unwanted noise is present in Figure 5.5 (b).

We therefore prefer to estimate (only) the *stable* part of the medial axis, which is not affected by any small perturbation of $\mathcal{S}$. A common solution is, therefore, to *prune* unstable components of the medial axis, which itself remains an active research topic [167]. We take inspiration from the scale axis transform (SAT) [51, 100], but design an alternative since SAT considers topology information of the medial axis, which is unnecessary for local feature size estimation. Our alternative is simpler and faster since topology information is omitted. We first scale all the medial balls by a constant factor $s > 1$. Then, for any pair of medial balls $B_{r_1}(\mathbf{x}_1)$ and $B_{r_2}(\mathbf{x}_2)$, we remove the smaller ball from the set of medial balls if it is fully contained in the other. That is, if $s \cdot r_1 < s \cdot r_2 + \|\mathbf{x}_1 - \mathbf{x}_2\|_2$, we remove the ball $B_{r_1}(\mathbf{x}_1)$.

Note that some of the medial balls may have a very large radius before pruning. For example, an exterior medial ball for a surface of a convex shape would have an infinite radius in theory (but our algorithm returns at most the length of the diagonal of the bounding box of $\mathcal{S}$). When such large medial balls are used in our pruning algorithm, they can easily (and undesirably) remove important, stable parts of the medial axis. The original SAT approach was applied only to the interior medial axis of closed surfaces. Therefore, this issue was not observed since the interior medial ball radii are always bounded and proportional to the size of local features of $\mathcal{S}$.

To address this problem, we consider each pair of tangent balls generated at the same surface point, and replace the larger one with a tangent ball having the radius of the smaller one. In other words, before we prune the medial axis, we shift the medial ball centres of the larger medial ball in the pair and shrink its radius. In Figure 5.2, we visualize the medial ball centres after this shifting operation. Figure 5.5 (c) and (d) also visualize using



(a) Parameterization

(b) Triangulation

(c) Tri. + Our Scale axis transform. (SAT)

(d) Tri. + Our SAT + Shifted

Figure 5.5: Medial axis point clouds computed (blue) using Algorithm 3 on different surface representations (green) and with different pruning methods.

our SAT with and without shifting the medial axis point cloud, respectively. It is not obvious from this example that shifting the medial axis helps to save the stable part of the medial axis (but we have observed this on other $\mathcal{S}$). However, it is possible for a medial ball outside of $\mathcal{S}$ whose radius is scaled by a factor $s > 1$ to completely engulf all of $\mathcal{S}$ (worst-case scenario). Adjusting the scaling parameter s allows us to control the pruning strength. Unless otherwise stated, we use the value $s = 1.15$ for our results.

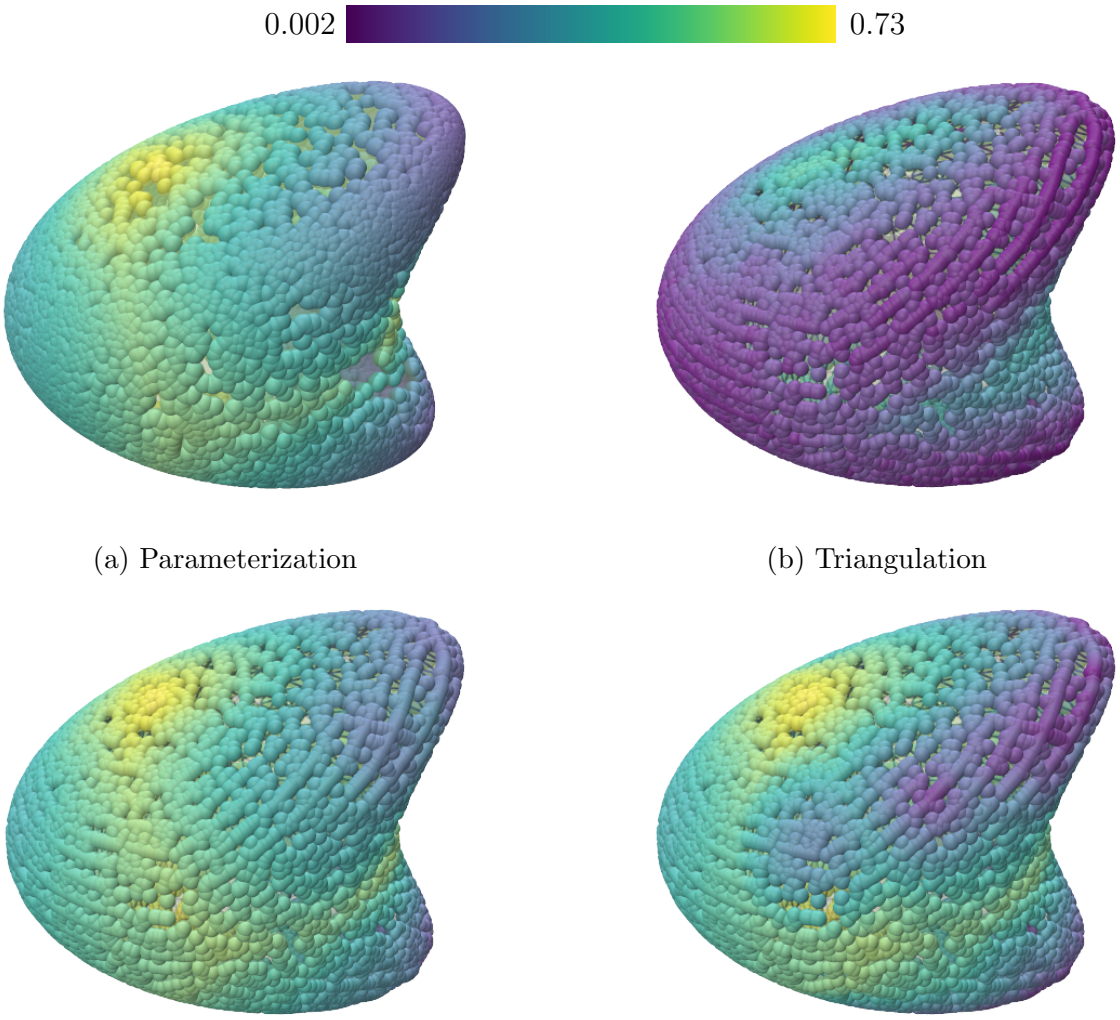
After this pruning, the set of medial ball centres gives the medial axis point cloud we use to estimate the local feature size. Figure 5.6 shows the local feature size computed from the medial axis point clouds in Figure 5.5. Figure 5.6 (a) is the most accurate since using the parameterization of the Dziuk surface gives the most accurate medial axis point cloud. The local feature size is drastically underestimated in Figure 5.6 (b) since the medial axis point cloud touches $\mathcal{S}$ almost everywhere. Figure 5.6 (c) and (d) show that using our SAT pruning gives more accurate local feature size estimation. As seen in Figure 5.6 (d), one disadvantage of shifting the medial axis point cloud before using our SAT is the underestimation of local feature size. However, the shifting will never cause an overestimate of the local feature size, since we only move the medial axis closer to $\mathcal{S}$.

Conservative and Nonzero Local Feature Size

As the medial axis is represented as a point cloud and the nearest point distance may give a larger value than the actual local feature size, we multiply by a small constant (0.9 in our implementation) to ensure a conservative estimate of the local feature size. When there are sharp corners in the geometry, the analytical local feature size is zero, and the walk will become stuck. To prevent this problem, when the estimated local feature size is smaller than a positive constant threshold λ, we return λ as the local feature size estimate instead. This process essentially rounds sharp corners with rounding radius λ. The uniform grid size adopted in grid-based CPM has a similar effect. We do not observe any significant error due to this rounding, as we show in Section 5.3.

5.2.2 Distance to Extended Dirichlet Boundaries

Dirichlet boundaries $\mathcal{C}$ are extended in the normal direction of $\mathcal{S}$ and the solution in the embedding space on this extended boundary is determined by the CP extension of Dirichlet values on $\mathcal{C}$. Therefore, we need to compute the minimum distance to the extended Dirichlet boundaries, and limit the sphere radius in PWoS further if it is less than the local feature size. To determine the distance to the extended Dirichlet boundary from point $\mathbf{x} \in \mathcal{S}$,



(a) Parameterization

(b) Triangulation

(c) Tri. + Our SAT

(d) Tri. + Our SAT + Shifted

Figure 5.6: Local feature size estimates for points on the Dzuik surface using the corresponding medial axis point clouds computed in Figure 5.5.

we first find the closest point that lies on the boundary before the extension, $\text{cp}_{\mathcal{C}}(\mathbf{x})$. The subset of the extended boundary that is extended from $\text{cp}_{\mathcal{C}}(\mathbf{x})$ takes the shape of a line segment when $\dim(\mathcal{S}) = 2$ and a disk when $\dim(\mathcal{S}) = 1$. We set the line segment's half-length or the disk radius to the local feature size at $\text{cp}_{\mathcal{C}}(\mathbf{x})$ using the algorithm in Section 5.2.1. We can compute the distance from the point $\mathbf{x} \in \mathcal{S}$ to this line segment or disk without explicitly constructing the extended boundary geometry. When $\dim(\mathcal{S}) = 2$, the distance δ to the extended boundary is given by

$$\begin{aligned} \mathbf{r} &= \text{cp}_{\mathcal{C}}(\mathbf{x}) - \mathbf{x}, \\ \delta &= \|\mathbf{r} - \text{clamp}(\mathbf{r} \cdot \mathbf{n}, -l, l) \cdot \mathbf{n}\|_2, \end{aligned} \tag{5.6}$$

where $\mathbf{n}$ and l are the surface normal and the local feature size estimate at $\text{cp}_{\mathcal{C}}(\mathbf{x})$, respectively. When $\dim(\mathcal{S}) = 1$, the normal direction is not uniquely defined, so we instead use a similar method based on the surface tangent $\mathbf{t}$:

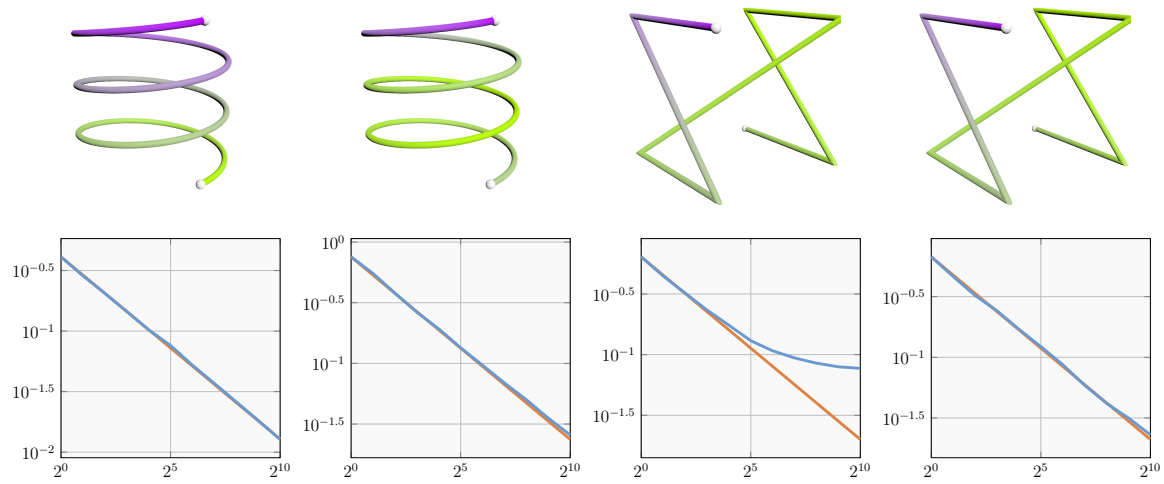
$$\begin{aligned} \mathbf{q} &= \mathbf{r} - (\mathbf{r} \cdot \mathbf{t})\mathbf{t}, \\ \delta &= \begin{cases} |\mathbf{r} \cdot \mathbf{t}|, & \text{if } \|\mathbf{q}\|_2 < l, \\ \|\mathbf{r} - l \cdot (\mathbf{q}/\|\mathbf{q}\|_2)\|_2, & \text{otherwise.} \end{cases} \end{aligned} \tag{5.7}$$

5.3 Convergence Studies

We conducted error convergence studies of our method using discretized surfaces. In each scene we change the number of sample paths N_P to plot the root mean squared error measured against the analytical or reference solution. The scene setup includes Laplace, Poisson, and screened-Poisson equations, and both smooth surfaces and surfaces with sharp corners. A generalization of our method allows us to handle the screened-Poisson equation. See [164, Section 3.3.1] for details.

In each of the examples in Figures 5.7 and 5.8, we show the visualization of the scene setup (top), error convergence plot (middle), and scene description (bottom). The scene visualizations show the analytical or reference solution of the problem by mapping the range of solution values to the green-to-purple colour gradient and placing a white point or curve on the Dirichlet boundary. In all scenes, we use $\epsilon = 0.001$ and $N_V = 32$ except for Laplace equations, for which we use $N_V = 0$.

While the expected $O(1/\sqrt{N_P})$ is achieved in most scenes, the bias remains high (i.e., the error plateaus as N_P increases) when an aggressive medial axis pruning parameter is used (see Figure 5.7 (c)) and when the source term of the problem is relatively complex (see

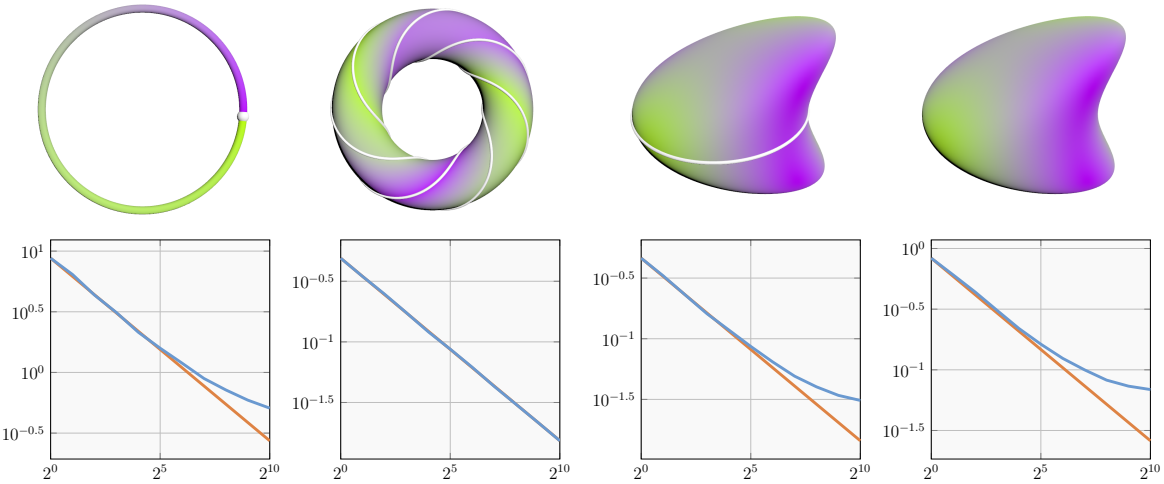


(a) Helix
Ends
Laplace

(b) Helix
Ends
Poisson

(c) Z-Order
($s = 1.15$)
Ends
Poisson

(d) Z-Order
($s = 1.05$)
Ends
Poisson



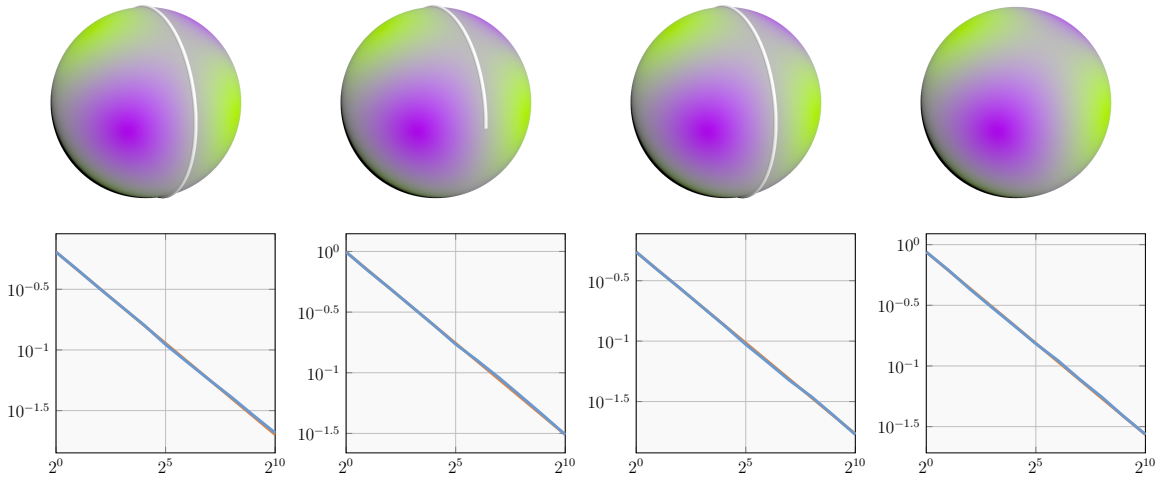
(e) Circle
Two-sided
Poisson

(f) Torus
Torus Knot
Laplace

(g) Dziuk Surface
Closed
screened-Poisson

(h) Dziuk Surface
No Boundary
Screened Poisson

Figure 5.7: Error convergence. The vertical axis of each error plot shows the root mean squared error of the estimates at a few points on the surface in a logarithmic scale, and the horizontal axis shows the number of samples N_P in a logarithmic scale. We show the result of the experiment (blue) and a line that corresponds to the desired $O(1/\sqrt{N_P})$ convergence rate (orange).

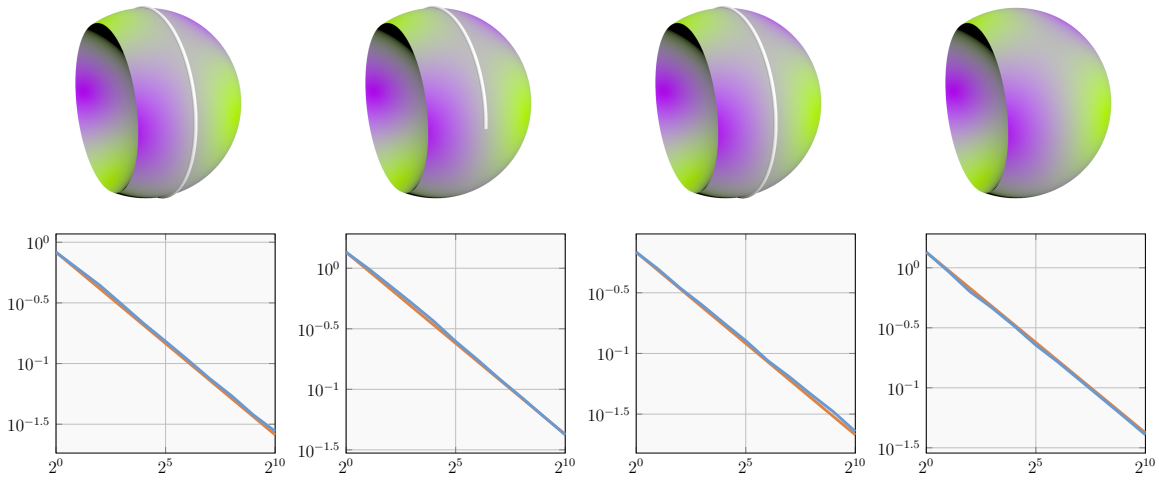


(a) Sphere Closed Poisson

(b) Sphere Open Poisson

(c) Sphere Closed Screened Poisson

(d) Sphere No Boundary Screened Poisson



(e) Punched Closed Poisson

(f) Punched Open Poisson

(g) Punched Closed Screened Poisson

(h) Punched No Boundary Screened Poisson

Figure 5.8: Error convergence. The vertical axis of each error plot shows the root mean squared error of the estimates at a few points on the surface in a logarithmic scale, and the horizontal axis shows the number of samples N_P in a logarithmic scale. We show the result of the experiment (blue) and a line that corresponds to the desired $O(1/\sqrt{N_P})$ convergence rate (orange).

Figure 5.7 (e, g, and h)). This indicates the need for future work to investigate estimating $g(\mathbf{x})$ from (5.4) to reduce the bias in such problems. The remainder of this section details the setup for each of the convergence studies in Figures 5.7 and 5.8.

Figure 5.7 (a)

The helix curve we use has three turns, has a radius of 1, and the endpoints have a height difference of 2. We solve the Laplace equation defined along the curve length ϕ as

$$\begin{aligned} \frac{\partial^2 u_{\mathcal{S}}}{\partial \phi^2} &= 0, \\ u_{\mathcal{S}}(0) &= 0, \\ u_{\mathcal{S}}(\psi) &= 1, \end{aligned} \tag{5.8}$$

where the boundary conditions are specified at the two ends of the curve, $\phi = 0$ and $\phi = \psi$. The analytical solution is $u_{\mathcal{S}}(\phi) = \phi/\psi$.

Figure 5.7 (b)-(d)

The problem we solve is defined along the curve length ϕ as

$$\begin{aligned} \frac{\partial^2 u_{\mathcal{S}}}{\partial \phi^2} &= 0.02, \\ u_{\mathcal{S}}(0) &= 0, \\ u_{\mathcal{S}}(\psi) &= 1, \end{aligned} \tag{5.9}$$

where the boundary conditions are specified at the two ends of the curve, $\phi = 0$ and $\phi = \psi$, similar to (a). The analytical solution is $u_{\mathcal{S}}(\phi) = 0.01\phi^2 + \frac{1-0.01\psi^2}{\psi}\phi$. The helix curve in (b) is identical to the one in (a). The z-order curve in (c) and (d) is defined using 8 points, $(\pm 1.0, \pm 1.0, \pm 1.0)$.

Figure 5.7 (e)-(h)

The remainder of Figure 5.7 uses examples from Chapter 4 (some with slight modifications). The problem in (e) is the same as the one detailed in Section 4.3.1.

For (f), the surface we used is a torus with a major radius $R = 3$ and a minor radius $r = 1$. The Dirichlet boundary curve is the torus knot given in (4.17). We solve the

Laplace equation on the torus with boundary condition $\sin(s)$ along the curve. We used the grid-based CPM implementation of King et al. [71] with a grid spacing $h = 0.02$ to generate a reference solution and measured the error of PWoS against it.

For (g) and (h) the screened-Poisson equation in Section 4.3.3 is solved on the Dziuk surface. For (g), we use a unit circle on the x_1x_2-plane with the analytical solution specified on it as the boundary value as the Dirichlet boundary. For (h), we did not use any boundary to show the algorithm's convergence for the screened-Poisson equation without any boundaries.

Figure 5.8

These scenes use the unit sphere with a spherical harmonic function as the analytical solution as is done in the study of mesh Laplacians [19]. The sphere mesh is punched inward at $x_3 = 0.25$ for (e) to (h) to test the algorithm on a geometry with sharp corners. Given a spherical harmonic $Y_2^3(\mathbf{x}) = \frac{1}{4}\sqrt{\frac{105}{\pi}}(x_1^2 - x_2^2)^2 x_3$ with eigenvalue -12 as the solution, we solve the Poisson equation

$$\Delta_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{x}) = -12 Y_2^3(\mathbf{x}), \quad \mathbf{x} \in \mathcal{S}, \tag{5.10}$$

for (a), (b), (e), and (f) and the screened-Poisson equation

$$\Delta_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{x}) - u_{\mathcal{S}}(\mathbf{x}) = -13 Y_2^3(\mathbf{x}), \quad \mathbf{x} \in \mathcal{S}, \tag{5.11}$$

for (c), (d), (g), and (h). For (a), (c), (e), and (g), we use the unit circle on the x_1x_2-plane as the Dirichlet boundary, and for (b) and (f), we use the unit semicircle where $x_2 > 0$ as the Dirichlet boundary. We observe the expected convergence behaviour with all of the cases in Figure 5.8 despite our method's inherent assumption of $g(\mathbf{x}) = 0$. We have not yet developed a theory that explains under what conditions this approach consistently converges or not; however, it is noteworthy that for these (successful) examples, the source term is a simple constant multiple of the solution.

5.4 Applications

5.4.1 Diffusion Curves

Diffusion curves [122] succinctly represent an image as a collection of curves with associated colours. As discussed previously in Section 4.4.1, the final image, exhibiting smooth colour

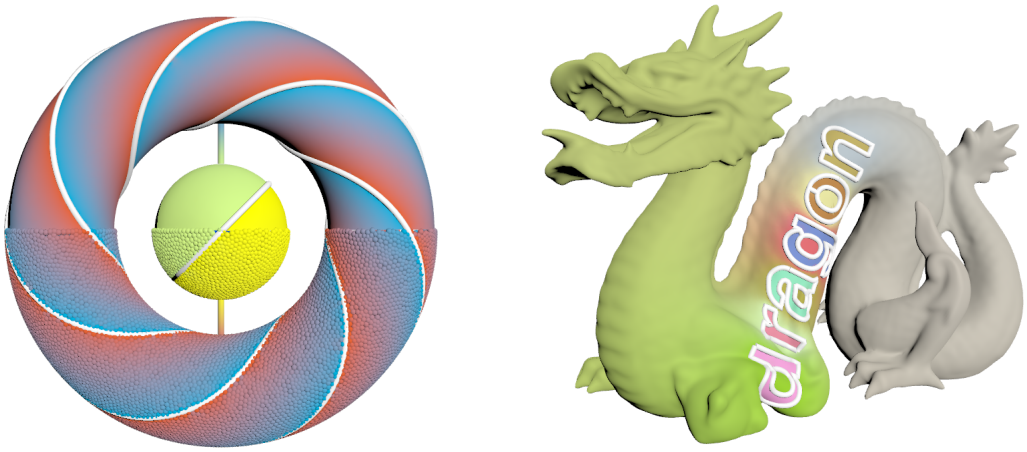


Figure 5.9: Surface diffusion curves solved on various surface representations. The surface on the left is represented as a combination of triangles, polylines, and oriented points. The surface on the right is represented as a quadrilateral mesh.

gradients, is recovered by solving a Laplace equation with the curves dictating boundary conditions. In our application, we solve the surface Laplace equation using PWoS.

With our approach, the surface need not have a boundary curve conforming to the discretized mesh, which contrasts with the common approach [38]. Figure 5.9 shows the reconstruction of colour at each point on the discretized surface, represented as a quadrilateral mesh (right) and a combination of triangles, polylines, and point clouds (left). Our method naturally supports two-sided boundaries, with different colours specified on each side of a curve, and surface geometries with mixed-codimension.

Additionally, the pointwise nature of PWoS allows it to be applied in a view-dependent manner. For example, given a camera configuration, for antialiasing, we sample points within each pixel to generate rays. We then generate PWoS samples at the ray-surface intersection points. No computational resources are wasted on surface points that are invisible to the camera (Figure 5.1), and we can obtain clean results without relying on a fine discretization of the surface. Boundary integral-based approaches [12, 165] would similarly allow domain discretization-free evaluation of diffusion curves, but they first require a global linear system solve. Moreover, such methods are not applicable to general curved surfaces, and would need to map the results in the 2D domain to the surface via UV coordinates, for example.

5.4.2 Geodesic Distance

Crane et al. [33] proposed the heat method, which solves two standard surface PDEs in series to compute the geodesic distance from the boundary $\mathcal{C}$. The steps are summarized in Section 4.4.2. Step (2) uses the gradient of the solution to the screened-Poisson equation found in Step (1). With a discretization-based method, a discrete gradient operator is used to estimate this gradient. In our method, we directly evaluate the gradient of u during Step (1) without needing u itself. This generalization of our method is detailed in [164, Section 3.3.2].

We evaluate the gradient at mesh vertices and normalize it to get $\mathbf{X}$ at mesh vertices. In Step (3), again, we do not rely on a discrete divergence operator to solve the Poisson equation, but instead use the generalization of our method described in [164, Section 3.3.2]. When our Poisson solver requires the evaluation of $\mathbf{X}$ at a point, we interpolate $\mathbf{X}$ from the mesh vertices and (re)normalize it. We can similarly compute the geodesic distance on a surface represented as a point cloud.

Figure 5.10 compares our PWoS-based heat method on surfaces represented as polygonal meshes or oriented point clouds against the heat method with grid-based CPM (Section 4.4.2) and the exact geodesic distance computed with `geometry-central` [154]. We compute the geodesic distance to the great circle in the center of the sphere, as well as the geodesic distance to the boundary of the car. Our results are consistent with the reference implementations, albeit with minor deviations.

5.5 Summary and Future Work

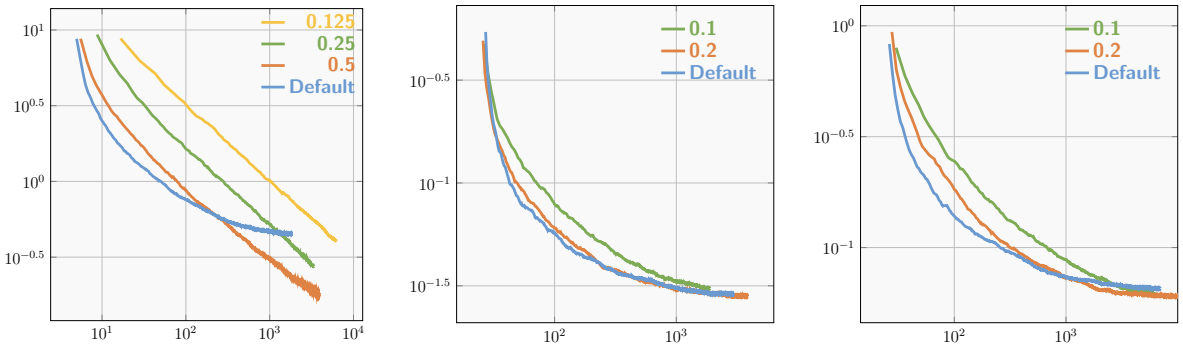
We have developed a Monte Carlo method for surface PDEs by augmenting the formulation of the walk-on-spheres method with a closest point projection step. Our algorithm is justified through its connection to the theory of CPM; an embedding PDE is solved in the space surrounding $\mathcal{S}$, and closest point projections are allowed since the solution is constant in the normal direction of $\mathcal{S}$. We have further analysed the method's convergence on representative analytical tests and demonstrated its application to geometry processing problems.

PWoS currently supports only Dirichlet boundary conditions; efficient Neumann or Robin boundary handling similar to the walk-on-stars method for volumetric PDEs [102, 142] would require the availability of a few more queries, such as a ray intersection query against the (extended) boundaries.

Figure 5.10: Geodesic distance computation with the heat method. For each of the two scenes, we compare our algorithm on a polygonal mesh representation (leftmost) and oriented point cloud representation (middle-left) against a grid-based CPM counterpart (middle-right) and the exact polyhedral distance computed with `geometry-central` [154] (rightmost). For the sphere surface (top), we compute the distance from the circle boundary curve in the centre, and for the car surface (bottom), we compute the distance from the surface boundary edges. Note that the rendering of the point clouds assigns a UV coordinate per point, resulting in larger visual differences.

While we used a local feature size estimation algorithm to allow walks with larger step sizes, the local feature size estimation itself imposes additional smoothness assumptions on the surface. To respect small-scale local features, the walk can require many iterations to reach a Dirichlet boundary. This effect is partly due to our algorithm (like WoS) being based on an integral equation that holds only locally inside a ball. Revisiting this choice using an integral equation based on a global relationship, such as the one underpinning the walk-on-*boundary* method [163], could lead to a more efficient alternative for surface PDEs.

Lastly, our method relied on the assumption that the CP extension compensation term (i.e., $g(\mathbf{x})$ in (5.4)) in the embedding PDE is negligible. We empirically showed that the algorithm designed with this assumption works well when the source term has a relatively simple expression, but we do not yet have a complete understanding of when this assumption is strictly valid. However, since $g(\mathbf{x})$ tends to zero continuously as $\mathbf{x}$ approaches the surface, the influence of ignoring this term is expected to decrease as we shrink the embedding space (i.e., shrink the sphere size). One can always take a smaller sphere size, albeit at a higher computational cost, as we show in Figure 5.11. While limiting the sphere size may reduce the bias, as we can observe from the intersections of the curves for the default option and the curves for the sphere-size-constrained option, the computation may take longer, and it is difficult to get a practical advantage. Instead, extending our method to consider the effect of the compensation term would further improve the reliability and broaden the applicability of our method.



(e) (g) (h)

Figure 5.11: Using bounded sphere size. For the Poisson and screened-Poisson problems (e), (g), and (h) in Figure 5.7, we compare the Default option of not constraining the sphere size (apart from the limit imposed by the local feature size estimate) against specified limits on the maximum sphere size as indicated in the legend. The vertical axis shows the root mean squared error, and the horizontal axis shows the time in seconds. For (e), we had 1024 evaluation points, and for (g) and (h), we used 100 sample evaluation points.

Chapter 6

Geodesic Paths

A geodesic path is the shortest path (locally or globally) between points on a manifold. Geodesics paths are the natural extension of straight lines in Euclidean spaces to curved spaces. Many computer graphics applications require the computation of geodesic paths. For example, a common task in computational fabrication is to segment and flatten digital objects so the object can be manufactured in the real world from flat sheets of material [151]. The borders of the flattened segments can be cut from materials using computer numerical control (CNC) machines, such as laser cutters [117] for cloth or CNC routers for wood. Segmenting the digital object with geodesic paths results in smooth, shorter curves for the CNC machine paths, which allows the manufacturing process to be more efficient. We propose a novel algorithm for computing geodesic paths on general manifold representations given only the ability to perform closest point queries.

Much work has been done to compute the global shortest path, called the *minimal* geodesic. Many of these methods focus on computing the global geodesic distance field ϕ first, and then the geodesic path using the distance field. The path is computed by tracing the path from the end point $\mathbf{q}$ to the start point $\mathbf{p}$ along the direction of the negative intrinsic-distance gradient, $-\nabla_{\mathcal{S}}\phi$.

This two-step approach is used by Mitchell et al. [104] for their exact polyhedral geodesics algorithm. Surazhsky et al. [166] give an acceleration of [104]. Kimmel and Sethian [69] provide a fast marching method to solve the eikonal equation on triangulated surfaces to compute geodesic distance and then geodesics. Mémoli and Sapiro [113] use the fast marching method in the embedding space to compute geodesics on level-set surfaces. Martin and Tsai [94] provide an improvement of [113] for computing the eikonal equation on surfaces that have both a level set and CP representation.

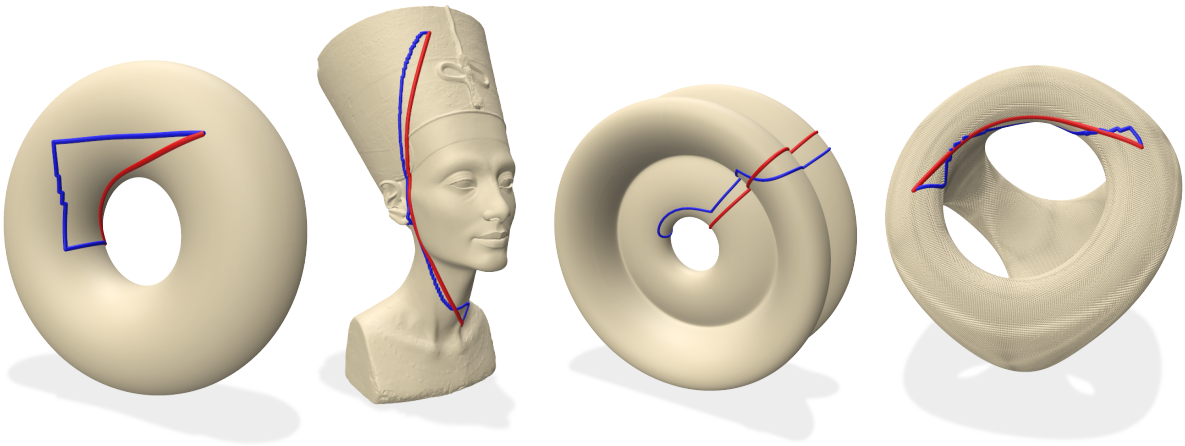


Figure 6.1: Our algorithm to compute geodesic paths is applicable to any manifold representation that supports closest point queries. The initial path (blue) is iteratively shortened to a geodesic (red) using our heat-based method. Manifold representation from left to right: exact closest point queries, mesh, parameterization, level set.

The necessity of first computing the geodesic distance can be inefficient for applications where only the geodesic path is needed (especially if only one path is computed since the geodesic distance computation is not amortized over many paths). Furthermore, in many applications (see e.g., [152]) only locally minimal geodesics are required, not necessarily the globally minimal geodesic.

Many algorithms for computing locally minimal geodesics involve minimizing the length (e.g., [193]) or the geodesic curvature (e.g., [95]). However, as Yuan et al. [193] pointed out, existing methods have mainly been designed specifically for meshes. Xin and Wang [187], Xin et al. [188], and Martínez et al. [95] provide iterative approaches to locally update a polyline to shorten an initial path. Xin and Wang [187] and Xin et al. [188] use geometry-based update rules within triangle strips, while Martínez et al. [95] uses local (per polyline vertex) update rules based only on the triangles containing the polyline vertices. Remešíková et al. [136] and Wu and Tai [186] compute geodesics based on geodesic curvature flow. Remešíková et al. [136] discretizes the problem directly on the polyline. Wu and Tai [186] use a level-set formulation of the problem and solve it on the triangle mesh using a finite-volume method. Most recently, Sharp and Crane [152] shortened initial paths to local geodesics on triangulated surfaces by simple edge flip operations.

The method of Yuan et al. [193] is most similar to ours. Their algorithm is based

on minimizing length. They show how to apply their method with meshes, point clouds, parameterizations, and level sets. However, many of the steps in their approach must be specifically tailored to each different discrete surface representation. Instead, we view geodesics in the setting of harmonic maps, which leads to a unified framework for all manifold representations that allow closest point queries. These representations include all the ones handled by Yuan et al. [193] and many more. Figure 6.1 shows some geodesics (red) computed on different manifold representations with our algorithm.

Our algorithm only involves computing heat flow on a 1D line segment and evaluating closest points on the manifold. It computes the harmonic map from a 1D line segment to the manifold, which is equivalent to the geodesic (see [45]). For this case, the CPM for computing harmonic maps [72] simplifies drastically, which is the basis of our method. We therefore review harmonic maps and the CPM for computing harmonic maps next. Then our method is introduced and compared with the methods of Yuan et al. [193] and Sharp and Crane [152]. The method of Yuan et al. [193] is the state-of-the-art (SOTA) in terms of manifold representation generality, while the method of Sharp and Crane [152] is the SOTA with respect to runtime to shorten the initial path (although it is restricted to meshes).

6.1 Harmonic Maps

Harmonic maps [109, 65, 145] have been used in many applications such as texture mapping [42], regularization of brain images [115], colour image enhancement [169], and interpolating intermediate poses from starting and final poses of a character [46]. King and Ruuth [72] developed a CPM for harmonic maps that works for general manifolds $\mathcal{S}_1$ and $\mathcal{S}_2$. We first review harmonic maps and their method, and then show how their approach can be used to compute geodesic paths via harmonic maps in the next section.

A harmonic map $\mathbf{u}(\mathbf{x}) : \mathcal{S}_1 \to \mathcal{S}_2$ is a mapping from a *source* manifold $\mathcal{S}_1$ to a *target* manifold $\mathcal{S}_2$ that minimizes the Dirichlet energy

$$E_H[\mathbf{u}] = \frac{1}{2} \int_{\mathcal{S}_1} \left\| \mathbf{J}_{\mathbf{u}}^{\mathcal{S}_1} \right\|_{\mathcal{F}}^2 \, d\mathcal{S}_1, \tag{6.1}$$

where $\mathbf{J}_{\mathbf{u}}^{\mathcal{S}_1}$ is the intrinsic Jacobian of the map on $\mathcal{S}_1$ and $\| \cdot \|_{\mathcal{F}}$ is the Frobenius norm. Note that $\mathbf{u} = (u_1, u_2, \ldots, u_{d_2})^T$ has the pointwise constraint $\mathbf{u}(\mathbf{x}) \in \mathcal{S}_2$ for any $\mathbf{x} = (x_1, x_2, \ldots, x_{d_1})^T \in \mathcal{S}_1$, where d_1 and d_2 are the dimensions of the embedding space for $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. The intrinsic Jacobian can be written in terms of the standard

Jacobian as $\mathbf{J}_{\mathbf{u}}^{\mathcal{S}_1} = \mathbf{J}_{\mathbf{u}} \Pi_{\nabla T_{\mathbf{u}} \mathcal{S}_1}$, where $\Pi_{\nabla T_{\mathbf{u}} \mathcal{S}_1} = \mathbf{I} - \mathbf{n}_{\mathcal{S}_1} \mathbf{n}_{\mathcal{S}_1}^T$ is the projection operator onto the tangent space of $\mathcal{S}_1$. Physically, a map is harmonic when $\mathcal{S}_1$ corresponds to a membrane that is constrained to $\mathcal{S}_2$ in elastic equilibrium [45].

Mémoli et al. [114] derived the Euler-Lagrange equations for (6.1) in terms of the level set representation of $\mathcal{S}_2$ under the assumption that $\mathcal{S}_1$ is flat and open. The same calculation is carried out by Moser [109] in terms of the closest point representation of $\mathcal{S}_2$. There, the closest point function is called the nearest point projection and is used to prove regularity results of harmonic maps [109, Chapter 3].

The Euler-Lagrange equations for (6.1) are $\Pi_{T_{\mathbf{u}} \mathcal{S}_2}(\Delta_{\mathcal{S}_1} \mathbf{u}) = 0$, where $\Pi_{T_{\mathbf{u}} \mathcal{S}_2}$ is the projection operator at the point $\mathbf{u}$ onto the tangent space of $\mathcal{S}_2$. The vector $\Delta_{\mathcal{S}_1} \mathbf{u}$ is defined component-wise, i.e., $\Delta_{\mathcal{S}_1} \mathbf{u} = (\Delta_{\mathcal{S}_1} u_1, \Delta_{\mathcal{S}_1} u_2, \ldots, \Delta_{\mathcal{S}_1} u_{d_2})^T$. The Euler-Lagrange equations are solved by evolving the corresponding gradient descent flow

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} = \Pi_{T_{\mathbf{u}} \mathcal{S}_2}(\Delta_{\mathcal{S}_1} \mathbf{u}), \\ \mathbf{u}(\mathbf{x}, 0) = \mathbf{u}^0(\mathbf{x}), \\ \mathbf{J}_{\mathbf{u}}^{\mathcal{S}_1} \mathbf{n}|_{\partial \mathcal{S}_1} = 0, \end{cases} \tag{6.2}$$

to steady state. The map $\mathbf{u}^0(\mathbf{x})$ is a given initial map and the homogeneous Neumann boundary conditions are justified in [114, Appendix A].

To use CPM to solve (6.2) a corresponding embedding PDE must be constructed first. As shown in Chapter 2, the term $\Delta_{\mathcal{S}_1} \mathbf{u}$ can be replaced by $\Delta \mathbf{u}(\mathrm{cp}_{\mathcal{S}_1})$. Furthermore, the projection operator $\Pi_{T_{\mathbf{u}} \mathcal{S}_2}$ equals the Jacobian of the closest point function, $\mathbf{J}_{\mathrm{cp}_{\mathcal{S}_2}(\mathbf{u})}$, for $\mathbf{u} \in \mathcal{S}_2$, as mentioned in Section 4.2.7. Applying these replacements gives the embedding PDE for the gradient descent flow as

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{J}_{\mathrm{cp}_{\mathcal{S}_2}(\mathbf{u})}(\Delta \mathbf{u}(\mathrm{cp}_{\mathcal{S}_1})). \tag{6.3}$$

The closest point function, $\mathrm{cp}_{\mathcal{S}_2}$, is itself a projection operator onto $\mathcal{S}_2$. King and Ruuth [72] realized that the computation of $\mathbf{J}_{\mathrm{cp}_{\mathcal{S}_2}}$ can be eliminated by splitting the evolution of (6.3) into two steps. The first step solves a PDE on $\mathcal{S}_1$ alone and the second projects the solution onto $\mathcal{S}_2$ using $\mathrm{cp}_{\mathcal{S}_2}$. They prove that this splitting approach gives a first-order approximation [72, Theorem 1].

To solve (6.3), the embedding PDE on $\mathcal{S}_1$ alone,

$$\frac{\partial \mathbf{v}}{\partial t} = \Delta \mathbf{v}(\mathrm{cp}_{\mathcal{S}_1}), \tag{6.4}$$

is evolved for one time step of size Δt to give $\mathbf{v}_i$ at each grid node $\mathbf{x}_i \in \Omega(\mathcal{S}_1)$. The second step projects $\mathbf{v}_i$ onto $\mathcal{S}_2$ via $\mathrm{cp}_{\mathcal{S}_2}(\mathbf{v}_i)$. The result, $\mathbf{u}_i^{k+1} \approx \mathbf{u}(\mathbf{x}_i, t^{k+1})$, approximates the solution at $t^{k+1} = (k+1)\Delta t$. Starting from $\mathbf{u}_i^0 = \mathbf{u}(\mathrm{cp}_{\mathcal{S}_1}(\mathbf{x}_i), 0)$, the steps to advance from time t^k to time t^{k+1} are given in Algorithm 4.

Algorithm 4: A time step of CPM for harmonic maps.

1. Solve (6.4) using an explicit Euler time-step of size Δt and CPM appproach of Ruuth and Merriman [138]:
   - *Evolution.* For $\mathbf{x}_i \in \Omega(\mathcal{S}_1)$ solve

$$\begin{cases} \dfrac{\partial \mathbf{v}}{\partial t}(\mathbf{x}, t) = \Delta \mathbf{v}(\mathrm{cp}_{\mathcal{S}_1}(\mathbf{x}), t), \\ \mathbf{v}(\mathbf{x}_i, 0) = \mathbf{u}_i^k. \end{cases}$$

   - *CP extension.* Set $\mathbf{v}_i^{\Delta t} = \mathbf{v}(\mathrm{cp}_{\mathcal{S}_1}(\mathbf{x}_i), \Delta t)$.
2. Project $\mathbf{v}_i$ onto $\mathcal{S}_2$ by setting $\mathbf{u}_i^{k+1} = \mathrm{cp}_{\mathcal{S}_2}(\mathbf{v}_i^{\Delta t})$.

6.2 Geodesic Paths via Harmonic Maps

We now seek to compute geodesic paths on general manifolds using the CPM-based harmonic map computation technique. The core insight was stated by Eells and Lemaire [45]: if $\dim(\mathcal{S}_1) = 1$, then harmonic maps are the geodesics of $\mathcal{S}_2$. Such a map will yield a closed or open geodesic on $\mathcal{S}_2$ depending on if $\mathcal{S}_1$ is closed (e.g., a circle) or open (e.g., a line segment), respectively. Our approach is therefore to compute a geodesic on the manifold $\mathcal{S}_2$ by computing a harmonic map from the 1D line segment $\mathcal{S}_1 = [0, 1]$ to $\mathcal{S}_2$. A line segment can be used for both open and closed $\mathcal{S}_1$ by enforcing Dirichlet BCs for open geodesics or periodic BCs for closed geodesics.

Our resulting algorithm requires only heat flow on the 1D line segment and the closest point projection $\mathrm{cp}_{\mathcal{S}_2}$ onto $\mathcal{S}_2$ — it inherits these attractive features from Algorithm 4. Since King and Ruuth [72] did not consider the geodesic problem, we discuss some intricacies in computing initial paths and stopping criteria in the following subsections.

For the line segment $x \in \mathcal{S}_1 = [0, 1]$, the Laplace-Beltrami operator is simply $\Delta_{\mathcal{S}_1} = \frac{\partial^2}{\partial x^2}$. Thus, standard 1D finite differences can be used to discretize the heat flow on $\mathcal{S}_1$. The

initial condition for the gradient descent flow is some path $\mathbf{u}^0(x) \in \mathcal{S}_2$. Starting from $k = 0$, the geodesic path is computed by iterating the following two steps:

(I) Solve $\dfrac{\partial \mathbf{v}}{\partial t} = \dfrac{\partial^2 \mathbf{v}}{\partial x^2}$, $\mathbf{v}(x, 0) = \mathbf{u}^k(x)$, for one time step of size Δt using explicit Euler.

(II) Project $\mathbf{v}(x, \Delta t)$ onto $\mathcal{S}_2$ via $\mathbf{u}^{k+1}(x) = \mathrm{cp}_{\mathcal{S}_2}(\mathbf{v}(x, \Delta t))$.

The notation in step (I) means to solve the heat equation independently for each component of $\mathbf{v}$. For open geodesics with endpoints $\mathbf{p}$, $\mathbf{q} \in \mathcal{S}_2$, Dirichlet boundary conditions $\mathbf{u}(0) = \mathbf{p}$ and $\mathbf{u}(1) = \mathbf{q}$ are imposed. For closed geodesics, we use periodic boundary conditions $\mathbf{u}(0) = \mathbf{u}(1)$ on the line segment to avoid complicating step (I) when $\mathcal{S}_1$ is curved (e.g., a circle).

6.2.1 Discretization

For efficiency, the embedding space $\mathbb{R}^{d_{\mathcal{S}_2}}$ for $\mathcal{S}_2$ is taken to be as small as possible, i.e., $d_{\mathcal{S}_2} = \dim(\mathcal{S}_2) + 1$. For example, $d_{\mathcal{S}_2} = 3$ if $\mathcal{S}_2$ is a surface (2D manifold). The line segment $\mathcal{S}_1 = [0, 1]$ is discretized using equally spaced grid points $x_i = ih$ for $i = 0, 1, \ldots, N + 1$ and $h = \frac{1}{N+1}$. The geodesic path $\mathbf{u} \in \mathcal{S}_2 \subseteq \mathbb{R}^{d_{\mathcal{S}_2}}$ is represented discretely as a polyline with vertices $\mathbf{u}_i = \mathbf{u}(x_i)$.

Step (I) of the algorithm is applied independently for each of the $d_{\mathcal{S}_2}$ dimensions of $\mathbf{u}$. Let u and v denote one of the particular $d_{\mathcal{S}_2}$ components of $\mathbf{u}$ and $\mathbf{v}$, respectively. Second-order centred differences are used to discretize $\partial^2/\partial x^2$. On iteration k, we set $\mathbf{v}_i = \mathbf{u}_i^k$, then apply one step of explicit Euler to the heat equation, with $\Delta t = 0.4h^2$:

$$v_i^{\Delta t} = v_i + \frac{\Delta t}{h^2}(v_{i-1} - 2v_i + v_{i+1}). \tag{6.5}$$

Then step (II) couples the $d_{\mathcal{S}_2}$ dimensions again for each vertex i of the path via $\mathbf{u}_i^{k+1} = \mathrm{cp}_{\mathcal{S}_2}(\mathbf{v}_i^{\Delta t})$.

6.2.2 Stopping Criteria

Different stopping criteria have previously been used for iterative geodesic algorithms. Perhaps the most obvious stopping criterion would monitor the change in length of the curve. However, as Martínez et al. [95, Section 3.3.1] discuss, the difference in lengths

between consecutive iterations can be small even when the iteration has not converged. We observe the same behaviour in practice for our method. Instead, we stop iterating if the movement of each individual vertex falls below a tolerance, i.e.,

$$\max_i \|\mathbf{u}_i^{k+1} - \mathbf{u}_i^k\| \leq \epsilon, \tag{6.6}$$

with $\epsilon = 10^{-5}$ for all examples. Our condition draws on the observation that individual vertices still move when the algorithm is far from converged, even when the length of $\mathbf{u}$ only changes slightly. Therefore, monitoring the motion of each vertex provides a more effective convergence indicator.

Yuan et al. [193] use the stopping criterion $\max \|\nabla E_L\| \leq \epsilon$, where E_L is defined below in (6.11). Our stopping criterion (6.6) is also related to the norm of our energy gradient. To see this, we first write our algorithm in one step as

$$\mathbf{u}_i^{k+1} = \mathrm{cp}_{\mathcal{S}_2}\Big(\mathbf{u}_i^k + \frac{\Delta t}{h^2}\left(\mathbf{u}_{i-1}^k - 2\mathbf{u}_i^k + \mathbf{u}_{i+1}^k\right)\Big). \tag{6.7}$$

Then Taylor expanding the closest point function about $\mathbf{u}_i^k$ gives

$$\mathbf{u}_i^{k+1} \approx \mathrm{cp}_{\mathcal{S}_2}(\mathbf{u}_i^k) + \mathbf{J}_{\mathrm{cp}_{\mathcal{S}_2}(\mathbf{u}_i^k)}\left(\frac{\mathbf{u}_{i-1}^k - 2\mathbf{u}_i^k + \mathbf{u}_{i+1}^k}{h^2}\right).$$

Since $\mathbf{u}_i^k \in \mathcal{S}_2$ we have that $\mathrm{cp}_{\mathcal{S}_2}(\mathbf{u}_i^k) = \mathbf{u}_i^k$ and $\mathbf{J}_{\mathrm{cp}_{\mathcal{S}_2}(\mathbf{u}_i^k)} = \Pi_{\mathcal{T}_{\mathbf{u}_i^k}\mathcal{S}_2}$. Therefore, after rearranging we have

$$\frac{\mathbf{u}_i^{k+1} - \mathbf{u}_i^k}{\Delta t} \approx \Pi_{\mathcal{T}_{\mathbf{u}_i^k}\mathcal{S}_2}\left(\frac{\mathbf{u}_{i-1}^k - 2\mathbf{u}_i^k + \mathbf{u}_{i+1}^k}{h^2}\right). \tag{6.8}$$

The term in (6.8) in brackets on the right-hand-side (RHS) approximates $\Delta_{\mathcal{S}_1}\mathbf{u}$. Hence, comparing with (6.2) we see that the RHS of (6.8) corresponds to $-\nabla E_H$. Thus $\|\mathbf{u}_i^{k+1} - \mathbf{u}_i^k\| \approx \Delta t \|\nabla E_H\|$.

6.2.3 Initial Path Construction

Since our method applies to general manifold representations via closest point queries, our path initialization should also. We explore two path initialization algorithms here: Dijkstra's algorithm in a computational tube $\Omega(\mathcal{S}_2)$ and rapidly-exploring random trees (RRT) [78].

For the first approach, we construct $\Omega(\mathcal{S}_2)$ using the memory and runtime efficient implementation detailed in Section 3.2 with $r_{\Omega(\mathcal{S}_2)} = 3h/2$. Dijkstra's algorithm is then used to compute a path between the nearest grid points in $\Omega(\mathcal{S}_2)$ to $\mathbf{p}$ and $\mathbf{q}$. We then replace the grid points $\mathbf{y}_i \in \Omega(\mathcal{S}_2)$ in Dijkstra's path with their previously computed $\text{cp}_{\mathcal{S}_2}(\mathbf{y}_i)$. Finally, we spatially adapt the initial path, splitting and collapsing edges until all edge midpoints $\mathbf{m}$ satisfy $\|\mathbf{m} - \text{cp}_{\mathcal{S}_2}(\mathbf{m})\| \leq \gamma$.

The blue initial paths in Figure 6.1 are computed with this approach for the torus, Nefertiti mesh, and DecoTetrahedron level set with $\gamma = 10^{-3}$. A grid spacing of $h = 0.1$ is used for both the torus and DecoTetrahedron and $h = 0.01$ is used for Nefertiti. The initial path for the parameterized surface in Figure 6.1 was not constructed using a general approach. It instead projected a circle onto the surface using $\text{cp}_{\mathcal{S}_2}$, which was done just to show our method can handle closed paths.

Using Dijkstra's algorithm in $\Omega(\mathcal{S}_2)$ provides generality with respect to the manifold representation, but it can also be more efficient even for meshes. Dijkstra's algorithm in $\Omega(\mathcal{S}_2)$ will be faster if h can be taken large enough compared to the mesh resolution. However, h must be small enough to give a sufficiently accurate initial path. Furthermore, the overhead time of the construction of $\Omega(\mathcal{S}_2)$ can also outweigh the benefit of using this approach as $h \to 0$. One approach to address this issue may be to use the spatially adaptive framework in Section 3.4. However, the second approach we explore here, RRT, eliminates the need for $\Omega(\mathcal{S}_2)$ altogether.

RRT was also used by Yuan et al. [193] for path initialization for manifolds represented as a level set. An illustration of RRT in $\mathbb{R}^2$ is shown in Figure 6.2. A tree $\mathcal{T}$ is initialized with the starting point $\mathbf{p}$ and then grown through an iterative process. On each iteration, a random point $\mathbf{x}_{\text{rand}}$ is generated and the nearest neighbour $\mathbf{x}_{\text{near}}$ in $\mathcal{T}$ is found. A new point $\mathbf{x}_{\text{new}}$ is then generated by stepping a distance δ in the direction of $\mathbf{x}_{\text{rand}}$ from $\mathbf{x}_{\text{near}}$. The point $\mathbf{x}_{\text{new}}$ is added to $\mathcal{T}$ if it lies in a valid region of space. Invalid regions of space can be due to obstacles or physical robot constraints, for example. RRT terminates when a point in $\mathcal{T}$ is within a distance δ to $\mathbf{q}$.

Yuan et al. [193] take their valid region of space to be a half tube of radius $r_{\mathcal{N}(\mathcal{S})}$ on the positive (exterior) side of the level set. They consider $\mathcal{S}$ and the boundary of the half tube to be obstacles. They then use the standard RRT method in the half tube. A final post-processing step projects the path generated in the half tube onto $\mathcal{S}$ by computing the closest points of the path vertices. Figure 6.3 (left) gives an example path computed using this implementation of RRT on the sphere (except with the full tube instead of just the positive half). We make some changes to the implementation of RRT of Yuan et al. [193], as discussed below, which allows for more general manifolds and decreases the number of

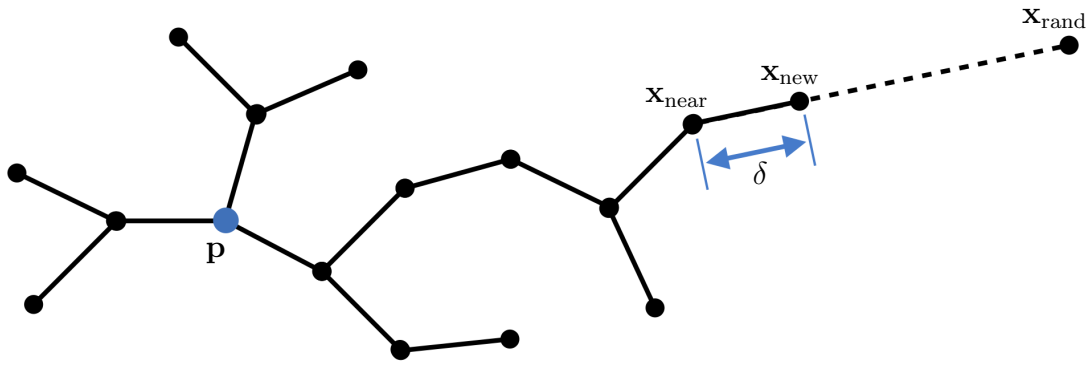


Figure 6.2: One iteration of the RRT algorithm where $\mathbf{x}_{\text{new}}$ is connected to $\mathbf{x}_{\text{near}}$ in the tree $\mathcal{T}$ originally grown from the start point $\mathbf{p}$. The point $\mathbf{x}_{\text{new}}$ is generated by moving a distance δ from $\mathbf{x}_{\text{near}}$ in the direction of $\mathbf{x}_{\text{rand}}$.

iterations without drastically increasing error.

Algorithm 5 summarizes our RRT implementation (note we abbreviate $\text{cp}_{\mathcal{S}_2}(\mathbf{x}_{\text{text}})$ as cp_{text}). We take the full tube $\mathcal{N}(\mathcal{S}_2)$ of radius $r_{\mathcal{N}(\mathcal{S}_2)}$ to be the valid region, which allows the algorithm to be applicable to open and/or unorientable manifolds. First, a random point in the valid region must be generated. A random point $\mathbf{x}_{\text{rand}}$ is generated in a ball of radius R that completely encloses $\mathcal{S}_2$, but is not necessarily in the valid region $\mathcal{N}(\mathcal{S}_2)$. Therefore, a point $\mathbf{x}$ is created by moving a random distance $r \in [-r_{\mathcal{N}(\mathcal{S}_2)}, r_{\mathcal{N}(\mathcal{S}_2)}]$ from the associated projected point cp_{rand} of $\mathbf{x}_{\text{rand}}$ in the manifold normal direction, which ensures $\mathbf{x} \in \mathcal{N}(\mathcal{S}_2)$. Instead of constructing the path in $\mathcal{N}(\mathcal{S}_2)$, we add closest points to $\mathcal{T}$ on each iteration, which keeps the path on $\mathcal{S}_2$. The nearest neighbour step, therefore, finds the point cp_{near} already added to $\mathcal{T}$ that is the nearest neighbour of $\mathbf{x}$. The point $\mathbf{x}_{\text{new}}$ is still constructed as in the original RRT algorithm by moving a δ distance from cp_{near} in the direction of $\mathbf{x}$, but if $\mathbf{x}_{\text{new}}$ is valid (i.e., $\mathbf{x}_{\text{new}} \in \mathcal{N}(\mathcal{S}_2)$), then cp_{new} is added to $\mathcal{T}$ instead of $\mathbf{x}_{\text{new}}$. In short, Yuan et al. [193] perform RRT in the half-tube and project onto $\mathcal{S}$ afterwards. We instead use the full tube to handle general manifolds, and incorporate projections on $\mathcal{S}$ during each iterative step of the algorithm for greater efficiency. Figure 6.3 (right) shows an example path computed using our RRT implementation on the sphere.

Table 6.1 compares the number of iterations and errors when constructing initial paths with the implementation of RRT of Yuan et al. [193] (but on the full tube) and ours. The test is performed on the unit sphere for 150 unique pairs of $\mathbf{p}$ and $\mathbf{q}$. The path is computed 100 times for each of the 150 pairs since RRT is a random algorithm. We used $\delta = 0.1$, $r_{\mathcal{N}(\mathcal{S}_2)} = 0.25$, and $R = 2$. The maximum number of iterations of our RRT implementation

Algorithm 5: Rapidly-exploring random trees on manifold $\mathcal{S}$

Given path start and end points $\mathbf{p}$, $\mathbf{q}$
Given max iterations K, step distance δ, tube radius $r_{\mathcal{N}(\mathcal{S})}$, enclosing radius R

Initialize tree $\mathcal{T}$.init($\mathbf{p}$)
while $k < K$ **do**
  $k \leftarrow k + 1$
  Set $\mathbf{x}_{\text{rand}}$ to a random point in ball of radius R enclosing $\mathcal{S}$
  $\mathbf{x} \leftarrow \text{cp}_{\text{rand}} + r\left(\mathbf{x}_{\text{rand}} - \text{cp}_{\text{rand}}\right) / \|\mathbf{x}_{\text{rand}} - \text{cp}_{\text{rand}}\|$, for random $r \in [-r_{\mathcal{N}(\mathcal{S})}, r_{\mathcal{N}(\mathcal{S})}]$
  $\text{cp}_{\text{near}} \leftarrow \text{NearestNeighbour}(\mathcal{T}, \mathbf{x})$
  $\mathbf{x}_{\text{new}} \leftarrow \text{cp}_{\text{near}} + \delta\left(\mathbf{x} - \text{cp}_{\text{near}}\right)$
  if $\|\mathbf{x}_{\text{new}} - \text{cp}_{\text{new}}\| < r_{\mathcal{N}(\mathcal{S})}$ **then**
    $\mathcal{T}$.AddVertex(cp_{new})
    $\mathcal{T}$.AddEdge($\text{cp}_{\text{near}}, \text{cp}_{\text{new}}$)
    if $\|\text{cp}_{\text{new}} - \mathbf{q}\| \leq \delta$ **then**
      return $\mathcal{T}$
    end
  end
end

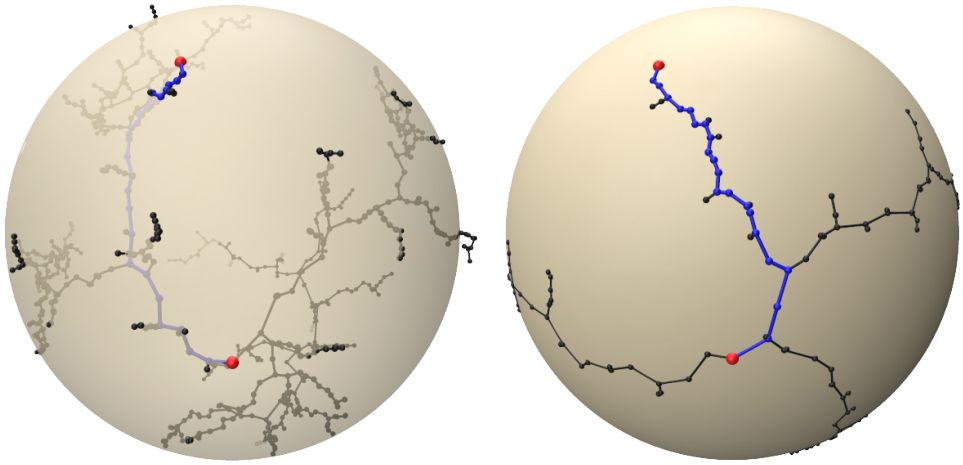


Figure 6.3: An initial path (blue) between the red points computed using RRT as discussed by Yuan et al. [193] (left) and our improved RRT implementation (right). The full tree $\mathcal{T}$ computed by each algorithm is shown in black. Note that Yuan et al. [193] projects the final blue path onto the surface with $\text{cp}_{\mathcal{S}}$, but the final post-projection path is not shown here.

over all tests is 67% of the standard RRT with nearly the same error. The error in the path is defined as

$$\text{Error} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{u}_i - \text{cp}_{\mathcal{P}}(\mathbf{u}_i)\|\,, \tag{6.9}$$

where $\text{cp}_{\mathcal{P}}$ is the closest point function of the exact geodesic through $\mathbf{p}$ and $\mathbf{q}$. For a sphere, the exact geodesic is the great circle intersecting $\mathbf{p}$ and $\mathbf{q}$. Our mean number of iterations over all tests is 85% of the standard RRT, but the median has a 2% increase. There is a slight increase in error for the mean and median, but this error is not detrimental since the resulting path is only used to initialize our algorithm.

6.3 Comparison to Yuan et al. [193]

One variant of the method of Yuan et al. is similar to ours. However, their perspective is based on the length minimization property of geodesics. For a continuously differentiable

Table 6.1: Iterations and errors for RRT path initialization on the unit sphere.

	Method	Max	Min	Mean	Median
Iterations	Yuan et al.	50,363	2	3221	1217
	Ours	33,570	2	2740	1240
Errors	Yuan et al.	0.7477	0.0044	0.2157	0.1832
	Ours	0.7476	0.0045	0.2281	0.1931

curve $\gamma : [0, 1] \to \mathcal{S}_2$ the length of the curve is given by

$$\int_0^1 \|\gamma'(t)\| \, dt. \tag{6.10}$$

Yuan et al. instead consider minimizing the energy functional

$$E_L(\gamma) = \int_0^1 H(\|\gamma'(t)\|) \, dt, \tag{6.11}$$

since it has the same critical points as the length functional (6.10). The function $H(s)$ must be convex and satisfy $H'(s) > 0$ and $H''(s) \geq 0$. Yuan et al. minimize $E_L(\gamma)$ using LBFGS [135] and show (albeit with a simple example of shortening an "S"-shaped curve in the plane) that $H(s) = s^2$ and $H(s) = e^{s^2} - 1$ require far fewer iterations than $H(s) = s$ (which corresponds to the actual length functional (6.10)). They chose $H(s) = e^{s^2} - 1$ since slightly fewer iterations were needed compared to $H(s) = s^2$. In our comparisons in Section 6.4 we use $H(s) = e^{s^2} - 1$ for the Yuan et al. [193] method also.

In the discrete setting, there are subtle differences between our method and theirs with $H(s) = s^2$. For this specific case, the ∇E_L used in LBFGS is a scalar multiple of the finite difference scheme we use for $\partial^2/\partial x^2$ in (6.5). The $\partial^2/\partial x^2$ term is related to the gradient of E_H, but only the specific combination in steps (I)-(II) give a first-order consistent discretization of the gradient descent flow (see King and Ruuth [72, Section 3.2] for the proof).

Importantly, our approach does not require computing and applying the projection operator $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$. Yuan et al. use $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$ to project ∇E_L onto the tangent space of $\mathcal{S}_2$; otherwise their minimization does not converge. Even with these projected gradients their method can produce geodesics that are not strictly in $\mathcal{S}_2$. Therefore, after each iteration, they put the vertices of their path on $\mathcal{S}_2$ by computing closest points. They give no theoretical justification for using $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}(\nabla E_L)$ or $\text{cp}_{\mathcal{S}_2}$, whereas, our approach has firm theoretical justification for not needing $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$ and computing $\text{cp}_{\mathcal{S}_2}$ in step (II).

6.4 Numerical Results

Our approach to compute geodesic paths is compared with the methods of Yuan et al. [193] and Sharp and Crane [152] in this section. We first focus on just the runtime required to shorten an initial path to a geodesic on different surfaces provided as meshes. Then a more thorough comparison of the full pipeline to compute geodesic paths is done on a single surface, the unit sphere. The closest points to the meshes are computed using `fcpw` [139]. The CPU version of `fcpw` is used, but its GPU counterpart may be able to provide more efficient closest point computation for meshes. Enoki vectorization [63] within `fcpw` was also not used since experiments were performed on a Macbook Air with M1 chip, which is not supported by Enoki.

6.4.1 Path Shortening Comparison

The same initial path is used for all three methods computed using Dijkstra's algorithm directly on a mesh. Six different meshes are used in this comparison: Blub the fish, a higher resolution fish, a Möbius strip, a sphere, a dragon, and a Buddha statue. The meshes have approximately 10K, 173K, 328K, 655K, 3.6M, 5M vertices, respectively. Roughly 2000, 1000, 667, 500, 400, and 333 different paths are computed between vertices $\mathbf{p}$ and $\mathbf{q}$ on the meshes, respectively.

Figure 6.4 (top row) gives histograms of the shortening time normalized with respect to Dijkstra's algorithm runtime (left) or the shortening time of our method (right). From Figure 6.4 (top left), we see that our method and that of Yuan et al. [193] take longer than Dijkstra's algorithm for most of the geodesics computed. There is, however, a cluster of geodesics computed by our algorithm that require a similar amount of time to the method of Sharp and Crane [152], which is faster than Dijkstra's algorithm. A more direct comparison of our method to the other two is provided in Figure 6.4 (top right) where the runtimes of Yuan et al. [193] and Sharp and Crane [152] are normalized by the runtime of our method. The method of Yuan et al. [193] can be up to approximately $1000\times$ slower than ours, but never faster. In contrast, the method of Sharp and Crane [152] is always faster by at most approximately $500\times$.

Avoiding computation of $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$ is the main reason we enjoy faster runtimes than the Yuan et al. [193] method. This result is significant since their method was faster than 9 others they compared against (only Dijkstra's algorithm was faster, but it does not provide a smooth path due to its restriction to edges). Yuan et al. [193] need the normal vectors of each face in the mesh that the polyline goes through during the iterations to apply $\Pi_{T_{\mathbf{u}}\mathcal{S}_2}$.

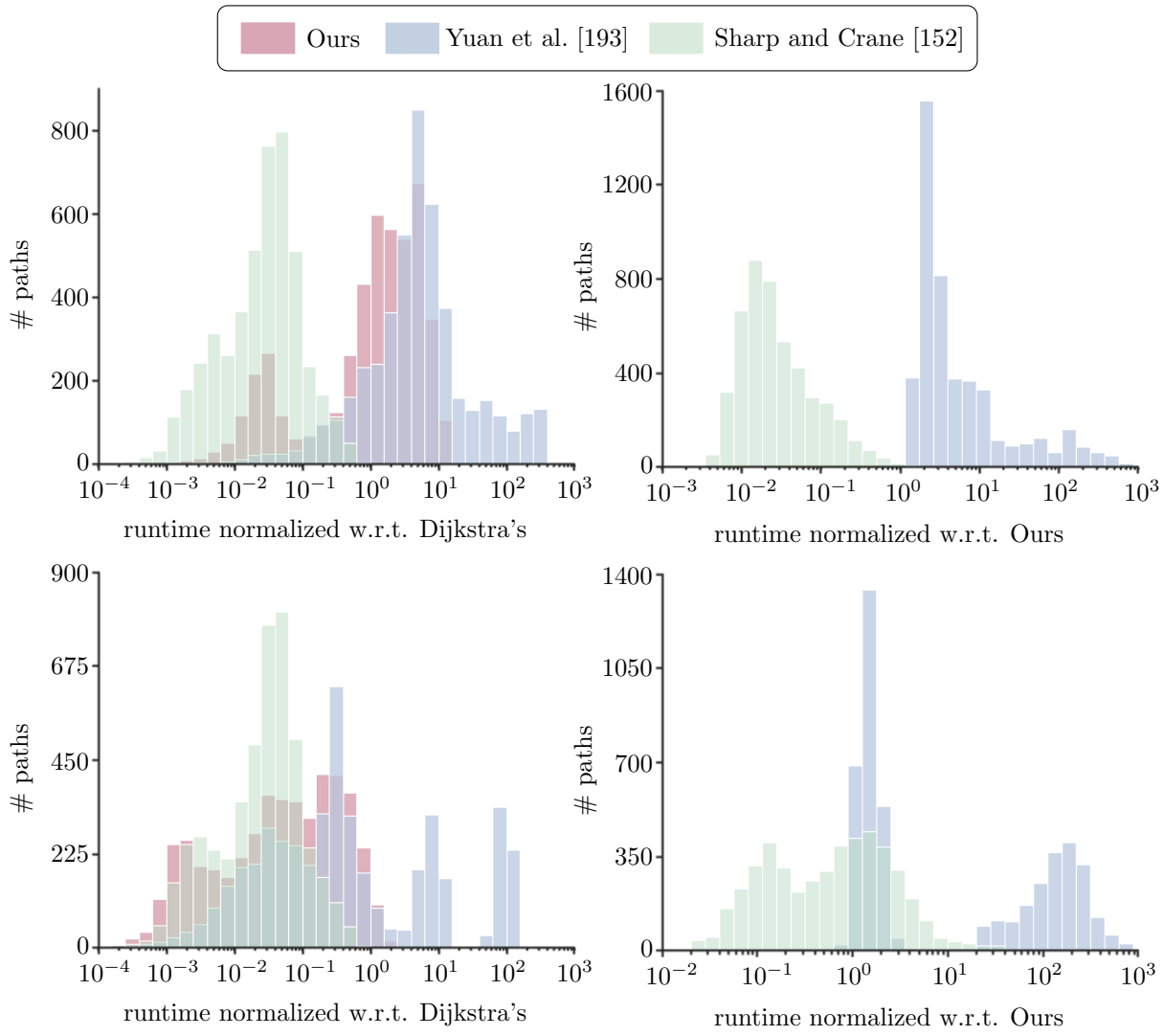


Figure 6.4: Histograms of path shortening timings when computing geodesics paths with 6 different meshes. Top row: All algorithms use the same initial path from Dijkstra's algorithm. Bottom row: The initial path is coarsened for our method and the method of Yuan et al. [193]. Left column: Shortening runtime normalized by the runtime of Dijkstra's algorithm for path initialization. Right column: Shortening runtimes normalized by the shortening runtime of our method.

In our implementation of their approach, we compute the per-face normals for the whole mesh as a preprocessing step. If many geodesic paths are to be computed, this approach amortizes the cost of the face normal computation. For the meshes used in our tests the cost of normal computation is 0.07, 1.3, 3.5, 4.9, 29.2, and 35.9 seconds for the fish, higher resolution fish, Möbius strip, sphere, dragon, and Buddha statue, respectively. The cost of computing the normal vectors can therefore be a large bottleneck for the Yuan et al. [193] method depending on how many paths are to be computed and the number of faces in the mesh. If only one geodesic path is needed, it would be possible to speed up the normal computation by only computing on faces in the mesh that the polyline travels through during minimization.

Note also that computing the per-face normals to be used with the Yuan et al. [193] method does not generalize to other manifold representations. Representation specific methods could be used, but this complicates the implementation. A single implementation for the normal vectors could be accomplished using CPM (see [138, Appendix A] and/or Section 4.2.7), but the added cost of constructing a computational tube around the manifold and interpolating the normals may be disadvantageous. One could instead compute the normal when the polyline path vertices are put back on the manifold via the closest point computation. However, this will be inaccurate when a path vertex is close to the manifold, since the normal direction vector will have a small norm. Our method does not need normal vectors, which eliminates this extra computation.

The method of Sharp and Crane [152] is guaranteed to return a geodesic polyline consisting of vertices and edges that all lie strictly within the triangle mesh surface. For some applications this is an important feature, however, this is not always necessary. Our algorithm and that of Yuan et al. [193] only guarantee that the polyline vertices are in the mesh (while the edges may not be), since they are designed for general manifold representations where $\mathcal{S}$ is not necessarily piecewise planar. We can therefore coarsen the initial path polyline to speed up our algorithms. Figure 6.4 (bottom row) shows histograms for the same tests above (top row), but with edges of the polyline collapsed if the midpoint of the resulting edge is still a distance less than 10^{-3} away from $\mathcal{S}$. All methods are now faster or about the same runtime as Dijkstra's algorithm, see Figure 6.4 (bottom left), except some of the geodesics computed by the Yuan et al. [193] method.

Figure 6.4 (bottom right) shows that our method has roughly the same speedup over the Yuan et al. [193] method with the non-coarsened initial path (but due to the Yuan et al. [193] method converging less often, discussed below, there is a larger cluster above $100\times$ slower than ours). The figure also shows that our method with the coarsened initial path can now be faster than the method of Sharp and Crane [152] and is always less than $100\times$ slower.

Table 6.2: Path differences between the three methods for all paths computed in Figure 6.4.

Methods Compared	Initial Path	Max	Mean	Median
Ours & Sharp and Crane [152]	Dijkstra's	0.3012	0.0177	0.0033
	Coarsened	0.1460	0.0027	0.0011
Ours & Yuan et al. [193]	Dijkstra's	0.0752	0.0062	0.0018
	Coarsened	0.0589	0.0007	0.0002
Yuan et al. [193] & Sharp and Crane [152]	Dijkstra's	0.2743	0.0138	0.0018
	Coarsened	0.1456	0.0025	0.0009

Interestingly, the difference between paths computed by all three algorithms decreases when the initial path is coarsened. To illustrate this effect, we compute the path difference using (6.9), but with the exact geodesic $\mathcal{P}$ replaced by one of the two polyline paths being compared. Table 6.2 gives the path differences between all pairs of methods for the Dijkstra's initial path and the coarsened version. As expected, the difference between our paths and the paths of Yuan et al. [193] is smaller than that of those methods compared with the method of Sharp and Crane [152]. The mean and median differences are quite small, but the max difference can become quite large, especially for ours and Yuan et al. [193] compared to Sharp and Crane [152].

Another way our method shows superior runtimes compared to Yuan et al. [193] is due to superior robustness. A maximum of 50,000 iterations was used for both methods. With the initial Dijkstra's path, the method of Yuan et al. [193] converged for $4102/4690 = 87.5\%$ of the tests before the max iteration count was reached, whereas, ours converged for all 4690 tests. With the coarsened initial path (same initial path for both), the method of Yuan et al. [193] only converged for $2636/4690 = 56.2\%$ of the tests and ours converged for $4681/4690 = 99.8\%$ of the tests. These tests that failed to converge are responsible for the cluster of tests towards the right of the histograms in Figure 6.4 for the Yuan et al. [193] method.

A final aspect we examined is how the shortening time scales for each method as the number of points in the polyline path increases. Figure 6.5 plots the shortening time versus the number of points in the (not coarsened) Dijkstra's initial path. The tests that did not converge with the Yuan et al. [193] method are removed; they would otherwise skew the scaling result since all failed tests took the maximum 50,000 iterations. A line of best fit is computed for each method, which shows that the method of Sharp and Crane [152] scales the best with a slope of 2.8×10^{-5}. Our method scales about 26 times worse than Sharp and Crane [152], but the method of Yuan et al. [193] is much worse by a factor of about 360. Figure 6.5 verifies the importance of reducing the number of points in the path

to speed up our method. It would be interesting to explore other adaptive strategies to improve performance in future work.

6.4.2 Full Pipeline Comparison

The runtime to shorten the initial path to a geodesic is not always the bottleneck of the overall geodesic computation. Of course, if geodesic computation is part of a larger application pipeline that provides the initial path, only the shortening time needs to be considered. However, as pointed out by Sharp and Crane [152, Section 5.2.1], the cost of shortening the initial path for most methods they examined was faster than constructing the initial path with Dijkstra's algorithm. We have also shown in Figure 6.4 that our method and that of Yuan et al. [193] (when convergent) can run faster than Dijkstra's algorithm (especially with a coarsened initial path).

In practice, features such as accuracy, robustness, and types of input/output are more important. Our method is more robust than the method of Yuan et al. [193]. Our method also allows more general representations of $\mathcal{S}$ to be input than any other method (including Yuan et al. [193]) by only requiring closest point queries. In this section, we also explore the runtime cost of the full pipeline needed to compute geodesics.

Other runtime costs in geodesic computation are the initial path construction and the setup of the discrete representation. Since we are not restricted to a mesh, we can use other approaches instead of Dijkstra's algorithm for path initialization. A mesh is usually saved as a list of vertices and faces which must be read into memory before computation proceeds. Whereas, a closest point function can be represented directly in the code in different ways, e.g., analytically or as an optimization problem (see Appendix A). Obviously, the cost of the discrete setup will be the same for our method and mesh-based methods if the closest point function is computed with a mesh. In addition, the discrete setup runtime is less important if geodesics are computed as part of a larger application pipeline. The cost of the discrete setup can also be amortized if many geodesic paths are required on the same $\mathcal{S}$.

Figure 6.6 compares our method using RRT path initialization against Sharp and Crane [152] using Dijkstra's path initialization on the unit sphere. Different resolution meshes are used with the method of Sharp and Crane [152] to show the resolution dependence. The different resolution meshes of the sphere are constructed with 4:1 subdivision of a base mesh (with 162 vertices) and projecting new vertices onto the sphere using $\text{cp}_{\mathcal{S}}$. Our method uses exact closest point queries for the sphere (which also requires a small but nonzero discrete setup cost).

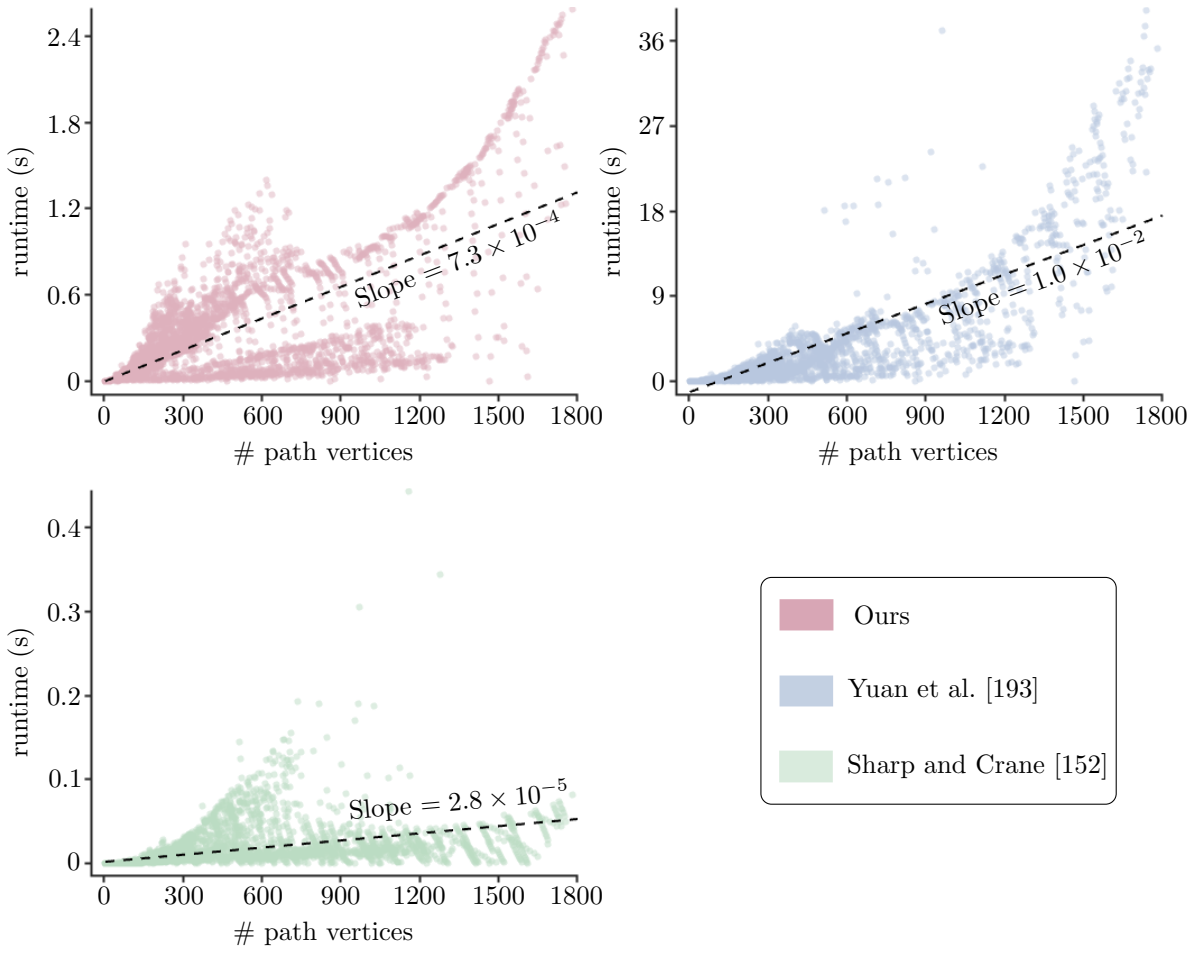


Figure 6.5: The runtime scaling of the three methods with respect to the number of vertices in the path for the paths computed in Figure 6.4 (top row).

Table 6.3: Path error for all the geodesic paths computed for Figure 6.6.

Method	Max	Mean	Median
Ours	0.0645	0.0057	0.0028
Base Mesh	0.1517	0.0209	0.0122
Subdivide 5	0.0109	0.0012	0.0003
Subdivide 9	0.0108	0.0012	0.0003
Subdivide 10	0.0108	0.0011	0.0003

Figure 6.6 (top row) gives histograms for the shortening runtime normalized by the path initialization runtime (left) and by our method's runtime (right). The histogram for the shortening runtime normalized by the path initialization runtime (top row, left) has a wider spread for our method compared to Sharp and Crane [152]. Two correlated factors are responsible for this. First, the RRT algorithm has a wide range of iteration counts (see Table 6.1). Second, the wide range of RRT iterations directly corresponds to a wide range in the number of points in the initial path. As shown in Figure 6.5, our method scales with the number of points in the path. Figure 6.6 (top row, right) shows that, for finer resolution meshes, our method can be up to $10,000\times$ faster than Sharp and Crane [152].

Figure 6.6 (bottom row) gives histograms for the shortening + path initialization runtime (left) and total runtime including the discrete setup (right), both normalized by our method's runtime. Our method can be up to approximately $100,000\times$ faster than Sharp and Crane [152] when considering the path initialization runtime also (left). The histograms become more skewed to the right when the total runtime is considered (right), corresponding to our method being faster more often in this scenario.

Table 6.3 gives the max, mean, and median error over all the tests from Figure 6.6 computed using (6.9). Our method has higher errors than Sharp and Crane [152] with finer resolution meshes (but lower errors than with the base resolution mesh), but only by a factor of about 5-10.

6.5 Summary and Future Work

An algorithm to compute geodesics paths on general manifold representations was presented in this chapter. We can handle more manifold representations than any previous method, since only closest point queries to the manifold are required. We compared our method with two SOTA methods: the work of Yuan et al. [193] (SOTA for representation

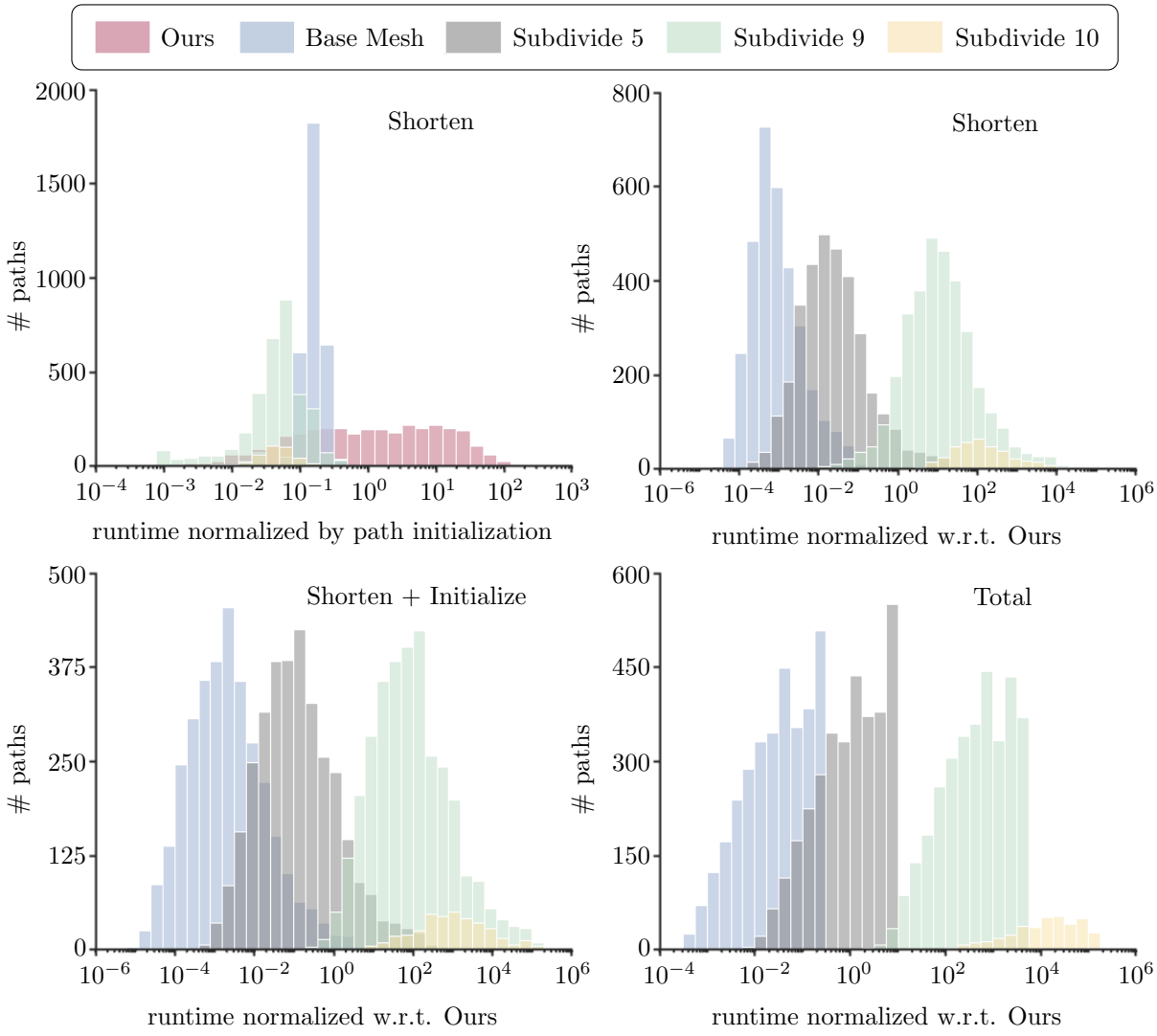


Figure 6.6: Runtime comparison of our method and the method of Sharp and Crane [152] for the full pipeline to compute geodesics.

generality) and Sharp and Crane [152] (SOTA for path shortening runtime on triangle meshes). We showed that our method is more robust and faster (up to $1000\times$) than the method of Yuan et al. [193]. When initializing the path using Dijkstra's algorithm our method can be up to $500\times$ slower than the method of Sharp and Crane [152]. However, when using the RRT method for path initialization our approach can be up to $100,000\times$ faster than the method of Sharp and Crane [152] on high-resolution meshes.

Our algorithm is an ideal candidate for parallelization. The closest point computation in step (II) of the algorithm can be performed independently for each vertex of the polyline path, while step (I) only requires data from the two neighbouring vertices. When the path contains a large number of vertices, parallelizing our method would allow superior runtime scaling compared to the SOTA.

Further improvement in runtime of our method can be achieved by optimizing the RRT path initialization implementation. Currently, it is inefficient since a brute-force nearest neighbour search is used (checking all points in the tree $\mathcal{T}$). However, incremental distance algorithms could speed up the nearest neighbour search, as mentioned in [77, Section 2]. There have also been many extensions of RRT that could be explored to provide further improvement of our approach [176, 189].

Chapter 7

Future Work for CPM

Previous to the work in this thesis, the representation of objects using closest points showed great potential in terms of generality. Closest points can be computed for a vast array of discrete object representations and these objects can have diverse characteristics, i.e., manifold or nonmanifold, open or closed, orientable or not, and of any codimension or even mixed codimension. The ability to solve general classes of PDEs via CPM further added to the potential of representing objects with closest points. However, limitations existed that made closest point representations seem impractical when applied to problems in geometry processing (and computer graphics in general) due to CPM's inability to handle interior BCs, and the fine level of detail objects exhibit that we are accustomed to in this field.

This thesis focused on multiple ways of addressing these limitations, which arise mainly due to the use of an embedding space. However, the use of the embedding space is vital; the concept of closest points exists only if there is space surrounding the object. Furthermore, the simplicity of CPM is due to the ability to solve PDEs on curved objects using Cartesian numerical methods in the embedding space. Fortunately, only a relatively small portion of the embedding space $\mathbb{R}^d$ is required, i.e., a tubular neighbourhood of $\mathcal{S}$.

First, to handle objects with fine details, we showed in Chapter 3 that the grid-based CPM theoretically scales with $\dim(\mathcal{S})$ (rather than d) and developed a technique to construct the computational tube $\Omega(\mathcal{S})$ with this optimal scaling in memory and runtime. We also created a spatial adaptivity framework to allow the tube radius to vary over $\mathcal{S}$ and concentrate the DOFs where they are needed most. In Chapter 4, we devised an accurate and general approach to handle interior BCs with CPM, which allowed us to handle applications from geometry processing that were previously impossible. Our further development of CPM with a Monte Carlo method (Chapter 5) provided common advantages of

volumetric Monte Carlo methods, such as view-dependent solutions of PDEs. (Note that Monte Carlo methods are not a "silver bullet" since they come with tradeoffs compared to grid-based methods, which also exist for us.) Finally, in Chapter 6, we introduced a method to compute geodesic paths that only requires closest point queries (but, unlike earlier chapters, does not involve solving PDEs on curved objects). In this chapter, we discuss other outstanding limitations and provide some insight into possible remedies.

7.1 Spatial Grid Resolution

Existing CPM theory assumes a unique closest point function $\text{cp}_{\mathcal{S}}$. As discussed in Section 2.1, $\text{cp}_{\mathcal{S}}$ is unique for $\mathbf{x}$ in a tubular neighbourhood $\mathcal{N}(\mathcal{S})$ of constant (uniform) tube radius $r_{\mathcal{N}(\mathcal{S})}$ satisfying $r_{\mathcal{N}(\mathcal{S})} < \text{reach}(\mathcal{S})$. Therefore, in the discrete setting, the computational tube-radius $r_{\Omega(\mathcal{S})}$ must be less than $\text{reach}(\mathcal{S})$ also. Rearranging (3.4) means h must satisfy

$$h < \frac{\text{reach}(\mathcal{S})}{\sqrt{(d-1)\left(\frac{p+1}{2}\right)^2 + \left(q + \frac{p+1}{2}\right)^2}}$$

to ensure a unique $\text{cp}_{\mathcal{S}}$ on $\Omega(\mathcal{S})$. However, in practice CPM can often be used successfully with larger h, depending on the PDE to be solved and the accuracy requirements of the application.

In many graphics applications the visual appearance is paramount. Consider a diffusion curves example on a dragon [171]. Figure 7.1 shows the resultant surface colouring at different grid resolutions. Artifacts can be observed for $h = 0.0125$: unintended blending of blue and red on the head yields purple, while the zoomed-in dragon scale incorrectly shows hints of blue appearing in a red region. For $h = 0.003125$ (and arguably $h = 0.00625$) the result has converged to a visually acceptable, artifact-free result. However, the h required to give a unique $\text{cp}_{\mathcal{S}}$ for the dragon is $h < 1.28 \times 10^{-6}$. This assumes no thin bottlenecks exist, i.e., $\text{reach}(\mathcal{S})$ is computed based on only principal curvatures (computed directly on the mesh using `geometry-central` [154]). Instead, one could estimate $\text{reach}(\mathcal{S})$ using the medial axis approximation discussed in Section 5.2.1. Ultimately, an h based on an estimate of $\text{reach}(\mathcal{S})$ can be unnecessarily restrictive, since Figure 7.1 shows that $h \gg \text{reach}(\mathcal{S})$ still produces satisfactory results.

Therefore, h has always been empirically determined for practical applications of CPM, but this is a limitation that costs the user time. A priori determination of a "correct" grid spacing h is an open challenge: it will require knowledge about the specific PDE to be solved, the manifold it is to be solved on, and the accuracy requirements (perceptual,

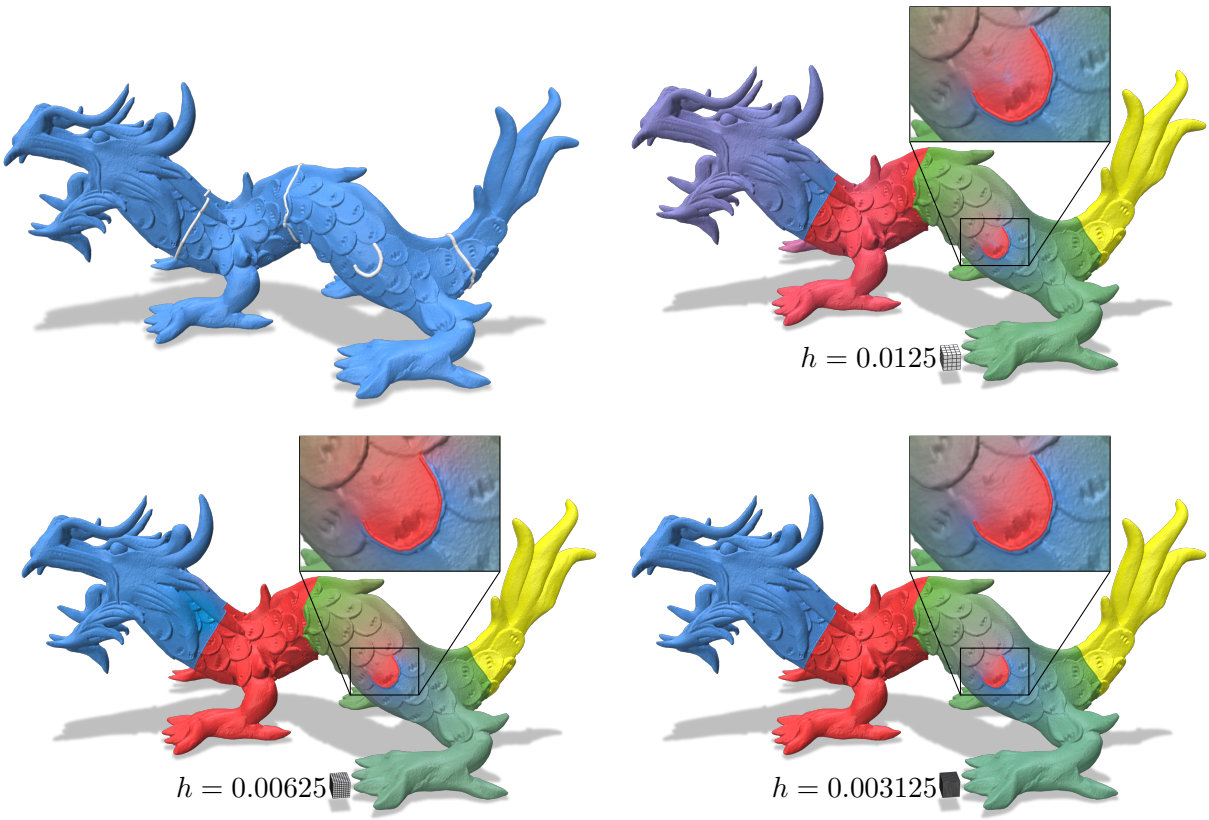


Figure 7.1: Results for three grid resolutions used to solve a diffusion curves problem to colour the surface of a dragon. The resolution is illustrated by a small block of grid cells (best viewed by zooming). The $\text{cp}_{\mathcal{S}}$ are computed from a triangulation, while the $\text{cp}_{\mathcal{C}}$ are from polylines.

numerical, etc.) of the user. In general, a priori error estimation has been rare in computer graphics applications. A notable exception is the p-refinement FEM scheme of Schneider et al. [144], which uses an a priori error estimate based on the geometry of the (volumetric) domain.

Error estimates would also be useful for our spatial adaptivity framework (Section 3.4) to determine h_m for each subset $\Omega(\mathcal{S}_m)$ of the adaptive computational tube. It would be interesting to explore whether the uniqueness requirement of $\text{cp}_{\mathcal{S}}$ can be relaxed for adaptive tubes as well, i.e., if $r_{\Omega(\mathcal{S}_m)} < \text{LFS}(\text{cp}_{\mathcal{S}_m}(\mathbf{x}_i))$ can be violated and still produce satisfactory results.

In scenarios where $r_{\Omega(\mathcal{S})} < \text{reach}(\mathcal{S})$ is violated, neighbouring grid points can receive data through the CP extension from geodesically distant parts of $\mathcal{S}$ (when the edge between neighbours intersects with $\text{med}(\mathcal{S})$, see Figure 7.2 (left)). Data being similar on these geodesically distant parts of $\mathcal{S}$ is one possible explanation for why $r_{\Omega(\mathcal{S})} < \text{reach}(\mathcal{S})$ can be violated. Exploring the use of duplicate DOFs on either side of the medial axis could ensure data is CP extended from the correct part of $\mathcal{S}$ (similar to how Chapter 4 distinguishes different sides of an IBC). This would result in a nonmanifold grid similar to the work of Mitchell et al. [105] and Chuang [27]. However, more than just the closest points would be necessary; the points of intersection $\mathbf{p_{i,j}}$ between rays emanating from $\mathbf{x}_i$ that intersect $\mathcal{S}$ orthogonally would be needed (see Figure 7.2 (right)). This is a generalization of closest points since the $\mathbf{p}_{i,j}$ that gives the minimum distance over all j between $\mathbf{x}_i$ and $\mathbf{p}_{i,j}$ is the closest point, i.e., $\text{cp}_{\mathcal{S}}(\mathbf{x}_i) = \mathbf{p}_{i,j^\star}$ with $j^\star = \arg\min_j \|\mathbf{x}_i - \mathbf{p}_{i,j}\|$.

7.2 Additional PDEs, CPM Convergence, and Manifold Smoothness

We primarily focused on Poisson and diffusion problems, but the grid-based CPM has been applied to numerous other PDEs (see Section 2). In principle, our approach to allow efficient scaling of CPM to larger problems (Chapter 3) and our IBC enforcement (Chapter 4) should readily extend to those other PDEs. This was confirmed for our IBC treatment with reaction-diffusion equations in Section 4.4.5. Extending CPM to approximate previously unexplored operators, such as the relative Dirac operator [81] or the connection Laplacian [155, 47], would allow other geometry processing applications to benefit from CPM. Furthermore, our Monte Carlo CPM in Chapter 5 is only applicable to the Poisson equation (and some other closely related PDEs, e.g., screened-Poisson equation, using generalizations detailed in [164]) with Dirichlet BCs. This is not an inherent limitation of our method

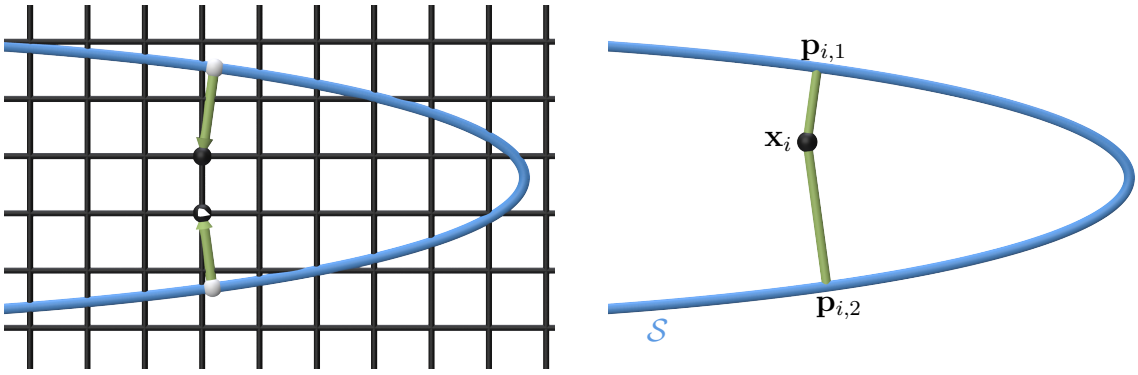


Figure 7.2: Left: When $r_{\Omega(\mathcal{S})} < \text{reach}(\mathcal{S})$ is violated, neighbouring grid points (black points) can be assigned data from their closest points (white) that are far apart in terms of geodesic distance. Right: Two segments (green) of the rays starting from $\mathbf{x}_i$ that intersect $\mathcal{S}$ (blue) orthogonally. The points of intersection $\mathbf{p}_{i,1}$ and $\mathbf{p}_{i,2}$ lie at the intersection of the green and blue curves.

but a limitation of (volumetric) Monte Carlo methods in general; different Monte Carlo methods must be developed for different classes of PDEs. For example, the *walk-on-stars* method [142] extends the (volumetric) walk-on-spheres method [110] to handle Neumann BCs.

Currently, convergence of the numerical scheme for CPM has not been proven for the first three variants detailed in Sections 2.2.1–2.2.3. However, in practice, those variants of CPM do indeed converge. Chen and Macdonald [24] provide proof for the convergence of the fourth variant (Section 2.2.4), but only for closed curves embedded in 2D. They remark that their proof does not extend to even 3D. A complete convergence theory for all CPM variants with any type of manifold embedded in any dimension is a difficult but important problem for future work.

Most CPM work and theory is based on smooth manifolds. However, WENO interpolation has been used to improve the grid-based CPM for nonsmooth surfaces (e.g., surfaces with sharp features) [87, 7]. Sharp features have zero local feature size, so they can never be fully resolved, even as $h \to 0$. The CP extension can cause neighbouring grid points in $\Omega(\mathcal{S})$ to be assigned data from parts of $\mathcal{S}$ that are geodesically distant, i.e., from either side of the sharp feature. The ability of WENO interpolation to adapt its stencil allows data from the correct side of the sharp feature to be used more often. Cheung et al. [25] used duplicated DOFs (similar to our IBC work in Chapter 4) near the sharp feature with a radial-basis function discretization of CPM. However, such discretizations can suffer from

ill-conditioned linear systems. Therefore, it would be interesting to instead explore the alteration of stencils (similar to our IBC approach in Chapter 4) for the grid-based CPM near sharp features to use data from the correct side of the sharp feature. In this context, the BC curve $\mathcal{C}$ would instead be the sharp feature and the PDE is still imposed on $\mathcal{C}$ instead of a BC.

The theoretical restriction of smoothness also applies to the curve $\mathcal{C}$. Therefore, our IBC approach is theoretically restricted to curves without kinks or intersections. In practice, we are still able to obtain the expected result when $\mathcal{C}$ has sharp features or intersections, e.g., Figure 4.9 involves many intersecting curves (in the band of the headdress) that also create sharp corners. Similarly, CPM gives expected results in practice for mixed-codimensional objects as seen in Figure 4.10 where sharp features are present when differing codimensional pieces meet (one does however observe a decrease in the empirical convergence order). The development of a sound theoretical understanding of CPM's behaviour near sharp features and intersections is interesting future work.

7.3 Neural Representation

Using neural networks to encode the closest point representation of objects would provide a non-discrete (continuous everywhere except on the medial axis) formulation that further broadens research directions. There has been much recent work involving neural geometry processing with different representations, e.g., meshes [184], voxelizations [180] and signed distance fields (SDFs) [190].

A neural closest point representation is well suited for many geometry processing tasks, which alleviates the need to convert a neural representation into a discrete representation (e.g., a mesh) to apply classical geometry processing algorithms. Geometric quantities of the object, such as curvatures, normals, tangent spaces, first and second fundamental forms, are easily computed using automatic differentiation of neural closest point representations. The gradient, divergence, Laplace-Beltrami operator, and other higher-order differential operators of scalar/vector fields on the neural closest point representation can be computed using ideas from CPM. This would provide the ability to solve PDEs in a discretization-free manner.

Similar recent work by Williamson and Mitra [185] used a spherical embedding of genus-0 surfaces, called spherical neural surfaces (SNSs). They map genus-0 surfaces to the unit sphere and train a multi-layer perceptron to give a continuous representation of the spherical parametrization. They show how to compute geometric quantities such as

curvatures, normals, and first and second fundamental forms, with SNSs. Furthermore, the computation of surface gradients, surface divergence, and the Laplace-Beltrami operator of scalar/vector fields defined on SNSs is shown.

The approach of Williamson and Mitra [185] is restricted to closed, orientable, genus-0 manifolds embedded in 3D. They mainly show the construction of an SNS from a triangle mesh but also show one example from a neural SDF (albeit still utilizing an intermediate mesh). The use of a closest point representation instead would allow much more general objects while retaining the ability to compute all the geometric quantities and differential operators.

Further interesting directions of future work involving the use of a neural closest point representation include a wide range of applications such as rendering neural closest point representations directly, deforming them, segmenting or classifying shapes, etc.

7.4 Conclusion

Realistic and complex digital objects are commonplace in our daily experience. They are created in numerous ways, e.g., from data captured of real-world objects, manually with the aid of 3D modelling software, and using generative AI. Due to the many ways objects are created, vastly different discrete representations exist, which causes wasted efforts in the research and development community; time would be better spent creating novel and fundamental methods than adapting existing methods to be compatible with another object representation.

Therefore, it is paramount that a single representation is found that has the ability to represent any object we desire and can be used to accomplish any application. We believe that the *closest point representation* has the potential to satisfy these requirements. The closest point representation can handle objects that are embedded in any dimension. Even though we focused on curved objects embedded in a higher-dimensional space, solid objects with irregular boundaries can also be represented with codimension zero. Even mixed-codimensional objects can be represented with closest points, such as a 2D balloon surface tied to a 1D string curve. The objects can also be manifold or nonmanifold, open or closed, and orientable or not.

Also, to our knowledge, no other single representation is simultaneously implicit and explicit, which is an attractive feature for application versatility. Poursaeed et al. [134] developed a neural network that simultaneously outputs an implicit and explicit surface

representation. They showed that coupling their implicit and explicit representations provided more accurate implicit occupancy functions and produced smoother explicit surfaces with more accurate normals. The closest point representation circumvents the need for their consistency loss term that aligns the explicit representation with the implicit one. Moreover, the closest point implicit representation, $\mathbf{x} - \text{cp}_{\mathcal{S}}(\mathbf{x}) = 0$, contains more information because it is vector-valued (the common scalar-valued implicit representation is also readily available as $\|\mathbf{x} - \text{cp}_{\mathcal{S}}(\mathbf{x})\| = 0$, although it is unsigned).

From the above, it seems the closest point representation satisfies the goal of finding a representation that can handle any object of interest. The implicit and explicit nature of the closest point representation increases the likelihood that it can be used in any application. We will have to wait for future research to know if this latter requirement is satisfied, but we began showing in this thesis that the closest point representation can indeed handle many PDE-based geometry processing applications when augmented with the *closest point method*. Our work is also directly applicable in other areas of computer graphics and in other fields that require solving PDEs.

CPM is a relatively young method compared to other methods that solve PDEs. For example, the finite element method can be traced back to 1941 with the work of Hrennikoff [59] (see [82] for a historical overview of its development) and the walk-on-spheres Monte Carlo method was introduced in 1956 by Muller [110]. In contrast, the foundational CPM paper of Ruuth and Merriman [138] was only published in 2008, more than 50 years later. In this chapter, we outlined a partial roadmap of the significant untapped potential of closest point representations and CPM; we hope that others in the computer graphics community will join us in exploring it.

References

[1] Eddie Aamari, Jisu Kim, Frédéric Chazal, Bertrand Michel, Alessandro Rinaldo, and Larry Wasserman. Estimating the reach of a manifold. *Electronic Journal of Statistics*, 13(1):1359–1399, 2019.

[2] David Adalsteinsson and James A Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[3] Nora Al-Badri and Jan Nikolai Nelles. Nefertiti, 2024. Downloaded from `https://cs.cmu.edu/~kmcrane/Projects/ModelRepository`, original source `https://nefertitihack.alloversky.com/`.

[4] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999. doi: 10.1007/PL00009475. URL `https://doi.org/10.1007/PL00009475`.

[5] Reynaldo J. Arteaga and Steven J. Ruuth. Laplace-Beltrami spectra for shape comparison of surfaces in 3D using the closest point method. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4511–4515. IEEE, 2015.

[6] S. Auer and R. Westermann. A semi-Lagrangian closest point method for deforming surfaces. *Computer Graphics Forum*, 32(7):207–214, 2013. doi: https://doi.org/10.1111/cgf.12228.

[7] S. Auer, C. B. Macdonald, M. Treib, J. Schneider, and R. Westermann. Real-time fluid effects on surfaces using the closest point method. *Computer Graphics Forum*, 31(6):1909–1923, 2012. doi: https://doi.org/10.1111/j.1467-8659.2012.03071.x.

[8] Stefan Auer and Rüdiger Westermann. Direct Contouring of Implicit Closest Point Surfaces. In M.-A. Otaduy and O. Sorkine, editors, *Eurographics 2013 - Short Papers*. The Eurographics Association, 2013. doi: /10.2312/conf/EG2013/short/001-004.

[9] Autodesk. AutoCAD, URL accessed April 2025. https://www.autodesk.com/ca-en/products/autocad/overview.

[10] Autodesk. Maya, URL accessed April 2025. https://www.autodesk.com/ca-en/products/maya/overview.

[11] Vinicius C. Azevedo, Christopher Batty, and Manuel M. Oliveira. Preserving geometry and topology for fluid flows with thin obstacles and narrow gaps. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[12] Seungbae Bang, Kirill Serkh, Oded Stein, and Alec Jacobson. An adaptive fast-multipole-accelerated hybrid boundary integral equation method for accurate diffusion curves. *ACM Transactions on Graphics (TOG)*, 42(6), December 2023. ISSN 0730-0301. doi: 10.1145/3618374. URL https://doi.org/10.1145/3618374.

[13] Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.

[14] Sören Bartels. Stability and convergence of finite-element approximation schemes for harmonic maps. *SIAM Journal on Numerical Analysis*, 43(1):220–238, 2005.

[15] Jacob Bedrossian, James H. Von Brecht, Siwei Zhu, Eftychios Sifakis, and Joseph M. Teran. A second order virtual node method for elliptic problems with interfaces and irregular domains. *Journal of Computational Physics*, 229(18):6405–6426, 2010.

[16] Alexander G. Belyaev and Pierre-Alain Fayolle. On variational and PDE-based distance function approximations. *Computer Graphics Forum*, 34(8):104–118, 2015. doi: https://doi.org/10.1111/cgf.12611. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12611.

[17] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004.

[18] Marcelo Bertalmío, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *Journal of Computational Physics*, 174(2):759–780, 2001. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.2001.6937. URL https://www.sciencedirect.com/science/article/pii/S0021999101969372.

[19] A. Bunge and M. Botsch. A survey on discrete Laplacians for general polygonal meshes. *Computer Graphics Forum*, 42(2):521–544, 2023. doi: https://doi.org/10.1111/cgf.14777.

[20] Erik Burman, Susanne Claus, Peter Hansbo, Mats G. Larson, and André Massing. CutFEM: Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7):472–501, 2015.

[21] Erik Burman, Peter Hansbo, and Mats G. Larson. A stabilized cut finite element method for partial differential equations on surfaces: the Laplace–Beltrami operator. *Computer Methods in Applied Mechanics and Engineering*, 285:188–207, 2015.

[22] Erik Burman, Peter Hansbo, Mats G. Larson, and Sara Zahedi. Stabilized CutFEM for the convection problem on surfaces. *Numerische Mathematik*, 141:103–139, 2019.

[23] Chieh Chen and Richard Tsai. Implicit boundary integral methods for the Helmholtz equation in exterior domains. *Research in the Mathematical Sciences*, 4(1):19, 2017.

[24] Yujia Chen and Colin B. Macdonald. The closest point method and multigrid solvers for elliptic equations on surfaces. *SIAM Journal on Scientific Computing*, 37(1): A134–A155, 2015. doi: 10.1137/130929497.

[25] Ka Chun Cheung, Leevan Ling, and Steven J. Ruuth. A localized meshless method for diffusion on folded surfaces. *Journal of Computational Physics*, 297:194–206, 2015. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2015.05.021. URL `https://www.sciencedirect.com/science/article/pii/S0021999115003551`.

[26] Jay Chu and Richard Tsai. Volumetric variational principles for a class of partial differential equations defined on surfaces and curves. *Research in the Mathematical Sciences*, 5(2):19, 2018.

[27] Ming Chuang. *Grid-based finite elements system for solving Laplace-Beltrami equations on 2-manifolds*. PhD thesis, Johns Hopkins University, 2013.

[28] Ming Chuang, Linjie Luo, Benedict J. Brown, Szymon Rusinkiewicz, and Michael Kazhdan. Estimating the Laplace-Beltrami operator by restricting 3D functions. *Computer Graphics Forum*, 28(5):1475–1484, 2009. doi: https://doi.org/10.1111/j.1467-8659.2009.01524.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01524.x`.

[29] Keenan Crane. *Conformal geometry processing*. PhD thesis, California Institute of Technology, 2013.

[30] Keenan Crane. Project page for the heat method for distance computation, 2013. `https://www.cs.cmu.edu/~kmcrane/Projects/HeatMethod/index.html`.

[31] Keenan Crane. Fish, 2019. Downloaded modified version from odedstein-meshes `https://github.com/odedstein/meshes/tree/master/objects/fish`, originally from `https://cs.cmu.edu/~kmcrane/Projects/ModelRepository`.

[32] Keenan Crane. Bob, 2024. Downloaded from `https://cs.cmu.edu/~kmcrane/Projects/ModelRepository`.

[33] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):1–11, 2013.

[34] Keenan Crane, Marco Livesu, Enrico Puppo, and Yipeng Qin. A survey of algorithms for geodesic paths and distances. *arXiv preprint arXiv:2007.10430*, 2020.

[35] Fang Da, Christopher Batty, and Eitan Grinspun. Multimaterial mesh-based surface tracking. *ACM Transactions on Graphics (TOG)*, 33(4):112–1, 2014.

[36] Fernando de Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *SIGGRAPH Asia 2015 Courses*, SA '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450339247. doi: 10.1145/2818143.2818167. URL `https://doi.org/10.1145/2818143.2818167`.

[37] Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. Subdivision exterior calculus for geometry processing. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016.

[38] Fernando de Goes, William Sheffler, and Kurt Fleischer. Character articulation through profile curves. *ACM Transactions on Graphics (TOG)*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530060. URL `https://doi.org/10.1145/3528223.3530060`.

[39] Ismail Demir and Rüdiger Westermann. Vector-to-closest-point octree for surface ray-casting. In David Bommes, Tobias Ritschel, and Thomas Schultz, editors, *Vision, Modeling & Visualization*. The Eurographics Association, 2015. ISBN 978-3-905674-95-8. doi: 10.2312/vmv.20151259.

[40] Ismail Demir, Mihaela Jarema, and Rüdiger Westermann. Visualizing the central tendency of ensembles of shapes. In *SIGGRAPH ASIA 2016 Symposium on Visualization*, SA '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345477. doi: 10.1145/3002151.3002165. URL `https://doi.org/10.1145/3002151.3002165`.

[41] Christian Dick, Joachim Georgii, and Rüdiger Westermann. A hexahedral multigrid approach for simulating cuts in deformable objects. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1663–1675, 2010.

[42] Huong Quynh Dinh, Anthony Yezzi, and Greg Turk. Texture transfer during shape transformation. *ACM Transactions on Graphics (TOG)*, 24(2):289–310, 2005.

[43] Gerhard Dziuk. *Finite elements for the Beltrami operator on arbitrary surfaces*, pages 142–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988. ISBN 978-3-540-46024-4. doi: 10.1007/BFb0082865. URL `https://doi.org/10.1007/BFb0082865`.

[44] Gerhard Dziuk and Charles M. Elliott. Surface finite elements for parabolic equations. *Journal of Computational Mathematics*, pages 385–407, 2007.

[45] James Eells and Luc Lemaire. A report on harmonic maps. *Bulletin of the London Mathematical Society*, 10(1):1–68, 1978.

[46] Danielle Ezuz, Justin Solomon, and Mirela Ben-Chen. Reversible harmonic maps between discrete surfaces. *ACM Transactions on Graphics (TOG)*, 38(2):1–12, 2019.

[47] Nicole Feng and Keenan Crane. A heat method for generalized signed distance. *ACM Transactions on Graphics (TOG)*, 43(4):1–19, 2024.

[48] Blender Foundation. Blender: Free and open 3D creation software, URL accessed April 2025. `https://www.blender.org/`.

[49] Prerna Gera and David Salac. Cahn–Hilliard on surfaces: A numerical study. *Applied Mathematics Letters*, 73:56–61, 2017.

[50] Frederic Gibou, Ronald P. Fedkiw, Li-Tien Cheng, and Myungjoo Kang. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002.

[51] Joachim Giesen, Balint Miklos, Mark Pauly, and Camille Wormser. The scale axis transform. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, SCG '09, page 106–115, New York, NY, USA, 2009. Association for

Computing Machinery. ISBN 9781605585017. doi: 10.1145/1542362.1542388. URL https://doi.org/10.1145/1542362.1542388.

[52] Alfred Gray. *An Introduction to Weyl's Tube Formula*, pages 1–12. Birkhäuser Basel, Basel, 2004. ISBN 978-3-0348-7966-8. doi: 10.1007/978-3-0348-7966-8_1. URL https://doi.org/10.1007/978-3-0348-7966-8_1.

[53] John B. Greer. An improvement of a recent Eulerian method for solving PDEs on general geometries. *Journal of Scientific Computing*, 29(3):321–352, 2006.

[54] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[55] Peter Heiss-Synak, Aleksei Kalinov, Malina Strugaru, Arian Etemadi, Huidong Yang, and Chris Wojtan. Multi-material mesh-based surface tracking with implicit topology changes. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024.

[56] Jeffrey L. Hellrung Jr., Luming Wang, Eftychios Sifakis, and Joseph M. Teran. A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions. *Journal of Computational Physics*, 231(4):2015–2048, 2012.

[57] Yi Hong, Dengming Zhu, Xianjie Qiu, and Zhaoqi Wang. Geometry-based control of fire simulation. *The Visual Computer*, 26(9):1217–1228, 2010.

[58] Ben Houston, Michael B Nielsen, Christopher Batty, Ola Nilsson, and Ken Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics (TOG)*, 25(1):151–175, 2006.

[59] Alexander Hrennikoff. Solution of problems of elasticity by the framework method. *Journal of Applied Mechanics*, 8(4):A169–A175, 1941. doi: 10.1115/1.4009129. URL https://doi.org/10.1115/1.4009129.

[60] Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)*, 37(4):60:1–60:14, 2018.

[61] Intel. MKL PARDISO: Parallel direct sparse solver interface, URL accessed April 2025. https://www.intel.com/content/www/us/en/docs/onemkl/developer-reference-c/2023-0/onemkl-pardiso-parallel-direct-sparse-solver-iface.html.

[62] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

[63] Wenzel Jakob. Enoki: structured vectorization and differentiation on modern processor architectures, 2019. https://github.com/mitsuba-renderer/enoki.

[64] Stefan Jeschke, David Cline, and Peter Wonka. Rendering surface details with diffusion curves. *ACM Transactions on Graphics (TOG)*, 28(5):1–8, December 2009. ISSN 0730-0301. doi: 10.1145/1618452.1618463. URL https://doi.org/10.1145/1618452.1618463.

[65] Jürgen Jost. *Riemannian geometry and geometric analysis*, volume 42005. Springer, 2008.

[66] Peter Kaufmann, Sebastian Martin, Mario Botsch, Eitan Grinspun, and Markus Gross. Enrichment textures for detailed cutting of shells. *ACM Transactions on Graphics (TOG)*, 28(3), July 2009. ISSN 0730-0301. doi: 10.1145/1531326.1531356. URL https://doi.org/10.1145/1531326.1531356.

[67] Mark Kim and Charles Hansen. Surface flow visualization using the closest point embedding. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 17–23, 2015. doi: 10.1109/PACIFICVIS.2015.7156351.

[68] Theodore Kim, Jerry Tessendorf, and Nils Thuerey. Closest point turbulence for liquid surfaces. *ACM Transactions on Graphics (TOG)*, 32(2):1–13, 2013.

[69] Ron Kimmel and James A. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences*, 95(15):8431–8435, 1998. doi: 10.1073/pnas.95.15.8431. URL https://www.pnas.org/doi/abs/10.1073/pnas.95.15.8431.

[70] Nathan King, Steven Ruuth, and Christopher Batty. A simple heat method for computing geodesic paths on general manifold representations. In *SIGGRAPH Asia 2024 Posters*, SA '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400711381. doi: 10.1145/3681756.3697920. URL https://doi.org/10.1145/3681756.3697920.

[71] Nathan King, Haozhe Su, Mridul Aanjaneya, Steven Ruuth, and Christopher Batty. A closest point method for PDEs on manifolds with interior boundary conditions for geometry processing. *ACM Transactions on Graphics (TOG)*, 43(5), August 2024. ISSN 0730-0301. doi: 10.1145/3673652. URL https://doi.org/10.1145/3673652.

[72] Nathan D. King and Steven J. Ruuth. Solving variational problems and partial differential equations that map between manifolds via the closest point method. *Journal of Computational Physics*, 336:330–346, 2017. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2017.02.019. URL `https://www.sciencedirect.com/science/article/pii/S0021999117301122`.

[73] Tobias Kirschstein, Javier Romero, Artem Sevastopolsky, Matthias Nießner, and Shunsuke Saito. Avat3r: Large animatable Gaussian reconstruction model for high-fidelity 3D head avatars. *arXiv preprint arXiv:2502.20220*, 2025.

[74] Ravikrishna Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms (TALG)*, 4(2):1–25, 2008.

[75] Catherine Kublik and Richard Tsai. Integration over curves and surfaces defined by the closest point mapping. *Research in the Mathematical Sciences*, 3(1):3, 2016.

[76] Catherine Kublik, Nicolay M. Tanushev, and Richard Tsai. An implicit interface boundary integral method for Poisson's equation on arbitrary domains. *Journal of Computational Physics*, 247:279–311, 2013.

[77] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000. doi: 10.1109/ROBOT.2000.844730.

[78] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[79] Randall J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[80] Jian Liang and Hongkai Zhao. Solving partial differential equations on point clouds. *SIAM Journal on Scientific Computing*, 35(3):A1461–A1486, 2013.

[81] Hsueh-Ti D. Liu, Alec Jacobson, and Keenan Crane. A Dirac operator for extrinsic shape analysis. In *Computer Graphics Forum*, volume 36, pages 139–149. Wiley Online Library, 2017.

[82] Wing Kam Liu, Shaofan Li, and Harold S Park. Eighty years of the finite element method: Birth, evolution, and future. *Archives of Computational Methods in Engineering*, 29(6):4431–4453, 2022.

[83] Yu-Shen Liu, Jean-Claude Paul, Jun-Hai Yong, Pi-Qiang Yu, Hui Zhang, Jia-Guang Sun, and Karthik Ramani. Automatic least-squares projection of points onto point clouds with applications in reverse engineering. *Computer-Aided Design*, 38(12): 1251–1263, 2006.

[84] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (TOG)*, 23(3):457–462, August 2004. ISSN 0730-0301. doi: 10.1145/1015706.1015745. URL `https://doi.org/10.1145/1015706.1015745`.

[85] Frank Losasso, Ronald Fedkiw, and Stanley Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids*, 35(10):995–1010, 2006.

[86] Jaehwan Ma, Sang Won Bae, and Sunghee Choi. 3D medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28 (1):7–19, 2012. doi: 10.1007/s00371-011-0594-7. URL `https://doi.org/10.1007/s00371-011-0594-7`.

[87] Colin B. Macdonald and Steven J. Ruuth. Level set equations on surfaces via the closest point method. *Journal of Scientific Computing*, 35(2-3):219–240, 2008.

[88] Colin B. Macdonald and Steven J. Ruuth. The implicit closest point method for the numerical solution of partial differential equations on surfaces. *SIAM Journal on Scientific Computing*, 31(6):4330–4350, 2010.

[89] Colin B. Macdonald, Jeremy Brandman, and Steven J. Ruuth. Solving eigenvalue problems on curved surfaces using the closest point method. *Journal of Computational Physics*, 230(22):7944–7956, 2011.

[90] Colin B. Macdonald, Barry Merriman, and Steven J. Ruuth. Simple computation of reaction–diffusion processes on point clouds. *Proceedings of the National Academy of Sciences*, 110(23):9209–9214, 2013. doi: 10.1073/pnas.1221408110. URL `https://www.pnas.org/doi/abs/10.1073/pnas.1221408110`.

[91] Miles Macklin. Warp: A high-performance python framework for GPU simulation and graphics. `https://github.com/nvidia/warp`, March 2022. NVIDIA GPU Technology Conference (GTC).

[92] P.-L. Manteaux, C. Wojtan, R. Narain, S. Redon, F. Faure, and M.-P. Cani. Adaptive physically based models in computer graphics. *Computer Graphics Forum*, 36(6):312–337, 2017. doi: https://doi.org/10.1111/cgf.12941. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12941`.

[93] Zoë Marschner, Paul Zhang, David Palmer, and Justin Solomon. Sum-of-squares geometry processing. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.

[94] Lindsay Martin and Yen-Hsi R. Tsai. Equivalent extensions of Hamilton–Jacobi–Bellman equations on hypersurfaces. *Journal of Scientific Computing*, 84(3):1–29, 2020.

[95] Dimas Martínez, Luiz Velho, and Paulo C. Carvalho. Computing geodesics on triangular meshes. *Computers & Graphics*, 29(5):667–675, 2005.

[96] Thomas März and Colin B. Macdonald. Calculus on surfaces with general closest point functions. *SIAM Journal on Numerical Analysis*, 50(6):3303–3328, 2012. doi: 10.1137/120865537.

[97] Sean P. Mauch. *Efficient algorithms for solving static Hamilton-Jacobi equations*. PhD thesis, California Institute of Technology, 2003.

[98] Ian C. May, Ronald D. Haynes, and Steven J. Ruuth. Schwarz solvers and preconditioners for the closest point method. *SIAM Journal on Scientific Computing*, 42(6): A3584–A3609, 2020.

[99] Ian C.T. May, Ronald D. Haynes, and Steven J. Ruuth. A closest point method library for PDEs on surfaces with parallel domain decomposition solvers and preconditioners. *Numerical Algorithms*, pages 1–23, 2022.

[100] Balint Miklos, Joachim Giesen, and Mark Pauly. Discrete scale axis representations for 3D geometry. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302104. doi: 10.1145/1833349.1778838. URL `https://doi.org/10.1145/1833349.1778838`.

[101] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: representing scenes as neural radiance fields for view synthesis. *Commun. ACM*, 65(1):99–106, December 2021. ISSN 0001-0782. doi: 10.1145/3503250. URL `https://doi.org/10.1145/3503250`.

[102] Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. Walkin' robin: Walk on stars with robin boundary conditions. *ACM Transactions on Graphics (TOG)*, 43(4), July 2024. ISSN 0730-0301. doi: 10.1145/3658153. URL `https://doi.org/10.1145/3658153`.

[103] Chohong Min and Frédéric Gibou. A second order accurate projection method for the incompressible Navier–Stokes equations on non-graded adaptive grids. *Journal of Computational Physics*, 219(2):912–929, 2006.

[104] Joseph S.B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[105] Nathan Mitchell, Mridul Aanjaneya, Rajsekhar Setaluri, and Eftychios Sifakis. Non-manifold level sets: A multivalued implicit surface representation with applications to self-collision processing. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

[106] Nicolas Moës, John Dolbow, and Ted Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150, 1999.

[107] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics (TOG)*, 23(3): 385–392, 2004.

[108] D. Morgenroth, S. Reinhardt, D. Weiskopf, and B. Eberhardt. Efficient 2D simulation on moving 3D surfaces. *Computer Graphics Forum*, 39(8):27–38, 2020. doi: https://doi.org/10.1111/cgf.14098. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14098`.

[109] Roger Moser. *Partial regularity for harmonic maps and related problems*. World Scientific, 2005.

[110] Mervin E. Muller. Some continuous Monte Carlo methods for the Dirichlet problem. *The Annals of Mathematical Statistics*, 27(3):569 – 589, 1956. doi: 10.1214/aoms/1177728169.

[111] Ken Museth. VDB: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)*, 32(3):1–22, 2013.

[112] Ken Museth. Nanovdb: A GPU-friendly and portable VDB data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*, SIGGRAPH

'21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383738. doi: 10.1145/3450623.3464653. URL https://doi.org/10.1145/3450623.3464653.

[113] Facundo Mémoli and Guillermo Sapiro. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. *Journal of Computational Physics*, 173(2):730–764, 2001.

[114] Facundo Mémoli, Guillermo Sapiro, and Stanley Osher. Solving variational problems and partial differential equations mapping into general target manifolds. *Journal of Computational Physics*, 195(1):263–292, 2004.

[115] Facundo Mémoli, Guillermo Sapiro, and Paul Thompson. Implicit brain imaging. *NeuroImage*, 23:S179–S188, 2004. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2004.07.072. URL https://www.sciencedirect.com/science/article/pii/S1053811904003878. Mathematics in Brain Imaging.

[116] Takashi Nagata. Simple local interpolation of surfaces using normal vectors. *Computer Aided Geometric Design*, 22(4):327–347, 2005.

[117] Rajkishore Nayak and Rajiv Padhye. The use of laser in garment manufacturing: An overview. *Fashion and textiles*, 3:1–16, 2016.

[118] Yen Ting Ng, Chohong Min, and Frédéric Gibou. An efficient fluid–solid coupling algorithm for single-phase flows. *Journal of Computational Physics*, 228(23):8807–8829, 2009.

[119] Thien Nguyen, Kęstutis Karčiauskas, and Jörg Peters. C^1 finite elements on non-tensor-product 2D and 3D manifolds. *Applied Mathematics and Computation*, 272: 148–158, 2016.

[120] Michael B Nielsen and Ken Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26:261–299, 2006.

[121] nTop. nTop: Computational design software (formally nTopology), URL accessed April 2025. https://www.ntop.com/.

[122] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, New York,

NY, USA, 2008. Association for Computing Machinery. ISBN 9781450301121. doi: 10.1145/1399504.1360691. URL https://doi.org/10.1145/1399504.1360691.

[123] Stanley Osher and Ronald Fedkiw. Level set methods and dynamic implicit surfaces. *Springer*, 2004.

[124] Richard Palais, Hermann Karcher, et al. 3DXM virtual math museum, 2023. https://virtualmathmuseum.org.

[125] John E Pearson. Complex patterns in a simple system. *Science*, 261(5118):189–192, 1993.

[126] Danping Peng, Barry Merriman, Stanley Osher, Hongkai Zhao, and Myungjoo Kang. A PDE-based fast local level set method. *Journal of Computational Physics*, 155(2): 410–438, 1999.

[127] A. Petras and S.J. Ruuth. PDEs on moving surfaces via the closest point method and a modified grid based particle method. *Journal of Computational Physics*, 312:139–156, 2016. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2016.02.024. URL https://www.sciencedirect.com/science/article/pii/S0021999116000814.

[128] Argyrios Petras, Leevan Ling, and Steven J. Ruuth. An RBF-FD closest point method for solving PDEs on surfaces. *Journal of Computational Physics*, 370:43–57, 2018.

[129] Argyrios Petras, Leevan Ling, Cécile Piret, and Steven J. Ruuth. A least-squares implicit RBF-FD closest point method and applications to PDEs on moving surfaces. *Journal of Computational Physics*, 381:146–161, 2019.

[130] Argyrios Petras, Leevan Ling, and Steven J. Ruuth. Meshfree semi-Lagrangian methods for solving surface advection PDEs. *Journal of Scientific Computing*, 93(1):1–22, 2022.

[131] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[132] Cécile Piret. The orthogonal gradients method: A radial basis functions method for solving partial differential equations on arbitrary surfaces. *Journal of Computational Physics*, 231(14):4662–4675, 2012. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2012.03.007. URL https://www.sciencedirect.com/science/article/pii/S0021999112001477.

[133] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

[134] Omid Poursaeed, Matthew Fisher, Noam Aigerman, and Vladimir G Kim. Coupling explicit and implicit surface representations for generative 3D modeling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*, pages 667–683. Springer, 2020.

[135] Yixuan Qiu. Lbfgs++, 2023. `https://lbfgspp.statr.me/`.

[136] Mariana Remešíková, Marián Šagát, and Peter Novysedlák. Discrete Lagrangian algorithm for finding geodesics on triangular meshes. *Applied Mathematical Modelling*, 76:396–427, 2019.

[137] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. Laplace-Beltrami spectra as 'Shape-DNA' of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006.

[138] Steven J. Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.*, 227(3):1943–1961, January 2008. ISSN 0021-9991. doi: 10.1016/j.jcp.2007.10.009. URL `https://doi.org/10.1016/j.jcp.2007.10.009`.

[139] Rohan Sawhney. fcpw: Fastest closest points in the west, 2022. `https://github.com/rohan-sawhney/fcpw`.

[140] Rohan Sawhney and Keenan Crane. Monte Carlo geometry processing: a grid-free approach to PDE-based methods on volumetric domains. *ACM Transactions on Graphics (TOG)*, 39(4), August 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392374. URL `https://doi.org/10.1145/3386569.3392374`.

[141] Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. Grid-free Monte Carlo for PDEs with spatially varying coefficients. *ACM Transactions on Graphics (TOG)*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530134. URL `https://doi.org/10.1145/3528223.3530134`.

[142] Rohan Sawhney, Bailey Miller, Ioannis Gkioulekas, and Keenan Crane. Walk on stars: A grid-free Monte Carlo method for PDEs with Neumann boundary conditions. *ACM Transactions on Graphics (TOG)*, 42(4), August 2023. doi: 10.1145/3592398.

[143] Robert Saye. High-order methods for computing distances to implicitly defined surfaces. *Communications in Applied Mathematics and Computational Science*, 9(1): 107–141, 2014.

[144] Teseo Schneider, Yixin Hu, Jérémie Dumas, Xifeng Gao, Daniele Panozzo, and Denis Zorin. Decoupling simulation accuracy from mesh quality. *ACM Transactions on Graphics (TOG)*, 2018.

[145] Richard M. Schoen and Shing Tung Yau. Lectures on harmonic maps. 1997.

[146] Peter Schwartz, Michael Barad, Phillip Colella, and Terry Ligocki. A Cartesian grid embedded boundary method for the heat equation and Poisson's equation in three dimensions. *Journal of Computational Physics*, 211(2):531–550, 2006.

[147] Martin Seiler, Denis Steinemann, Jonas Spillmann, and Matthias Harders. Robust interactive cutting based on an adaptive octree simulation mesh. *The Visual Computer*, 27:519–529, 2011.

[148] Silvia Sellán and Alec Jacobson. Neural stochastic poisson surface reconstruction. In *SIGGRAPH Asia 2023 Conference Papers*, SA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703157. doi: 10.1145/3610548.3618162. URL `https://doi.org/10.1145/3610548.3618162`.

[149] Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)*, 33(6):1–12, 2014.

[150] Bobak Shahriari. The modified Cahn-Hilliard equation on general surfaces. Master's thesis, Simon Fraser University, 2010.

[151] Nicholas Sharp and Keenan Crane. Variational surface cutting. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

[152] Nicholas Sharp and Keenan Crane. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Trans. Graph.*, 39(6), November 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417839. URL `https://doi.org/10.1145/3414685.3417839`.

[153] Nicholas Sharp and Alec Jacobson. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG)*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530155. URL `https://doi.org/10.1145/3528223.3530155`.

[154] Nicholas Sharp, Keenan Crane, et al. GeometryCentral: A modern c++ library of data structures and algorithms for geometry processing. 2019.

[155] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. The vector heat method. *ACM Transactions on Graphics (TOG)*, 38(3):1–19, 2019.

[156] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. Navigating intrinsic triangulations. *ACM Transactions on Graphics (TOG)*, 38(4):55, 2019.

[157] Nicholas Sharp et al. Polyscope, 2019. www.polyscope.run.

[158] Lin Shi and Yizhou Yu. Visual smoke simulation with adaptive octree refinement. In *Computer Graphics and Imaging*, pages 13–19, 2004.

[159] Yonggang Shi, Paul M. Thompson, Ivo Dinov, Stanley Osher, and Arthur W. Toga. Direct cortical mapping via solving partial differential equations on implicit surfaces. *Medical image analysis*, 11(3):207–223, 2007.

[160] Yonggang Shi, Jonathan H. Morra, Paul M. Thompson, and Arthur W. Toga. Inverse-consistent surface mapping with Laplace-Beltrami eigen-features. In *International Conference on Information Processing in Medical Imaging*, pages 467–478. Springer, 2009.

[161] Side Effects Software, Inc. Houdini, 2023. URL `https://www.sidefx.com/products/houdini/`. Computer Software.

[162] John Strain. Fast tree-based redistancing for level set computations. *Journal of Computational Physics*, 152(2):664–686, 1999.

[163] Ryusuke Sugimoto, Terry Chen, Yiti Jiang, Christopher Batty, and Toshiya Hachisuka. A practical walk-on-boundary method for boundary value problems. *ACM Transactions on Graphics (TOG)*, 42(4), July 2023. ISSN 0730-0301. doi: 10.1145/3592109. URL `https://doi.org/10.1145/3592109`.

[164] Ryusuke Sugimoto, Nathan King, Toshiya Hachisuka, and Christopher Batty. Projected walk on spheres: A Monte Carlo closest point method for surface PDEs. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400711312. doi: 10.1145/3680528.3687599. URL `https://doi.org/10.1145/3680528.3687599`.

[165] Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. Diffusion curve textures for resolution independent texture mapping. *ACM Transactions on Graphics (TOG)*, 31(4):1–9, 2012.

[166] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Transactions on Graphics (TOG)*, 24(3):553–560, 2005.

[167] Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3D skeletons: A state-of-the-art report. *Computer Graphics Forum*, 35(2):573–597, 2016. doi: https://doi.org/10.1111/cgf.12865.

[168] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11358–11367, 2021.

[169] Bei Tang, Guillermo Sapiro, and Vicent Caselles. Color image enhancement via chromaticity diffusion. *IEEE Transactions on Image Processing*, 10(5):701–707, 2001.

[170] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.1.1 edition, 2020. URL `https://doc.cgal.org/5.1.1/Manual/packages.html`.

[171] The Stanford 3D Scanning Repository. Lucy and XYZ RGB Dragon, 2024. Downloaded modified version of Lucy from `https://animium.com/2013/11/lucy-angel-3d-model`. Original Lucy and XYZ RGB Dragon meshes at `https://graphics.stanford.edu/data/3Dscanrep`.

[172] Li Tian, Colin B. Macdonald, and Steven J. Ruuth. Segmentation on surfaces with the closest point method. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3009–3012. IEEE, 2009.

[173] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2004.

[174] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *ACM SIGGRAPH Computer Graphics*, 25(4):289–298, 1991.

[175] Greg Turk. Texture synthesis on surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 347–354, 2001.

[176] Luiz Gustavo DO Véras, Felipe LL Medeiros, and Lamartine NF Guimarães. Systematic literature review of sampling process in rapidly-exploring random trees. *IEEE Access*, 7:50933–50953, 2019.

[177] Ingrid von Glehn, Thomas März, and Colin B. Macdonald. An embedded method-of-lines approach to solving partial differential equations on surfaces, 2013.

[178] Hui Wang, Yongxu Jin, Anqi Luo, Xubo Yang, and Bo Zhu. Codimensional surface tension flow using moving-least-squares particles. *ACM Transactions on Graphics (TOG)*, 39(4):42–1, 2020.

[179] Peng Wang, Tom Abel, and Ralf Kaehler. Adaptive mesh fluid simulations on GPU. *New Astronomy*, 15(7):581–589, 2010.

[180] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[181] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 355–360, 2001.

[182] Hermann Weyl. On the volume of tubes. *American Journal of Mathematics*, 61(2): 461–472, 1939.

[183] Ross T Whitaker. A level-set approach to 3D reconstruction from range data. *International journal of computer vision*, 29:203–231, 1998.

[184] Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. CNNs on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (TOG)*, 39(4):92–1, 2020.

[185] Romy Williamson and Niloy J. Mitra. Neural geometry processing via spherical neural surfaces. *Eurographics*, 2025.

[186] Chunlin Wu and Xuecheng Tai. A level set formulation of geodesic curvature flow on simplicial surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):647–662, 2009.

[187] Shi-Qing Xin and Guo-Jin Wang. Efficiently determining a locally exact shortest path on polyhedral surfaces. *Computer-Aided Design*, 39(12):1081–1090, 2007.

[188] Shi-Qing Xin, Ying He, and Chi-Wing Fu. Efficiently computing exact geodesic loops within finite steps. *IEEE Transactions on Visualization and Computer Graphics*, 18 (6):879–889, 2011.

[189] Tong Xu. Recent advances in rapidly-exploring random tree: A review. *Heliyon*, 10(11):e32451, 2024. ISSN 2405-8440. doi: https://doi.org/10.1016/j.heliyon.2024.e32451. URL https://www.sciencedirect.com/science/article/pii/S2405844024084822.

[190] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34:22483–22497, 2021.

[191] Junxiang Yang, Yibao Li, and Junseok Kim. A practical finite difference scheme for the Navier–Stokes equation on curved surfaces in $\mathbb{R}^3$. *Journal of Computational Physics*, page 109403, 2020.

[192] Zhang Yingjie and Ge Liling. Improved moving least squares algorithm for directed projecting onto point clouds. *Measurement*, 44(10):2008–2019, 2011.

[193] Na Yuan, Peihui Wang, Wenlong Meng, Shuang-Min Chen, Jian Xu, Shiqing Xin, Ying He, and Wenping Wang. A variational framework for curve shortening in various geometric domains. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[194] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. *ACM Transactions on Graphics (TOG)*, 25(4):1294–1326, 2006.

[195] Yihuan Zhang, Liang Wang, Xuhui Jiang, Yong Zeng, and Yifan Dai. An efficient lidar-based localization method for self-driving cars in dynamic environments. *Robotica*, 40(1):38–55, 2022.

APPENDICES

Appendix A

Closest Point Computation

Some manifolds allow closest points to be computed analytically, e.g., lines, circles, planes, spheres, cylinders, and tori. We use the analytical expressions for exact closest points in all examples for which they exist. For parameterized manifolds, closest points can be computed using standard numerical optimization techniques, e.g., Ruuth and Merriman [138] used Newton's method for various manifolds, such as a helix. The following optimization problem is solved

$$\arg\min_{\mathbf{t}} \frac{1}{2}\|\mathbf{p}(\mathbf{t}) - \mathbf{x}_i\|^2,$$

for the parameters $\mathbf{t}$ (e.g., $\mathbf{t} = t$ for a 1D curves and $\mathbf{t} = [u, v]^T$ for a 2D surface), where $\mathbf{p}(\mathbf{t}) \in \mathcal{S}$ and $\mathbf{x}_i \in \Omega(\mathcal{S})$. Depending on the required accuracy, we use LBFGS++ [135] or Newton's method to solve the optimization problem. An initial guess for $\text{cp}_{\mathcal{S}}(\mathbf{x}_i)$ is taken as the nearest neighbour in a point cloud $\mathcal{P}_{\mathcal{S}}$ of the parametric manifold. The point cloud $\mathcal{P}_{\mathcal{S}}$ is constructed using N equispaced points of the parameter $\mathbf{t}$.

Computing closest points to triangulated surfaces is also well-studied [162, 97, 7]. Notably, the work of Auer et al. [7] implements the closest point evaluation on a GPU. There also exist open source libraries that support computing closest points to triangle meshes, e.g., `libigl` [62]. Here we use the library `fcpw` [139] to compute closest points to triangulated surfaces and polyline curves.

The simplest way to compute closest points to a point cloud is to take the nearest neighbour as the closest point. As discussed by Macdonald et al. [90] this choice can be inaccurate if the point cloud is not dense enough. Wang et al. [178] (Figure 17) showed the inaccuracy of using nearest neighbours as closest points with CPM on a diffusion problem.

Several more accurate approaches for closest points to point clouds have been developed [94, 130, 83, 192].

Closest points can also be computed from analytical signed-distance functions $d(\mathbf{x})$ as

$$\text{cp}_{\mathcal{S}}(\mathbf{x}) = \mathbf{x} - d(\mathbf{x})\nabla d(\mathbf{x}). \tag{A.1}$$

Equation (A.1), however, is not valid for more general level-set functions ϕ. High-order accuracy of closest points from level-set functions (sampled on a grid) can be obtained using the method of Saye [143]. For the examples in this thesis, we use the ideas of Saye [143] but with analytical expressions for ϕ. Specifically, an initial guess $\text{cp}^\star$ of the closest point is obtained using a Newton-style procedure, starting with $\text{cp}_0 = \mathbf{x}_i$, and iterating

$$\text{cp}_{k+1} = \text{cp}_k - \frac{\phi(\text{cp}_k)\nabla\phi(\text{cp}_k)}{\|\nabla\phi(\text{cp}_k)\|^2},$$

with stopping criterion $\|\text{cp}_{k+1} - \text{cp}_k\| < 10^{-10}$. Then Newton's method

$$\mathbf{y}_{k+1} = \mathbf{y}_k - (D^2 f(\mathbf{y}_k))^{-1}\nabla f(\mathbf{y}_k),$$

is used to optimize

$$f(\text{cp}, \lambda) = \frac{1}{2}\|\text{cp} - \mathbf{x}_i\|^2 + \lambda\phi(\text{cp}),$$

where $\mathbf{y} = [\text{cp}, \lambda]^T$ and $\|\mathbf{y}_{k+1} - \mathbf{y}_k\| < 10^{-10}$ is used as the stopping criterion. The initial Lagrange multiplier is $\lambda_0 = (\mathbf{x}_i - \text{cp}^\star) \cdot \nabla\phi(\text{cp}^\star)/\|\nabla\phi(\text{cp}^\star)\|^2$. Analytical expressions for $\nabla f(\mathbf{y})$ and $D^2 f(\mathbf{y})$ are computed using analytical expressions of $\nabla\phi$ and $D^2\phi$.

Closest points for objects composed of multiple parts can be computed by obtaining the closest point to each independent manifold first. Then the closest point to the combined object is taken as the closest of the independent manifold closest points (e.g., the torus and sphere joined by line segments in Figure 4.10).

Closest points can be computed for many other representations. For example, closest points to neural implicit surfaces can be computed using the work of Sharp and Jacobson [153]. Further references for closest point computation are given in Section 5.1 of [140].