Scalable Particulate Flow Simulations with Boundary Integral Equations

by

Matthew James Morse

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2021

Professor Denis Zorin

For Mom and Dad.

Acknowledgements

I would first like to thank my advisor, Denis Zorin, for his advice, support and guidance throughout my PhD. I also want to thank my committee members, Georg Stadler, Marsha Berger, Michael O'Neil and Michael Overton, for their valuable feedback on my thesis.

I'm grateful to Leslie Greengard and Alex Barnett for many valuable and insightful discussions. I'm also grateful to Michael Shelley at the Flatiron Institute and and to Harper Langston, Pierre-David Letourneau, and Rich Lethin at Reservoir Labs, for two wonderful internships and the opportunity to experience research from a new perspective. Bill Menasco, Nancy Wrinkle and Joan Birman have been great collaborators and very supportive, with a contagious excitement and passion for mathematics that kept me going. I also must thank Shenglong Wang at NYU HPC, who has helped me out of many last minute crises.

I have had the great fortune to meet and work with many wonderful people during my time at NYU: Abhinav Tamaskar, Azam Asl, Chelsea Tymms, Claudia Skok-Gibbs, Daniele Panozzo, David Stein, Dhairya Malhotra, Dimitri Leggas, Elman Manismov, Francis Williams, Hannah Lawerence, Ilya Kostrikov, Jérémie Kalfon, Jason Kaye, Julia Wei, Julian Panetta, Karina Koval, Ludvig af Klinteberg, Mitchell Harris, Naomi Oppenheimer, Nico Shertler, Qingnan Zhou, Reza Farhadifar, Roberta Raileanu, Shravas Rao, Siddartha Krishna, Wen Yan, Yixin Hu, Zachary Ferguson, Zhongshi Jiang, Zvonimir Pavlinovic, and others whom I have forgotten that I hope will forgive me.

I am deeply thankful for the guidance and support of Abtin Rahimian, for a fruitful collaboration and friendship, and for teaching me to be a proficient researcher and engineer. I'm equally grateful for Libin Lu, who has been not only a great collaborator but a great friend. I'm also indebted to Adam Cunningham, Clarence Beutel, Davi Colli Tozoni, and Francisca Gil-Ureta, who have been very understanding and encouraging throughout this process.

Finally, I am tremendously thankful for my mother, father, sister, and Ioanna, for their unconditional support and unending patience.

Abstract

Numerical simulation of complex particulate flows, and of red blood cell flows through capillaries in particular, is an important investigational tool in the biological sciences. The ability to rapidly evaluate the impact of vessel and cell geometries, plasma viscosity, and particulate densities on macroscopic physiology is crucial to pursuing further biological understanding. Experimental techniques are costly and time-consuming, while analytical approaches are often of limited practical use in realistic scenarios, ultimately underscoring the importance of a computational approach.

In this work, we construct such a simulation, capable of simulating microliters of blood flowing through realistic vasculature, along with more general particulate suspensions. Due to the micrometer length scales of typical capillaries, we can model the blood plasma as a Stokesian fluid and red blood cells as inextensible, deformable membranes. By reformulating the viscous flow as a set of boundary integral equations, we are able to produce a method that has optimal complexity with high-order accuracy that is capable of handling dense particulate suspensions in complex geometries.

This approach relies on a novel, robust solver for elliptic partial differential equations, applied to Stokes flow. A core component of the solver is a novel fast algorithm to compute the value of the solution near and on the domain boundary, which we have named `hedgehog` . We provide a set of algorithms to guarantee the accuracy of `hedgehog` on piecewise smooth surfaces, discuss the error behavior and complexity of `hedgehog` , and evaluate its performance.

Leveraging this solver in a confined blood flow simulation involves advecting deformable particulates along the flow trajectory. Large timesteps are required for an efficient simulation, but can cause collisions among cells and with the vessel wall if performed naively. We present collision detection and resolution algorithms for the red blood cells and the blood vessel. We parallelize `hedgehog` and the collision algorithms and scale the final simulation to nearly 35,000 cores.

Contents

List of Figures

List of Tables

1 Introduction

Microscale particulate flows play a central role in many biophysical settings. Red blood cell flows through vasculature in particular can be modeled in this fashion by a suspension of deformable particles in a Newtonian fluid. A clear understanding blood circulation through capillaries would shed light on important yet complex phenomena such as vasodilation, vasoconstriction, thrombosis and clotting. The ability to simulate such flows with modern computers enables computational investigation of these phenomena with an unprecedented level of control and resolution. This is a crucial milestone in the next generation of biological innovation.

The computational tools required for blood flow simulations can also be applied in many other biological settings. By simulating a deterministic lateral displacement microfluidic chip [McGrath et al. 2014], one can optimize the device geometry through simulation and avoid the typical expensive trial-and-error design process through manufacturing [Kabacaoğlu and Biros 2019]. Researchers simulating intra-cellular dynamics of organelles can gain insights into biophysical dynamics by adjusting model formulations and infer biological behaviors by comparing experimental and computational results [Nazockdast et al. 2017]. Advanced devices like optical tweezers [Zhong et al. 2013] and techniques like laser ablation [Yu et al. 2014] can be used to isolate and study single cells or bacteria, but can cost tens of thousands of dollars to purchase. Better computational tools can simulate such complex systems numerically, which will increase the pace of biological innovation while reducing cost.

In this work, we present a robust scalable platform to simulate complex flows of deformable particulates and apply the methodology to red blood cell (RBC) flows through capillaries. We first construct a solver for elliptic partial differential equations (PDEs) that robustly handles complex geometries. We then parallelize this solver and integrate it with a collision-aware time stepping scheme to advect RBCs along the flow trajectory. We scale this solver to nearly 35,000 cores and millions of RBCs, demonstrating its ability to capture high fidelity real-world biological phenomena.

1.1 Background

The methodology presented in this thesis is applicable to a broad range of applications, such as microfluidic chip design [McGrath et al. 2014], neurotransmission via synaptic vesicles [Gan and Watanabe 2018], surfactant laden drops [Schramm et al. 2003; Sorgentone and Tornberg 2018] in cosmetics [Lourith and Kanlayavattanakul 2009], pharmaceutics [Lourith and Kanlayavattanakul 2009], petroleum engineering [Schramm 2000], among others. Moreover, the model presented here accurately captures the dynamics of many types of biological cells, such as vacuoles, transport and secretory vesicles, lysosomes [Carreira et al. 2017], enveloped viruses [Barenholz et al. 1976] and gas bubbles released from bacteria to maintain buoyancy [Pfeifer 2012]. We restrict our focus here to red blood cell flow [Veerapaneni et al. 2009b] to highlight its applicability.

The fluid dynamics of blood flow, or *hemodynamics*, spans several regimes. The first regime is in larger veins and arteries and in regions near the heart, where the blood can be well characterized as a non-viscous continuum that is well-characterized by a Navier-Stokes incompressible fluid model [Griffith 2012]. The second flow regime is at very small length scales ($\leq 80\mu$m [Potter and Groom 1983]) such as capillaries, called microcirculation. In this setting, blood vessel diameter can equal the length of 1-10 RBCs [Potter and Groom 1983; Linden et al. 2012], so the overall flow dynamics are largely determined by RBCs. This also implies that the blood plasma has negligible inertial fluid forces (Reynolds number $Re \leq 10^{-3}$) and can be characterized as a viscous Newtontian fluid [Cortinovis et al. 2006]. The final, most complex length scale to study is the transition between these two regimes. This regime contains length scales varying over three orders of magnitude, implying variable plasma viscosity and non-trivial RBC flow contributions. In the non-viscous regime, there are many platforms to simulate the heart and associated vasculature [Randles et al. 2015; Vigmond et al. 2008; Peskin 1977; Griffith 2012]. As such, we will focus on microcirculation in this work, since target applications such as vasoconstriction, thombosis and clotting occuring in this regime are difficult to investigate experimentally.

RBCs are somewhat unique cellular structures. They contain no nucleus or mitochondira, in order to maximize their capacity to transport hemoglobin [Zhang et al. 2011] and have a biconcave shape at rest to maximize large-scale lamniar flow and minimize platelet scattering [Uzoigwe 2006]. Most importantly, RBCs are highly *deformable*, allowing them to travel through capillaries much narrower than their resting diameter [Huisjes et al. 2018]. Macroscopic physiological mutations, such as sickle cell anemia and thalassemia, are known to impact the degree of RBC deformability [Huisjes et al. 2018], indicated its importance in the overall dynamics. Three primary forces act on RBC membranes: bending, in-plane shear forces, and extensional forces, with expansion moduli in an approximate ratio of $1 : 50 : 10^6$ [Lee and Smith 2008]. Together, these characteristics allow RBC flow dynamics to be approximated by a thin, deformable, inextensible membrane filled with a viscous Newtontian fluid called a *vesicle* [Sackmann 1996; Kraus et al. 1996]. While modeling an RBC as a vesicle neglects the contribution of in-plane shear forces, they serve as an effective qualitative model of RBC. In some scenarios, shear forces can have a significant contribution to the overall flow dynamics [Mills et al. 2004; Horobin et al. 2019], but in more typical physiological regimes, their contribution is small.

In summary, in order to model microcirculation through capillaries, our model must:

- represent blood plasma as a viscous Newtontian fluid within a blood vessel
- approximate RBCs as a thin, deformable, fluid-filled membrane

- incorporate RBC membrane inextensibility along with tension and bending forces within the membrane.

1.2 Challenges

To faithfully simulate complex particulate flows, there are several outstanding computational obstacles. The first is *robustness*: can we design a set of algorithms that can handle arbtitrary flows? Typical flows are characterized by complex vascular geometry with rapidly varying curvature, high-volume fraction flows (>45% in humans) and long simulation times. This will result in nearly-touching and potentially colliding geometries that causes many standard simulation approaches to fail; our algorithms need to handle these adversarial cases. Another challenge is *accuracy*: can we reliably control the numerical error of our algorithms? We would like accuracy of our simulation to correspond to the accuracy of our physical model, not to the accuracy of our algorithms. Ideally, one would hope for a single parameter that can be tuned to control the overall numerical error.

Constructing accurate and robust numerical algorithms is demanding in its own right, but as little as a microliter of human blood can contain almost four million RBCs. This imposes a final computational challenge: in order to simulate several microliters of blood, our algorithms must be *fast* and *scalable*. Achieving algorithmic complexity proportional to the number of cells size while guaranteeing robustness and high accuracy is a tall order. Moreover, the representation of millions of RBCs and the blood vessel vastly exceeds the memory capcity of a single machine, so distributed algorithms are needed to complete the simulation with low wallclock times.

There has been a vast amount of literature trying to address these challenges, many successfully scaling blood flow simulations up to hundreds of thousands [Grinberg et al. 2011; Rossinelli et al. 2015] or even millions [Gounley et al. 2017; Randles et al. 2015] of cores. While these methods are able to simulate immense, complex geometries, the particle-based simulation approaches taken in these papers, such as Lattice Boltzmann methods and dissipative particle dynamics, naturally incur very high numerical errors. It remains to be seen whether these errors allow such methods to recover small-scale qualitative behaviors of RBC flows.

On the other extreme, many recent works have focused on producing highly accurate numerical algorithms of deformable particulate flows using the *boundary integral equation method* [Veerapaneni et al. 2009a, 2011; Rahimian et al. 2015; Sorgentone and Tornberg 2018; Sorgentone et al. 2019; af Klinteberg and Tornberg 2016; Zhao et al. 2010]. Several works [Lu et al. 2018; Malhotra et al. 2017; Rahimian et al. 2010] have produced scalable parallel versions of these algorithms applied to RBCs with great success. However, the majority of these works involved free-space or periodic boundary conditions, which are not relevant in a realistic biophysical setting. Similar mathematical formulations can be applied to simulating elliptic PDEs [Ying et al. 2006b; Wala and Klöckner 2018a,c; Bruno and Lintner 2013], providing efficient, accurate numerical solutions on complex geometries. A recent paper has leveraged these principles into an exascale parallel acoustic scattering simulation [Abduljabbar et al. 2019], which along with [Rahimian et al. 2010] demonstrate that large scale BIE approaches are achievable.

1.3 Overall Approach

Motivated by our discussion of the physical properties of RBC flows, we approximate the physical behavior of individual RBCs by vesicles and assume that the surrounding fluid is highly viscous and Newtonian. We will refer to RBCs and vesicles interchangeably throughout the remainder of this work. The blood vessel is approximated by a rigid closed fluid-filled domain. However, such flows through arbtitirary vessel geometries are far too complex for analytic solutions and require numerical simulation to study any large system of biological significance.

We adopt a *boundary integral formulation* of the fluid flow through the vessel. In brief, a boundary integral formulation allows us to express the fluid velocity at a point in the blood vessel as a sum of integrals, each defined on the distinct surfaces bounding the fluid. We can evaluate these integrals on the surface of RBCs and advect them along the fluid's trajectory.

This approach has many advantages. It avoids discretizing the fluid volume and avoids the expensive, error-prone process of remeshing at each timestep. We are able to leverage quadrature methods for smooth functions to integrate with high accuracy, with RBCs and the blood vessel are defined by smooth surfaces. Well-formulated integral equations have favorable conditioning when discretized, allowing iterative solvers like GMRES to converge in a constant number of steps. With an appropriate discretization, fast summation methods such as the Fast Multipole Method (FMM) can compute matrix-vector products of the resulting linear system matrix with optimal complexity.

The primary difficulty introduced by the boundary integral formulation is the need to evaluate integrals of singular functions or functions with rapidly varying high-order derivatives. These are call *singular* and *near-singular* integrals, respectively. These integrals are well-defined mathematically, but standard numerical quadrature methods fail to resolve their true value. Special algorithms are required in order to accurately evaluate such integrals.

Recent work [Barnett 2014; Klöckner et al. 2013a] introduced an elegant and intuitive method called *Quadrature by Expansion*, or QBX , which approximates a singular/near-singular integral with a local expansion, whose coefficients are defined by smooth integrals. QBX has evolved rapidly since, with target-specific [Siegel and Tornberg 2018], adaptive [af Klinteberg and Tornberg 2018], kernel-independent [Rahimian et al. 2018], and Stokes [af Klinteberg and Tornberg 2016] variations, analyses [Epstein et al. 2013; af Klinteberg and Tornberg 2017; af Klinteberg et al. 2020] for various PDEs and fast algorithmic accelerations [Rachh et al. 2017; Wala and Klöckner 2019a,b, 2020]. Although several 3D implementations have been developed, [Siegel and Tornberg 2018; Wala and Klöckner 2019b, 2020, 2019a], the large scale nature of realistic problems require efficient and scalable parallel implementations to be competitive with state-of-the-art finite element implementations. Moreover, in practical engineering scenarios, geometries often come from CAD files based on splines or Bézier curves that have rapidly varying curvatures and nearly touching, non-local surface patches, while most current QBX methods rely on less standard representations of local geometry.

There is also a rich literature regarding vesicle simulations based on boundary integral formulations [Veerapaneni et al. 2009b,a, 2011; Ghigliotti et al. 2011; Rahimian et al. 2015; Lu et al. 2017; Sorgentone and Tornberg 2018; Sorgentone et al. 2019]. Fast and accurate parallel algorithms have been developed and scaled to thousands of cores [Lu et al. 2018; Rahimian et al. 2010; Malhotra et al. 2017]. However, much of this work is either in 2D [Veerapaneni et al. 2009b,a, 2011; Lu et al. 2017] or a free-space or periodic setting in 3D [Rahimian et al. 2010, 2015; Lu et al. 2018; Malhotra et al. 2017; Sorgentone and Tornberg 2018; Sorgentone et al. 2019]. To recover qualitative properties of RBC flows, the vesicles need to simulated in a confined flow with representative vessel geometries (i.e., more complicated than a cylinder/torus).

1.4 Contributions and Outline

This thesis addresses these challenges, resulting in a practical simulation platform for realistic RBC flows through capillaries.

In Chapter 2, we introduce an $O(N)$ high-order solver for elliptic PDEs in 3D geometries. The core component of the solver is hedgehog, a straightforward PDE-independent singular/near-singular quadrature scheme for layer potentials arising from PDEs defined on domains with spline-based boundaries. We design adaptive geometric preprocessing and query algorithms to gurantee the accuracy of hedgehog while enabling good performance. We evaluate the method on a variety of complex geometries and boundary conditions and stress test it on several challenging geometries This is based on the work in [Morse et al. 2020a].

In Chapter 3, we present a robust scalable RBC flow simulation platform. We parallelize the PDE solver presented in Chapter 2 to scale to thousands of processors. We then integrate this with several parallel RBC simulation libraries. In partiular, we extend a collision-free time-stepping scheme for deformable bodies to also handle rigid boundaries. We explore strong and weak scaling of our plaform and scale this simulation to thousands of cores. This is based on the work in [Lu et al. 2019] in collaboration with Libin Lu [Lu 2019].

In Chapter 4, we summarize the thesis and discuss possible future directions of research.

2 | Quadrature Methods for Elliptic PDEs in 3D Complex Geometries

This chapter introduces a simple, high-order boundary solver for elliptic partial differential equations (PDEs) in 3D complex geometries called hedgehog . It is based on the joint work [Morse et al. 2020a] with Abtin Rahimian and Denis Zorin.

Boundary integral equation methods have many computational advantages over conventional approaches, but the necessity of evaluating singular/near-singular integrals remains a complicated challenge. The main contribution of hedgehog is its algorithmic simplicity. The singular/near-singular quadrature scheme decouples quadrature evaluation from the regularization required to accurately evaluate layer potentials. hedgehog only requires values of the solution at prescribed points, which can be computed with a standard point FMM . This overcomes a key hurdle in the development of fast singular quadrature methods.

This elliptic PDE solver ultimately allows us to solve Laplace, Stokes and linear elasticity equations in complex geometries. As will see in the next chapter, by assuming that the fluid flowing through capillaries behaves like a Stokesian fluid, we can leverage hedgehog as a building block in a large scale blood flow simulation.

2.1 Introduction

Linear elliptic homogeneous partial differential equations (PDEs) play an important role in modeling many physical interactions, including electrostatics, elastostatics, acoustic scattering, and viscous fluid flow. Using ideas from potential theory allow us to reformulate the associated boundary value problem (BVP) as an integral equation. The solution to the BVP can then be expressed as a layer potential, i.e., a surface convolution against the PDE's fundamental solution. Discretizing the integral equation formulation offers several potential advantages over common used direct PDE discretization methods such as finite element or finite volume methods.

First, the system of equations uses asymptotically fewer variables because only the domain boundary requires discretization. There is no need to discretize the volume, which is often the most time-consuming and error-prone task in the full simulation pipeline, especially if complex boundary geometry is involved. This aspect of integral formulations is particularly important for problems with changing geometries such as particulate flows, or flows with deforming boundaries, as well as moving boundaries. Second, while the algebraic system resulting from discretization is dense, efficient $O(N)$ methods are available to solve it. A suitable integral formulation can yield a well-conditioned system that can be solved using an iterative method like GMRES in relatively few iterations. Third, high-order quadrature rules for smooth functions can be leveraged to dramatically improve the accuracy for a given discretization size over a standard method. In other words, integral equation solvers can be both more efficient, usually if high accuracy is desired, and more robust, as they do not require volume meshing.

For elliptic problems with smooth (or mostly smooth) domain boundaries, high-order methods have a significant advantage over standard methods, drastically reducing the number of degrees of freedom needed to approximate a solution to a given accuracy. However, realizing this potential presents a significant challenge: for integral equation methods to achieve high-order accuracy, they require a high-order quadrature and a high-order surface approximation to compute the integrals accurately.

One of the main difficulties in constructing high-order boundary integral equation (BIE) solvers is the need for accurate quadrature rules for *singular* integrals, as the formulation requires the solution of an integral equation involving the singular fundamental solution of the PDE. Moreover, if the solution needs to be evaluated arbitrarily close to the boundary, then one must numerically compute *nearly singular* integrals with high-order accuracy. In some sense, the near singular integrals are even more difficult to handle compared to singular integrals, since simple change of variable techniques that are often used to eliminate singularities on the boundary are harder to apply. Precomputing high-order singular/near-singular quadrature weights also presents a considerable problem, as these necessarily depend on the local surface shape and different sets of weights are required for each sample point. Furthermore, the sampling density required for accurate singular/near-singular integration is highly dependent on the boundary geometry. For example, two nearly touching pieces of the boundary require a sampling density proportional to the distance between them; applying such a fine discretization globally will become prohibitively expensive.

2.1.1 Contributions

We introduce a new, high-order boundary integral solver for non-oscillatory elliptic PDEs. A preliminary parallel version of this method is used in [Lu et al. 2019] to simulate red blood cell flows through complex blood vessel with high numerical accuracy. We describe the singular/near-singular quadrature scheme for single- and double-layer potentials with the following features:

- **Surface representation.** We use standard Bézier patches to define the domain boundary, which simplifies the use of the solver on CAD geometry, increases the efficiency of surface evaluation and simplifies parallelization. We use a *quad-tree* of patches that allows us to approximate complex surfaces with nonuniform curvature distribution efficiently and to refine sampling as required by surface quadrature. Our method extends directly to other surface representation.

- **Singular and near-singular quadrature.** We introduce an approximation-based singular/near-singular quadrature scheme for single- and double-layer potentials in 3D : after computing the solution at a set of nearby *check points*, placed along a line intersecting the target, we extrapolate the solution to the target point. We have named this scheme hedgehog , for reasons that are apparent from Figure 2.2. In order to ensure accuracy of the scheme for complex geometries, a key component of our scheme is a set of geometric criteria for surface sampling needed for accurate integration along with fast algorithms to refine the sampling adaptively so that these criteria are satisfied.

  Our approach is originally motivated by the near-singular evaluation scheme of [Ying et al. 2006b; Quaife and Biros 2014], which implements a similar scheme that includes an additional on-surface singular evaluation to allow for interpolation of the solution. Removing this interpolation step and directly extrapolating solution values achieves optimal complexity without greatly affecting accuracy.

- **Refinement for geometric admissibilty and quadrature accuracy.** We present a set of criteria that allows for accurate integration via hedgehog called *geometric admissibilty.* This is similar in spirit to [Rachh et al. 2017] and [Wala and Klöckner 2019a], but adapted to the geometry of our particular quadrature scheme. To guarantee quadrature accuracy of our method, we detail an adaptive h-refinement approach of the integral equation discretization based on a simple criteria to enable a fast refinement pipeline.

- **Error convergence and comparison** We apply hedgehog to a variety of problems on various geometries to demonstrate high-order convergence. We also compare our method with [Ying et al. 2006b] to highlight the differences between global and local singular quadrature schemes. We also solve Laplace and Stokes problems on challenging domain boundaries.

2.1.2 Related Work

We will restrict our discussion to elliptic PDE solvers in 3D using boundary integral formulations. The common schemes to discretize boundary integral equations are the *Galerkin* method, the *collocation* method, and the *Nyström* method [Atkinson and Han 2009]. After choosing a set of basis functions to represent the solution, the Galerkin method forms a linear system for the coefficients of the solution by computing double integrals of the chosen basis functions multiplied by singular kernels. The collocation method computes a set of unknown functions that match the solution at a prescribed set of points. To form the required linear system, it assumes that an accurate quadrature rule is available for evaluating the layer potential at the discretization points. For a particular choice of quadratures, collocation and Nyström discretizations can lead to equivalent algebraic systems [Kress 1999, Chapter 13]. In this paper, we focus on the *Nyström* discretization, which is both simple (the integral in the equation is replaced by the quadrature approximation) and enables very efficient methods to solve the discretized integral equation. The Galerkin and collocation approaches are commonly referred to as *boundary element methods* (BEM) and have become very popular. There have been many optimized BEM implementations for elliptic (Laplace, Helmholtz) and Maxwell problems. One such implementation is BEM ++, presented in [Śmigaj et al. 2015], with extensions for adaptivity added in [Bespalov et al. 2019; Betcke et al. 2019]. [Chaillat et al. 2017a,b] present iterative solvers for high-frequency scattering problems in elastodynamics, based on a BEM implementation coupled with fast summation methods to enable accurate solutions on complex triangle meshes. For a more complete background of BEM , we refer the reader to [Steinbach 2007].

A significant advancement in the field of finite element methods, called *isogeometric analysis* (IGA)[Hughes et al. 2005], has been recently applied to boundary integral formulations. IGA couples the basis functions defining the surface geometry with the analytic approaches for the finite element scheme. Most relevant to this work, IGA has recently been applied to singular and hypersingular boundary integral equations with a collocation discretization [Taus et al. 2016] with great success. A Nyström IGA method coupled with a regularized quadrature scheme is detailed in [Zechner et al. 2016].

In the BIE literature, singular and near-singular integration schemes fall into one of the several categories: *singularity cancellation*, *asymptotic correction*, *singularity subtraction* or *custom quadrature* schemes. Singularity cancellation schemes apply a change of variables to remove the singularity in the layer potential, allowing for the application of standard smooth quadrature rules. The first polar change of variables was detailed in the context of acoustic scattering [Bruno and Kunyansky 2001], which leveraged a partition of unity and a polar quadrature rule to remove the singularity in the integrand of layer potential. Fast summations were performed with FFT 's and the periodic trapezoidal rule enables high-order convergence; the method was extended to open surfaces in [Bruno and Lintner 2013]. This methodology was applied to general elliptic PDEs in [Ying et al. 2006b] and coupled with the kernel-independent fast multipole method [Ying et al. 2004] and a C^∞ surface representation for complex geometries [Ying and Zorin 2004]. Recently, [Malhotra et al. 2019] demonstrated that the choice of partition of unity function used for the change of variables has a dramatic effect on overall convergence order, although not in the context of elliptic PDEs. The first singularity cancellation scheme in 3D on general surfaces composed of piecewise smooth triangles was presented in [Bremer and Gimbutas 2012, 2013] by splitting a triangle into three subtriangles at the singularity and computing a polar integral on each new triangle. [Ganesh and Graham 2004] introduced a change of variables method for acoustic scattering on 3D surfaces parametrized by spherical coordinates by integrating over a rotated coordinate system that cancels out the singularity. [Abduljabbar et al. 2019] outlines a fast exascale solver for soft body acoustic problems on triangle meshes 3D and using Duffy transforms as for singular/near-singular quadratures.

Asymptotic correction methods study the inaccuracies due to the singular PDE kernel with asymtotic analysis and apply a compensating correction. [Beale 2004; Beale et al. 2016; Tlupova and Beale 2019] compute the integral with a regularized kernel and add corrections for regularization and discretization for the single and double layer Laplace kernel in 3D , along with the Stokeslet and stresslet in 3D . [Carvalho et al. 2018a] computes an asymptotic expansion of the kernel itself, which is used to remove the aliasing error incurred when applying smooth quadrature rules to near-singular layer potentials. This method is extended to 3D in [Carvalho et al. 2018b] and a complete asymptotic analysis of the double-layer integral is performed in [Khatri et al. 2020].

Singularity subtraction methods [Järvenpää et al. 2003; Jarvenpaa et al. 2006; Nair et al. 2013] explicitly subtract the singular component of the integrand, which produces a smooth bounded integral that can be integrated with standard quadrature rules. Custom quadrature rules aim to integrate a particular family of functions to high-order accuracy. This can allow for arbitrarily accurate and extremely fast singular integration methods, since the quadrature rules can be precomputed and stored [Alpert 1999; Xiao and Gimbutas 2010].

The most noteworthy singular quadrature scheme that does not fit into one of the above categories is that of [Helsing and Ojala 2008]. This method is similar to the second-kind barycentric interpolation [Berrut and Trefethen 2004]; it forms a rational function whose numerator and denominator compensates for the error as the target point approaches the boundary. [Wu et al. 2020] has implemented an adaptive version of [Helsing and Ojala 2008] for complex Stokes flows in 2D and [af Klinteberg and Barnett 2019] have recently produced a remarkable extension to nearly-singular line integrals in 2D and 3D . While this method performs exceptionally well in practice, it does not immediately generalize to 3D surfaces in an efficient manner.

The *method of fundamental solutions*, which represents the solution as a sum of point charges on an equivalent surface outside of the PDE domain, removing the need for singular evaluation, has also seen a great deal of success in 2D [Barnett and Betcke 2008] and in axis-symmetric 3D problems [Liu and Barnett 2016]. Recently, [Gopal and Trefethen 2019] has introduced an 2D approach similar in spirit to the method of fundamental solutions for domains with corners, but formulated as a rational approximation problem in the complex plane rather than as a boundary integral equation. The lack of singular integration makes these methods advantageous, but placing the point charges robustly can be challenging in practice. General 3D geometries also remain a challenge.

There has been a great deal of recent work on special analyses of regions with corners [Serkh and Rokhlin 2016a; Serkh 2017, 2018; Hoskins et al. 2019; Rachh and Serkh 2017; Serkh and Rokhlin 2016b]. Rather than a dyadic refinement of the discretization toward corners to handle the artificial singularities, these works have shown that the solution can be appropriately captured with special quadratures for a certain class of functions. Although not yet generalized to 3D , this work has the potential to vastly improve the performance of 3D Nyström boundary integral methods on regions with corners and edges.

Our method falls into a final category: approximation-based quadrature schemes. The first use of a local expansion to approximate a layer potential near the boundary of a 2D boundary was presented in [Barnett 2014]. By using an refined, or *upsampled*, global quadrature rule to accurately compute coefficients of a Taylor series, the resulting expansion serves as a reasonable approximation to the solution near the boundary where quadrature rules for smooth functions are inaccurate. This scheme was then adapted to evaluate the solution both near and on the boundary, called Quadrature by Expansion (QBX) [Klöckner et al. 2013b]. The first rigorous error analysis of the truncation error of QBX was carried out in [Epstein et al. 2013].

Great progress has been made in this area since [Klöckner et al. 2013b]. A fast implementation of QBX in 2D , along with a set of geometric constraints required for well-behaved convergence, was presented in [Rachh et al. 2017]. However, the interaction of the expansions of QBX and the expansions used in the translation operators of the FMM resulted in a loss of accuracy, which required an artificially high multipole order to compensate. [Wala and Klöckner 2018b] addresses this shortcoming by enforcing a confinement criteria on the location of expansion disks relative to FMM tree boxes. [af Klinteberg and Tornberg 2017] provided extremely tight error heuristics for various kernels and quadrature rules using contour integration and the asymptotic approach of [Elliott et al. 2008]. [af Klinteberg and Tornberg 2018] then leveraged these estimates in a QBX algorithm for Laplace and Helmholtz problems in 2D that adaptively selects the amount of upsampled quadrature and the expansion order for each QBX expansion. In the spirit of [Ying et al. 2004], [Rahimian et al. 2018] generalizes QBX to any elliptic PDE by using potential theory to form a local, least-squares solution approximation using only evaluations of the PDE's Green's function.

The first extension of QBX to 3D was [Siegel and Tornberg 2018], where the authors present a *local, target-specific* QBX method on spheroidal geometries. In a local QBX scheme, an upsampled accurate quadrature is used as a local correction to the expansion coefficients computed from the coarse quadrature rule over the boundary. This is in contrast with a *global* scheme, where the expansion coefficients are computed from the upsampled quadrature with no need for correction. The first local QBX scheme appears in [Barnett 2014] in 2D , but the notion of local FMM corrections dates back to earlier work such as [Alpert 1999; Kapur and Rokhlin 1997]. The expansions in [Siegel and Tornberg 2018] computed in a target-specific QBX scheme can only be used to evaluate a single target point, but each expansion can be computed at a lower cost than a regular expansion valid in a disk. The net effect of both these algorithmic variations are greatly improved constants, which are required for complicated geometries in 3D . In [af Klinteberg and Tornberg 2017], very accurate error heuristics are derived for the tensor product Gauss-Legendre rule on a surface panel and a simple spheroidal geometry in 3D , which were then leveraged to estimate QBX quadrature errors. [af Klinteberg and Tornberg 2016] generalized QBX to Stokes problems on spheroidal geometries in 3D . [Wala and Klöckner 2019a] extends the QBX -FMM coupling detailed in [Wala and Klöckner 2018b], along with the geometric criteria and algorithms of [Rachh et al. 2017] that guarantees accurate quadrature, to 3D surfaces. [Wala and Klöckner 2019b] improves upon this by adding target-specific expansions to [Wala and Klöckner 2019a], achieving a 40% speed-up and [Wala and Klöckner 2020] provides a thorough error analysis of the interaction between computing QBX expansions and FMM local expansions.

The rest of the paper is organized as follows: In Section 2.2, we briefly summarize the problem formulation, geometry representation and discretization. In Section 2.3, we detail our singular evaluation scheme and with algorithms to enforce admissibility, adaptively upsample the boundary discretization, and query surface geometry to evaluate singular/near-singular integrals. In Section 2.4, we provide error estimates for hedgehog . In Section 2.5, we summarize the complexity of each of the algorithms described in Section 2.3. In Section 3.5, we detail convergence tests of our singular evaluation scheme and compare against other state-of-the-art methods.

2.2 Formulation

2.2.1 Problem Setup

We restrict our focus to interior Dirichlet boundary value problems of the form

$$Lu(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Omega, \tag{2.1}$$

$$u(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega = \Gamma, \tag{2.2}$$

with multiply- or singly-connected domain Ω of arbitrary genus. Our approach applies directly to standard integral equation formulations of exterior Dirichlet and Neumann problems; we include results for an exterior Dirichlet problem in Section 2.6.4. Here L is a linear elliptic operator and f is at least C^k. While our method can be applied to any non-oscillatory elliptic PDE, we use the following equations in our examples:

$$Lu = \begin{cases} \Delta u & \text{Laplace} \\ \Delta u - \nabla p, \quad \nabla \cdot u = 0 & \text{Stokes} \\ \Delta u + \frac{1}{1-2\nu} \nabla\nabla \cdot u & \text{Navier (linear elasticity)} \end{cases} \tag{2.3}$$

We follow the approach of [Ying et al. 2006b]. We can express the solution at a point $\boldsymbol{x} \in \Omega$ in terms of the double-layer potential

$$u(\boldsymbol{x}) = D[\phi](\boldsymbol{x}) = \int_\Gamma \frac{\partial G(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} \phi(\boldsymbol{y}) d\boldsymbol{y}_\Gamma, \tag{2.4}$$

where $G(\boldsymbol{x}, \boldsymbol{y})$ is the *fundamental solution* or *kernel* of Eq. (2.2), $\boldsymbol{n}(\boldsymbol{y})$ is the normal at $\boldsymbol{y}$ on Γ pointing into the exterior of Ω, and ϕ is an unknown function, or *density*, defined on Γ. We list the kernels associated with the PDEs in Eq. (2.3) in [Morse et al. 2020b, Section 1]. Using the jump relations for the interior and exterior limits of $u(\boldsymbol{x})$ as $\boldsymbol{x}$ tends towards Γ [Kress 1999; Mikhlin 2014; Pozrikidis 1992a; Parton and Perlin 1982], we know that Eq. (2.4) is a solution to Eq. (2.2) if ϕ satisfies

$$\left(\frac{1}{2} I + D + M\right)[\phi](\boldsymbol{x}) = f(\boldsymbol{x}), \boldsymbol{x} \in \Gamma \tag{2.5}$$

with identity operator I. We will refer to ϕ as the *density* and $u(\boldsymbol{x})$ as the *potential* at $\boldsymbol{x}$. The double-layer integrals in this equation are *singular*, due to the singularity in the integrand of Eq. (2.4). Additionally, as $\boldsymbol{x}$ approaches Γ, Eq. (2.4) becomes a *nearly singular* integral.

The operator M completes the rank of $\frac{1}{2}I + D$ to ensure invertibility of Eq. (2.5). If $\frac{1}{2}I + D$ is full-rank, $M = 0$. When $\frac{1}{2}I + D$ has a non-trivial null space, M accounts for the additional constraints to complete the rank of the left-hand side of Eq. (2.5). For example, for the exterior Laplace problem on ℓ multiply-connected domains, the null space of $\frac{1}{2}I + D$ has dimension ℓ [Siegel and Tornberg 2018]. The full set of cases for each kernel is considered in this work and their corresponding values of M have been detailed in [Ying et al. 2006b].

2.2.2 Geometry representation

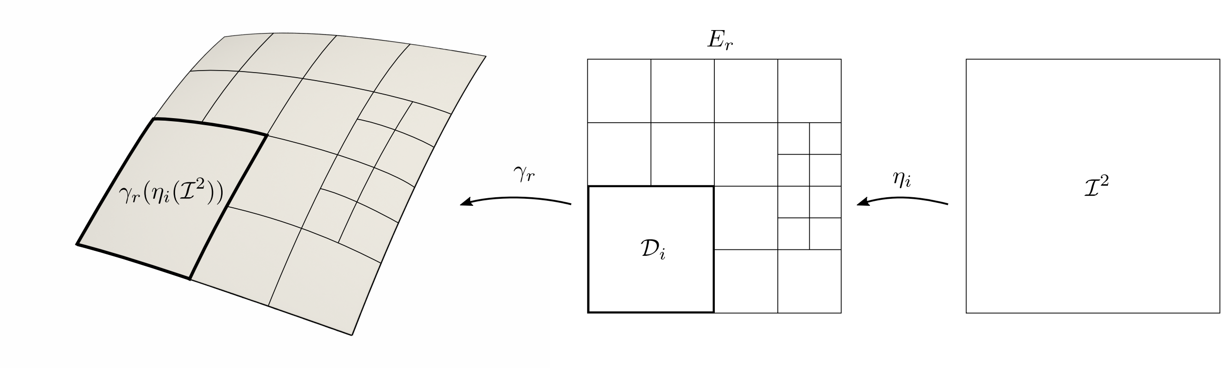


Figure 2.1: Patch Quadrisection. Right: the standard domain $\mathcal{I}^2$ of a single surface or quadrature patch. Middle: a collection of subdomains $\mathcal{D}_i$ of E_r, produced by quadrisection. Each $\mathcal{D}_i$ corresponds to a map η_i such that $\mathcal{D}_i = \eta_i(\mathcal{I}^2)$; a single $\mathcal{D}_i$ is highlighted in bold. Left: the image of E_r under the patch γ_r. The final image of each subdomain is outlined, with the image of $\mathcal{D}_i$ in bold.

We assume that the smooth domain boundary Γ is given by a *quadrilateral mesh* consisting of quadrilateral faces Q_r, referred to as *quads*. Each quad is associated with a parametric domain $\mathcal{I}^2 = [-1, 1]^2 = E_r$, along with embeddings $\gamma_r : E_r \longrightarrow \mathbb{R}^3$ for each quad such that $Q_r = \gamma_r(E_r)$. We assume that the quad mesh is *conforming*, i.e., two non-disjoint faces either share a whole edge or a single vertex; examples of this are shown in Figures 2.8 and 2.9. We assume that no two images $\gamma_r(E_r)$ intersect, except along the shared edge or vertex. The surface Γ is the union of patches $\cup_r \gamma_r(E_r) = \cup_r Q_r$. We also assume that Γ is sufficiently smooth to recover the solution of Eq. (2.2) up to the boundary [Kress 1999] and is at least C^k.

To represent the surface geometry, we approximate Γ with a collection of *Bézier patches*, given by a linear combination of tensor-product Bernstein polynomials

$$\boldsymbol{P}_i(s,t) = \sum_{\ell=0}^{n} \sum_{m=0}^{n} \boldsymbol{a}_{\ell m}^{(i)} B_\ell^n(s) B_m^n(t), \tag{2.6}$$

where $B_\ell^n(t) = \binom{n}{\ell} t^{n-\ell}(1-t)^\ell$ for each ℓ are the n-th degree Bernstein polynomials, i denotes the index of a patch in the collection and $\boldsymbol{a}^{(i)}_{\ell m} \in \mathbb{R}^3$. Each patch $\boldsymbol{P}$ is a vector function from $\mathcal{I}^2$ to $\mathbb{R}^3$, so $s, t \in [-1, 1]$. We will refer to this approximation of Γ as $\hat{\Gamma}$. This representation is advantageous because the Bézier coefficients provide a direct connection to surface geometry, as shown in Fig. 2.4.

The domain E_r of each embedding function γ_r is adaptively refined using *quadrisection*, i.e., splitting a square domain into four square subdomains of equal size. Quadrisection induces a *quadtree* structure on each E_r. The root of the quadtree is the original domain $\mathcal{I}^2$ and each node of the tree is related by a single quadrisection of a subdomain of E_r. The leaves of the quadtree form a collection of subdomains $\mathcal{D}_i$ whose union equals E_r, as shown in Fig. 2.1-middle. Given an indexing scheme of all $\mathcal{D}_i$'s over all E_r's, we define the function $r(i)$ that maps the leaf node index i to its root node index r in the quadtree forest, indicating that $\mathcal{D}_i \subset E_r$. For each r, E_r can have a distinct sequence of associated quadrisections and therefore a distinct quadtree structure. We refer to the process of *refinement* or *refining a patch* $\boldsymbol{P}$ as the construction of such quadtrees for each E_r subject to some set of criteria.

On each $\mathcal{D}_i$ at the quadtree leaves, we define a Bézier patch and reparametrize each patch over $\mathcal{I}^2$ by defining the affine map $\eta_i : \mathcal{I}^2 \to E_{r(i)}$ such that $\eta_i(\mathcal{I}^2) = \mathcal{D}_i \subseteq E_{r(i)}$. It follows that the set of subdomains $\{\eta_i(\mathcal{I}^2) \,|\, r(i) = \kappa\}$ form a cover of E_κ and $\{\gamma_\kappa(\eta_i(\mathcal{I}^2)) \,|\, r(i) = \kappa\}$ likewise covers $\gamma_\kappa(E_\kappa)$. We summarize this setup in Figure 2.1; examples of surfaces of this form can be seen in Figures 2.8, 2.9, 2.12 and 2.13.

2.2.3 Problem discretization

We use two collections of patches in the form described above: $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$. The patches in $\mathcal{P}_{\text{coarse}}$, called *surface patches*, determine $\hat{\Gamma}$ from Γ and the set of patches $\mathcal{P}_{\text{fine}}$, called *quadrature patches*, are obtained by further quadrisection of the surface patches in $\mathcal{P}_{\text{coarse}}$. The geometry of $\hat{\Gamma}$ is not changed by this additional refinement of $\mathcal{P}_{\text{coarse}}$, but the total number of subdomains $E_{r(i)}$ is increased. We will detail the geometric criteria that $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ must satisfy in Section 2.3.2.

Discretizing $\hat{\Gamma}$ with with a quadrature rule based on $\mathcal{P}_{\text{fine}}$ results in a denser sampling of $\hat{\Gamma}$ than a similar discretization of $\mathcal{P}_{\text{coarse}}$. We will refer to $\mathcal{P}_{\text{coarse}}$ as the *coarse discretization* of $\hat{\Gamma}$ and $\mathcal{P}_{\text{fine}}$ as the *upsampled* or *fine discretization* of $\hat{\Gamma}$.

We index the patches in $\boldsymbol{P}_i \in \mathcal{P}_{\text{coarse}}$ by $i = 1, \dots N$; we can then rewrite Eq. (2.4) as a sum of integrals over surface patches:

$$u(\boldsymbol{x}) = \sum_{i=1}^{N} \int_{P_i} \frac{\partial G(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} \phi(\boldsymbol{y}) d\boldsymbol{y}_{\boldsymbol{P}_i}. \tag{2.7}$$

We discretize functions defined on $\hat{\Gamma}$, such as Eq. (2.7), at q-node composite tensor-product Clenshaw-Curtis quadrature points on $\mathcal{I}^2$ of patches in $\mathcal{P}_{\text{coarse}}$. We refer to these points and weights on a single patch $\boldsymbol{P}_i$ as x_j and w_j^{CC} respectively, for $j = 1 \dots q^2$. The quadrature point $\boldsymbol{y}_{ij}$ from $\boldsymbol{P}_i$ is defined as $\boldsymbol{y}_{ij} = \boldsymbol{P}_i(\eta_i(x_j))$. We assume that the boundary condition f is given by a black-box evaluator on $\mathbb{R}^3$ that can be used to obtain values at $\boldsymbol{y}_{ij}$. For clarity, we reindex the surface points by a global index $I = 1, \dots, q^2 N$. We discretize the double layer integral Eq. (2.7) on $\mathcal{P}_{\text{coarse}}$ to approximate the solution $u(\boldsymbol{x})$:

$$u(\boldsymbol{x}, \mathcal{P}_{\text{coarse}}) \approx \hat{u}(\boldsymbol{x}, \mathcal{P}_{\text{coarse}}) = \sum_{i=1}^{N} \sum_{j=1}^{q^2} \frac{\partial G(\boldsymbol{x}, \boldsymbol{y}_{ij})}{\partial \boldsymbol{n}(\boldsymbol{y}_{ij})} \phi_{ij} \sqrt{g_{ij}} w_j^{\text{CC}} = \sum_{I=1}^{q^2 N} \frac{\partial G(\boldsymbol{x}, \boldsymbol{y}_I)}{\partial \boldsymbol{n}(\boldsymbol{y}_I)} \phi_I \hat{w}_I \tag{2.8}$$

with g_{ij} being the determinant of the metric tensor of $\boldsymbol{P}_i$ at x_j and $\hat{w}_{i \cdot q^2 + j} = \sqrt{g_{ij}} w_j^{\text{CC}}$. In other words, $\hat{u}(\boldsymbol{x}, \mathcal{P}_{\text{coarse}}) = \hat{D}[\phi](\boldsymbol{x})$, where $\hat{D}[\phi](\boldsymbol{x}) \approx D[\phi](\boldsymbol{x})$.

We can also discretize functions with tensor-product Clenshaw-Curtis nodes on the domains of patches in $\mathcal{P}_{\text{fine}}$. The values of functions on $\mathcal{P}_{\text{fine}}$ are *interpolated* from their values on the quadrature nodes of $\mathcal{P}_{\text{coarse}}$ rather than being computed directly on $\mathcal{P}_{\text{fine}}$. We call this interpolation from $\mathcal{P}_{\text{coarse}}$ to $\mathcal{P}_{\text{fine}}$ *upsampling*. We denote the quadrature nodes and weights on $\mathcal{P}_{\text{fine}}$ by $\tilde{x}_j$ and $\tilde{w}_j$ with a similar global index J and refer to them as the *upsampled* nodes and weights. Identical formulas are used for computing quadrature on $\mathcal{P}_{\text{fine}}$ with the nodes and weights $\tilde{x}_j$, $\tilde{w}_j$

on $\mathcal{P}_{\text{fine}}$, denoted $u(\boldsymbol{x}, \mathcal{P}_{\text{fine}})$ and $\hat{u}(\boldsymbol{x}, \mathcal{P}_{\text{fine}})$, repsectively.

In the next section, we describe the algorithm to compute an accurate approximation to the singular/near-singular double-layer integral in Eq. (2.4), using a quadrature rule for smooth functions (Eq. (2.8)) as a building block. This algorithm allows us to compute the matrix-vector products $A\phi$, for a vector of values ϕ defined at the quadrature points $\boldsymbol{y}_I$, where A is the discrete operator obtained from the left-hand side of Eq. (2.5) after approximating $D[\phi](\boldsymbol{y})$ with the singular integration scheme. As a result, we can solve the linear system using GMRES, which only requires a matrix-vector product

$$A\phi = f, \tag{2.9}$$

where f is the boundary condition sampled at the points $\boldsymbol{y}_I$. The evaluation of these integrals is accelerated in a standard manner using the fast multipole method (FMM)[Malhotra and Biros 2015a; Ying et al. 2004; Greengard and Rokhlin 1987].

2.3 Algorithms

We now detail a set of algorithms to solve the integral equation in Eq. (2.5) and evaluate the solution via the double layer integral in Eq. (2.4) at a given target point $\boldsymbol{x} \in \Omega$. As described in the previous section, both solving Eq. (2.5) and evaluating Eq. (2.4) require accurate evaluation of singular/near-singular integrals of functions defined on the surface $\hat{\Gamma}$. We first outline our unified singular/near-singular integration scheme, hedgehog , its relation to existing approximation-based quadrature methods and geometric problems that can impede accurate solution evaluation. We then describe two geometry preprocessing algorithms, *admissibility refinement* and *adaptive upsampling*, that address these issues to obtain the sets of patches $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ used by hedgehog .

2.3.1 Singular and Near-Singular Evaluation

We begin with an outline of the algorithm. For a point $\boldsymbol{s_x} \in \hat{\Gamma}$ on a patch $\boldsymbol{P}$ from $\mathcal{P}_{\text{coarse}}$ that is closest to $\boldsymbol{x}$, we first upsample the density ϕ from $\mathcal{P}_{\text{coarse}}$ to $\mathcal{P}_{\text{fine}}$ and compute the solution at a set of points $\boldsymbol{c}_s$, $s = 1, \ldots p$ called *check points*, sampled along the surface normal at $\boldsymbol{s_x}$ away from $\hat{\Gamma}$. We use Eq. (2.8) to approximate the solution at the check points. We then extrapolate the solution to $\boldsymbol{x}$.

For a given surface or quadrature patch $\boldsymbol{P} : \mathcal{I}^2 \to \mathbb{R}^3$, we define the *characteristic length* $L(P)$ as the square root of the surface area of $\boldsymbol{P}$, i.e., $L(\boldsymbol{P}) = \sqrt{\int_{\boldsymbol{P}} d\boldsymbol{y_P}}$. We use $L = L(\boldsymbol{P})$ or $L_{\boldsymbol{y}}$ for $\boldsymbol{y} \in \boldsymbol{P}(D)$ to denote the characteristic length when $\boldsymbol{P}$ is clear from context. For a point $\boldsymbol{x} \in \Omega$, we assume that there is a single closest point $\boldsymbol{s_x} \in \hat{\Gamma}$ to $\boldsymbol{x}$; all points to which the algorithm is applied will have this property by construction. Note that $\boldsymbol{n}(\boldsymbol{s_x})$, the vector normal to $\hat{\Gamma}$ at $\boldsymbol{s_x}$, is chosen to point outside of Ω.

We define three zones in Ω for which Eq. (2.4) is evaluated differently in terms of Eq. (2.8) and the desired solution accuracy $\varepsilon_{\text{target}}$. The *far field* $\Omega_F = \{\boldsymbol{x} \in \Omega \mid \|u(\boldsymbol{x}) - \hat{u}(\boldsymbol{x}; \mathcal{P}_{\text{coarse}})\|_2 \leq \varepsilon_{\text{target}}\}$, where the quadrature rule corresponding to $\mathcal{P}_{\text{coarse}}$ is sufficiently accurate, and the *intermediate field* $\Omega_I = \{\boldsymbol{x} \in \Omega \mid \|u(\boldsymbol{x}) - \hat{u}(\boldsymbol{x}; \mathcal{P}_{\text{fine}})\|_2 \leq \varepsilon_{\text{target}}\}$, where quadrature over $\mathcal{P}_{\text{fine}}$ is sufficiently accurate. The remainder of Ω is the *near field* $\Omega_N = \Omega \setminus \Omega_I$.

Non-singular integration To compute the solution at points $\boldsymbol{x}$ in Ω_F, Eq. (2.8) is accurate to $\varepsilon_{\text{target}}$, so we can simply compute $\hat{u}(\boldsymbol{x}, \mathcal{P}_{\text{coarse}})$ directly. Similarly for points in $\Omega_I \setminus \Omega_F$, we know by definition that $\hat{u}(\boldsymbol{x}, \mathcal{P}_{\text{fine}})$ is sufficiently accurate, so it can also be applied directly.

Singular/near-singular integration algorithm For the remaining points in Ω_N, we need an alternative means of evaluating the solution. In the spirit of the near-singular evaluation method of [Ying et al. 2006b], we construct a set of *check points* $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_p$ in Ω_I along a line intersecting $\boldsymbol{x}$ to approximate the solution near $\boldsymbol{x}$. However, instead of interpolating the solution

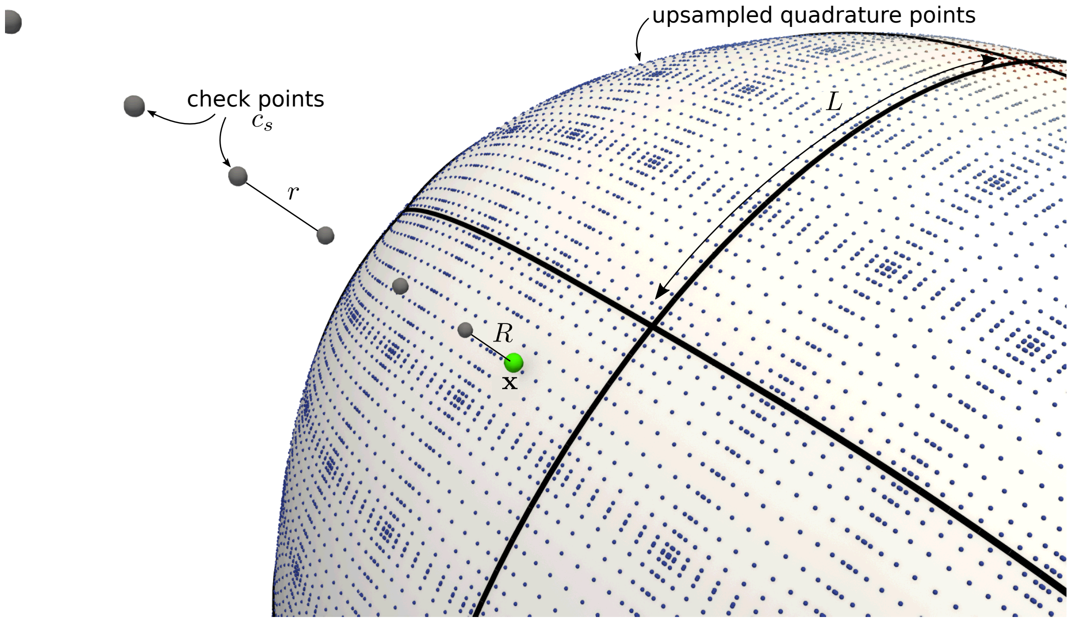


Figure 2.2: Schematic of singular/near-singular evaluation. A small piece of a boundary $\hat{\Gamma}$ is shown, along with the set of patches $\mathcal{P}_{\text{coarse}}$ (patch boundaries are drawn in black). The target point $\boldsymbol{x}$, in this case on $\hat{\Gamma}$, is shown in green. The solution is evaluated at the check points $\boldsymbol{c}_s$ (gray points off-surface) using the fine discretization $\mathcal{P}_{\text{fine}}$ (small dots on-surface). The distance from the first check point $\boldsymbol{c}_0$ to $\hat{\Gamma}$ is R and the distance between consecutive check points $\boldsymbol{c}_i$ and $\boldsymbol{c}_{i+1}$ is r. In this example, $\mathcal{P}_{\text{fine}}$ is computed from $\mathcal{P}_{\text{coarse}}$ with two levels of uniform quadrisection, producing 16 times more patches. The patch length L is roughly proportional to the average edge length of the patch.

as in [Ying et al. 2006b], we instead extrapolate the solution from the check points to $\boldsymbol{x}$. We define two distances relative to $\boldsymbol{s}_{\boldsymbol{x}}$: $R(\boldsymbol{s}_{\boldsymbol{x}}) = bL_{\boldsymbol{s}_{\boldsymbol{x}}} = \|\boldsymbol{c}_0 - \boldsymbol{s}_{\boldsymbol{x}}\|_2$, the distance from the first check point $\boldsymbol{c}_0$ to $\hat{\Gamma}$, and $r(\boldsymbol{s}_{\boldsymbol{x}}) = aL_{\boldsymbol{s}_{\boldsymbol{x}}} = \|\boldsymbol{c}_i - \boldsymbol{c}_{i+1}\|_2$, the distance between consecutive check points. We assume $0 < a, b < 1$.

The overall algorithm for the unified singular/near-singular evaluation scheme is as follows. A schematic for hedgehog is depicted in Figure 2.2.

1. Find the closest point $\boldsymbol{s}_{\boldsymbol{x}}$ on $\hat{\Gamma}$ to $\boldsymbol{x}$.

2. Given values a and b, generate check points $C = \{\boldsymbol{c}_0, \ldots, \boldsymbol{c}_p\}$

$$\boldsymbol{c}_s = \boldsymbol{s}_{\boldsymbol{x}} - (R(\boldsymbol{s}_{\boldsymbol{x}}) + sr(\boldsymbol{s}_{\boldsymbol{x}}))\boldsymbol{n}(\boldsymbol{s}_{\boldsymbol{x}}), \quad s = 0, \ldots, p \tag{2.10}$$

The center of mass of these check points $\hat{\boldsymbol{c}}$ is called the *check center* for $\boldsymbol{x}$. Note that $\mathcal{P}_{\text{fine}}$ must satisfy the condition that $\boldsymbol{c}_s$ are in Ω_I for a given choice of a and b.

3. Upsample ϕ. We interpolate the density values ϕ_I at x_I on patches in $\mathcal{P}_{\text{coarse}}$ to quadrature points $\tilde{x}_J$ on patches in $\mathcal{P}_{\text{fine}}$ with global indices I and J on $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ respectively. If a patch $\boldsymbol{P}_i$ in $\mathcal{P}_{\text{coarse}}$ is split into m_i patches in $\mathcal{P}_{\text{fine}}$, we are interpolating from q^2 points to $m_i q^2$ points.

4. Evaluate the potential at check points via smooth quadrature with the upsampled density, i.e. evaluate $\hat{u}(\boldsymbol{c}_s) = \hat{u}(\boldsymbol{c}_s, \mathcal{P}_{\text{fine}})$ for $s = 0, \ldots, p$.

5. Compute a Lagrange interpolant $\tilde{u}$ through the check points $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_p$ and values $\hat{u}(\boldsymbol{c}_0), \ldots, \hat{u}(\boldsymbol{c}_p)$ and evaluate at the interpolant at $\boldsymbol{x}$:

$$\tilde{u}(\boldsymbol{x}) = \sum_{s=0}^{p} \hat{u}(\boldsymbol{c}_s)\ell_s(t_{\boldsymbol{x}}), \tag{2.11}$$

   where $\ell_s(\boldsymbol{x})$ is the sth Lagrange basis function through the points $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_p$, and $t_{\boldsymbol{x}} \in \mathbb{R}$ is such that $\boldsymbol{x} = \boldsymbol{s}_{\boldsymbol{x}} - t_{\boldsymbol{x}}\boldsymbol{n}(\boldsymbol{s}_{\boldsymbol{x}})$ (see Fig. 2.6 for a schematic of the check points). Since $\boldsymbol{x}$ lies between $\boldsymbol{c}_0$ and $\hat{\Gamma}$, we are extrapolating when computing $\tilde{u}(\boldsymbol{x})$.

Ill-conditioning of the discrete integral operator This evaluation scheme can be used directly to extrapolate all the way to the surface and obtain the values of the singular integral in Eq. (2.5). However, in practice, due to a distorted eigenspectrum of this approximate operator, GMRES tends to stagnate at a level of error corresponding to the accuracy of hedgehog when it is used to compute the matrix-vector product. This is a well-known phenomenon of approximation-based singular quadrature schemes; [Klöckner et al. 2013b, Section 3.5][Rahimian et al. 2018, Section 4.2] present a more detailed study. To address this, we average the interior and exterior limits of the solution at the quadrature nodes, computed via hedgehog, to compute the on-surface potential and add $\frac{1}{2}I$ to produce the interior limit. This shifts the clustering of eigenvalues from around zero to around $\frac{1}{2}$, which is ideal from the perspective of GMRES. We call this *two-sided* hedgehog , while the standard version described above is called *one-sided* hedgehog . We observe stable and consistent convergence of GMRES when two-sided hedgehog is used to evaluate the matrix-vector multiply to solve Eq. (2.9). In light of this, we always use two-sided hedgehog within GMRES and set the stopping tolerance for GMRES to $\varepsilon_{\text{GMRES}} = 10^{-12}$, regardless of the geometry, boundary condition or quadrature order.

2.3.2 Geometric criteria for accurate quadrature

The accuracy of the method outlined above is controlled by two competing error terms: *quadrature error* incurred from approximating the layer potential Eq. (2.4) with Eq. (2.8) in Step 4 and *extrapolation error* due to approximating the singular integral with an extratpolated value in Step 5. Both errors are determined by the location of check points relative to the patches in $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ (see Heuristic 2.1 and Theorem 2.2).

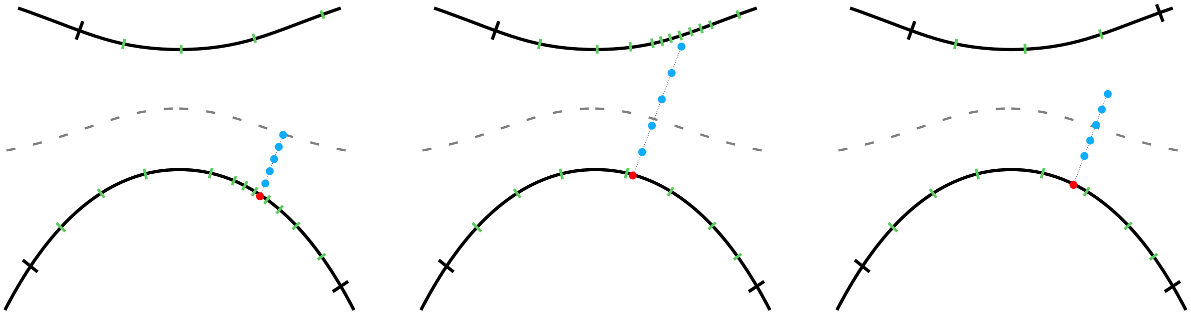


Figure 2.3: Possible check point configurations. A 2D example depicting three choices of a and b in Eq. (2.10). Shown is the boundary $\hat{\Gamma}$, with black tick marks denoting patch boundaries of $\mathcal{P}_{\text{coarse}}$, green tick marks denoting patch boundaries of $\mathcal{P}_{\text{fine}}$, the target point (red dots), its check points (blue dots) along the normal closest to the target point, and the medial axis of $\hat{\Gamma}$ (gray dotted line). Large (left) and small (middle) values of a and b can cause clustering of check points near to $\hat{\Gamma}$, which requires large amounts of upsampling to compute the potential accurately. Using the medial axis as a heuristic to for admissibility (right), we can minimize the amount of adaptive upsampling required.

In Figure 2.3, we show three examples of different choices of check point locations to evaluate the potential at a point with hedgehog . In Fig. 2.3-left, $\boldsymbol{c}_0$ is placed close to the target point, while in Fig. 2.3-middle, $\boldsymbol{c}_0$ is far from the target point, but $\boldsymbol{c}_p$ is close to a non-local piece of $\hat{\Gamma}$. Both cases will require excessive refinement of $\mathcal{P}_{\text{coarse}}$ in order to resolve Eq. (2.8) accurately with $\mathcal{P}_{\text{fine}}$. On the other hand, in Fig. 2.3-right, we can either perform one refinement step on $\mathcal{P}_{\text{coarse}}$ or adjust a and b, which will result in fewer patches in $\mathcal{P}_{\text{fine}}$, and therefore provide a faster integral evaluation, while maintaining accuracy.

In an attempt to strike this balance between speed and accuracy, we need certain constraints on the geometry of $\hat{\Gamma}$ to ensure the efficient and accurate application of hedgehog , which we impose on the patch sets $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$. We will first outline our constraints on the quadrature patch sets $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ which allow for accurate evaluation with hedgehog .

We note here that this is largely a consequence of the global nature of our method. We compute accurate potentials at each check point with a single large quadrature evaluation over the boundary. Alternatively, one could use a *local* approach which performs a local correction to an inaccurate FMM evaluation of a singular/near-singular integral. The net effect of this is a faster FMM evaluation without much concern for where check points are placed, but many small quadratic computations on individual patches. We explore this tradeoff by comparing hedgehog with a competing approach in Section 2.6.2.

2.3.2.1 Admissibility criteria

A set of patches $\mathcal{P}$ is *admissibile* if the following statements are satisfied on each quadrature patch in $\mathcal{P}$:

1 The error of a surface patch $\boldsymbol{P}_i$ approximating an embedding γ_r is below some absolute target accuracy ε_{g}

2 The interpolation error of the boundary condition f is below some absolute target accuracy ε_{f}

3 For each check center $\hat{\boldsymbol{c}}_j$ corresponding to the quadrature point $\boldsymbol{y}_j$ on the surface, the closest point on $\hat{\Gamma}$ to $\hat{\boldsymbol{c}}_j$ is $\boldsymbol{y}_j$.

Criterion 1 is required to ensure that $\hat{\Gamma}$ approximates Γ with sufficient accuracy to solve the integral equation. We discuss how to choose ε_{g} in [Morse et al. 2020b, Section 6]; for the tests in this paper, we simply choose $\varepsilon_{\mathrm{g}} < \varepsilon_{\mathrm{target}}$. Criterion 2 guarantees that f can be represented at least as accurately as the desired solution accuracy. We therefore similarly choose $\varepsilon_{\mathrm{f}} < \varepsilon_{\mathrm{target}}$. Criterion 3 balances the competing geometric constraints of cost and accuracy by flexibly placing check points as far as possible from $\hat{\Gamma}$ without causing too much upsampling on other patches. If a check point $\boldsymbol{c}$ constructed from a surface patch $\boldsymbol{P}$ is too close to another surface patch $\boldsymbol{P}'$, Criterion 3 will indicate that $\boldsymbol{P}$ is inadmissible. If $\boldsymbol{P}$ is subdivided into its children, new check points $\boldsymbol{c}'$ generated from these children of $\boldsymbol{P}$ will be closer to $\boldsymbol{P}$ and further from $\boldsymbol{P}'$. Since check points are placed at distances proportional to $L(\boldsymbol{P})$, repeated refinement of $\boldsymbol{P}$ will eventually satisfy Criterion 3.

2.3.2.2 Upsampling criteria

Once we have a set of admissible surface patches satisfying Criteria 1 to 3, we need to determine the upsampled quadrature patches $\mathcal{P}_{\text{fine}}$ that ensure that the check points generated from $\mathcal{P}_{\text{coarse}}$ are in Ω_I, i.e., $\|u(\boldsymbol{c}) - \hat{u}(\boldsymbol{c}, \mathcal{P}_{\text{fine}})\| < \varepsilon_{\mathrm{target}}$. To achieve this, we need a criterion to determine which patches are "too close" to a given check point for the error to be below $\varepsilon_{\mathrm{target}}$. We make the following assumption about the accuracy of our smooth quadrature rule: *Eq.* (2.8) *is accurate to* $\varepsilon_{\mathrm{target}}$ *at points further than* $L(\boldsymbol{P})$ *from* $\boldsymbol{P}$*, for* $\varepsilon_{\mathrm{target}} > 10^{-12}$. This is motivated by [af Klinteberg and Tornberg 2017; Barnett 2014], which demonstrate the rapid convergence of the layer potential quadrature error with respect to $\|\boldsymbol{x} - \boldsymbol{s}_{\boldsymbol{x}}\|_2$. For sufficiently high quadrature orders, such as $q = 20$, this assumption seems to hold in practice. We say that a point $\boldsymbol{x}$ is *near* to $\boldsymbol{P}$ if the distance from $\boldsymbol{x}$ to $\boldsymbol{P}$ is less than $L(\boldsymbol{P})$; otherwise, $\boldsymbol{x}$ is *far* from $\boldsymbol{P}$. We would like all check points required for the singular/near-singular evaluation of the discretization of Eq. (2.4) using hedgehog to be far from all patches in $\mathcal{P}_{\text{fine}}$. If this is satisfied, then we know that the Clenshaw-Curtis quadrature rule will be accurate to 10^{-12} at each check point.

2.3.3 Refinement algorithm preliminaries

Computing the distance from a check point to a given patch is a fundamental step in verifying the constraints on $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ from Sections 2.3.2.1 and 2.3.2.2. Before detailing our refinement algorithms to enforce these criteria, we introduce several geometric algorithms and data structures that will be used to compute the closest point on piecewise polynomial surfaces.

2.3.3.1 AABB trees

In order to implement our algorithms to enforce admissibility efficiently, we use a fast spatial data structure to find the patches that are close to a query point $\boldsymbol{x}$. In [Rachh et al. 2017; Wala and Klöckner 2019a], the quadtree and octree within an FMM is extended to support the geometric queries needed for a fast QBX algorithm. In this work, we use an axis-aligned bounding box (AABB) tree, which is a type of bounding volume hierarchy [Samet 2006], implemented in `geogram` [Lévy 2015]. An AABB is a tree with nodes corresponding to bounding boxes and leaves corresponding to bounding boxes containing single objects. A bounding box B_0 is a child of another box B_1 if $B_0 \subset B_1$; the root node is a bounding box of the entire domain of interest. Operations supported by AABB trees include: (i) finding all bounding boxes containing a query point, (ii) finding all bounding boxes that intersect another query box, (iii) finding the closest triangle to a query point (because triangles have trivial bounding boxes). By decoupling geometric queries from fast summation, the individual algorithms can be more thoroughly optimized, in exchange for the additional memory overhead of maintaining two distinct data structures. The query algorithm presented in [Lu et al. 2019] likely has better parallel scalability, but AABB trees are faster for small to medium problem sizes on a single machine due to less redundant computation.

To define an AABB tree for our patch-based surface $\hat{\Gamma}$, we make use of the following fact: the control points of a Bézier surface ($\boldsymbol{a}_{\ell m}$'s from Eq. (2.6)) form a convex hull around the surface that they define [Farin 1988]. As a result, we can compute a bounding box of a surface or quadrature patch $\boldsymbol{P}$ directly from the Bézier coefficients simply by computing the maximum and minimum values of each component of the $\boldsymbol{a}_{\ell m}$'s, as shown in Fig. 2.4-middle. This bounding box can then be inserted into the AABB tree as a proxy for a surface or quadrature patch.

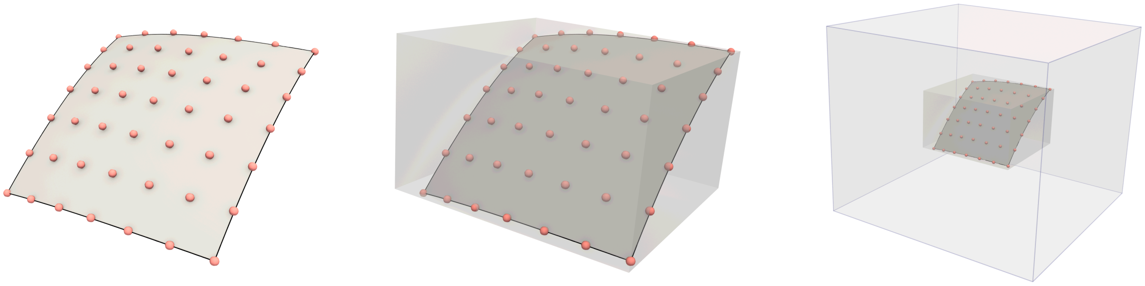


Figure 2.4: Relationship between control points and bounding boxes. Left: a patch in the tensor product Bézier basis, with control points ($\boldsymbol{a}_{\ell m}$'s from Eq. (2.6)) plotted. The convex hull of the control points of a patch are guaranteed to contain the patch. Center: The patch bounding box, computed from the control points. Right: The near-zone bounding box of the patch from Section 2.3.5 computed by inflating the bounding box by $L(\boldsymbol{P})$.

2.3.3.2 Computing the closest point to a patch

To find a candidate closest patch $\boldsymbol{P}_{i_0}$ to $\boldsymbol{x}$, we construct a fine triangle mesh and bounding boxes of each patch in $\mathcal{P}_{\text{coarse}}$ and insert them into an AABB tree. We can query the AABB tree for the nearest triangle to $\boldsymbol{x}$ with the AABB tree, which corresponds to $\boldsymbol{P}_{i_0}$. We then compute the accurate true distance d_{i_0} to $\boldsymbol{P}_{i_0}$ using a constrained Newton method, presented in detail in Appendix A.2.

However, there may be other patches whose distance to $\boldsymbol{x}$ is less than d_{i_0}, as shown in Fig. 2.5. To handle this case, we then query the AABB tree for all patches $\boldsymbol{P}_{i_1}, \dots, \boldsymbol{P}_{i_k}$ that are distance at most d_{i_0} from $\boldsymbol{x}$. This is achieved by forming a query box centered at $\boldsymbol{x}$ with edge length $2d_{i_0}$ and querying the AABB tree for all intersection bounding boxes. The precise distance is then computed for each patch $\boldsymbol{P}_{i_1}, \dots, \boldsymbol{P}_{i_k}$ with Appendix A.2 and the smallest distance is chosen. We summarize this process in Algorithm 1.

Algorithm 1: Compute the closest point to $\boldsymbol{x}$.

```
   Data: A set of quadrature patches $\mathcal{P}$, a query point $\boldsymbol{x}$, Newton method tolerance $\varepsilon_{\text{opt}}$
   Result: The closest point $\boldsymbol{s_x}$ on $\mathcal{P}$ to $\boldsymbol{x}$
1  Construct an AABB tree $T_T$ from a fine triangle mesh of the quadrature patches of $\mathcal{P}$
2  Construct an AABB tree $T_B$ from bounding boxes of quadrature patches in $\mathcal{P}$.
3  $\tau_0$ = closest triangle to $\boldsymbol{x}$ computed with $T_T$
4  $\boldsymbol{P}_{i_0}$ = patch corresponding to $\tau_0$
5  Find the closest point $\boldsymbol{s}_{\boldsymbol{x},0}$ on $\boldsymbol{P}_{i_0}$ to $\boldsymbol{x}$ with [Morse et al. 2020b, Section 2].
6  $d_{i_0} = \|\boldsymbol{x} - \boldsymbol{s}_{\boldsymbol{x},0}\|_2$
7  $B_{d_{i_0}}(\boldsymbol{x})$ = a box centered a $\boldsymbol{x}$ with edge length $2d_{i_0}$
8  Find the boxes $B_{i_1}, \ldots B_{i_k}$ in $T_B$ that intersect $B_{d_{i_0}}(\boldsymbol{x})$
9  for $B_{i_j} \in B_{i_1}, \ldots B_{i_k}$ do
10     $\boldsymbol{P}_{i_j}$ = quadrature patch corresponding to $B_{i_j}$
11     Find the closest point $\boldsymbol{s}_{\boldsymbol{x},j}$ on $\boldsymbol{P}_{i_j}$ to $\boldsymbol{x}$ with [Morse et al. 2020b, Section 2] to precision
         $\varepsilon_{\text{opt}}$.
12     $d_{i_j} = \|\boldsymbol{x} - \boldsymbol{s}_{\boldsymbol{x},j}\|_2$
13 $j^* = \text{argmin}_j\{d_{i_j}\}$
14 return $\boldsymbol{s}_{\boldsymbol{x},j^*}$
```

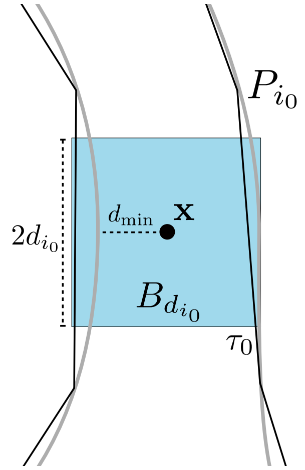


Figure 2.5: A 2D schematic of near-patch candidate selection. A visual depiction of the quantities defined in lines 3-7 of Algorithm 1 (shown here in 2D for simplicity), with notation matching Algorithm 4. The triangle-mesh proxy is drawn in as black lines and patches are drawn as gray curves. We have found an initial closest triangle τ_0 to $\boldsymbol{x}$ corresponding to patch $\boldsymbol{P}_{i_0}$ and computed $d(\boldsymbol{x}, \boldsymbol{P}_{i_0}) = d_{i_0}$. We then query the AABB tree for all patches that intersect box $B_{d_{i_0}}$ with edge length $2d_{i_0}$, shown in blue. There is clearly a patch that is closer to $\boldsymbol{x}$ than $\boldsymbol{P}_{i_0}$ that will be returned from the query, which will be distance d_{min} from $\boldsymbol{x}$.

2.3.4 Admissibility algorithm

Our algorithm to enforce Criteria 1 to 3 proceeds as follows:

- To enforce Criterion 1, we adaptively fit a set of surface patches to the embeddings γ_r representing Γ. We construct a bidegree (n, n) piecewise polynomial least-squares approximation $\boldsymbol{P}_i$ in the form of Eq. (2.6) to γ_r on I^2. If $\boldsymbol{P}_i$'s domain $\mathcal{D}_i$ is obtained by refinement of E_r, we fit $\boldsymbol{P}_i \circ \eta_i$ to γ_r on $\mathcal{I}^2$, using $4n \times 4n$ samples on $\mathcal{I}^2$. If the pointwise error of $\boldsymbol{P}_i$ and its partial derivatives is greater than ε_{g}, then it is quadrisected and the process is repeated.

- Once the embeddings are resolved, we resolve f on each surface patch produced from the previous step in a similar fashion to enforce Criterion 2. However, rather than a least-squares approximation in this stage, we use piecewise polynomial interpolation.

- To enforce Criterion 3, we construct the set of check centers $\hat{\boldsymbol{c}}_I$ which correspond to the check points required to evaluate the solution at the quadrature nodes $\boldsymbol{y}_I$. For each check center $\hat{\boldsymbol{c}}_I$, we find the closest point $\boldsymbol{s}_{\hat{\boldsymbol{c}}_I} \in \hat{\Gamma}$. If $\|\boldsymbol{s}_{\hat{\boldsymbol{c}}_I} - \boldsymbol{y}_I\| \geq \varepsilon_{\mathrm{opt}}$, we split the quadrature patch $\boldsymbol{P}$ containing $\boldsymbol{y}_I$. The tolerance $\varepsilon_{\mathrm{opt}}$ is used in the Newton's method in [Morse et al. 2020b, Section 2]; we usually choose $\varepsilon_{\mathrm{opt}} = 10^{-14}$. Since $d(\hat{\boldsymbol{c}}_I, \hat{\Gamma})$ is proportional to $L_{\boldsymbol{y}_I}$, the new centers $\hat{\boldsymbol{c}}_I$ for the refined patches will be closer to the surface. We use Algorithm 1 to compute $\boldsymbol{s}_{\hat{\boldsymbol{c}}_I}$. However, in the case of check points, we can skip lines 1-6 to compute d_{i_0}, since $\hat{\boldsymbol{c}}_I$ is $R + r(p+1)/2$ away from $\boldsymbol{y}_I \in \boldsymbol{P}(D)$ by construction. We can apply lines 7-14 of Algorithm 1 with $d_{i_0} = R + r(p+1)/2$ to compute $\boldsymbol{s}_{\hat{\boldsymbol{c}}_I}$.

We summarize the algorithm to enforce Criterion 3 in Algorithm 2. At each refinement iteration, the offending patches are decreased by quadrisection, which reduces the distance from the quadrature point $\boldsymbol{y}_I$ to its checkpoints. This eventually satisfies Criterion 3 and the algorithm terminates.

Algorithm 2: Enforce admissibility Criterion 3 on a set of quadrature patches.

```
Data: A set of quadrature patches $\mathcal{P}$, optimization tolerance $\varepsilon_{\text{opt}}$
Result: An admissible set of quadrature patches $\mathcal{P}$
$\mathcal{P} = \mathcal{P}_{\text{coarse}}$
Mark all patches in $\mathcal{P}$ as inadmissible.
while any patch in $\mathcal{P}$ is inadmissible do
    Construct an AABB tree $T$ as described in Section 2.3.3.2 from $\mathcal{P}$
    for $\boldsymbol{P} \in \mathcal{P}$ do
        if $\boldsymbol{P}$ is inadmissible then
            Construct a set of check centers $C_{\boldsymbol{P}}$ for each $\boldsymbol{y}_J \in \boldsymbol{P}(D)$
            for $\hat{\boldsymbol{c}} \in C_{\boldsymbol{P}}$ do
                $d_{i_0} = R + r(p+1)/2$
                Compute $\boldsymbol{s}_{\hat{\boldsymbol{c}}}$ with lines 7-14 of Algorithm 1 with precision $\varepsilon_{\text{opt}}$ and $d_{i_0}$.
                if $\|\boldsymbol{s}_{\hat{\boldsymbol{c}}} - \boldsymbol{y}_J\|_2 < \varepsilon_{\text{opt}}$ then
                    Mark $\boldsymbol{P}$ as admissible.
                else
                    Mark $\boldsymbol{P}$ as inadmissible.
                    break // only need one bad check center to mark $\boldsymbol{P}$ for
                        refinement
    for $\boldsymbol{P} \in \mathcal{P}$ do
        if $\boldsymbol{P}$ is inadmissible then
            Split $\boldsymbol{P}$ into its four child patches, mark each as inadmissible, and replace $\boldsymbol{P}$
              with its children in $\mathcal{P}$.
return $\mathcal{P}$
```

2.3.5 Adaptive upsampling algorithm

Before detailing our upsampling algorithm to satisfy the criteria outlined in Section 2.3.2.2, we must define the notion of a *near-zone bounding box* of a quadrature patch $\boldsymbol{P}$, denoted $B_{\text{near}}(\boldsymbol{P})$. The near-zone bounding box of $\boldsymbol{P}$ is computed as described in Section 2.3.3.1, but then is inflated by $2L(\boldsymbol{P})$, as shown in Fig. 2.4-right. This inflation guarantees that any point $\boldsymbol{x}$ that is near $\boldsymbol{P}$ is contained in $B_{\text{near}}(\boldsymbol{P})$ and, for an admissible set of quadrature patches $\mathcal{P}_{\text{coarse}}$, that any $\boldsymbol{x} \in \Omega_N$ must be contained in some quadrature patch's near-zone bounding box. This means that by forming $B_{\text{near}}(\boldsymbol{P})$ for each quadrature patch in $\mathcal{P}_{\text{fine}}$, a check point is in Ω_I if it is not contained in any near-zone bounding boxes.

To compute the upsampled patch set from $\mathcal{P}_{\text{coarse}}$, we initially set $\mathcal{P}_{\text{fine}} = \mathcal{P}_{\text{coarse}}$, compute the near-zone bounding boxes of each patch in $\mathcal{P}_{\text{fine}}$ and insert them into an AABB tree. We also construct the set of check points C required to evaluate our discretized layer-potential with hedgehog (Section 2.3.1). For each check point $\boldsymbol{c} \in C$, we query the AABB tree for all near-zone bounding boxes that contain $\boldsymbol{c}$. If there are no such boxes, we know $\boldsymbol{c}$ is far from all quadrature patches and can continue. If, however, there are near-zone bounding boxes $B_{i_0}, \dots, B_{i_k}$ containing $\boldsymbol{c}$, we compute the distances d_{i_k} from $\boldsymbol{c}$ to $\boldsymbol{P}_{i_1}, \dots, \boldsymbol{P}_{i_k}$ using Appendix A.2. If $d_{i_k} < L(\boldsymbol{P}_{i_k})$, we replace $\boldsymbol{P}_{i_k}$ in $\mathcal{P}_{\text{fine}}$ with its four children produced by quadrisection.

To improve the performance of this refinement procedure, we allow for the option to skip the Newton method in Algorithm 1 and immediately refine all patches $\boldsymbol{P}_{i_0}, \dots \boldsymbol{P}_{i_k}$. This is advantageous in the early iterations of the algorithm, when most check points are near to patches by design. We allow for a parameter n_{skip} to indicate the number of iterations to skip the Newton optimization and trigger refinement immediately. We typically set $n_{\text{skip}} = 2$. We summarize our algorithm in Algorithm 3.

2.3.6 Marking target points for evaluation

Once we have solved Eq. (2.9) for ϕ on $\hat{\Gamma}$, we need the ability to evaluate Eq. (2.4) at an arbitrary set of points in the domain. For a target point $\boldsymbol{x}$, in order apply the algorithm in Section 2.3.1, we need to determine whether or not $\boldsymbol{x} \in \Omega$ and, if so, whether $\boldsymbol{x}$ is in Ω_N, Ω_I or Ω_F. Both of these questions can be answered by computing the closest point $\boldsymbol{s_x}$ on $\hat{\Gamma}$ to $\boldsymbol{x}$. If $\boldsymbol{n}(\boldsymbol{s_x}) \cdot (\boldsymbol{x} - \boldsymbol{s_x}) < 0$, then $\boldsymbol{x} \in \Omega$. As we have seen in Section 2.3.2.2, the distance $\|\boldsymbol{x} - \boldsymbol{s_x}\|$ determines whether $\boldsymbol{x} \in \Omega_N, \Omega_I$ or Ω_F. However, for large numbers of target points, a brute force calculation of closest points on $\hat{\Gamma}$ to all target points is prohibitively expensive. We present an accelerated algorithm combining Algorithm 1 and an FMM evaluation to require only constant work per target point.

Algorithm 3: Adaptively upsample to accurately evaluate Eq. (2.8) at check points.

```
Data: An admissible patch set $\mathcal{P}$, number of iterations $n_{\text{skip}}$ before using [Morse et al. 2020b, Section 2]
Result: An upsampled set of quadrature patches
Compute inflated near-zone bounding boxes $B_1, \ldots, B_N$ of each $\boldsymbol{P} \in \mathcal{P}$.
Construct an AABB tree $T$ from the near-zone bounding boxes.
Construct all check points $C$ required to evaluate the Eq. (2.5) on $\mathcal{P}$.
$\mathcal{P}_{\text{fine}} = \mathcal{P}$
Mark all check points in $C$ as near.
$i = 0$
while any $\boldsymbol{c} \in C$ is marked near do
    for $\boldsymbol{c} \in C$ do
        if $\boldsymbol{c}$ is marked near then
            Query $T$ for all bounding boxes $B_{i_1}, \ldots B_{i_k}$ containing $\boldsymbol{c}$.
            $\boldsymbol{P}_{i_1}, \ldots \boldsymbol{P}_{i_k}$ = patches corresponding to boxes $B_{i_1}, \ldots B_{i_k}$
            Mark $\boldsymbol{c}$ as far
            for $P \in \boldsymbol{P}_{i_1}, \ldots \boldsymbol{P}_{i_k}$ do
                if $i > n_{\text{skip}}$ then
                    Find the closest point $\boldsymbol{s_c}$ on $\boldsymbol{P}$ to $\boldsymbol{c}$ with Algorithm 1.
                    if $\|\boldsymbol{s_c} - \boldsymbol{c}\|_2 < L(\boldsymbol{P})$ then
                        Split $\boldsymbol{P}$ and replace it in $\mathcal{P}_{\text{fine}}$ with its children.
                        Mark $\boldsymbol{c}$ as near
                else
                    Split $\boldsymbol{P}$ and replace it in $\mathcal{P}_{\text{fine}}$ with its children.
                    Mark $\boldsymbol{c}$ as near
    $i = i + 1$
```

2.3.6.1 Marking and culling far points

A severe shortcoming of Algorithm 1 is that its performance deteriorates as the distance from $\boldsymbol{x}$ to $\hat{\Gamma}$ increases. Consider the case where $\hat{\Gamma}$ is a sphere with radius r with $\boldsymbol{x}$ at its center. The first stage of Algorithm 1 returns a single quadrature patch that is distance r from $\boldsymbol{x}$; the next stage will return all quadrature patches. This will take $O(N)$ time to check the distance to each patch. Even on more typical geometries, we observe poor performance of Algorithm 1 when $\boldsymbol{x}$ is far from $\hat{\Gamma}$.

To address this, we use an additional FMM -based acceleration step to mark most points far from $\hat{\Gamma}$ before using applying Algorithm 1. Our approach is based on computing the generalized winding number [Jacobson et al. 2013] of $\hat{\Gamma}$ at the evaluation points. For closed curves in $\mathbb{R}^2$, the *winding number* at a point counts the number of times the curve travels around that point. The *generalized winding number* of a surface $\hat{\Gamma}$ at a point $\boldsymbol{x} \in \mathbb{R}^3$ can be written as

$$\omega_{\hat{\Gamma}}(\boldsymbol{x}) = -\frac{1}{4\pi} \int_{\hat{\Gamma}} \frac{(\boldsymbol{x} - \boldsymbol{y}) \cdot \boldsymbol{n}}{\|\boldsymbol{x} - \boldsymbol{y}\|^3} d\boldsymbol{y}_{\hat{\Gamma}} \tag{2.12}$$

We recognize this integral as the double-layer potential in Eq. (2.4) for a Laplace problem with $\phi = 1$. Its values in $\mathbb{R}^3$ are [Kress 1999]:

$$\omega_{\hat{\Gamma}}(x) = \begin{cases} 1 & \boldsymbol{x} \in \Omega \setminus \hat{\Gamma} \\ 1/2 & \boldsymbol{x} \in \hat{\Gamma} \\ 0 & \boldsymbol{x} \in \mathbb{R}^3 \setminus \overline{\Omega} \end{cases} \tag{2.13}$$

Eq. (2.12) can be evaluated using the same surface quadrature in Eq. (2.8) using an FMM in $O(N)$ time. While the quadrature rule is inaccurate close to the surface, Ω_F is defined precisely as the zone where the quadrature rule is sufficiently accurate. For this reason, we use

$$|\omega_{\hat{\Gamma}}(\boldsymbol{x}) - 1| < \varepsilon_{\text{target}} \tag{2.14}$$

to mark points $\boldsymbol{x} \in \Omega_F \subset \Omega$ and a similar relation

$$|\omega_{\hat{\Gamma}}(\boldsymbol{x})| < \varepsilon_{\text{target}} \tag{2.15}$$

to mark points $\boldsymbol{x} \notin \Omega$. This approach is similar in spirit to the spectrally accurate collision detection scheme of [Quaife and Biros 2014, Section 3.5]. Unlike [Quaife and Biros 2014], however, we do *not* use singular integration to mark all points. This isn't possible since at this stage since we do not yet know which target points require singular integration. We use the FMM evaluation purely as a culling mechanism before applying the full marking algorithm.

Remark: Since the quadrature rule may be highly inaccurate for points close to the surface, due the near-singular nature of the integrand, $\omega_{\hat{\Gamma}}(\boldsymbol{x})$ may happen to be close to one or zero. We highlight that it is possible that points outside Ω_F may be mismarked, although we have not observed this in practice.

2.3.6.2 Full marking algorithm

We combine the algorithms of the previous two sections into a single marking pipeline for a general set of target points in $\mathbb{R}^3$, by first applying the algorithm of Section 2.3.6.1 to mark all points satisfying Eq. (2.14) then passing the remaining points to Algorithm 1. The full marking algorithm is summarized as Algorithm 4.

Algorithm 4: Mark points in regions Ω_F, Ω_I and Ω_N.

```
Data: An admissible set of quadrature patches $\mathcal{P}$, $\varepsilon_{\text{target}}$, target points $\boldsymbol{X}$
Result: A marked set of target points $\boldsymbol{X}$
$\phi_0 = 1$
$\omega_{\hat{\Gamma}}$ = Laplace_FMM($\mathcal{P}, \boldsymbol{X}, \phi_0$)
for $\boldsymbol{x} \in X$ do
    if $|\omega_{\hat{\Gamma}}(\boldsymbol{x}) - 1| < \varepsilon_{\text{target}}$ then
        Mark $\boldsymbol{x}$ as inside $\Omega$.
        Mark $\boldsymbol{x}$ as in $\Omega_{\text{F}}$.
    else if $|\omega_{\hat{\Gamma}}(\boldsymbol{x})| < \varepsilon_{\text{target}}$ then
        Mark $\boldsymbol{x}$ as outside $\Omega$.
for $\boldsymbol{x} \in X$ do
    if $\boldsymbol{x}$ is unmarked then
        Compute the closest point $\boldsymbol{s_x}$ to $\boldsymbol{x}$ with Algorithm 1
        $d_{\min} = \|\boldsymbol{s_x} - \boldsymbol{x}\|_2$
        if $d_{\min} \leq L_{\boldsymbol{s_x}}$ then
            Mark $\boldsymbol{x}$ as in $\Omega_N$
        else
            Mark $\boldsymbol{x}$ as in $\Omega_I$
        if $\boldsymbol{n}(\boldsymbol{s_x}) \cdot (\boldsymbol{x} - \boldsymbol{s_x}) < 0$ then
            Mark $\boldsymbol{x}$ as inside $\Omega$
        Mark $\boldsymbol{x}$ as outside $\Omega$
```

2.4 Error Analysis

As with other approximation-based quadrature methods, hedgehog has two primary sources of error: the quadrature error e_Q incurred as a result of evaluating potential at the check points and the extrapolation error e_E due to evaluating the polynomial approximation of the potential at the target point, assuming $\mathcal{P}_{\text{coarse}}$ is admissible. Let

$$e_Q(\boldsymbol{x}) = \left|\sum_{s=0}^{p}(u(c_s) - \hat{u}(c_s, \mathcal{P}_{\text{fine}}))\ell_s(t_{\boldsymbol{x}})\right|, \tag{2.16}$$

$$e_E(\boldsymbol{x}) = \left|u(\boldsymbol{x}) - \sum_{s=0}^{p} u(c_s)\ell_s(t_{\boldsymbol{x}})\right|, \tag{2.17}$$

$$e_{\text{hedgehog}}(\boldsymbol{x}) \leq e_Q(\boldsymbol{x}) + e_E(\boldsymbol{x}), \tag{2.18}$$

$$\tag{2.19}$$

where $u(\boldsymbol{x})$ and $\hat{u}(\boldsymbol{x}, \mathcal{P}_{\text{fine}})$ are defined in Eqs. (2.4) and (2.8) and $\ell_s(t)$ is the s-th Lagrange polynomial defined on the points $\{0, 1, \ldots, p\}$. We define $t_{\boldsymbol{x}}$ such that $\boldsymbol{x} = -\boldsymbol{n}(\boldsymbol{y})(R+t_{\boldsymbol{x}}r)$, so $t_{\boldsymbol{x}} = \frac{\|\boldsymbol{x}-\boldsymbol{y}\|-R}{r}$. In this section, we first prove that we achieve high-order accuracy with our singular/near-singular evaluation scheme in Section 2.3.1 with respect to extrapolation order p and quadrature order q. We then detail the impact of surface approximation on overall solution accuracy.

2.4.1 Quadrature error

We briefly state a tensor-product variation of known Clenshaw-Curtis quadrature error results as applied to smooth functions in 3D. This estimate is derived based on assumptions detailed in Appendix A.4 that, in general, are difficult to verify in practice and may not hold for all functions we consider. For this reason, we refer to it as a heuristic.

Heuristic 2.1. *Let the boundary $\hat{\Gamma}$ be discretized by quadrature patches over the domains* $[-h, h]$

and the boundary condition f in Eq. (2.2) be at least C^k. Apply the q-th order Clenshaw-Curtis quadrature rule to the double-layer potential $u(\boldsymbol{x})$ given in Eq. (2.7) and let $\boldsymbol{x}$ be in the interior of Ω. Then for all sufficiently large q:

$$e_{\mathrm{Q}}(\boldsymbol{x}) \lesssim \frac{128h^{k+1}}{15\pi k(2q+1-k)^k}\tilde{V}, \tag{2.20}$$

where

$$\tilde{V} = \max_{i=1,\ldots,N}\max_{\alpha,\beta\leq k}\left\|\frac{\partial^{\alpha+\beta}}{\partial u^{\alpha}\partial v^{\beta}}\left(\frac{\partial G(\boldsymbol{x},\boldsymbol{P}_i(s,t))}{\partial \boldsymbol{n}}\phi(\boldsymbol{P}_i(s,t))g_{\boldsymbol{P}_i}(s,t)\right)\right\|_T, \tag{2.21}$$

$g_{\boldsymbol{P}}$ is the determinant of the metric tensor of a patch $\boldsymbol{P}$ implicit in Eq. (2.7), $\lesssim$ means "approximately less than or equal to," and $\|\zeta\|_T = \|\zeta'/\sqrt{1-x^2}\|_1$.

This heuristic captures the qualitative behavior of the error. We present the derivation of Heuristic 2.1 in Appendix A.4. The presence of the derivatives of ϕ in heuristic 2.1 largely captures two troublesome cases for integral equations: (a) when the underlying singularities of f are close to Γ; and (b) when pieces of non-local are nearly touching. Both of these cases, if not properly resolved with adaptive refinement, can induce singularities in the harmonic extension of ϕ and cause hedgehog to accumulate error. These sources of error are handled by Criteria 2 and 3 in Section 2.3.2 and the algorithms to enforce them.

As $\boldsymbol{x} \to \hat{\Gamma}$, the value of k required in Heuristic 2.1 grows rapidly due to growing higher order derivatives of the integrand. Such large values of q and k imply that smooth quadrature rules are cost-prohibitive; this is the problem that singular/near-singular quadrature schemes like hedgehog aim to address. This means that heuristic 2.1 is insufficient for direct application to Eq. (2.7), as we know. Moreover, this estimate is too loose to determine whether hedgehog or smooth quadrature is required to evaluate the potential. The assumption in Section 2.3.2.2 addresses this problem by providing a cheap, reasonably robust criterion for refinement that is motivated by existing analyses [af Klinteberg and Tornberg 2017; Barnett 2014] instead of relying on Heuristic 2.1.

2.4.2 Extrapolation error

A reasonable critique of hedgehog is its reliance on an equispaced polynomial interpolant to extrapolate values of u to the target point. Despite using the first-kind barycentric interpolation formula [Webb et al. 2012], polynomial interpolation and extrapolation in equispaced points is well-known for an exponentially growing Lebesgue constant and poor stability properties as the number of points p increases [Trefethen and Weideman 1991; Platte et al. 2011]. Recently [Demanet and Townsend 2016] demonstrated stable extrapolation in equispaced $p + 1$ points using least-sqaures polynomials of degree $\sqrt{p}$. However, these results are asymptotic in nature and don't tell the full story for small to moderate values of p, as in the hedgehog context.

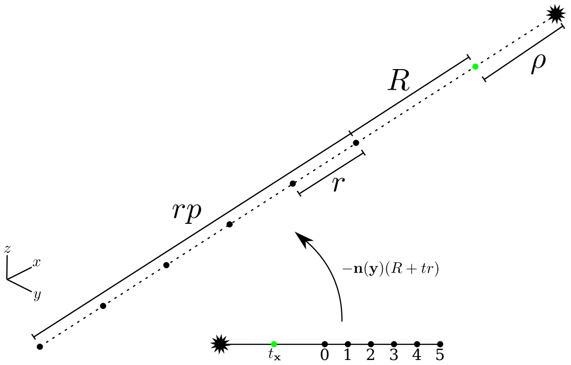
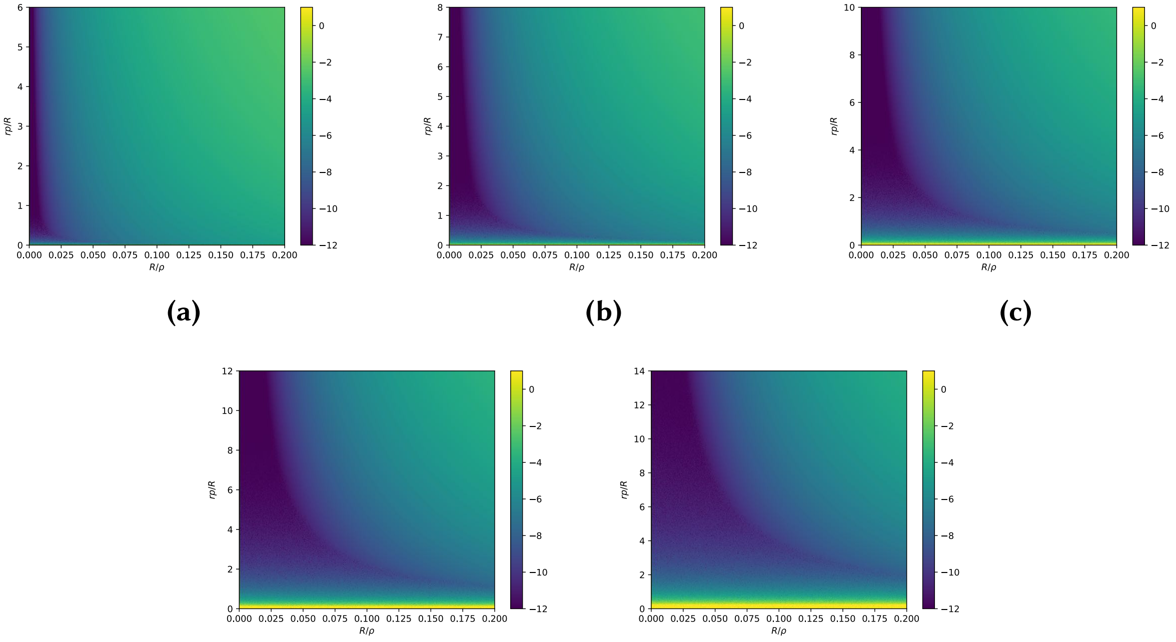


Figure 2.6: Diagram of extrapolation setup. The toy setup used to study the extrapolation error of a singular function. We choose a simple point singularity $\mu(t) = \frac{1}{\|t-q\|}$ where $q = (\rho, 0, 0)$ (black star) with $\rho = -.1$. We choose samples at the points $t_i = (R + ir, 0, 0)$ for $i = 0, \ldots, p$ (black dots) and extrapolate the values $\mu(t_0), \ldots, \mu(t_p)$ to $t = 0$ (green dot).

We begin our discussion with a simple representative experiment in equispaced extrapolation. Figure 2.6 depicts a minimal extrapolation setup in 3D of a simple singular function $\mu(t) = 1/\|t-q\|$ along a line, with $q = (\rho, 0, 0)$ and $\rho = -.1$. We extrapolate exact values of μ from p points, located at $t_i = (R + ir, 0, 0)$, to the origin. This closely mimics the worse-case extrapolation error in 1D of a function analytic in a Bernstein ellipse with a real axis intercept of $\rho + R + rp/2$. We repeat this for a large range of values of r and R for various values of p. The log of the relative error is plotted in Figures 2.7(a) to 2.7(e) as a function of the relative extrapolation interval size rp/R and the scaled extrapolation distance R/ρ.

As mentioned in [Rahimian et al. 2018, Section 3.4], the adaptive refinement of $\mathcal{P}_{\text{coarse}}$ resolves the boundary data f, and therefore u and ϕ, on the length scale L of the patch. This means we can reasonably assume that the distance of the nearest singularity is $O(L)$ from $\hat{\Gamma}$, i.e., $\rho = \lambda L$ for some λ. In the context of hedgehog , we know that $R = bL(P)$ and $r = aL(P)$. Figures 2.7(a) to 2.7(e) are a study of extrapolation error as a function of a/b, b/λ and p.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 2.7: Empirical extrapolation error behavior. We sweep over a range of R and r values to vary Figure 2.6 and plot the log of the relative error in Figures 2.7(a) to 2.7(e), for values $p = 6, 8, 10, 12, 14$, in increasing order, from (a) to (e). In these figures, the x-axis is the extrapolation distance R normalized by ρ and the y-axis is the ratio rp/R.The top of the y-axis corresponds to $r = R$; $rp/R = 1$ corresponds to our choice of the parameter a. Assuming that $\rho = O(L)$, $r/R = a/b$ and $R/\rho = b/\lambda$ for some constant λ.

There are several important observations to make from these plots:

- Extrapolation error decreases as R/ρ decreases, as expected.
- For a fixed value of R/ρ, the extrapolation error *decreases* rapidly as rp decreases, up to a certain value r^*p. This is somewhat counterintuitive, since this means placing points closer

together and extrapolating a further distance relative to rp. For a fixed p in exact arithmetic, letting the interpolation interval size tend to zero produces an order p Taylor expansion of the solution u centered at the interval's origin, which accounts for this phenomenon.

- Beyond r^*p, the extrapolation error *increases*. The effects of finite precision eventually pollutes the convergence behavior described above. Moreover, the spacing r^* appears to be a function of p. For $p = 6$, r can be reduced to $1/p$ without any numerical issues, but by $p = 14$, only $r > \frac{1}{2}$ is a safe choice for extrapolation.

We do not aim to rigorously analyze these phenomena in this work. We highlight them to provide empirical evidence that equispaced extrapolation is a reasonable, but not optimal, choice for our problem of singular/near-singular integration and to provide some intuition for our parameter choices.

The following simple result describes the behavior of the extrapolation error in Eq. (2.17).

Theorem 2.2. *Let $u(\boldsymbol{c}(t))$ be the solution to Eq. (2.2) given by Eq. (2.4), restricted to the line $\boldsymbol{c}(t)$ in 3D intersecting $\boldsymbol{x}$, let $\boldsymbol{c}(t)$ be given by*

$$\boldsymbol{c}(t) = \boldsymbol{s_x} - (R + tr)\boldsymbol{n}(\boldsymbol{s_x}), \tag{2.22}$$

where $\boldsymbol{s_x}$ is the closest point on $\hat{\Gamma}$ to $\boldsymbol{x}$, $R = bL_{s_x}$, $r = aL_{s_x}$, $\boldsymbol{n}(\boldsymbol{s_x})$ is the outward surface normal at $\boldsymbol{s_x}$, and let $|u^{(p)}(\boldsymbol{c}(t))|$ be bounded above by C_p on the interval $[-R, R + pr]$. Let $\mathfrak{P}(t)$ be the p-th order polynomial interpolant of $u(\boldsymbol{c}(t))$ constructed from the check points $\boldsymbol{c}_0, \ldots, \boldsymbol{c}_p$, where $\boldsymbol{c}_i = \boldsymbol{c}(i)$. Then the extrapolation error associated with hedgehog *behaves according to:*

$$|u(\boldsymbol{c}(t_{\boldsymbol{x}})) - \mathfrak{P}(t_{\boldsymbol{x}})| \leq \frac{C_p}{(p+1)!}|R + rp|^p = \frac{C_p}{(p+1)!}|b + ap|^p \cdot |L|^p, \tag{2.23}$$

where $t_{\boldsymbol{x}} = \frac{\|\boldsymbol{x}-\boldsymbol{s_x}\|-R}{r}$.

Proof. We know that for a smooth function f and points $x_0, \ldots x_p$ in a 1D interval I_0, for some $\xi \in I_0$, the following relation holds for all $x \in I_0$:

$$f(x) - \mathfrak{P}(x) = \frac{f^{(p)}(\xi)}{(p+1)!} \prod_{i=0}^{p} (x - x_i). \tag{2.24}$$

Let $\mathfrak{P}$ be the pth order polynomial interpolating the points $x_0, \ldots x_p$. In the hedgehog setup, since $R + rp$ is the distance of the furthest check point to $\boldsymbol{y}$, we know that $x - x_i < R + rp$ for each i. Since $f(t) = u(\boldsymbol{c}(t))$ is harmonic, and therefore C^∞, in Ω, $|f^{(p)}(\xi)|$ can be uniformly bounded on I_0 by some constant C_p, Noting that $R = bL$ and $r = aL$ yields our result. □

For fixed values of a and b, as we let $L \longrightarrow 0$, the extrapolation error is bounded by $O(L^p)$. In practice, however, this means that we can choose a and b to minimize the constant factor $|b + ap|^p$ in Theorem 2.2. Since $p > 1$, a must be chosen to balance out the contribution of p, yet our extrapolation study shows that we can't simply set $a = 0$. We therefore choose $a \leq 1/p$ for $p = 6$ and 8, motivated by Figs. 2.7(a) and 2.7(b). Moreover, since $b < 1$, we can choose $a \leq b/p$, which allows a and b to decay at the same rate. The advantage of choosing $a \leq b/p$ is that b is a single parameter that controls the accuracy of hedgehog . Since we have fixed the quadrature order $q = 20$ to satisfy the assumption in Section 2.3.2.2, a smaller value of b will trigger more upsampling in Algorithm 3, keeping quadrature error fixed while reducing extrapolation error.

It is important to keep in mind that Theorem 2.2 only provides insight for moderate values of p; our conclusions are largely irrelevant for large p. We use $p = 6$ and $a = b/6$, leaving the construction of an optimal extrapolation extrapolation scheme to future work.

2.4.3 Geometry approximation error

Let θ be a smooth scalar function defined on the surface of $\partial\Omega$ with $|\theta| \leq 1$ and let δ be a small real constant. Suppose the boundary of the domain Ω is perturbed by δ along the normal field of $\partial\Omega$, scaled by θ, to produce the perturbed domain Ω_δ with boundary $\partial\Omega_\delta$. More concretely, for

$\boldsymbol{y} \in \partial\Omega$ and $\boldsymbol{y}_\delta \in \partial\Omega_\delta$, $\boldsymbol{y}_\delta = \boldsymbol{y} + \delta\theta\boldsymbol{n}(\boldsymbol{y})$. We can define the *Eulerian shape derivative* of u with respect to θ, denoted u_θ, at a point $\boldsymbol{x} \in \Omega_\delta \cap \Omega$ as the rate of change in u at $\boldsymbol{x}$ as $\delta \longrightarrow 0$. This quantity is of interest to us because the solution to [Morse et al. 2020a, Equation 2] on $\Omega_\delta \cap \Omega$ can be written as $u + \delta u_\theta$, where u is the solution to [Morse et al. 2020a, Equation 2] on Ω. Moreover, we can compute the shape derivative by solving a Laplace problem on the unperturbed domain [Pironneau 1982]:

$$\Delta u_\theta = 0 \text{ in } \Omega,\ u_\theta = -\theta\frac{\partial u}{\partial n}\text{on } \partial\Omega. \tag{2.25}$$

where u is the solution of the [Morse et al. 2020a, Equation 2] on Ω. For small δ, this means that the error in the solution introduced by a boundary perturbation along the field θ can be estimated by $\delta\sup_\Omega \|u_\theta\|$. Assuming the boundary is smooth and the gradient of the solution u is bounded, then

$$\|u_\theta\| \leq C_g \sup_{\partial\Omega}\left|\theta\frac{\partial u}{\partial n}\right| \leq C_g \sup_{\partial\Omega}\left|\frac{\partial u}{\partial n}\right| \tag{2.26}$$

for some real constant C_g. The right-hand side of Eq. (2.26) yields a constant C'_g, such that if $\varepsilon_\mathrm{g} < \zeta\varepsilon_\mathrm{target}/C'_g$ for some $\zeta < 1$, the change in the solution is less than $\varepsilon_\mathrm{target}$ for a sufficiently small ε_g. The constant depends implicitly on the surface geometry: for example, if an area element of $\partial\Omega$ is close to a sharp, concave corner, then $\frac{\partial u}{\partial n}$ can be arbitrarily large.

2.4.4 Limitations

Our error discussion reveals several limitations of our method. The first and most apparent shortcoming is that extrapolation instability fundamentally limits convergence order. However, for reasonable orders of convergence, up to 14, we have discussed an empirical scheme to choose parameters to maximize the available convergence behavior. Moreover, low-order surface geometries used in engineering applications will likely limit the convergence rate before it is limited by the extrapolation order, making this a non-issue in practical scenarios.

Another downside of the chosen extrapolation approach is lack of direct extension of hedgehog to oscillatory problems like the Helmholtz equation. Due to the limitation on the values of p, we can't guarantee the ability to resolve high-frequency oscillations in the solution. A new extrapolation procedure is required to do so robustly without compromising efficiency.

In [Wala and Klöckner 2019a], the authors demonstrate a relationship between the truncation error of a QBX expansion and the local curvature of $\hat{\Gamma}$. Our scheme also is susceptible to this form of error and we do not address nor analyze this in this work. This is a subtle problem that requires a detailed analysis of the surface geometry with respect to the chosen extrapolation scheme. Another limitation is the lack of an accurate error estimate to serve as an upsampling criteria in place of the criteria in Section 2.3.2.2, such as [af Klinteberg and Barnett 2019]. Extending [af Klinteberg and Barnett 2019] to 3D surfaces is non-trivial and whether the size of $\mathcal{P}_{\text{fine}}$ would be reduced enough to outweigh the added cost of the additional Newton iterations required by their scheme remains to be seen.

Finally, for certain accuracy targets and geometries, the algorithm above may lead to an impractically high number of patches in $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$. Geometries with nearly-touching non-local regions, as shown in Fig. 2.12, will see large amounts of refinement. If the nearly-touching embeddings γ_r are close enough, i.e., less than 10^{-10} apart, there is little hope of an accurate solution with a fixed computational budget. We allow the user to enforce a minimal patch size $L_{\min}$, limiting the time and memory consumption at the expense of not reaching the requested target accuracy.

2.5 Complexity Analysis

In this section, we analyze the complexity of the algorithms required by hedgehog . The input to our overall algorithm is a domain boundary Γ with N_{init} patches and boundary condition f. We begin with a summary of algorithm parameters that impact complexity:

- The number of patches N *after* admissibility refinement. This is a function of N_{init}, the geometry of Γ, the definition of f, and the choices of parameters a and b in check point construction.

- Quadrature order q and the degree of smoothness k of Γ and f. We assume that k is sufficiently high to obtain optimal error behavior for a given q by letting $k = 2q$ in Eq. (2.21).
- hedgehog interpolation order p.
- The numbers of evaluation points in different zones $\mathcal{N}_{\text{far}}$, $\mathcal{N}_{\text{inter}}$, and $\mathcal{N}_{\text{near}}$, with $\mathcal{N}_{\text{tot}} = \mathcal{N}_{\text{far}} + \mathcal{N}_{\text{inter}} + \mathcal{N}_{\text{near}}$.

The complexity is also affected by the geometric characteristics of Γ. These include:

- The *maximum patch length* $L_{\max} = \max_P L(P)$
- The *relative minimal patch length* $L_{\min} = \beta_0 L_{\max}$, $\beta_0 \leq 1$.
- The *minimal feature size relative to* $L_{\max}$, $\ell_{min} = \alpha_0 L_{\max}$, which is defined in terms of the *local feature size* and the *medial axis* of Γ. The medial axis of Γ, denoted $M(\Gamma)$, is the set of points in $\mathbb{R}^3$ with more than one closest point on Γ. For $\boldsymbol{y} \in \Gamma$, the local feature size $\ell(\boldsymbol{y})$ is the distance from $\boldsymbol{y}$ to $M(\Gamma)$. We assume that the local feature size is bounded below by $\alpha_0 L_{\max}$, i.e., $\ell(\boldsymbol{y}) \geq \alpha_0 L_{\max} = \ell_{min}$ for $\boldsymbol{y} \in \Gamma$.
- The *maximum variation of area distortion* of the parametrization C_J. The variation of the area distortion of a patch P is $C_J(P) = \max_{(u,v)} |J_P(u,v)| / \min_{(u,v)} |J_P(u,v)|$, where $J_P(u,v)$ is the Jacobian of P at the point (u,v). We define $C_J = \max_{P \in \Gamma} C_J(P)$. This value is an indicator of how non-uniform the parametrization of P is and allows us to estimate how the patch length decreases with refinement.

We assume that the α_0, β_0 and C_J are independent of N_{init}. We also assume that principal curvatures are bounded globally on Γ and independent of N_{init}. We now briefly summarize the results of this section:

- *Admissibility.* (Section 2.5.1) The complexity of this step is $O(N_{\text{init}} \log N_{\text{init}})$, with constants dependent on α_0, β_0 and C_J. The logarithmic factor is due to use of an AABB tree for closest surface point queries.

- *Upsampling.* (Section 2.5.2) The complexity of upsampling is $O(mN \log(N))$, where m is the upsampling ratio. The logarithmic factor appears for similar reason to admissibility, with constants that depend on geometric parameters and the boundary condition through the error estimate of Section 2.4. We show that the upsampling ratio is independent of N.

- *Point marking.* (Section 2.5.3) Identifying which zone an evaluation point belongs to (Ω_F, Ω_I or Ω_N) depends on N and the total number of points to be classified $\mathcal{N}_{\text{tot}} = \mathcal{N}_{\text{far}} + \mathcal{N}_{\text{inter}} + \mathcal{N}_{\text{near}}$. The complexity is $O(\mathcal{N}_{\text{tot}} \log N)$ with constants dependent on geometric parameters, due to the cost of closest surface point queries.

- *Far, intermediate and near zone integral evaluation.* (Section 2.5.4) The complexity of these components depends on N and $\mathcal{N}_{\text{far}}$, $\mathcal{N}_{\text{inter}}$ and $\mathcal{N}_{\text{near}}$ respectively, with the general form $O(s_1 N + s_2 N')$, where N' is the number of evaluation points in the corresponding class. For the far field, $s_1 = s_2 = 1$. For the intermediate evaluation, $s_2 = 1$, and $s_1 = mq^2$; finally, for the near zone, $s_2 = p$, and $s_1 = mq^2$, the same as in the intermediate zone. If b is chosen appropriately, the intermediate and near zone error is $\varepsilon_{\text{target}}$.

- *GMRES solve.* Due to the favorable conditioning of the double-layer formulation in Eq. (2.5), GMRES converges rapidly to a solution in a constant number of iterations for a given Γ that is independent of N. This means that the complexity to solve Eq. (2.5) is asymptotically equal (up to a constant dependent on Γ) to the complexity equal to a near-zone evaluation with $\mathcal{N}_{\text{near}} = N(q + 1)^2$.

2.5.1 Admissibility

The patch refinement procedure Section 2.3.2.1 to enforce Criteria 1 and 2 of admissibility and achieve given approximation errors of the geometry ε_{g} and boundary data ε_{f} is a local operation on each patch. If we assume that L_{min}, L_{max}, the partial derivatives of all patches composing $\hat{\Gamma}$, and the partial derivatives of f are bounded, then errors ε_{g} and ε_{f} can always be achieved after a fixed number of refinement steps. As a consequence, this stage must have complexity $O(N_{\text{init}})$.

We focus on the additional refinement needed to satisfy Criterion 3: ensuring that each check center $\hat{\boldsymbol{c}}$ is closest to its corresponding quadrature point $\boldsymbol{y}$. This can be restated in terms of local feature size: for a quadrature patch $P \in \Gamma$ and quadrature node $\boldsymbol{x} \in P$ with check center $\hat{\boldsymbol{c}}$, $\|\boldsymbol{x} - \hat{\boldsymbol{c}}\|_2 \leq \ell(\boldsymbol{x}) \leq \alpha_0 L_0$. We will first relate the number of required refinement steps η to satisfy Criterion 3 to the shape parameters α_0 and C_J, then we will show that this number does not depend on N under our assumptions.

Recall that the distance from a check center to the surface for a patch P is given by $R + r(p+1)/2 = (a + (p+1)b/2)L(P) = KL(P)$. After η refinement steps, the area of each child of P relative to P itself will have decreased by at least by $C_J(P)(1/4)^{\eta}$. Since the distance from $\hat{\boldsymbol{c}}$ to the surface is proportional to $L(P)$, we can estimate the required level of uniform refinement to satisfy Criterion 3 by requiring that the check center distance is less than the minimal local feature size, then taking the maximum value of $L(P)$ over all patches:

$$KL_{\text{max}}\sqrt{C_J}(1/2)^{\eta} \leq \ell_{min} = \alpha_0 L_{\text{max}}$$

This yields

$$\eta = \lceil -\log_2 \frac{\alpha_0}{K\sqrt{C_J}} \rceil, \tag{2.27}$$

which we note depends only on nondimensional quantities α_0, K and C_J characterizing the shape of the surface and its parametrization. If we assume these to be independent of N, then the number of required levels of refinement η are also independent of N. This means that the number of patches N generated Algorithm 2 is a linear function of N_{init}, bounded by $4^{\eta} N_{\text{init}}$.

Next, we estimate the complexity of work per patch in Algorithm 2 to determine if a given patch requires refinement. As described in Section 2.3.2.1, for each patch, we query the AABB tree T_B for patches that are at the distance $R + r(p+1)/2 = KL(P)$ from a check center $\hat{\boldsymbol{c}}$. The cost of the query is logarithmic in the number of patches N_{init} and proportional to the number of patches $N(\hat{\boldsymbol{c}})$ returned. This means that we need to estimate the number of patches that can be within the distance $KL(P)$ from $\hat{\boldsymbol{c}}$.

Consider an area element dA of $\hat{\Gamma}$ at a point $\boldsymbol{x}_0$. The parallel surface of dA, given by $\boldsymbol{x}_0 + h\boldsymbol{n}(\boldsymbol{x}_0)$ does not have self-intersections when $|h| \leq \ell_{min}$ and has a corresponding area element given by $dA^h = (1 + h\kappa_1)(1 + h\kappa_2)dA$ [Kress 1999, Section 6.2], where κ_1 and κ_2 are the principal curvatures of $\hat{\Gamma}$ at $\boldsymbol{x}_0$. The volume of the truncated cone bounded by dA and dA^h of height ℓ_{min} can be computed directly from the integral $\int_0^{\ell_{min}} dA^h dh$:

$$dV = dA\ell_{min}(1 + \frac{1}{2}(\kappa_1 + \kappa_2)\ell_{min} + \frac{1}{3}\kappa_1\kappa_2\ell_{min}^2) = dA\ell_{min}(1 + \frac{1}{2}H\ell_{min} + \frac{1}{3}K\ell_{min}^2)$$

where K and H are Gaussian and mean curvatures respectively. As principal curvatures satisfy $\kappa_i \geq -1/\ell_{min}$, this expression has minimal value for $\kappa_1 = \kappa_2 = -1/\ell_{min}$:

$$dV \geq \frac{1}{3}\ell_{min}dA \tag{2.28}$$

In other words, each surface element dA has (at least) a volume $\frac{1}{3}\ell_{min}dA$ with no other surface elements inside associated with it. From this, we can estimate the total area of surface contained within distance $KL(P)$ from $\hat{\boldsymbol{c}}$ by equating Eq. (2.28) with the volume of a sphere of raidus $KL(P)$, producing $4\pi K^3 L(P)^3/\ell_{min}$. Since the area of each patch is at least L_{min}^2, the number of patches $KL(P)$ from $\hat{\boldsymbol{c}}$ is bounded by

$$N(\hat{\boldsymbol{c}}) \leq 4\pi K^3 \frac{L(P)^3}{\ell_{min} L_{\min}^2} \leq 4\pi K^3 \frac{L_{\max}^3}{\ell_{min} L_{\min}^2} = \frac{4\pi K^3}{\alpha_0 \beta_0^2} \tag{2.29}$$

This is independent of N_{init}, which means that the complexity of nearest patch retrieval is $O(N_{\text{init}} \log N_{\text{init}})$, with constant given by the product of (2.29) and 4^η, with η given by (2.27).

To complete the complexity estimate of the admissibility refinement, we need to estimate the cost of computing the closest point on each patch. The complexity of the Newton's method for finding roots of polynomials in Appendix A.2 depends only on the polynomial degree and the desired accuracy of the optimization, which we can assume to be bounded by floating-point precision [Schleicher and Stoll 2017]. We conclude that the overall complexity of admissibility refinement is $O(N_{\text{init}} \log N_{\text{init}})$ with constants proportional to the patch degree and optimization accuracy.

2.5.2 Upsampling

We estimate the complexity of the upsampling algorithm in Section 2.3.2.2 in terms of N, the number of patches produced by admissibility refinement, and a parameter ε, which is the desired accuracy achieved by the final upsampled patches at the check points. As the distance from the surface to the check points $\boldsymbol{c}_i$ is bounded from below by $aL_{\min}$, the $\tilde{V}$ term in Eq. (2.21) is bounded from above by $CL_{\min}^{-2q-1}$, for a constant C independent of q. Furthermore, since $\hat{\Gamma}$ and f are assumed to be smooth, the density and its derivatives can also be assumed to be bounded. The overall form of the estimate in Eq. (2.21) can then be bounded and written as $\tilde{C}(q)L_{\min}^{-2q-1}\tilde{L}^{2q}$ for some constant $\tilde{C}(q)$. The maximum patch length obtained by refinement $\tilde{L}$ is

$$\tilde{L} = L_{\max}^{\text{fine}} \leq L_{\max} 2^{-\tilde{\eta}}, \tag{2.30}$$

where $\tilde{\eta}$ is the maximum amount of required patch refinement. By setting $C(q)L_{\min}^{-2q-1}\tilde{L}^{2q} \le \varepsilon$ and using Eq. (2.30), we can obtain an upper bound for $\tilde{\eta}$ as a function of $L_{\min}$, $L_{\max}$, and ε:

$$\tilde{\eta} \le -\frac{1}{2q}\log_2\left(\frac{\varepsilon}{L_{\min}^{-2q-1}L_{\max}^{2q}C(q)}\right) = \log_2 \varepsilon^{-1/(2q)} + \bar{C}(q, L_{\min}, L_{\max}), \tag{2.31}$$

for some constant $\bar{C}(q, L_{\min}, L_{\max})$.

The number of points generated by upsampling is $O(4^{\tilde{\eta}}N)$. Taking powers of both sides of Eq. (2.31) yields an estimate in terms of $\varepsilon_{\text{target}}$: $O((2^{\tilde{\eta}})^2 N) \le O(\varepsilon^{-2/(2q)}N) = O(\varepsilon^{-1/q}N)$. As discussed in Section 2.5.1, the closest point computation needed to determine if a checkpoint is in Ω_I has $\log(N)$ cost per point, leading to $O(\varepsilon^{-1/q}N\log(N))$ overall complexity and an upsampling factor of $\varepsilon^{-1/q}$. Since we desire upsampled quadrature with an accuracy of 10^{-12}, we set ε as such to arrive at the desired complexity.

2.5.3 Point marking

In the point marking algorithm of Section 2.3.6, we first use the Laplace FMM to cull points far from Γ, which requires $O(N + \mathcal{N}_{\text{tot}})$ time. Let $\bar{L} = \frac{1}{M}\sum_{P\in\mathcal{P}_{\text{coarse}}} L(P)$ be the average patch length. After FMM culling, the remaining unmarked evaluation points are those whose distances from Γ are approximately $\bar{L}$ or less. For each unmarked point $\boldsymbol{x}$, we query the AABB tree T_T for the nearest triangle in the linear approximation of $\mathcal{P}_{\text{coarse}}$.

Since there are $O(N)$ such triangles in T_T, we can perform this query in $O(\log N)$ time [Samet 2006]. This triangle provides a candidate closest patch that is distance d_0 from $\boldsymbol{x}$. We then use to query T_B for all bounding boxes at distance d_0 from $\boldsymbol{x}$. This query too can be performed in $O(\log N)$ time [Samet 2006] and returns a bounded number of boxes and that each is processed in constant time, as discussed in Section 2.5.1. As the number of unmarked points after culling is bounded above by $\mathcal{N}_{\text{tot}}$, the overall complexity of our marking scheme is $O(\mathcal{N}_{\text{tot}}\log N)$.

2.5.4 Integral evaluation complexity

We assume that geometric admissibility criteria are already satisfied. All integral evaluation is accelerated using an FMM with complexity $O(N + \mathcal{N}_{\mathrm{tot}})$.

Far zone The complexity of far evaluation is just the complexity of computing the integrals on $\mathcal{P}_{\mathrm{coarse}}$ using standard quadrature and FMM acceleration, i.e., $O(q^2 N + \mathcal{N}_{\mathrm{far}})$.

Intermediate zone The complexity of the intermediate zone evaluation is similar to that of the far zone. However the computation is performed on $\mathcal{P}_{\mathrm{fine}}$ rather than $\mathcal{P}_{\mathrm{coarse}}$, which is up to m times finer than $\mathcal{P}_{\mathrm{coarse}}$, with $m = O(\varepsilon^{-1/q})$ and $\varepsilon = 10^{-12}$. The density values must be interpolated from points in $\mathcal{P}_{\mathrm{coarse}}$ to points in $\mathcal{P}_{\mathrm{fine}}$: this can be computed in $O(mq^4 N)$ time using a 2D version of the barycentric interpolation formula [Berrut and Trefethen 2004]. This yields an overall complexity of $O(mq^4 N + mq^2 N + \mathcal{N}_{\mathrm{inter}})$. Although not asymptotically dominant, for all practical target errors, the quadrature evaluation is the dominant cost in practice due to suppressed FMM -related constants, as demonstrated in Section 2.6.2.

Near zone Section 2.3.1 requires a closest point computation, an intermediate-zone evaluation at p check points and an extrapolation for each target point in Ω_N. The intermediate zone calculation is the dominant cost, resulting in a complexity of $O(mq^4 N + mq^2 N + p\mathcal{N}_{\mathrm{near}})$.

GMRES solve As a result of the second-kind integral formulation in Section 2.2, the cost of solving Eq. (2.5) via GMRES is asymptotically equal to the cost of a single singular integral evaluation, since the low number of iterations are independent of N. In our algorithm, this is a special case of near-zone evaluation with $\mathcal{N}_{\mathrm{near}} = q^2 N$, producing a complexity of $O(mq^4 N + mq^2 N + pq^2 N) = O((m + p + mq^2)q^2 N)$.

Overall complexity for uniform point distribution We now suppose that we wish to evaluate the solution u determined by a density ϕ at a set of uniformly distributed points throughout Ω. We also assume that $\hat{\Gamma}$ is discretized uniformly by N patches, i.e., $L_{\max} = O(N^{-1/2})$ and that the distances between samples in Ω and from samples to $\hat{\Gamma}$ are also $O(N^{-1/2})$. Since the total number of evaluation points is proportional to $1/L^3_{\max}$, this implies that $\mathcal{N}_{\text{tot}} = O(N^{3/2})$.

The size of the intermediate zone Ω_I is bounded by the estimate discussed in Section 2.5.2. Letting d_I be the shortest distance along a normal vector of $\hat{\Gamma}$ which is contained in Ω_I, following the discussion in Section 2.5.2 yields the following relation:

$$\tilde{C}(n) d_I^{-2q-1} L_{\max}^{2q} \leq \varepsilon. \tag{2.32}$$

Solving for d_I gives us

$$d_I \leq \left(\frac{\varepsilon}{C(n)}\right)^{-\frac{1}{2q-1}} (L_{\max})^{\frac{2q}{2q-1}} . \tag{2.33}$$

We are interested in the regime as $N \longrightarrow \infty$, or $L_{\max} \longrightarrow 0$. Since $L_{\max}^{\frac{2q}{2q-1}} \leq \sqrt{L_{\max}} = O(N^{-1/4})$, this gives us

$$d_I \leq \left(\frac{\varepsilon}{C(n)}\right)^{-\frac{1}{2q-1}} N^{-1/4} = O(\varepsilon^{-1/2q} N^{-1/4}) = O(\sqrt{m} N^{-1/4}), \tag{2.34}$$

after recalling from above that $m = O(\varepsilon^{-1/q})$ is the average upsampling rate to produce $\mathcal{P}_{\text{fine}}$ from $\mathcal{P}_{\text{coarse}}$. The size of the near zone is, by construction, of the order $L_{\max}$. It follows that $\mathcal{N}_{\text{inter}} = O(\sqrt{m} N^{5/4})$, and $\mathcal{N}_{\text{near}} = O(N)$.

The overall complexity for this evaluation is the sum of the cost of each separate evaluation:

$$\begin{aligned} &O(q^2 N + \mathcal{N}_{\text{far}} + mq^4 N + mq^2 N + \mathcal{N}_{\text{inter}} + mq^4 N + mq^2 N + p\mathcal{N}_{\text{near}}) \\ &= O\left((m + mq^2) q^2 N + \mathcal{N}_{\text{tot}} + (p-1)\mathcal{N}_{\text{near}}\right) \end{aligned}$$

Using the estimates for $\mathcal{N}_{\text{tot}}$ and $\mathcal{N}_{\text{near}}$ and dropping dominated terms, we obtain $O((m+mq^2)q^2N+$

$N^{3/2}$) for the overall complexity. This suggests that for a given q and ε, the minimal cost is obtained from choosing the number of discretization points $N = O(m^2)$, i.e., $N = O(\varepsilon^{-2/q})$.

2.6 Results

We now demonstrate the accuracy and performance of hedgehog to evaluate singular/near-singular layer potentials on various complex geometries to solve the integral equation in Eq. (2.5) and evaluate the solution as defined in Eq. (2.4).

2.6.1 Classical convergence with patch refinement

We will first demonstrate the numerical convergence behavior of hedgehog. As discussed in [Klöckner et al. 2013b, Section 3.1], approximation-based schemes such as hedgehog do not converge classically but do so up to a controlled precision if r and R scale with proportional to the patch size. In order to observe classical convergence as we refine $\mathcal{P}_{\text{coarse}}$, we must allow R and r to decrease slower than $O(L)$, such as with rate $O(\sqrt{L})$. In this section, we choose the hedgehog parameters a and b proportional to $1/\sqrt{L}$ to achieve this and demonstrate numerical convergence with refinement of L.

In our examples, we use analytic solutions to Eq. (2.2) obtained as sums of point charge functions of the form

$$u_c(\boldsymbol{x}) = \sum_{i=1}^{m} G(\boldsymbol{x}, \boldsymbol{y}_i)\psi_i \tag{2.35}$$

where the charge locations $\boldsymbol{y}_i$ with strengths ψ_i are outside of Ω. To construct specific solutions, we sample a sphere of radius one with point charges, as shown in Figures 2.8 and 2.9. We choose charge strengths ψ_i randomly from $[0, 1]^d$, where $d = 1$ for Laplace problems and $d = 3$ for Stokes and elasticity problems.

We use the multipole order $m = 20$ with 5000 points per leaf box for the kernel-independent

FMM . This ensures that the FMM error does not dominate; sufficiently large number of points per leaf box is needed to minimize the additional error due to tree depth. We choose a high quadrature order $q = 20$, or 400 quadrature points per patch in $\mathcal{P}_{\text{coarse}}$, relative to overall convergence order to satisfy the assumption in Section 2.3.2.2. We also use two levels of uniform upsampling to demonstrate convergence.

2.6.1.1 Green's Identity

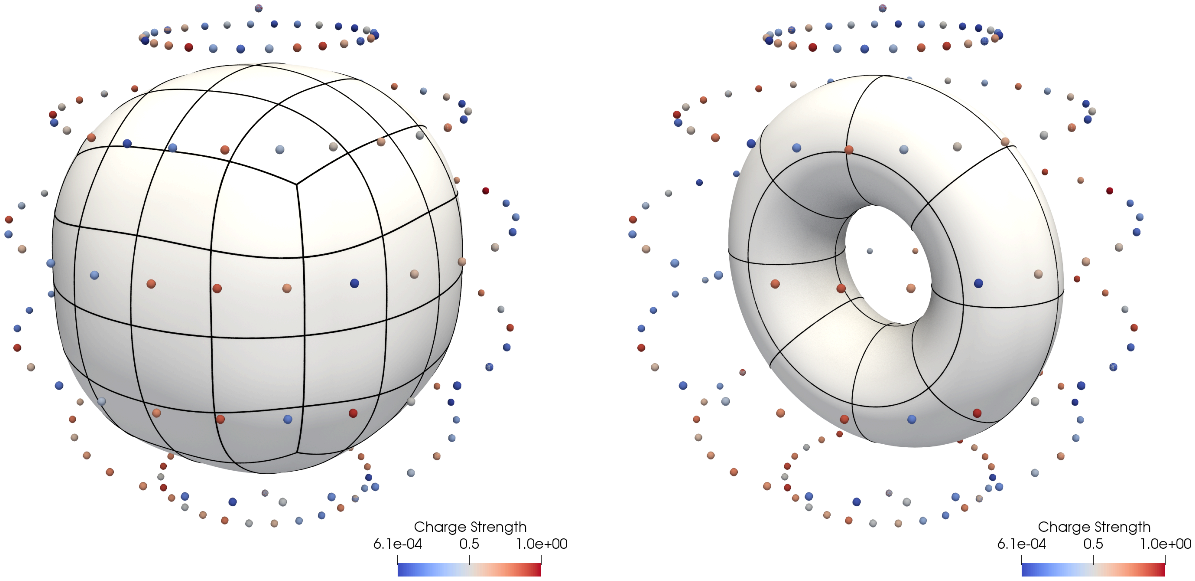


Figure 2.8: Geometry and singularities used for Green's Identity convergence tests.. Shown are polynomial patches defining boundary geometry (black lines) and point singularities placed on the surface on a sphere of radius one. Singularity strengths are randomly selected values in $[0, 1]$; shown is the strength intensity for Laplace problems, which varies from blue to red. We use 96 20th-order polynomial patches for the spheroid (left) and 32 cubic patches for the torus (right).

We report the accuracy of the hedgehog evaluation scheme in Table 2.1, where we verify Green's Identity for a random known function u_c in Eq. (2.35). We evaluate the Dirichlet and Neumann boundary data due to u_c at the discretization points of $\hat{\Gamma}$ and use one-sided hedgehog to evaluate the corresponding single- and double-layer potentials at the same discretization points. With each column of Table 2.1, we subdivide $\mathcal{P}_{\text{coarse}}$ to more accurately resolve the boundary

Geometry	PDE	Relative ℓ^∞ error (Number of patches)				EOC
Spheroid (Fig. 2.8-left)	Laplace	1.06×10^{-4} (96)	4.78×10^{-6} (384)	9.14×10^{-8} (1536)	4.35×10^{-9} (6144)	4.77
	Elasticity	1.68×10^{-3} (96)	6.94×10^{-5} (384)	1.53×10^{-6} (1536)	1.33×10^{-8} (6144)	5.74
	Stokes	1.92×10^{-3} (96)	7.95×10^{-5} (384)	1.74×10^{-6} (1536)	1.53×10^{-8} (6144)	5.72
Torus (Fig. 2.8-right)	Laplace	2.05×10^{-3} (32)	7.52×10^{-5} (128)	3.79×10^{-6} (512)	8.48×10^{-8} (2048)	5.45
	Elasticity	4.38×10^{-2} (32)	1.17×10^{-3} (128)	5.08×10^{-5} (512)	1.42×10^{-6} (2048)	5.09
	Stokes	5.03×10^{-2} (32)	1.33×10^{-3} (128)	5.81×10^{-5} (512)	1.65×10^{-6} (2048)	5.09

Table 2.1: ℓ^∞ Relative error in Green's Identity versus number of patches. The solution to Eq. (2.2) due to a known function u_c, shown in Fig. 2.8 is computed via Green's Identity. We evaluate the single- and double-layer potentials with hedgehog due to the Dirichlet and Neumann boundary data and compare against the known value of u_c on the boundary. Each column is the result of an additional level of uniform quadrisection of the patches in $\mathcal{P}_{\text{coarse}}$. The final column (EOC) is the estimated convergence order, computed via least-squares log-log fit of the error as a function of max patch size.

condition. The error shown in Table 2.1 is the ℓ^∞-relative error in the solution value

$$\frac{\left\| \hat{S}\left[\frac{\partial u_c}{\partial \boldsymbol{n}}\right](\boldsymbol{x}) - \hat{D}\left[u_c\right](\boldsymbol{x}) - u_c(\boldsymbol{x}) \right\|_\infty}{\|u_c\|_\infty}, \tag{2.36}$$

where $\hat{S}$ and $\hat{D}$ are the single- and double-layer singular integral operators discretized and evaluated with hedgehog . In these tests, we choose $p = 6$, $r = .004\sqrt{L}$ ($a = .004/\sqrt{L}$) and $R = .03\sqrt{L}$ ($b = .03/\sqrt{L}$). We observe roughly 5th order convergence on both the spheroid and torus test geometries in Fig. 2.8 for each of the tested PDE's. In Table 2.2, we present the number of target points evaluated per second per core with one-sided hedgehog . We see that performance is best for Laplace and worst for elasticity problems, as expected.

Geometry	PDE	Target points/second/core			
Spheroid (Fig. 2.8-left)	Laplace	3684	5438	5077	5629
	Elasticity	1325	1731	1687	1790
	Stokes	1635	2075	2016	2120
Torus (Fig. 2.8-right)	Laplace	2729	3373	4564	5477
	Elasticity	984	1171	1347	1502
	Stokes	1134	1331	1609	1727

Table 2.2: Performance of singular evaluation in Green's Identity. For each test in Table 2.1, we report the number of target points evaluated with one-sided hedgehog per second per core.

2.6.1.2 Solution via GMRES

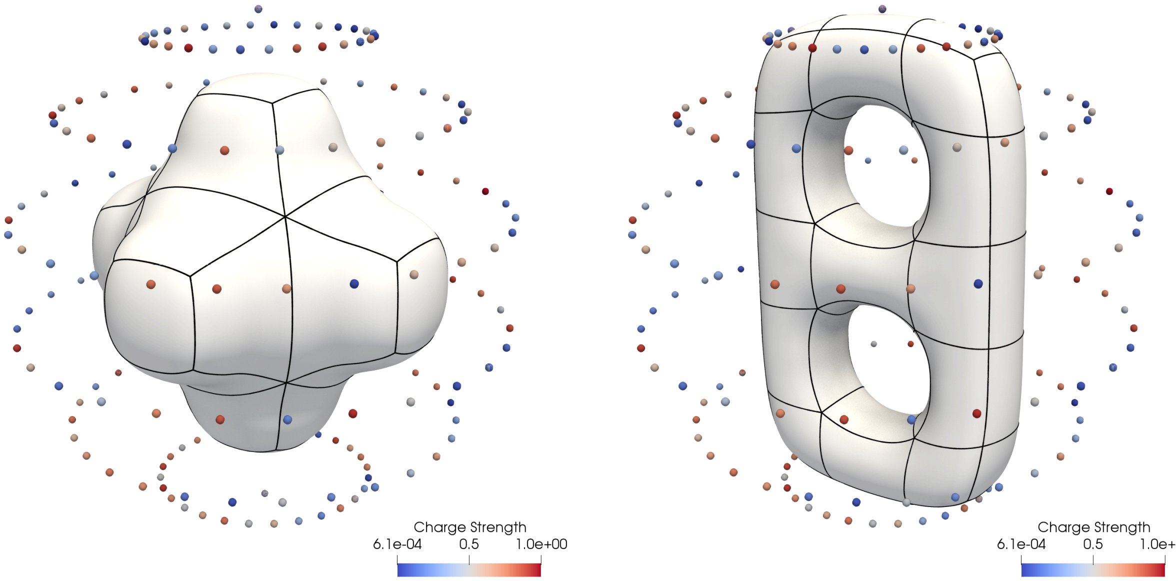


Figure 2.9: Geometry and singularities used for solver convergence tests.. Figures are similar to Fig. 2.8, but displaying geometries for testing the convergence of hedgehog within a GMRES solver. We use 30 16th-order polynomial patches for the pipe (left) and 50 20th-order patches for the genus two surface (right). Note the proximity of the singularities to the domain of the genus two surface; the nearest singularity is less than $.05L$ from $\hat{\Gamma}$.

We report the accuracy of the hedgehog scheme when used to solve Eq. (2.2) via the integral equation in Eq. (2.5). Two-sided hedgehog is used in the matrix-vector multiply inside GMRES to solve Eq. (2.5) for the values of the density ϕ at the discretization points. Then one-sided hedgehog is used to evaluate Eq. (2.8) at a slightly coarser discretization. Since GMRES minimizes the residual at the original discretization of Eq. (2.5), this final step prevents an artificially accurate solution by changing discretizations. Table 2.3 lists the ℓ^∞ relative error values for the total solve and evaluation steps using Section 2.3.1 as we refine $\mathcal{P}_{\text{coarse}}$ by subdivision as in the previous section. In these tests, we choose $p = 6$, $r = .005\sqrt{L}$ ($a = .005/\sqrt{L}$), and $R = .03\sqrt{L}$ ($b = .03/\sqrt{L}$). As for previous examples, we observe at least 5th order convergence on all tested geometries in Fig. 2.9 and Fig. 2.8-left and all PDE's. We include the spheroid example as an ad-

Geometry	PDE	Relative ℓ^∞ error (Number of patches)				EOC
Spheroid (Fig. 2.8-left)	Laplace	2.70×10^{-6} (96)	1.92×10^{-7} (384)	4.47×10^{-9} (1536)	5.13×10^{-11} (6144)	5.35
Pipe	Laplace	5.99×10^{-4} (30)	3.03×10^{-5} (120)	6.68×10^{-7} (480)	2.27×10^{-8} (1920)	5.92
(Fig. 2.9-left)	Elasticity	7.17×10^{-2} (30)	3.57×10^{-3} (120)	8.90×10^{-5} (480)	4.14×10^{-6} (1920)	5.45
	Stokes	8.53×10^{-2} (30)	4.12×10^{-3} (120)	1.03×10^{-4} (480)	4.73×10^{-6} (1920)	5.43
Genus 2	Laplace	4.00×10^{-2} (50)	1.25×10^{-4} (200)	1.54×10^{-6} (800)	5.73×10^{-10} (3200)	8.76
(Fig. 2.9-right)	Elasticity	9.20×10^{-2} (50)	1.05×10^{-3} (200)	1.00×10^{-5} (800)	9.44×10^{-8} (3200)	6.89
	Stokes	1.03×10^{-1} (50)	1.18×10^{-3} (200)	1.15×10^{-5} (800)	1.03×10^{-7} (3200)	6.88

Table 2.3: ℓ^∞ Relative error in GMRES solve and solution evaluation versus number of patches. We solve Eq. (2.2) by discretizing and evaluating the layer potential in the integral equation in Eq. (2.5) as described in Section 2.3.1. We use two-sided hedgehog inside of GMRES to solve for ϕ, then evaluate Eq. (2.8) with one-sided hedgehog at a new set of points on $\hat{\Gamma}$. Each column is the result of an additional level of uniform quadrisection of the patches in $\mathcal{P}_{\text{coarse}}$. The final column (EOC) is the estimated convergence order, computed via least-squares log-log fit of the error as a function of max patch size.

Geometry	PDE	Target points/second/core			
Spheroid	Laplace	2737	3149	2846	2950
Pipe	Laplace	3046	2178	2832	2982
(Fig. 2.8-left)	Elasticity	991	993	1189	1261
	Stokes	1048	1140	1335	1422
Genus 2	Laplace	1862	2886	3122	2879
(Fig. 2.8-right)	Elasticity	729	1125	1255	1295
	Stokes	929	1304	1450	1504

Table 2.4: Performance of singular evaluation in GMRES matrix-vector multiply. For each test in Table 2.3, we report the number of target points per second per core evaluated with two-sided hedgehog in a single GMRES matrix-vector multiplication.

ditional demonstration of a high accuracy solution via GMRES with our approach. We report the number of target points evaluated per second per core with two-sided hedgehog in Table 2.4. The results are similar to Table 2.2; the slower performance is because evaluation via two-sided hedgehog is more expensive than one-sided hedgehog .

2.6.2 Comparison with [Ying et al. 2006b]

In this section, we compare our method to [Ying et al. 2006b], a previously proposed high-order, kernel-independent singular quadrature method in 3D for complex geometries. These characteristics are similar to hedgehog shares these characteristics. Appendix A.3 presents additional comparisons.

The metric we are interested is *cost for a given relative error.* Assuming the surface discretization is $O(N)$, we measure the cost of a method as its total wall time during execution T divided by the total wall time of an FMM evaluation on the same $O(N)$ discretization, T_{FMM}. By normalizing by the FMM evaluation cost, we minimize the dependence of the cost on machine- and implementation-dependent machine-dependent parameters.

We run the tests in this section on the spheroid geometry shown in Fig. 2.8-left. We focus on the singular quadrature scheme of [Ying et al. 2006b]. The near-singular quadrature of [Ying et al. 2006b] is algorithmically similar to hedgehog , but since an expensive singular quadrature rule is used as a part of near-singular evaluation, it has a higher total cost. As a result, the accuracy and cost of near-singular evaluation of [Ying et al. 2006b] is bounded by the accuracy and cost of the singular integration scheme.

To compare the full hedgehog method with [Ying et al. 2006b], we fit polynomial patches to the C^∞ surface of [Ying and Zorin 2004], denoted Γ_b, to produce $\hat{\Gamma}$ during the first step of Section 2.3.4. We apply the remaining geometry preprocessing algorithms of Section 2.3.4 to $\hat{\Gamma}$ to produce $\mathcal{P}_{\text{coarse}}$. After producing $\mathcal{P}_{\text{fine}}$ with two levels of uniform upsampling, we solve Eq. (2.5) with two-sided hedgehog on $\hat{\Gamma}$ and evaluate the solution on the boundary with one-sided hedgehog . We then solve for the solution to Eq. (2.5) on Γ_b using [Ying et al. 2006b].

For each of the tests in this section, we choose some initial spacing parameter h_0 to discretize the surface of [Ying and Zorin 2004], as in [Ying et al. 2006b], and use the 16× upsampled grid and floating partition of unity radius proportional to $O(\sqrt{h})$, as in the original work. We apply hedgehog to $\hat{\Gamma}$ and the scheme of [Ying et al. 2006b] to Γ_b with spacing $h_0/2^i$, for $i = 1, \dots 4$.

As in the previous section, we choose the parameters r and R of hedgehog to be $O(\sqrt{L})$. For both quadrature methods, we use a multipole order of 16 for PVFMM with at most 250 points in each leaf box. The results are shown in Fig. 2.10. From left to right, each plot details the total

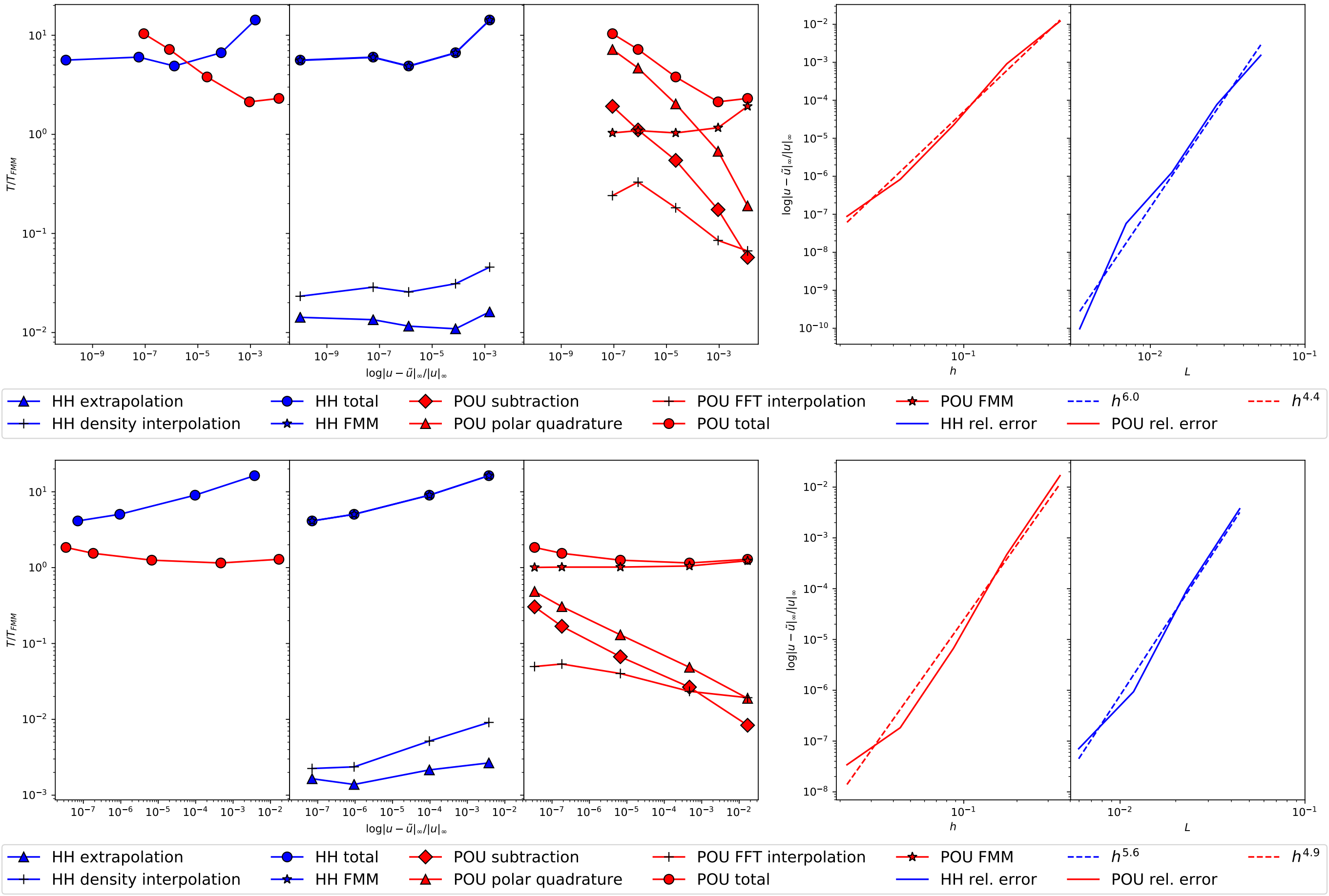


Figure 2.10: Comparison of hedgehog on polynomial patches (HH) versus [Ying et al. 2006b] on the surface representation of [Ying and Zorin 2004] (POU) solving via GMRES for u_c. Laplace (top) and elasticity (bottom) problems solved on the spheroid shown in Fig. 2.8. From left to right, we plot the total cost of each scheme, the cost of each subroutine for hedgehog (blue) and the singular quadrature scheme of [Ying et al. 2006b] (red), and the relative error as a function of h. We plot error convergence of [Ying et al. 2006b] as a function of h and hedgehog as a function of L, due to the distinct discretizations. For hedgehog parameters, we choose $r = .013\sqrt{L}$, $R = .075\sqrt{L}$ for the Laplace problem; for the elasticity problem, we choose $r = .013\sqrt{L}$, $R = .08\sqrt{L}$. We choose $p = 6$ and $q = 15$ for both problems. For [Ying et al. 2006b] the spacing is $h_0 = .35$. Note that in the hedgehog timing breakdown, since the FMM time is dominant, the FMM cost lies directly on top of the total cost.

cost of each scheme, the cost of each subroutine for hedgehog (denoted HH) and the singular quadrature scheme of [Ying et al. 2006b] (denoted POU), and the relative error as a function of h and L, respectively, for all refinement levels. We plot the cost of both schemes the cost of each algorithmic step as a function of their computed relative error. In each figure, we present results for a Laplace problem (top) and an elasticity problem (bottom).

In Fig. 2.10, as expected, we observe a higher convergence rate for hedgehog compared to [Ying et al. 2006b]. [Ying et al. 2006b] outperforms hedgehog in terms of cost for all tested discretizations. We observe that the FMM evaluation in Fig. 2.10 accounts for at least 95% of the hedgehog cost. This means that a local singular quadrature method (based on corrections to an FMM evaluation, Section 2.1.2) of *worse* complexity can beat a global method, simply by virtue of reducing the FMM size. By noting the large difference between the hedgehog FMM cost and the hedgehog density interpolation, we can reasonably infer that a local hedgehog scheme should narrow this performance gap and outperform [Ying et al. 2006b] for larger problems, assuming that switching to a local scheme does not dramatically affect error convergence.

Remark: A recent novel solver using a local quadrature approach, called FMM3DBIE , [Greengard et al. 2021], achieves this performance improvement using an adaptive refinement approach. This solver is highly optimized and is $O(N)$ complexity, achieving an order of magnitude speed up over hedgehog . This further supports our conclusion that a local quadrature approach should rapidly outpace a global one. It remains to be seen if a local version hedgehog will outperform FMM3DBIE and we leave this to future work.

2.6.3 Requested target precision vs. computed accuracy

In this section, we study the performance of the full algorithm outlined in Section 2.3. We test hedgehog on the torus domain shown in Fig. 2.8-right. We choose a reference solution of the form of Eq. (2.35) with a single point charge located at the origin, in the middle of the hole of the torus. We solve the integral equation with two-sided hedgehog and evaluate the singular integral on a

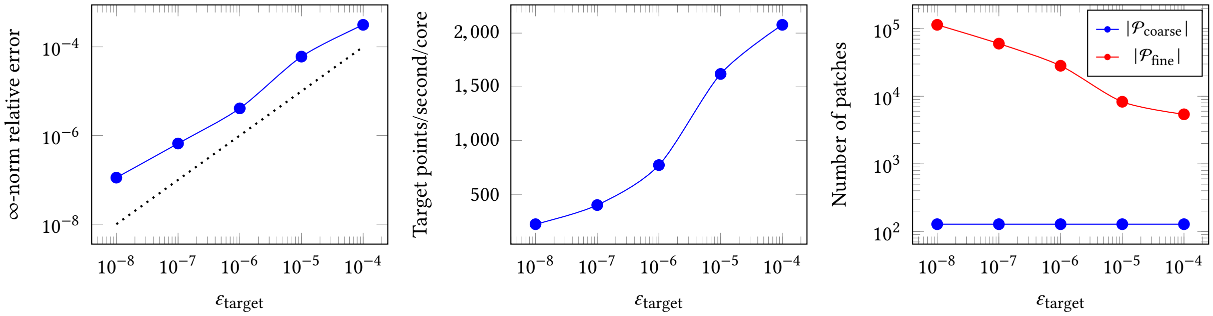


Figure 2.11: Performance of full algorithm. Left: ∞-norm relative error in singular integral vs requested target accuracy (blue). The dotted line is the ideal behavior $y = x$. Middle: Performance in terms of target points evaluated per second per core with hedgehog . Right: Number of patches in $\mathcal{P}_{\text{coarse}}$ and $\mathcal{P}_{\text{fine}}$ computed by the preprocessing algorithms.

distinct discretization with one-sided hedgehog . We choose $q = 20$, $p = 6$ and $a = b/6$. We select various values for $\varepsilon_{\text{target}}$ using the plot in Fig. 2.7(a) to choose b to ensure sufficiently accurate extrapolation. We plot the results of our tests in Fig. 2.11.

We see in Fig. 2.11-left that we are consistently close to the requested target precision. We see a decline in target points per second per core as accuracy increases in Fig. 2.11-middle. This is explained by Fig. 2.11-right, which shows an increase in the size $\mathcal{P}_{\text{fine}}$ as $\mathcal{P}_{\text{coarse}}$ remains a fixed size. The initial 128 patches in $\mathcal{P}_{\text{coarse}}$ are enough to resolve the boundary condition and Γ, but we need greater quadrature accuracy for lower values of $\varepsilon_{\text{target}}$. Decreasing the number of points in passed to the FMM , i.e., decreasing the size of $\mathcal{P}_{\text{fine}}$, is the main way to improve performance of our method. This is further indication that a local version of hedgehog will outperform a global approach.

2.6.4 Full algorithm on interlocking torii

We now demonstrate the full algorithm pipeline on an exterior Laplace problem, whose boundary is defined by four interlocking torii shown in Fig. 2.12. The domain boundary is contained in the box $[-3.8, 2.4]\times[-1.1, 1.1]\times[-1, 1]$. The shortest distance between two adjacent torii is less than 10% of a polynomial patch length defining the boundary. We again use a boundary condition of the form Eq. (2.35) with a single point charge located at (0, .03, .875), inside the upper half of the second torus from the right in Fig. 2.12. This problem is challenging due to the nearly touching geometry of the torii, along with the singularity placed close to the boundary. We run the admissibility and adaptive upsampling algorithms outlined in Section 2.3, solve Eq. (2.5) using two-sided hedgehog , and evaluate the solution on the boundary using one-sided hedgehog . The absolute error in the ∞-norm of the singular evaluation is plotted on the boundary surface.

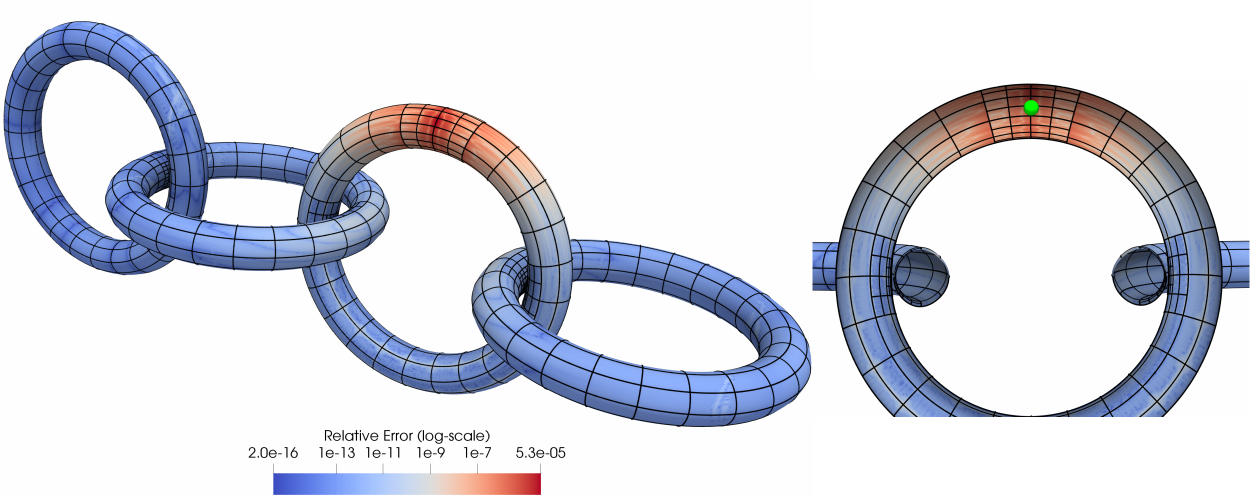


Figure 2.12: Absolute error of GMRES solve via hedgehog on interlocking torii. Left: The admissible set of 1128 patches in $\mathcal{P}_{\text{coarse}}$ used to solve Eq. (2.5) is shown (black lines denote patch boundaries). The point charge generated the boundary condition is located within the second torus from the right. Right: a cross-section of the torii geometry through the xz-plane, showing the second torus from the right and the location of the singularity (green point).

Using $a = .1$, $b = .025$, $p = 6$ and $q = 20$, we achieve a maximum pointwise error of 1.29×10^{-5}. GMRES was able to reduce the residual by a factor of 10^{-13} over 109 iterations. There are 288768 quadrature points in the coarse discretization, 18235392 quadrature points in the fine discretization, and 3465216 check points used in the two-sided hedgehog evaluation inside GMRES. We evaluate the solved density at 451200 points on the boundary with one-sided hedgehog to produce the render in Fig. 2.12. On a machine with two Intel Xeon E-2690v2 3.0GHz CPU's, each with 10 cores, and 100 GB of RAM, the GMRES solve and interior evaluation required 5.7 hours and can evaluate the singular integral at a rate of 1709 target points per second per core.

2.6.5 Solution on complex geometry

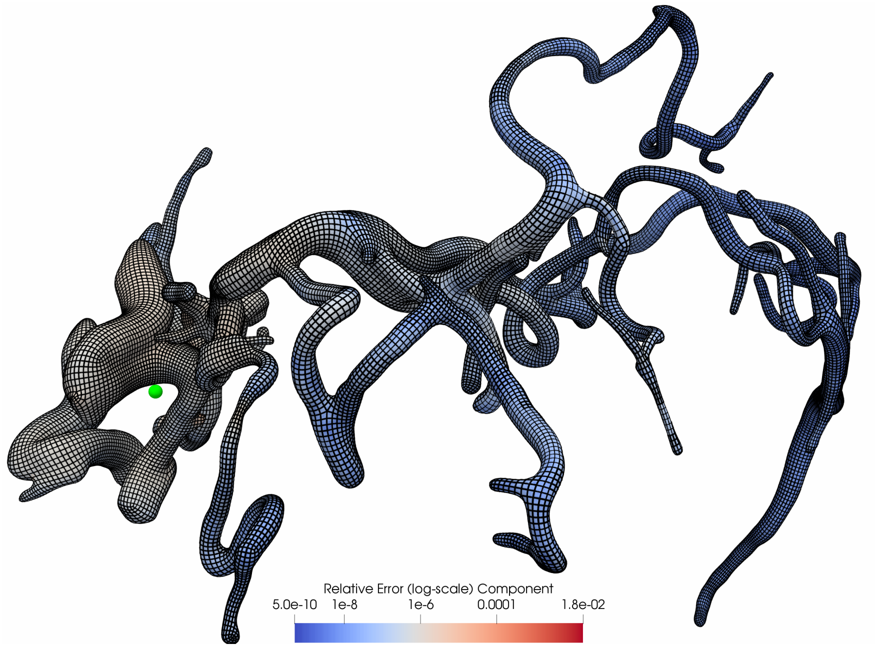


Figure 2.13: Absolute error of GMRES solve via hedgehog on complex blood vessel geometry used in [Lu et al. 2019]. The blood vessel uses 40,960 8th order polynomial patches (black edges denote patch boundaries). The geometry is admissible by construction. The point charge is located on left side of the figure (green)

We have demonstrated in [Lu et al. 2019] a parallel implementation of Section 2.3.1, applied to simulating red blood cell flows. The surface geometry of the blood vessel shown in Fig. 2.13 is complex, with rapidly varying curvatures and geometric distortions due to singular vertices in the surface mesh. Since the surface is admissible, we are able to apply parallel hedgehog directly without geometric preprocessing to solve an interior Dirichlet Stokes problem. We use $a = .125$, $b = .125$, $p = 6$ and $q = 16$ as simulation parameters.

Using 32 machines each with twenty 2.6 Ghz cores with 100GB of RAM, we achieve a maximum pointwise error of 3×10^{-6} when solving a Stokes problem with constant density. We then place a random vector point charge two patch lengths away (relative to the patches in $\mathcal{P}_{\text{coarse}}$) from the domain boundary (on the left side of Fig. 2.13, solve Eq. (2.5) using two-sided hedgehog , and evaluate the solution on the boundary using one-sided hedgehog . The absolute error in the ∞-norm of the singular evaluation is plotted on the boundary surface. There are 10,485,760 quadrature points in the coarse discretization, 167,772,160 quadrature points in the fine discretization, and 125,829,120 check points used in the two-sided hedgehog evaluation inside GMRES. We evaluate the solved density at 209,715,200 points on the boundary with one-sided hedgehog to produce the render in Fig. 2.12. We achieve a maximum pointwise error of 1.8×10^{-2} and can evaluate the singular integral at rate of 3529 target points per second per core.

2.7 Conclusion

We have presented hedgehog , a fast, high-order, kernel-independent, singular/near-singular quadrature scheme for elliptic boundary value problems in 3D on complex geometries defined by piecewise tensor-product polynomial surfaces. The primary advantage of our approach is *algorithmic simplicity*: the algorithm can implemented easily with an existing smooth quadrature rule, a point FMM and 1D and 2D interpolation schemes. We presented fast geometry processing algorithms to guarantee accurate singular/near-singular integration, adaptively upsample the discretization and query local surface patches. We then evaluated hedgehog in various test cases, for Laplace, Stokes, and elasticity problems on various patch-based geometries and compared our approach with [Ying et al. 2006b].

[Lu et al. 2019] demonstrates a parallel implementation of hedgehog , but the geometric preprocessing and adaptive upsampling algorithms presented in Section 2.3 are not parallelized. This is a requirement to solve truly large-scale problems that exist in engineering applications. Our method can also be easily restructured as a local method. The comparison in Section 2.6.2 highlights an important point: a local singular quadrature method can outperform a global method for moderate accuracies, *even when the local scheme is asymptotically slower.* This simple change can also dramatically improve both the serial performance and the parallel scalability of hedgehog shown in [Lu et al. 2019], due to the decreased communication of a smaller parallel FMM evaluation. The most important improvement to be made, however, is the equispaced extrapolation. Constructing a superior extrapolation procedure, optimized for the boundary integral context, is the main focus of our current investigations.

3 | Scalable Simulation of Realistic Red Blood Cell Flows

In this chapter, we construct a scalable, high-fidelity simulation platform for red blood cell flows through capillaries. It is based on the joint work [Lu et al. 2019] with Libin Lu, Abtin Rahimian, Georg Stadler and Denis Zorin.

While there have been many large scale or numerically accurate blood flow simulations in the past, there has been little work attempting to handle both numerical regimes simultaneously. After parallelizing the boundary solver from Chapter 2, we approximate red blood cells as *vesicles*: deformable inextensible membranes without in-plane shear forces.

The remaining challenging task of the simulation is robust, cost-efficient time stepping. Large timesteps are known to cause interpenetration, or *collisions*, between discretized bodies. Handling these collisions without introducing artifical errors is critical in long-time flow simulations. We use the parallel collision handling framework of [Lu et al. 2018] and extend it to arbitrary static rigid bodies to allow for large collision-free time steps among the cells and betweeen cells and the blood vessel wall. We study the strong and weak scaling behavior of our platform up to 34,816 cores with good parallel efficiency.

Portions of this chapter also appear in [Lu 2019]. RBC collision handling and vesicle modeling are contributed largely from [Lu 2019], the boundary integral solver is contributed from Chapter 2, and overall parallelization was a joint contribution.

3.1 Introduction

Achieving a fast, numerically accurate, and robust blood flow simulation requires that the system meets a number of stringent requirements. While previous work has made significant progress [Malhotra et al. 2017; Lu et al. 2018; Rahimian et al. 2010], we focus on several new infrastructure components essential for handling confined flows and arbitrarily long-time, high volume fractions RBC flows; in particular, our work is able to realize each of these goals.

We formulate the viscous flow in blood vessels as an integro-differential equation and make use of fast scalable summation algorithms for efficient implementation, as in prior RBC simulations [Veerapaneni et al. 2011]. This is the only approach to date that maintains high accuracy at the microscopic level while avoiding expensive discretization of fluid volume: all degrees of freedom reside on the surfaces of RBCs and blood vessels.

The most important novel aspects of our system include: (a) handling the RBC-blood vessel interaction with a fully parallel, high-order boundary integral equation solver; (b) explicit handling of collisions with a parallel constraint-based resolution and detection algorithm. The former is essential for modeling confined flows, while the latter is essential for handling high-volume fraction flows at long time scales without excessively small time steps or fine spatial discretizations.

Our contributions

1. We present a parallel platform for long-time simulations of RBCs through complex blood vessels. The extension to suspensions of various particulates (fibers, rigid bodies etc.) is straightforward from the boundary integral formulation. Flows through several complicated geometries are demonstrated.

2. We have parallelized a boundary solver for elliptic PDEs on smooth complex geometries in 3D . By leveraging the parallel fast-multipole method of [Malhotra and Biros 2015b] and the

parallel forest of quadtrees of [Burstedde et al. 2011], we are able to achieve good parallel performance and load balancing.

3. We have extended the parallel collision handling of [Lu et al. 2018] to include rigid 3D boundaries composed of patches.

4. We present weak and strong scalability results of our simulation on the Skylake cluster and weak scaling results on the Knights Landing cluster on Stampede2 at the Texas Advanced Computing Center along with several visualizations of long-time, large-scale blood cell flows through vessels. We observe 49% strong scaling efficiency for a 32-fold increase of compute cores. In our largest test on 12288 cores, we simulate 1,048,576 RBCs in a blood vessel composed of 2,097,152 patches with weak scaling efficiency of 71% compared to 192 cores (Fig. 3.3). In each time step, this test uses over three billion degrees of freedom and over four billion surface elements (triangles) for collision.

5. We are able to simulate realistic human blood flows with RBC volume fractions over 47% (Fig. 3.5).

Limitations. Despite the advantages and contributions of the computational framework presented here, our work has some limitations. We have made several simplifications in our model for RBCs. We are restricted to the low Reynolds number regime, i.e., small arteries and capillaries. We use a simplified model for RBCs, assuming the cell membranes to be inextensible and with no in-plane shear rigidity. It has been shown that flows in arterioles and capillaries with diameter of <50 μm and RBCs with 5 μm diameter have a Reynolds number of $< 5 \times 10^{-3}$ [Wang et al. 2013][Caro et al. 2012, Section 5.4] with roughly 2% error in approximating confined flows [Al Quddus et al. 2008]. This is sufficient for our interest in the qualitative behavior of particulate flows, with the possibility of investigating rheological dynamics in larger channels.

Regarding algorithms, each RBC is discretized with an equal number of points, despite the varied behavior of the velocity through the vessel. Adaptive refinement is required in order to resolve the velocity accurately. Finally, the blood vessel is constructed to satisfy certain geometric constraints that allow for the solution of Eq. (3.5) via singular integration. This can be overcome through uniform refinement, but a parallel adaptive algorithm is required to maintain good performance.

Related work: blood flow. Large-scale simulation of RBC flows typically fall into four categories: (a) *Immersed boundary (IB)* and *immersed interface methods*; (b) particle-based methods such as *dissipative particle dynamics (DPD)* and *smoothed particle hydrodynamics (SPH)* (c) multiscale network-based approaches and (d) *boundary integral equation (BIE)* approaches. For a comprehensive review of general blood flow simulation methods, see [Freund 2014]. IB methods can produce high-quality simulations of heterogeneous particulate flows in complex blood vessels [Balogh and Bagchi 2017b,a; Xu et al. 2013]. These methods typically require a finite element solve for each RBC to compute membrane tensions and use IB to couple the stresses with the fluid. This approach quickly becomes costly, especially for high-order elements, and although reasonably large simulation have been achieved [Saadat et al. 2018, 2019], large-scale parallelization has remained a challenge. A different approach to simulating blood flow is with multiscale reduced-order models. By making simplifying assumptions about the fluid behavior throughout the domain and transforming the complex fluid system into a simpler flow problem, the macroscopic behaviors of enormous capillary systems can be characterized [Peyrounette et al. 2018; Perdikaris et al. 2016] and scaled up to thousands of cores [Perdikaris et al. 2015]. This comes at a cost of local accuracy; by simulating the flows directly, we are able to accurately resolve local RBC dynamics that are not captured by such schemes.

Particle-based methods have had the greatest degree of success at large-scale blood flow simulations [Gounley et al. 2017; Grinberg et al. 2011; Randles et al. 2015; Rossinelli et al. 2015]. These types of approaches are extremely flexible in modeling the fluid and immersed particles, but are computationally demanding and usually suffer from numerical stiffness that requires very small time steps for a given target accuracy. For a comprehensive review, see [Ye et al. 2016]. There have also been recent advances in coupling a particle-based DPD-like scheme with IB in parallel [Ye et al. 2017, 2018], but the number of RBCs simulated and the complexity of the boundary seems to be limited.

BIE methods have successfully realized large-scale simulations of millions of RBCs [Rahimian et al. 2010] in free space. Recently, new methods for robust handling of collisions between RBCs in high-volume fraction simulations have been introduced [Lu et al. 2018; Malhotra et al. 2017]. This approach is versatile and efficient due to only requiring discretization of RBCs and blood vessel surfaces, while achieving high-order convergence and optimal complexity implementation due to fast summation methods [Veerapaneni et al. 2009a, 2011; Rahimian et al. 2015; Sorgentone and Tornberg 2018; Sorgentone et al. 2019; af Klinteberg and Tornberg 2016; Zhao et al. 2010]. To solve elliptic partial differential equations, BIE approaches have been successful in several application domains [Ying et al. 2006b; Wala and Klöckner 2018a,c; Bruno and Lintner 2013]. However, to our knowledge, there has been no work combining a Stokes boundary solver on arbitrary complex geometries in 3D with a collision detection and resolution scheme to simulate RBC flows at large scale. This work aims to fill this gap, illustrating that this can be achieved in a scalable manner.

Related work: collisions. Parallel collision detection methods are a well-studied area in computer graphics for both shared memory and GPU parallelism [Liu et al. 2010; Mazhar et al. 2011; Kim et al. 2009]. [Iglberger and Rüde 2009; Du et al. 2017] detect collisions between rigid bodies in a distributed memory architecture via domain decomposition. [Pabst et al. 2010] constructs a spatial hash to cull collision candidates and explicitly check candidates that hash to the same value. The parallel geometry and physics-based collision resolution scheme detailed in [Yan et al. 2019] is most similar to the scheme used in this work. However, such discrete collision detection schemes require small time steps to guarantee detections which can become costly for high-volume fraction simulations.

3.2 Formulation and solver overview

3.2.1 Problem summary

We simulate the flow of N cells with deformable boundary surfaces γ_i, $i = 1, \dots, N$ in a viscous Newtonian fluid in a domain $\Omega \subset \mathbb{R}^3$ with a fixed boundary Γ. The governing partial differential equations (PDEs) describing the conservation of momentum and mass are the incompressible Stokes equations for the velocity $\boldsymbol{u}$ and pressure p, combined with velocity boundary conditions on Γ. Additionally, we model cell membranes as massless, so the velocity $\mathbf{X}_t$ of the points on the cell surface coincides with the flow velocity:

$$-\mu\Delta\boldsymbol{u}(\boldsymbol{x}) + \nabla p(\boldsymbol{x}) = \mathbf{F}(\boldsymbol{x}) \quad \text{and} \quad \nabla \cdot \boldsymbol{u}(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \Omega, \tag{3.1}$$

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma, \tag{3.2}$$

$$\mathbf{X}_t = \boldsymbol{u}(\mathbf{X}), \quad X \in \gamma_i(t), \tag{3.3}$$

where μ is the viscosity of the ambient fluid; in our simulations, we use a simplified model with the viscosity of the fluid inside the cells also being μ although our code supports arbitrary viscosity contrast. The right-hand side force in the momentum equation is due to the sum of tension and bending forces $\mathbf{f} = \mathbf{f}_\sigma + \mathbf{f}_b$; it is concentrated on the cell surfaces. We assume that cell surfaces are inextensible, with bending forces determined by the Canham-Helfrich model [Canham 1970; Helfrich 1973], based on the surface curvature, and surface tension determined by the surface incompressibility condition $\nabla_{\gamma_i} \cdot \boldsymbol{u} = 0$ resulting in

$$\mathbf{F}(\boldsymbol{x}) = \sum_i \int_{\gamma_i} \mathbf{f}(\boldsymbol{y})\delta(\boldsymbol{x} - \boldsymbol{y})d\boldsymbol{y}$$

(see, e.g., [Rahimian et al. 2015] for the expressions for $\mathbf{f}$). Except on inflow and outflow regions of the vascular network, the boundary condition $\boldsymbol{g}$ is zero, modeling no-slip boundary condition on blood vessel walls.

3.2.1.1 Boundary integral formulation

To enforce the boundary conditions on Γ, we use the standard approach of computing $\boldsymbol{u}$ as the sum of the solution $\boldsymbol{u}^{\text{fr}}$ of the free-space equation Eq. (3.1) without boundary conditions but with non-zero right-hand side $\mathsf{F}(\boldsymbol{x})$, and the second term $\boldsymbol{u}^{\Gamma}$ obtained by solving the homogeneous equation with boundary conditions on Γ given by $\boldsymbol{g} - \boldsymbol{u}^{\text{fr}}$.

Following the approach of [Power and Miranda 1987; Pozrikidis 1992b; Lu et al. 2018; Nazockdast et al. 2015], we reformulate Eqs. (3.1) and (3.2) in the integral form. The free-space solution $\boldsymbol{u}^{\text{fr}}$ can be written directly as the sum of the single-layer Stokes potentials $\boldsymbol{u}^{\gamma_i}$:

$$\boldsymbol{u}^{\gamma_i}(\boldsymbol{x}) = (S_i\mathbf{f})(\boldsymbol{x}) = \int_{\gamma_i} S(\boldsymbol{x},\boldsymbol{y})\mathbf{f}(\boldsymbol{y})d\boldsymbol{y}, \quad \boldsymbol{x} \in \Omega, \tag{3.4}$$

where $S(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{8\pi\mu}\left(\frac{1}{\boldsymbol{r}} + \frac{\boldsymbol{r}\otimes\boldsymbol{r}}{|\boldsymbol{r}|^3}\right)$ for viscosity μ and $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{y}$.

To obtain $\boldsymbol{u}^{\Gamma}$, we reformulate the homogeneous volumetric PDE with nonzero boundary conditions as a boundary integral equation for an unknown double-layer density ϕ defined on the domain boundary Γ:

$$\left(\frac{1}{2}I + D + N\right)\phi = \tilde{D}_{\Gamma}\phi = \boldsymbol{g} - \boldsymbol{u}^{\text{fr}}, \quad \boldsymbol{x} \in \Gamma, \tag{3.5}$$

where the double-layer operator is $D\phi(\boldsymbol{x}) = \int_{\Gamma} D(\boldsymbol{x},\boldsymbol{y})\phi(\boldsymbol{y})d\boldsymbol{y}$ with double-layer Stokes kernel $D(\boldsymbol{x},\boldsymbol{y}) = \frac{6}{8\pi}\left(\frac{\boldsymbol{r}\otimes\boldsymbol{r}}{|\boldsymbol{r}|^5}(\boldsymbol{r}\cdot\boldsymbol{n}\right)$ for outward normal $\boldsymbol{n} = \boldsymbol{n}(\boldsymbol{y})$. The null-space operator needed to make the equations full-rank is defined as $(N\phi)(\boldsymbol{x}) = \int_{\Gamma}(\boldsymbol{n}(\boldsymbol{x})\cdot\phi(\boldsymbol{y}))\boldsymbol{n}(\boldsymbol{y})d\boldsymbol{y}$ (cf. [Lu et al. 2017]). The favorable eigenspectrum of the integral operator in Eq. (3.5) is well-known and allows GMRES to rapidly converge to a solution. One of the key differences between this work and previous free-space large-scale simulations is the need to solve this equation in a scalable way due to differing formulations [Grinberg et al. 2011; Balogh and Bagchi 2017b]. Once the density ϕ is computed, the velocity correction $\boldsymbol{u}^{\Gamma}$ is evaluated directly as $\boldsymbol{u}^{\Gamma} = D\phi$.

The equation for the total velocity $\boldsymbol{u}(\boldsymbol{x})$ at any point $\boldsymbol{x} \in \Omega$ is then given by

$$\boldsymbol{u} = \boldsymbol{u}^{\mathrm{fr}} + \boldsymbol{u}^{\Gamma} = \sum_{i=1}^{N} \boldsymbol{u}^{\gamma_i} + \boldsymbol{u}^{\Gamma}. \tag{3.6}$$

In particular, this determines the update equation for the boundary points of cells; see Eq. (3.3).

Contact formulation . In theory, the contacts between surfaces are prevented by the increasing fluid forces as surfaces approach each other closely. However, ensuring accuracy of resolving forces may require prohibitively fine sampling of surfaces and very small time steps, making large-scale simulations in space and time impractical. At the same time, as shown in [Lu et al. 2017], interpenetration of surfaces results in a catastrophic loss of accuracy due to singularities in the integrals.

To guarantee that our discretized cells remain interference-free, we augment Eqs. (3.1) and (3.2) with an explicit inequality constraint preventing collisions. We define a vector function $V(t)$ with components becoming strictly negative if any cell surfaces intersect each other, or intersect with the vessel boundaries Γ. More specifically, we use the *space-time interference volumes* introduced in [Harmon et al. 2011] and applied to 3D cell flows in [Lu et al. 2018]. Each component of V corresponds to a single connected overlap. The interference-free constraint at time t is then simply $V(t) \geq 0$.

For this constraint to be satisfied, the forces $\mathbf{f}$ are augmented by an artificial collision force, i.e., $\mathbf{f} = \mathbf{f}_b + \mathbf{f}_\sigma + \mathbf{f}_c$, $\mathbf{f}_c = \nabla_u V^T \lambda$, where λ is the vector of Lagrange multipliers, which is determined by the additional *complementarity* conditions:

$$\lambda(t) \geq 0, \quad V(t) \geq 0, \quad \lambda(t) \cdot V(t) = 0, \tag{3.7}$$

at time t, where all inequalities are to be understood component-wise.

To summarize, the system that we solve at every time step can be formulated as follows, where we separate equations for different cells and global and local parts of the right-hand side, as it is important for our time discretization:

$$\boldsymbol{X}_t = \left(\sum_{j \neq i} S_j \mathbf{f}_j + D\phi \right) + S_i \mathbf{f}_i, \quad \text{for points on } \gamma_i, \tag{3.8}$$

$$\nabla_{\gamma_i} \cdot \boldsymbol{X}_t = 0, \quad \mathbf{f}_j = \mathbf{f}(\boldsymbol{X}_j, \sigma_j, \lambda), \tag{3.9}$$

$$B_\Gamma \phi = \boldsymbol{g} - \sum_j S_j \mathbf{f}_j, \quad \text{for points on } \Gamma, \tag{3.10}$$

$$\lambda(t) \geq 0, \quad V(t) \geq 0, \quad \lambda(t) \cdot V(t) = 0. \tag{3.11}$$

At every time step, (3.11) results in coupling of all close γ_i's, which requires a non-local computation. We follow the approach detailed in [Lu et al. 2018, 2017] to define and solve the *nonlinear complementarity problem* (NCP) arising from cell-cell interactions in parallel, and extend it to prevent intersection of cells with the domain boundary Γ, as detailed in Section 3.4.

3.2.2 Algorithm Overview

Next, we summarize the algorithmic steps used to solve the constrained integral equations needed to compute cell surface positions and fluid velocities at each time step. In the subsequent sections, we detail the parallel algorithms we developed to obtain good weak and strong scalability, as shown in Section 3.5.

Overall Discretization. RBC surfaces are discretized using a spherical harmonic representation, with surfaces sampled uniformly in the standard latitude-longitude sphere parametrization. The blood vessel surfaces Γ are discretized using a collection of high-order tensor-product polynomial patches, each sampled at Clenshaw-Curtis quadrature points. The space-time interference volume function $V(t)$ is computed using a piecewise-linear approximation as described in [Lu et al. 2018]. For time discretization, we use a locally-implicit first order time-stepping (higher-order time stepping can be easily incorporated). Interactions between RBCs and the blood vessel surfaces are computed *explicitly*, while the self-interaction of a single RBC is computed *implicitly*.

The state of the system at every time step is given by a triple of distributed vectors $(\boldsymbol{X}, \sigma, \lambda)$. The first two (cell surface positions and tensions) are defined at the discretization points of cells. The vector λ has variable length and corresponds to connected components of collision volumes. We use the subscript i to denote the subvectors corresponding to i-the cell. $\boldsymbol{X}$ and σ are solved as a single system that includes the incompressibility constraint Eq. (3.9). To simplify exposition, we omit σ in our algorithm summary, which corresponds to dropping $\mathbf{f}_\sigma$ in the Stokes equation, and dropping the surface incompressibility constraint equation.

Algorithm summary. At each step t, we compute the new positions $\boldsymbol{X}_i^+$ and collision Lagrange multipliers λ^+ at time $t^+ = t + \Delta t$. We assume that in the initial configuration there are no collisions, so the Lagrange multiplier vector λ is zero. Discretizing in time, Eq. (3.8) becomes

$$\boldsymbol{X}_i^+ = \boldsymbol{X}_i + \Delta t \left(\sum_{j \neq i} S_j \mathbf{f}_j(\boldsymbol{X}_j, \lambda) + D\phi(\boldsymbol{X}_j, \lambda) \right) + \Delta t S_i \mathbf{f}_i(\boldsymbol{X}_i^+, \lambda^+).$$

At each single time step, we perform the following steps to obtain $(\boldsymbol{X}^+, \lambda^+)$ from $(\boldsymbol{X}, \lambda)$. Below evaluation of integrals implies using appropriate (smooth, near-singular or singular) quadrature rules on cell or blood vessel surfaces.

1. Compute the explicit part $\boldsymbol{b}$ of the position update (first term in Eq. (3.8)).

   (a) Evaluate $\boldsymbol{u}^{\text{fr}}$ from $(\boldsymbol{X}, \lambda)$ on Γ with Eq. (3.4).

   (b) Solve Eq. (3.5) for the unknown density ϕ on Γ using GMRES.

   (c) For each cell, evaluate $\boldsymbol{u}_i^\Gamma = D\phi$ at all cell points $\boldsymbol{X}_i$.

   (d) For each cell i, compute the contributions of other cells to $\boldsymbol{X}_i^+$: $\boldsymbol{b}_i^c = \boldsymbol{u}^{\text{fr}} - u^{\gamma_i} = \sum_{j \neq i} S_j \mathbf{f}_j$.

   (e) Set $\boldsymbol{b}_i = \boldsymbol{u}_i^\Gamma + \boldsymbol{b}_i^c$.

2. Perform the implicit part of the update: solve the NCP obtained by treating the second (self-interaction) term in Eq. (3.8) while enforcing the complementarity constraints Eq. (3.7), i.e.,

solve

$$\boldsymbol{X}_i^+ = \boldsymbol{X}_i + \Delta t(\boldsymbol{b}_i + S_i \boldsymbol{f}_i(\boldsymbol{X}_i^+, \lambda^+)), \tag{3.12}$$

$$\lambda(t^+) \geq 0, \quad V(t^+) \geq 0, \quad \lambda(t^+) \cdot V(t^+) = 0. \tag{3.13}$$

Items 1a to 1d all require evaluation of global integrals, evaluated as sums over quadrature points; we compute these sums in parallel with PVFMM . In particular, Item 1b uses PVFMM as a part of each matrix-vector product in the GMRES iteration. These matrix-vector product, as well as Items 1a, 1c and 1d require near-singular integration to compute the velocity accurately near RBC and blood vessel surfaces; this requires parallel communication to find non-local evaluation points. Details of these computations are discussed in Section 3.3.

The NCP is solved using a sequence of *linear complementarity problems* (LCPs). Algorithmically, this requires parallel searches of collision candidate pairs and the repeated application of the distributed LCP matrix to distributed vectors. Details of these computations are provided in Section 3.4.

Other parallel quadrature methods. Various other parallel algorithms are leveraged to perform boundary integrals for the vessel geometry and RBCs. To compute $\boldsymbol{u}^{\gamma_i}(\boldsymbol{X})$ for $\boldsymbol{X} \in \gamma_i$, the schemes presented in [Veerapaneni et al. 2011] are used to achieve spectral convergence for single-layer potentials by performing a spherical harmonic rotation and apply the quadrature rule of [Graham and Sloan 2002]. We use the improved algorithm in [Malhotra et al. 2017] to precompute the singular integration operator and substantially improve overall complexity. To compute $\boldsymbol{u}^{\gamma_i}(\boldsymbol{X})$ for $\boldsymbol{X}$ close to, but not on γ_i, we follow the approaches of [Sorgentone and Tornberg 2018; Malhotra et al. 2017], which use a variation of the high-order near-singular evaluation scheme of [Ying et al. 2006a]. Rather than extrapolating the velocity from nearby check points as in Section 3.3, we use [Veerapaneni et al. 2011] to compute the velocity on surface, upsampled quadrature on γ_i to compute the velocity at check points and interpolate the velocity between them to the desired location. We mention these schemes for the sake of completeness; they are not the primary contribution of this work, but are critical components of the overall simulation.

3.3 Boundary Solver

The main challenge in incorporating prescribed flow boundary conditions $\boldsymbol{g}$ on the domain boundary Γ is the approximation and solution of the boundary integral problem Eq. (3.5). Upon spatial discretization, this is an extremely large, dense linear system that must be solved at every time step due to the changing free space solution $\boldsymbol{u}^{\text{fr}}$ on the right hand side. Since we aim at a scalable implementation, we do not assemble the operator on the left hand side but only implement the corresponding matrix-vector multiplication, i.e., its application to vectors. Combined with an iterative solver such as GMRES, this matrix-vector multiply is sufficient to solve Eq. (3.5). Application of the double-layer operator D to vectors amounts to a near-singular quadrature for points close to Γ. Controlling the error in this computation requires a tailored quadrature scheme. This scheme is detailed below, where we put a particular emphasis on the challenges due to our parallel implementation.

3.3.1 Quadrature for integral equation

The domain boundary Γ is given by a collection of non-overlapping patches $\Gamma = \bigcup_i P_i(Q)$, where $P_i : Q \to \mathbb{R}^3$ is defined on $Q = [-1, 1]^2$. We use the Nyström discretization for Eq. (3.5). Since $D(\boldsymbol{x}, \boldsymbol{y})$ is singular, this requires a singular quadrature scheme for the integral on the right-hand side. We proceed in several steps, starting with the direct non-singular discretization, followed by a distinct discretization for the singular and near-singular case.

Non-singular integral quadrature. We discretize the integral in Eq. (3.5), for $\boldsymbol{x} \notin \Gamma$, by rewriting it as an integral over a set of patches and then apply a tensor-product qth order

Clenshaw-Curtis rule to each patch:

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_i \int_{P_i} D(\boldsymbol{x}, \boldsymbol{y})\phi(\boldsymbol{y})d\boldsymbol{y}_{P_i} \approx \sum_i \sum_{j=0}^{q^2} D(\boldsymbol{x}, \boldsymbol{y}_{ij})w_{ij}\phi(\boldsymbol{y}_{ij}), \tag{3.14}$$

where $\boldsymbol{y}_{ij} = P_i(\boldsymbol{t}_j)$ and $\boldsymbol{t}_j \in [-1, 1]^2$ is the jth quadrature point and w_{ij} is the corresponding quadrature weight. We refer to the points $\boldsymbol{y}_{ij}$ as the *coarse discretization of* Γ and introduce a single global index $\boldsymbol{y}_\ell = \boldsymbol{y}_{ij}$ with $\ell = \ell(i, j) = (i - 1)q^2 + j$, $\ell = 1, \ldots, N$, where N is the total number of quadrature nodes. We can then rewrite the right-hand side of (3.14) compactly as the vector dot product $W(\boldsymbol{x}) \cdot \phi$, where $\phi_\ell = \phi(\boldsymbol{y}_\ell)$ and $W_\ell(\boldsymbol{x}) = D(\boldsymbol{x}, \boldsymbol{y}_\ell)w_\ell$ are the quadrature weights in Eq. (3.14).

As $\boldsymbol{x} \rightarrow \Gamma$ for $\boldsymbol{x} \in \Omega$, the integrand becomes more singular and the accuracy of this quadrature rapidly decreases due to the singularity in the kernel D. This requires us to construct a singular integral discretization for $\boldsymbol{x} = \boldsymbol{y}_\ell$, $\ell = 1, \ldots, N$, and general points on Γ, which is discussed next. Note that the same method is used for evaluation of the velocity values at points close to the surface, once the equation is solved (*near-singular integration*).

Singular and near-singular integral quadrature. We take an approach similar to [Klöckner et al. 2013c]. The idea is to evaluate the integral sufficiently far from the surface using the non-singular quadrature rule (3.14) on an upsampled mesh, and then to extrapolate the accurate values towards the surface. Concretely, to compute the singular integral at a point $\boldsymbol{x}$ near or on Γ, we use the following steps:

1. Upsample ϕ using qth order interpolation, i.e., $\phi^{\text{up}} = U\phi$, where ϕ^{up} is the vector of Nk samples of the density and U is the interpolation operator. To be precise, we subdivide each patch P_i into k square subdomains P_{ik} and use Clenshaw-Curtis nodes in each subdomain. We subdivide uniformly, i.e., P_i is split into $k = 4^\eta$ patches for an integer η. This is the *fine discretization of* Γ. We use W^{up} to denote the weights for Eq. (3.14) the fine discretization quadrature points.

2. Find the closest point $\boldsymbol{y} = P(u^*, v^*)$ to $\boldsymbol{x}$ on Γ for some patch P on Γ with $u^*, v^* \in [-1, 1]$ ($\boldsymbol{y} = \boldsymbol{x}$ if $\boldsymbol{x} \in \Gamma$).

3. Construct *check points* $c_q = c_q(\boldsymbol{x}) = \boldsymbol{y} - (R + ir)\boldsymbol{n}(u^*, v^*)$, $i = 0, \ldots, p$, where $\boldsymbol{n}(u, v)$ is the outward normal vector to Γ at $P(u, v)$.

4. Evaluate the velocity at the check points:

$$\boldsymbol{u}(c_q(\boldsymbol{x})) \approx W^{\mathrm{up}}(c_q) \cdot \phi^{\mathrm{up}}, \quad i = 0, \ldots, p. \tag{3.15}$$

5. Extrapolate the velocity from the check points to $\boldsymbol{x}$ with 1D polynomial extrapolation:

$$\begin{aligned} \boldsymbol{u}(\boldsymbol{x}) \approx \sum_q e_q \boldsymbol{u}(c_q(\boldsymbol{x})) &= \left(\sum_q e_q W^{\mathrm{up}}(c_q)\right) U\phi & (3.16) \\ &= W^{\mathrm{s}}(\boldsymbol{x}) \cdot \phi, & (3.17) \end{aligned}$$

   where e_q are the extrapolation weights.

In this work, the parameters R, p, r and η are chosen empirically to balance the error in the accuracy of $W^{\mathrm{up}}(c_q) \cdot \phi^{\mathrm{up}}$ and the extrapolation to $\boldsymbol{x}$. A schematic of this quadrature procedure is shown in Fig. 2.2.

Discretizing the integral equation. With the singular integration method described above, we take $\boldsymbol{x} = \boldsymbol{y}_\ell$, $\ell = 1 \ldots N$, and obtain the following discretization of Eq. (3.5):

$$\left(\frac{1}{2}I + A\right)\phi = \boldsymbol{g}, \quad A_{\ell m} = W^{\mathrm{s}}_m(\boldsymbol{y}_\ell) + N_{ij}, \tag{3.18}$$

where $\boldsymbol{g}$ is the boundary condition evaluated at $\boldsymbol{y}_\ell$, $W^s_m(\boldsymbol{x})$ is the mth component of $W^s(\boldsymbol{x})$ and N_{ij} is the appropriate element of the rank-completing operator in Eq. (3.5).

The dense operator A is never assembled explicitly. We use GMRES to solve Eq. (3.18), which only requires application of A to vectors ϕ. This matrix-vector product is computed using the steps summarized above.

Extrapolation and upsampling are local computations that are parallelized trivially if all degrees of freedom for each patch are on a single processor. The main challenges in parallelization of the above singular evaluation are 1) initially distributing the patches among processors, 2) computing the closest point on Γ and 3) evaluating the velocity at the check points. The parallelization of these computations is detailed in the remainder of this section.

Far evaluation. To compute the fluid velocity away from Γ, where Eq. (3.5) is non-singular, i.e., at the check points, the integral can be directly evaluated using Eq. (3.14). Observing that Eq. (3.14) has the form of an N-body summation, we use the *fast-multipole method* [Greengard and Rokhlin 1987] to evaluate it for all target points at once. We use the parallel, kernel-independent implementation Parallel Volume Fast Multipole Method (PVFMM) [Malhotra and Biros 2015b, 2016], which has been demonstrated to scale to hundreds of thousands of cores. PVFMM handles all of the parallel communication required to aggregate and distribute the contribution of non-local patches in $O(N)$ time.

3.3.2 Distributing geometry and evaluation parallelization

We load pieces of the blood vessel geometry, which is provided as a quad mesh, separately on different processors. Each face of the mesh has a corresponding polynomial P_i defining the ith patch.

The k levels of patch subdivision induce a uniform quadtree structure within each quad. We use the `p4est` library [Burstedde et al. 2011] to manage this surface mesh hierarchy, keep track of neighbor information, distribute patch data and to refine and coarsen the discretization in parallel. The parallel quadtree algorithms provided by `p4est` are used to distribute the geometry without replicating the complete surface and polynomial patches across all processors. `p4est` also determines parent-child patch relationships between the coarse and fine discretizations and the coordinates of the child patches to which we interpolate.

Once the geometry is distributed, constructing check points, all necessary information for upsampling and extrapolation are either available on each processor or communicated by PVFMM . This allows these operations to be embarassingly parallel.

3.3.3 Parallel closest point search

To evaluate the solution at a point $\boldsymbol{x}$, we must find the closest point $\boldsymbol{y}$ on the boundary to $\boldsymbol{x}$. The distance $\|\boldsymbol{x}-\boldsymbol{y}\|_2$ determines whether or not near-singular integration is required to compute the velocity at $\boldsymbol{x}$. If it is, $\boldsymbol{y}$ is used to construct check points.

In the context of this work, the point $\boldsymbol{x}$ is on the surface of an RBC, which may be on a different processor than the patch containing $\boldsymbol{y}$. This necessitates a parallel algorithm to search for $\boldsymbol{y}$. For that purpose, we extend the spatial sorting algorithm from [Lu et al. 2018, Algorithm 1] to support our fixed patch-based boundary and detect near pairs of target points and patches.

a. *Construct a bounding box $B_{P,\varepsilon}$ for the near-zone of each patch.* We choose a distance d_ε so that for all points $\boldsymbol{z}$ further away than d_ε from P, the quadrature error of integration over P is bounded by ε. The set of points closer to P than d_ε is the *near-zone of P*. We inflate the bounding box B_P of P by d_ε along the diagonal to obtain $B_{P,\varepsilon}$ to contain all such points.

b. *Sample $B_{P,\varepsilon}$ and compute a spatial hash of the samples and $\boldsymbol{x}$.* Let H be the average diagonal length of all $B_{P,\varepsilon}$. We sample the volume contained in $B_{P,\varepsilon}$ with equispaced samples of spacing $h_P < H$. Using a spatial hash function, (such as Morton ordering with a spatial grid of spacing H), we assign hash values to bounding box samples and $\boldsymbol{x}$ to be used as a sorting key. This results in a set of hash values that define the near-zone of Γ.

c. *Sort all samples by the sorting key.* Use the parallel sort of [Sundar et al. 2013] on the sorting key of bounding box samples and that of $\boldsymbol{x}$. This collects all points with identical sorting key (i.e., close positions) and places them on the same processor. If the hash of $\boldsymbol{x}$ matches the hash of a bounding box sample, then $\boldsymbol{x}$ could require near-singular integration, which we check explicitly. Otherwise, we can assume $\boldsymbol{x}$ is sufficiently far from P and does not require singular integration.

d. *Compute distances* $\|\boldsymbol{x} - P_i\|$. For each patch P_i with a bounding box key of $\boldsymbol{x}$, we locally solve the minimization problem $\min_{(u,v)\in[-1,1]^2} \|\boldsymbol{x} - P_i(u, v)\|$ via Newton's method with a backtracking line search. This is a local computation since $\boldsymbol{x}$ and P_i were communicated during the Morton ID sort.

e. *Choose the closest patch* P_i. We perform a global reduce on the distances $\|\boldsymbol{x}-P_i\|$ to determine the closest P_i to $\boldsymbol{x}$ and communicate back all the relevant information required for singular evaluation back to $\boldsymbol{x}$'s processor.

3.4 Parallel collision handling

We prevent collisions of RBCs with other RBCs and with the vessel surface Γ by solving the NCP given in Eqs. (3.19) and (3.20). This is a nonsmooth and non-local problem, whose assembly and efficient solution is particularly challenging in parallel.In this section, we summarize our constraint-based approach and algorithm.

We have integrated piecewise polynomial patches into the framework of [Lu et al. 2018] for parallel collision handling, to which we refer the reader for a more detailed discussion. The key step to algorithmically unify RBCs and patches is to *form a linear triangle mesh approximation* of both objects. We now want to enforce that these meshes are collision-free subject to the physics constraints in Eq. (3.19).

We linearize the NCP and solve a sequence of LCPs whose solutions converge to the NCP solution. At a high-level, the collision algorithm proceeds as follows:

1. Find triangle-vertex pairs of distinct meshes that are candidates for collision.

2. Compute $V(t^+) = V(t^{+,0})$. If any triangle-vertex pairs on distinct meshes collide, the corresponding component of $V(t)$ will be negative.

3. While $V_i(t^{+,k}) < 0$ for any i:

   (a) Suppose m components of $V(t)$ are negative

   (b) Solve the following linearized version of Eqs. (3.19) and (3.20)

$$\boldsymbol{X}_i^{+,k} = \boldsymbol{X}_i + \Delta t(\boldsymbol{b}_i + S_i(\mathbf{f}_i(\boldsymbol{X}_i^{+,k}, \lambda^{+,k})), \tag{3.19}$$

$$\lambda(t^+) \geq 0, \quad L(t^{+,k}) \geq 0, \quad \lambda(t^{+,k}) \cdot L(t^{+,k}) = 0, \tag{3.20}$$

$$\text{where} \quad L(t) = V(t) + \nabla_u V^T \Delta \boldsymbol{X}_i(t) \tag{3.21}$$

   for the kth iteration of the loop and $\boldsymbol{X}_i^{+,k} = \boldsymbol{X}_i + \Delta \boldsymbol{X}_i(t^{+,k})$.

   (c) Find new candidate triangle-vertex pairs and compute $V(t^{+,k})$.

Here, $t^{+,k}$ is the intermediate time step at which a new candidate position $\boldsymbol{X}_i^{+,k}$ occurs. This approach of iteratively solving an NCP with sequence of LCPs was shown to converge superlinearly in [Fang 1984]. In [Yan et al. 2019], the authors demonstrate that one LCP linearization can approximate the NCP accurately; our algorithm uses around seven LCP solves to approximately solve the NCP. Upon convergence of this algorithm, we are guaranteed that our system is collision-free.

To solve the LCP in Item 3b, we follow the approach detailed in [Lu et al. 2017, Section 3.2.2, Section 3.3]. We reformulate the problem first in standard LCP form with diagonally-dominant system matrix $\boldsymbol{B}$, then solve an equivalent root finding problem by applying a minimum-map Newton's method. This can be restructured to use GMRES, so we only need to repeatedly apply $\boldsymbol{B}$ to vectors to solve the LCP. Each entry $\boldsymbol{B}_{ij}$ is the change in the jth contact volume induced by the kth contact force, which is explicitly defined in [Lu et al. 2018, Algorithm 3]. This means that $\boldsymbol{B}$ is of size $m \times m$, where m is the number of collisions, but is extremely sparse. We need not store the entire matrix explicitly; we only compute the non-zero entries and store them in a distributed

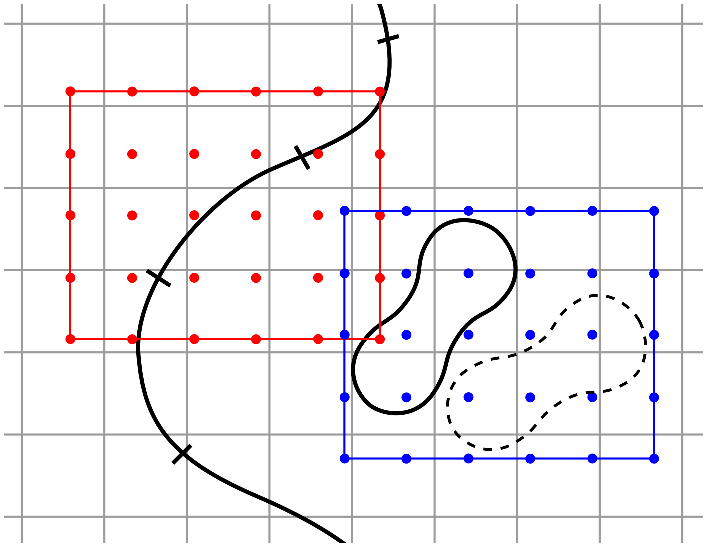


Figure 3.1: A 2D DEPICTION OF THE PARALLEL CANDIDATE COLLISION PAIR ALGORITHM. Shown is the implicit spatial grid (gray), a piece of the blood vessel Γ (open black curve), an RBC γ_i at the current time step (closed black curve) and at the next time step (dotted closed back curve). Also shown is the space-time bounding box and bounding box samples of a single patch (red square and red dots) and an RBC (blue square and blue dots).

hash-map. Computing these matrix elements requires an accumulation of all coupled collision contributions to the velocity, which requires just a sparse `MPI_All_to_Allv` to send each local contribution to the process containing $V_i(t^{+,k})$.

An important step to ensure good scaling of our collision handling algorithm is to minimize the number of triangle-vertex pairs that are found in Item 1. One could explicitly compute an all-to-all collision detection on all meshes in the system, but this requires $O(N^2)$ work and global communication. We perform a high-level filtering first to find local *candidate collision mesh pairs*, then only communicate and compute the required $O(m)$ information. Since spatially-near mesh pairs may be on different processors, we need a parallel algorithm to compute these collision candidates.

To address this, we reuse Items a. to c. from Section 3.3.3 and adapt it to this problem. For each mesh in the system, we form the *space-time bounding box* of the mesh: the smallest axis-aligned bounding box containing the mesh at positions $\boldsymbol{X}_i$ and $\boldsymbol{X}_i^+$, as shown in Fig. 3.1. For patches P_i, note that $P_i^+ = P_i$. This means one can reuse the bounding box of P_i constructed in Section 3.3.3 for this purpose and simply set d_ε to zero. After forming all space-time bounding boxes for the meshes of all patches and RBCs, we apply steps Items b. and c. directly to these boxes. Item c. will communicate meshes with the same spatial sorting key to the same processor; these meshes are collision candidate pairs. Once the computation is local and candidate collision pairs are identified, we can proceed with the NCP solution algorithm described above.

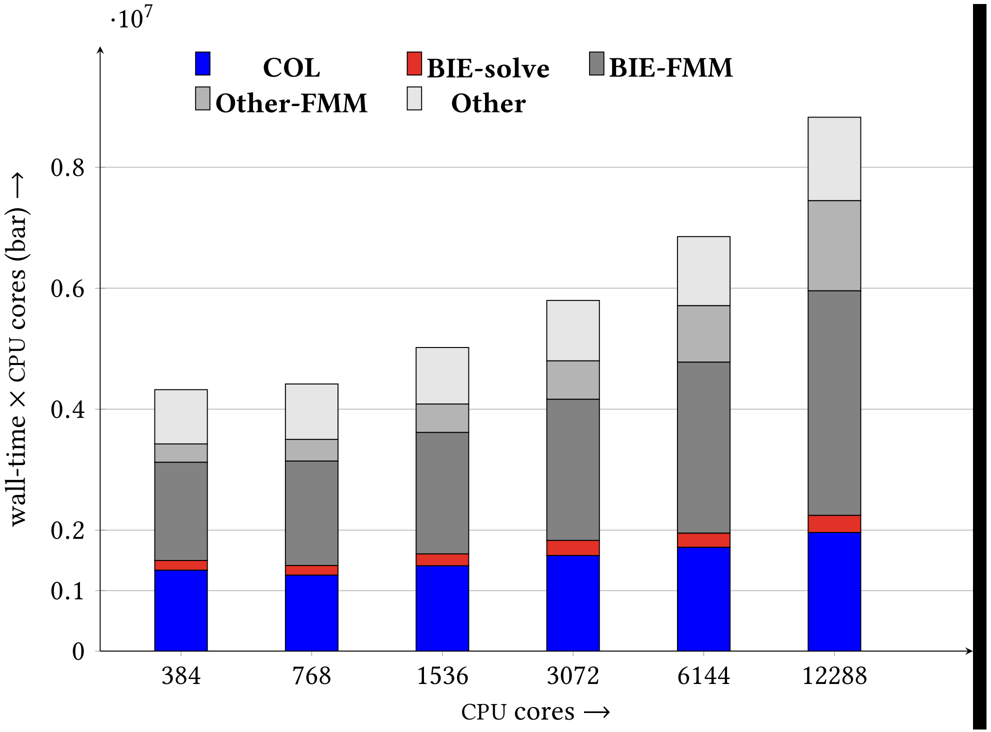
3.5 Results

In this section, we present scalability results for our blood flow simulation framework on various test geometries, simulations with various volume fractions and demonstrate the convergence behavior of our numerical methods.

3.5.1 Implementation and example setup

Architecture and software libraries. We use the Stampede2 system at the Texas Advanced Computing Center (TACC) to study the scalability of our algorithms and implementation. Stampede2 has two types of compute nodes, the Knights Landing (KNL) compute nodes and the Skylake (SKX) compute nodes. The SKX cluster has 1,736 dual-socket compute nodes, each with two 24-core 2.1GHZ CPUs and 192GB of memory. The KNL cluster has 4,200 compute nodes, with a 68-core Intel Xeon Phi 7250 1.4GHZ CPUs and 96GB of memory plus 16GB of high-speed MCDRAM. We run our simulations in a hybrid distributed-shared memory fashion: we run one MPI process per node, with one OpenMP thread per hardware core. Our largest simulations use 256 SKX and 512 KNL nodes.

We leverage several high-performance libraries in our implementation. We use PETSC's [Balay et al. 2017] parallel matrix and vector operations, and its parallel GMRES solver. Management and distribution of patches describing the blood vessel geometry uses the p4est library [Burstedde

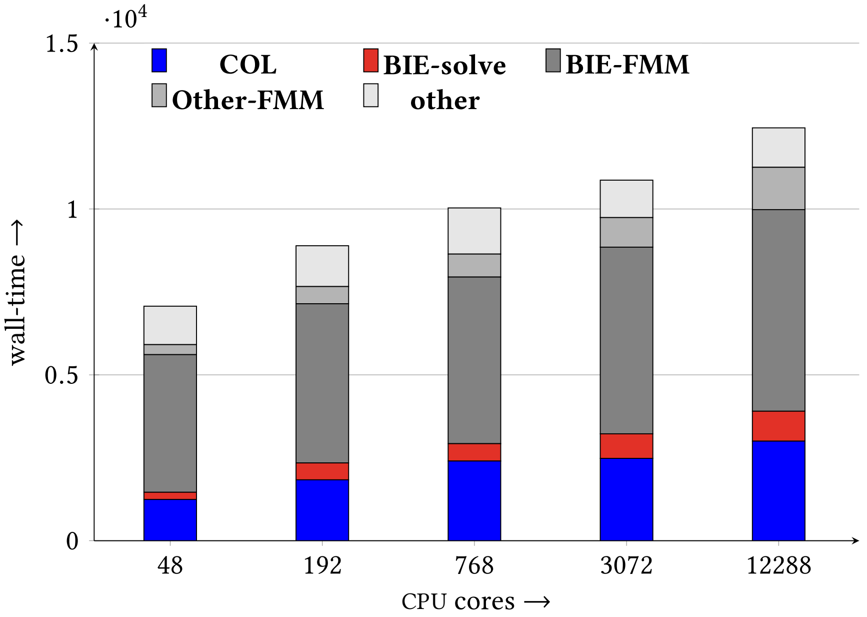


cores	384	768	1536	3072	6144	12288
total time (sec)	11257	5751	3268	1887	1116	718
efficiency	1.00	0.98	0.86	0.75	0.63	0.49
COL+**BIE-solve** (sec)	3901	1843	1046	596	317	183
efficiency	1.00	1.05	0.93	0.82	0.77	0.66

Figure 3.2: Strong scalability of a simultion with 40960 RBCs on Stampede’s SKX partition for the vessel network geometry shown in Fig. 3.7.. The vessel is discretized with 40960 polynomial patches. Shown in the bar graph is a breakdown of the compute resources (wall-time × CPU cores) required by the individual components for a simulation with 10 time steps on 384 to 12288 cores. The compute resources used by the main algorithms presented in this paper are ***COL*** (collision handling), ***BIE-solve*** (computation of $\boldsymbol{u}^{\Gamma}$, not including FMM calls). Shown in different gray scales are the compute resources required by FMM (***BIE-FMM*** and ***Other-FMM***) and other operations (***Other***). Shown in the table are the compute time and the parallel efficiency for the overall computation and for the sum of ***COL*** and ***BIE-solve***. For the collision avoidance and the boundary solve we observe a parallel efficiency of 66% for a 32-fold increase from 384 to 12288 CPU cores.

et al. 2011], and we use PVFMM [Malhotra and Biros 2015b] for parallel FMM evaluation. We also heavily leverage `Intel MKL` for fast dense linear algebra routines at the core of our algorithms and `paraview` for our visualizations.

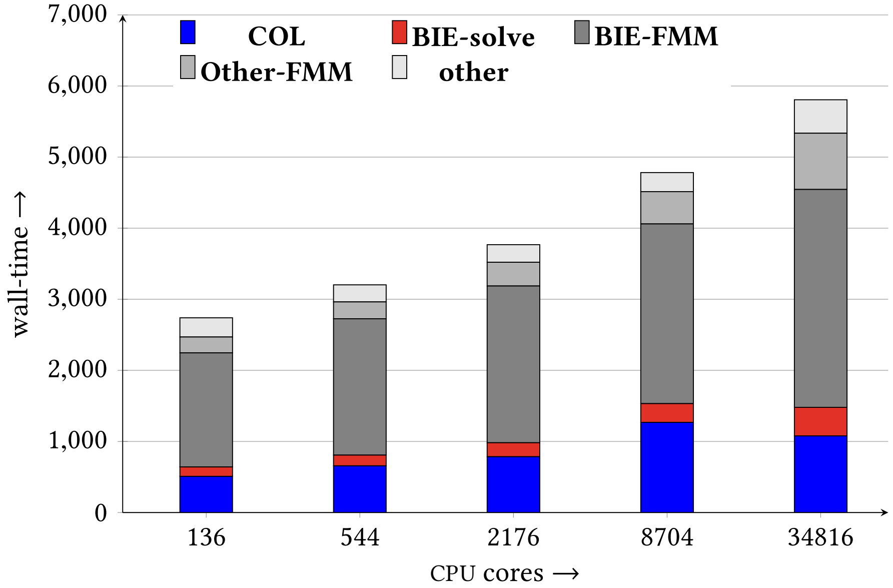
Discretization and example setup. For all test cases we present, we discretize each RBC



cores	48	192	768	3072	12288
vol fraction	19%	20%	23%	26%	27%
#collision/ #RBCs	15%	13%	17%	15%	16%
total time (sec)	7070	8892	10032	10869	12446
efficiency	–	1.00	0.88	0.81	0.71
COL+BIE-solve (sec)	1461	2345	2926	3222	3904
efficiency	–	1.00	0.80	0.73	0.60

Figure 3.3: Weak scalability on Stampede's SKX partition with node grain size of 4096 RBCs and 8192 polynomial patches per compute node (each node has 48 cores) for the vessel geometry shown in Fig. 3.6.. Increasing the number of RBCs and boundary patches is realized by decreasing the size of the RBCs as discussed in Section 3.5.2. Shown in the bar graph is a breakdown of wall-time spent in individual components for a simulation with 10 time steps on 136 to 12288 cores (i.e., 4 to 256 nodes). The explanation of the labels used in the legend is detailed in Fig. 3.2. Additionally, we show the volume fraction of RBCs for each simulation, as well as the percentage of vesicles where the RBC-RBC or RBC-vessel collision prevention is active. We report the parallel scalability with respect to 192 cores, as the smallest simulation is in a single node and no MPI communication is necessary. The largest simulation has 1,048,576 RBCs and 2,097,152 polynomial patches and an overall number of 3,042,967,552 unknowns per time step.

with 544 quadrature points and 2,112 points for collision detection. The blood vessel geometry is represented with 8th order tensor-product polynomial patches with 121 quadrature points per patch and 484 equispaced points for collision detection. The parameters chosen for singular/near-singular integration are $p = 8$ and $\eta = 1$, with $R = r = .15L$ for strong scaling tests and $R = r =$



cores	136	544	2176	8704	34816
vol fraction	17%	19%	20%	23%	26%
#collision/ #RBCs	10%	15%	13%	17%	15%
total time (sec)	2739	3203	3768	4782	5806
efficiency	1.00	0.86	0.73	0.57	0.47
COL+BIE-solve (sec)	642	808	982	1532	1480
efficiency	1.00	0.79	0.65	0.42	0.43

Figure 3.4: Weak scalability on Stampede's KNL parition with node grain size of 4096 RBCs and 8192 polynomial patches per compute node for the vessel geometry shown in Fig. 3.6. Same as Fig. 3.3 but on Stampede2's KNL partition with 512 RBCs and 1024 vessel boundary patches per node (each node has 68 cores). We find an overall parallel scalability of 47% for a 256-fold increase of the problem size.

$.1L$ for weak scaling tests. The value of L is the square root of the surface area of the patch containing the closest point to the target, called the *patch size*; this choice allows for a consistent extrapolation error over the entirety of Γ.

Since our scaling tests are performed on complex, realistic blood vessel geometries, we must algorithmically generate our initial simulation configuration. We prescribe portions of the blood vessel as inflow and outflow regions and appropriately prescribe positive and negative parabolic flows (inlet and outlet flow) as boundary conditions, such that the total fluid flux is zero. To

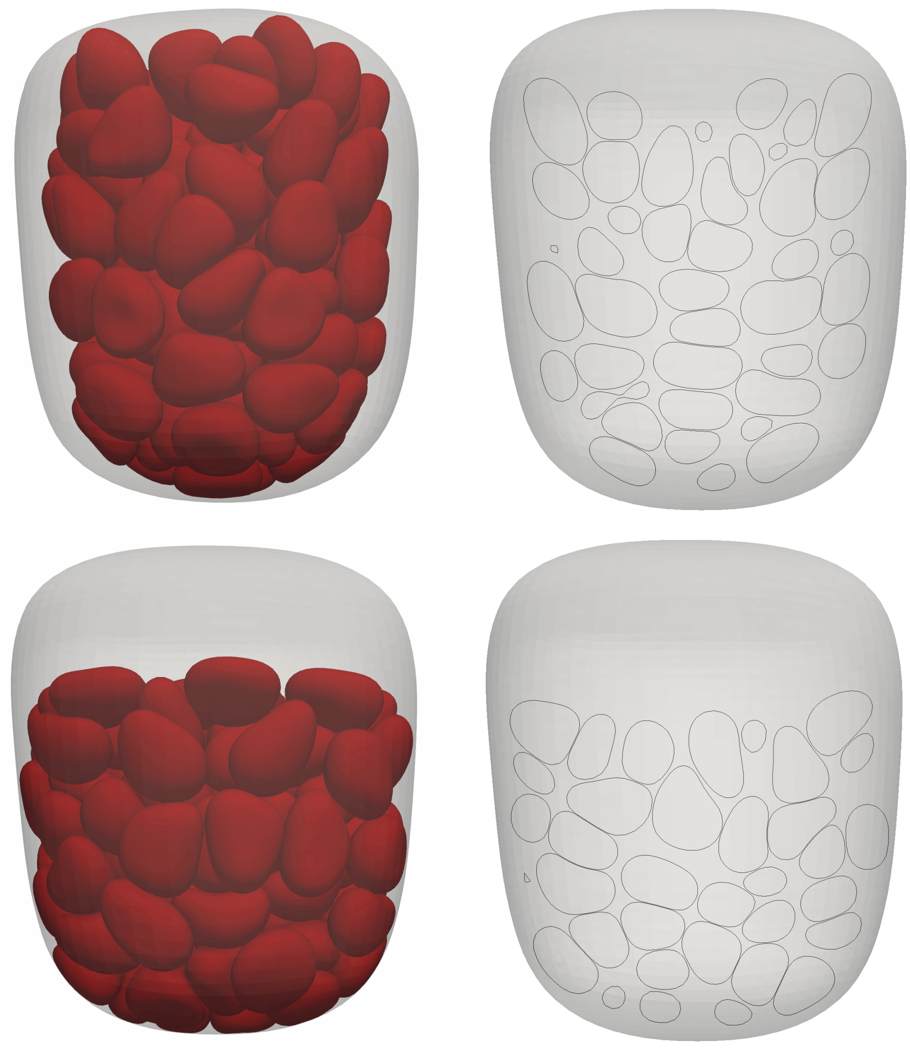


Figure 3.5: High-volume fraction sedimentation due to gravitational force.. The initial configuration (top figures) has a volume fraction of 47%. As the cells sediment to the lower part of the domain (bottom figures), the local volume fraction of the final state in this lower part of the domain is around 55%. Shown on the right side are slices through the center of the domain together with the RBC boundaries in the initial and final configuration. The full simulation video is available at https://vimeo.com/329509435.

populate the blood vessel with RBCs, we uniformly sample the volume of the bounding box of the vessel with a spacing h to find point locations inside the domain at which we place RBCs in a random orientation. We then slowly increase the size of each RBC until it collides with the vessel boundary or another RBC; this determines a single RBC's size. We continue this process until all RBCs stop expanding; this means that we are running a simulation of RBCs of various sizes. We refer to this process as *filling the blood vessel with* RBCs. This typically produces RBCs of radius r with $r_0 < r < 2r_0$ with r_0 chosen proportional to h. This is a precomputation for our simulation, so we do not include this step in the timings we report for weak and strong scaling. We emphasize that these simulations are primarily for scaling purposes of our algorithms and are not expected to represent true blood flows. The platform can of course be applied to length scales where viscous flow is a valid assumption.

Additionally, RBCs in such a confined flow will collide with the blood vessel wall if special care is not taken near the outflow part of the boundary. We define regions near the inlet and outlet flows where we can safely add and remove RBCs. When an RBC γ_i is within the outlet region, we subtract off the velocity due to γ_i from the entire system and move γ_i into an inlet region such that the arising RBC configuration is collision-free.

Limiting GMRES iterations. We have observed that the GMRES solver typically requires 30 iterations or less for convergence for almost all time steps, but the number of needed iterations may vary more in the first steps. To simulate the amount of work in a typical simulation time step, we cap the number of GMRES iterations at 30 and report weak and strong scaling for these iterations. A more detailed analysis of this behavior is needed.

3.5.2 Parallel scalability

Here, we present strong and weak scalability results for our RBC simulations. We decompose the time required for a complete simulation into the following categories:

- ***COL***: detection and resolution of collisions among RBCs and between RBCs and the vessel walls;
- ***BIE-solve***: computing $\boldsymbol{u}^\Gamma$, not including FMM calls. This includes all of the steps for singular/near-singular integration in Section 3.3 except the evaluation $\boldsymbol{u}^\Gamma$ at the check points.
- ***BIE-FMM***: FMM calls required to evaluate $\boldsymbol{u}^\Gamma$ at the check points and at points on RBCs

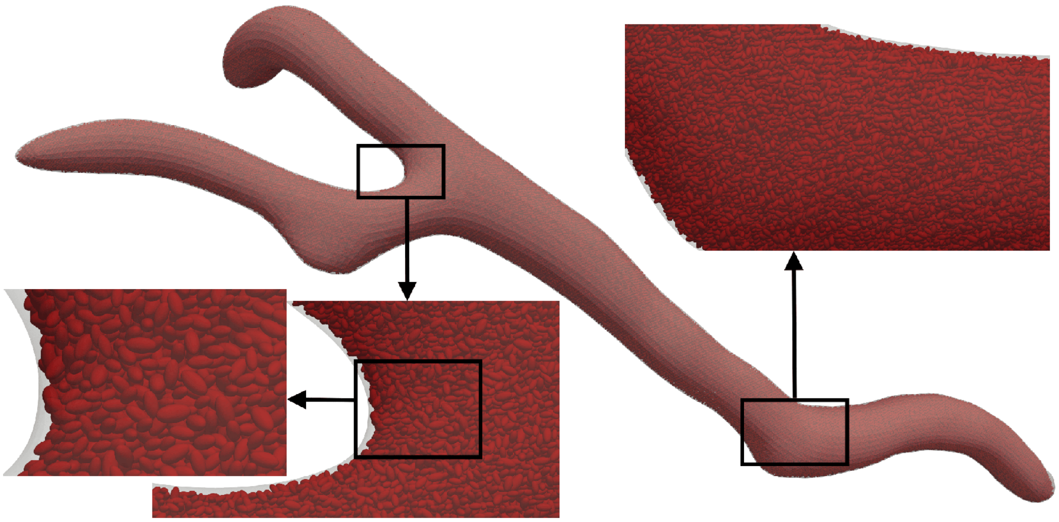


Figure 3.6: Weak scaling vessel geometry. For our weak scaling experiments, we use the the vessel geometry shown above with inflow boundary conditions on the right side and outflow boundary condition on the two left sides. To setup the problem, we fill the vessel with nearly-touching RBCs of different sizes to obtain a desired number, and refine the vessel geometry patches. The figure above shows a setup with overall 262,144 RBCs at a volume fraction of 26%.

- ***Other-FMM***: FMM calls required by other algorithms
- ***Other***: all other operations

In the discussion below, we focus on ***COL*** and ***BIE-solve***, as they are the primary algorithmic contribution of this work, and discuss how to reduce the computational time required for ***BIE-FMM***.

Strong scalability.

To study the strong scalability of our algorithms, we use the blood vessel geometry and RBC configuration in Fig. 3.7-left. This simulation contains 40,960 RBCs and the blood vessel is represented with 40,960 patches. With four degrees of freedom per RBC quadrature point and three per vessel quadrature point, this amounts to 89,128,960 and 14,868,480 degrees for the RBCs and blood vessel, respectively (103,997,440 in total). As can be seen from Fig. 3.2, we achieve a 15.7-fold speed-up in total wall-time scaling from 384 to 12288 cores, corresponding to 49% parallel

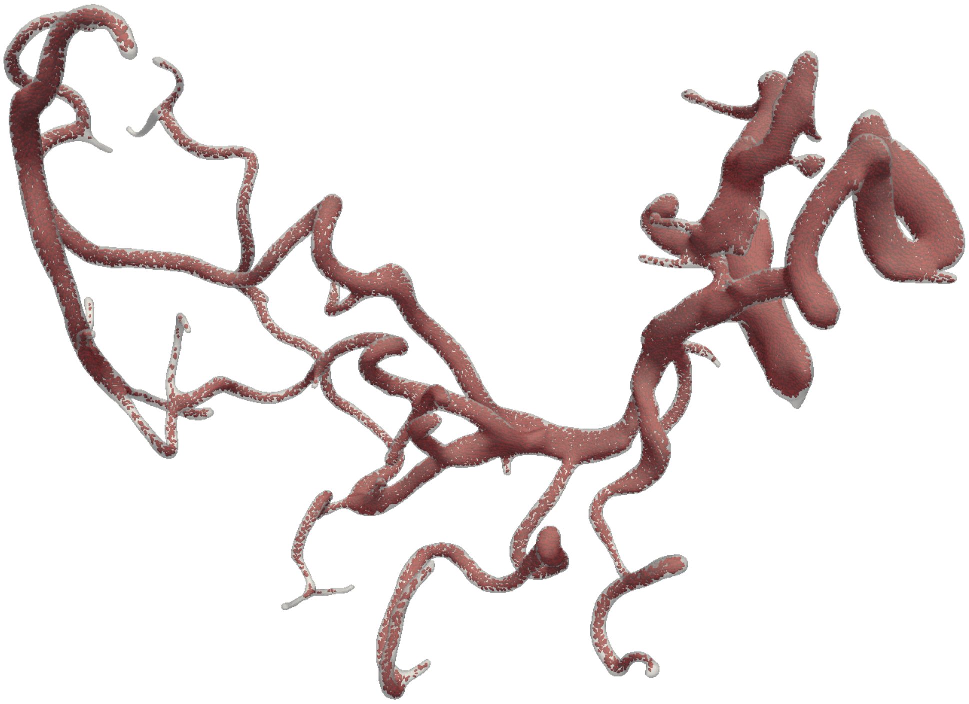


Figure 3.7: STRONG SCALING AND LONG-TIME SIMULATION VESSEL GEOMETRY. Simulation results for 40,960 RBCs in a complex vessel geometry. For our strong scaling experiments, we use the vessel geometry shown on the left, with inflow-outflow boundary conditions at various regions of the vessel geometry. To setup the problem, we fill the vessel with nearly-touching RBCs of different sizes. The figure above shows a setup with overall 40,960 RBCs at a volume fraction of 19%, and 40,960 polynomial patches. The full simulation video is available at https://vimeo.com/329509229.

efficiency. This level of parallel efficiency is partially due to the calls to the fmm library PVFMM. The strong scalability of PVFMM we observe is largely consistent with the results reported in [Malhotra and Biros 2016]. Neglecting the time for calls to FMM, i.e., only counting the time for the boundary solver to compute $\boldsymbol{u}^\Gamma$ and for collision prevention, we find 66% parallel efficiency when scaling strongly from 384 to 12288 cores. We see that the parallel collision handling and integral equation solver computations, excluding FMM, scale well as the number of cores is increased.

Weak Scalability. Our weak scalability results are shown in Figs. 3.3 and 3.4. Both tests are performed on the blood vessel displayed in Fig. 3.6. We use an initial boundary composed of a fixed number M of polynomial patches and fill the domain with roughly $M/2$ RBCs (which requires spacing h). To scale up our simulation by a factor of four, we: (1) subdivide the M polynomial

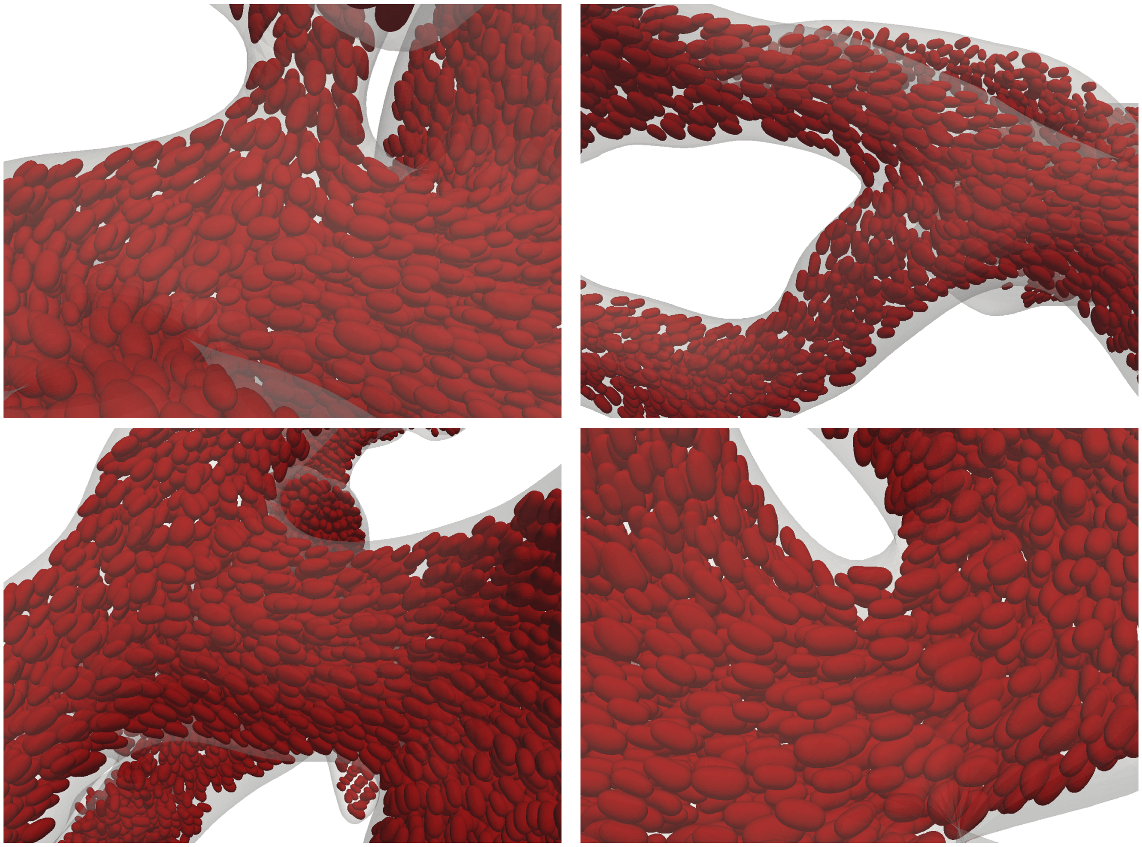


Figure 3.8: Snapshots of Fig. 3.7 during long-time simulation.

patches into $4M$ new but equivalent polynomial patches (via subdivision rules for Bezier curves); (2) refill the domain with RBCs using spacing $h/\sqrt[3]{4}$. This places $2M$ RBCs in the domain volume. We repeat this process each time we increase the number of cores by a factor of four in order to keep the number of patches and RBCs per core constant. In the tables in Figs. 3.3 and 3.4, we report parallel efficiency with respect to the first multi-node run on both SKX and KNL architectures, i.e, with respect to 192 and 136 cores, respectively.

The largest weak scaling test contains 1,048,576 RBCs and 2,097,152 polynomial patches on the blood vessel; we solve for 3,042,967,552 unknowns at each time step and are able to maintain a collision-free state between 4,194,304,000 triangular surface elements at each time step. Comparing the weak scalability results for SKX (Fig. 3.3) and KNL (Fig. 3.4), we observe similar qualitative behavior. Note that the smallest test on the SKX architecture only uses a single node, i.e., no MPI communication was needed. This explains the increased time for the collision prevention algorithms when going from 1 (48 cores) to 4 nodes (192 cores). Note also that the simulation on the KNL architecture used a significantly lower number of RBCs and geometry patches per node. Thus, this simulation has a larger ratio of communication to local work. This explains the less perfect scalability compared to the results obtained on the SKX architecture. As with strong scaling, we see good parallel scaling of the non-FMM -related parts of the computation of $\boldsymbol{u}^{\Gamma}$ and the collision handling algorithm.

Note that there is a slight variation in the number of collisions for the run on 8704 cores on KNL. This is an artifact of the RBC filling algorithm. Since we place RBCs in random orientations and distribute RBCs randomly among processors, we do not have complete control over the percentage of collisions or the volume fraction for each simulation in Figs. 3.3 and 3.4, as can be seen from the tables under these figures. This can affect the overall scaling: For the run on 8704 cores, the percentage of collisions is larger, explaining the longer time spent in ***COL***. Despite this phenomenon, we achieve good weak scaling overall.

Discussion. The parts of the algorithm introduced in this paper scale as well as or better than the FMM implementation we are using. However, our overall run time is diminished by the multiple expensive FMM evaluations required for solving Eq. (3.5). This can be addressed by using a *local* singular quadrature scheme, i.e., compute a singular integral using the FMM on Eq. (3.14) directly, then compute a singular correction locally. This calculation has a three-fold impact on parallel scalability: (1) the FMM evaluation required is proportional to the size of the coarse discretization rather than the fine discretization ($O((p+1)N)$ vs. $O((k+p)N)$); (2) after the FMM evaluation, the local correction is embarrassingly parallel; (3) the linear operator Eq. (3.16) can be precomputed, making the entire calculation extremely fast with MKL linear algebra routines. These improvements together will allow our algorithm to scale well beyond the computational regime explored in this work.

3.5.3 Verification

There are few analytic results known about RBCs in confined Stokes flows against which we can verify our simulations. However, exact solutions can be obtained for a part of our setup, invariants (e.g., surface area) can be considered and solutions for smaller examples can be verified against solutions with fine spatial and temporal discretizations. In particular, in this section, we demonstrate the accuracy of the parallel boundary solver presented in Section 3.3 and numerically study the collision-free time-stepping in Section 3.4.

Boundary solver. The error of the boundary integral solver is determined by the error of integration and the GMRES error, the latter not depending on the number of discretization points due to good conditioning of the equation. The integration error, in turn, can be separated into smooth quadrature error and interpolation error. The former is high-order accurate [Trefethen 2013]. Although our extrapolation is ill-conditioned, we observe good accuracy for $p \leq 8$. The singular evaluation in Section 3.3 converges with rate $O(L^p + L^q)$ corresponding to pth order extrapolation and qth order quadrature. To confirm this numerically, we solve an interior Stokes problem on the surface in Fig. 3.9-right. We evaluate a prescribed analytic solution at the discretization points to obtain the boundary condition. We then solve Eq. (3.18) and compare the numerical solution at on-surface samples different from discretization points, evaluated using the algorithms of Section 3.3. We use $\eta = 2$, $q = 16$, $p = 8$, $R = .04\sqrt{L}$ and $r = R/8$. In Fig. 3.9-left, we report the relative error in the infinity-norm of the velocity. By choosing check point distances proportional to $\sqrt{L}$, we observe the expected $O(L^7)$ convergence.

RBCs with collision resolution and convergence. Our choice of RBC representation and discretization is spectrally accurate in space for the approximation, differentiation and integration of functions on RBC surfaces, as shown in [Veerapaneni et al. 2011]. Although we use first-order time-stepping in this work, *spectral deferred correction* (SDC) can be incorporated into the algorithm exactly as in the 2D version described in [Lu et al. 2017]. This present work demonstrates

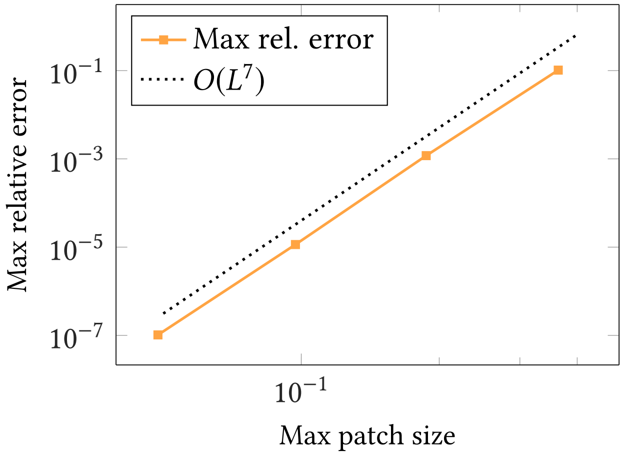


Figure 3.9: Error convergence test solving Eq. (3.18) with hedgehog on the domain Fig. 2.9-right .. We evaluate a known solution on the coarse discretization and solve for ϕ. On the left, we plot the maximum relative error in the infinity norm of $\boldsymbol{u}^\Gamma$ evaluated on the surface. On the right, we show the coarse discretization of the domain boundary and patches.

second-order convergence in time; however, SDC can be made arbitrarily high-order accurate.

For collision-resolution accuracy verification, we study the convergence of our contact-free time-stepping with two RBCs in shear flow. As shown in Fig. 3.10, at $T = 0$, two RBCs are placed in a shear flow $\boldsymbol{u} = [z, 0, 0]$ in free-space. We first compute a reference solution without collision handling but with expensive adaptive fully implicit time-stepping to ensure accurate resolution of the lubrication layer between RBCs. This reference simulatation used spherical harmonics of order 32 and the time step had to be reduced to 6.5×10^{-4} to prevent collisions. In Fig. 3.11, we show the convergence for the error in the centers of mass of each RBC as a function of the time-step size. We use spherical harmonic orders 16 and 32 for the spatial discretization to demonstrate the dominance of the time-stepping error. We observe first-order convergence with our locally-implicit backward Euler scheme which confirms that our collision resolution algorithm does not have a significant impact on time-stepping accuracy.

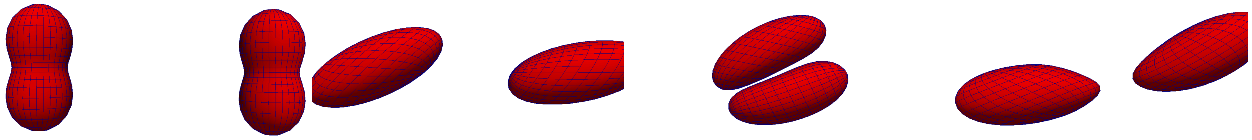


Figure 3.10: Snapshots of two vesicles in shear flow. At $T = 0$, two vesicles are placed at $[-5.5, 0, 0]$ and $[0, 0, 0]$ respectively.

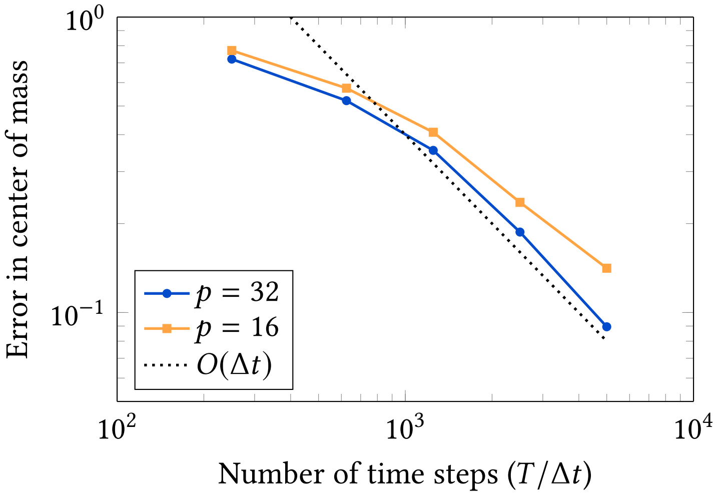


Figure 3.11: Temporal error convergence of shear flow simulation in Fig. 3.10. Shown is the error in the final ($T = 25$) centroid location as we decrease the time step size for two spherical harmonic orders 16 and 32. We observe $O(\Delta t)$ convergence in time and hence the collission detection algorithm converges at the same order as the time stepper.

3.5.4 High volume fraction

The RBC volume fraction, i.e., the ratio of volume occupied by RBCs compared to the overall blood volume is 36-48% in healthy women and 40-54% in healthy men [Billett 1990]. As can be seen in the tables in Figs. 3.3 and 3.4, the volume fraction in our weak scaling simulations is below these values, which is mostly due to the procedure used to fill the blood vessel with RBCs (see the discussion in Section 3.5.1). However, RBC volume fractions in capillaries and small arteries are known to be be around 10-20% [Wang et al. 2013; Saadat et al. 2019], which our scaling simulations achieve. To demonstrate that we can simulate even higher volume fraction blood flows, Fig. 3.5 shows a test of 140 RBCs sedimenting under a gravitational force in a small capsule. The volume fraction for this example is 47%, calculated by dividing the amount of volume occupied by RBCs by the volume of the capsule. By the end of the simulation, we achieve a volume fraction of 55% in the lower part of the domain (determined by bounding the RBCs by a tighter cylinder than the original domain boundary) since the RBCs have become more tightly packed. While such high volume fractions typically do not occur in capillary flow on average, in some scenarios (local fluctuations, sedimentation, microfluidics) these high concentrations need to be handled.

3.6 Conclusion

We have shown that our parallel platform for the simulation of red blood cell flows is capable of accurately resolved long-time simulation of red blood cell flows in complex vessel networks. We are able to achieve realistic cell volume fractions of over 47%, while avoiding collisions between cells or with the blood vessel walls. Incorporating blood vessels into red blood cell simulations requires solving a boundary integral equation, for which we use GMRES. Each GMRES iteration computes a matrix-vector product, which in turn involves singular quadrature and an FMM evaluation; the latter dominates the computation time. To avoid collisions, we solve a nonlinear complementarity problem in the implicit part of each time step. This requires repeated assembly of sparse matrices that, in principle, couple all cells globally. Nevertheless, solving this complementarity system yields close-to-optimal strong and weak scaling in our tests.

4 Conclusion

We have a scalable simulation platform with for RBC flows through capillaries using boundary integral equations. We first presented a robust solver for elliptic PDEs on 3D rigid geometries. We thoroughly studied the behavior and performance of this solver on a variety of geometries and compared it with a competitive state-of-the-art solver. We then parallelized the solver, combined it with boundary integral-based vesicle simulation algorithms and adapted collision-free time stepping to include rigid boundaries. We have scaled our simulations and the parallel solver to thousands of cores and demonstrated the practicality of using such simulations to reproduce qualitatively representative physical RBCs flows.

4.1 Future Work

The comparison between hedgehog and [Ying et al. 2006b] in Section 2.6.2 demonstrated the efficiency of a local quadrature scheme compared to a global one. Moreover, the scaling results in Section 3.5.2 demonstrate that parallel hedgehog is the dominant cost of the simulation. In order to scale RBC simulations beyond the regime explored here, the parallel boundary solver needs to be improved. A key improvement will be the adpotion of a *local* singular quadrature approach. Parallel scaling is largely determined by communication costs and hedgehog performs parallel communication entirely through PVFMM . Reducing the number of total points passed to PVFMM is the best way to improve parallel scaling, since this reduces the overall size of the distributed octree. The local corrections to an inaccurate FMM evaluation can be highly vectorized and require no additional parallel communication. Moreover, the local corrections can be precomputed when solving the integral equation with GMRES. These two facts will dramatically increase the performance of hedgehog .

Another area for improvement is in extrapolation procedure of hedgehog . Equally spaced points serve as a fairly bad interpolant, but we have shown that we're able to use low order polynomials to extrapolate reliably. An important question in the future of hedgehog is how to construct an optimal 1D extrapolation procedure for harmonic functions. An equally important concern is the scheme's inability to resolve oscillatory PDEs such as the Helmholtz equation, due to the difference in resolution achieved on the boundary by q^2 quadrature points and the p check points. A trigonometric extrapolation procedure, coupled with a sampling rate comensurate with the solution's underlying frequency, is one possible approach.

[Wala and Klöckner 2018a, Section 3.3.1] has shown that QBX truncation error is influenced by surface curvature; hedgehog experiences a similar phenomenon. An adaptive placement of check points, determined by surface curvature and overall extrapolation error, would further increase accuracy without increasing cost. Additionally, an approach like Richardson extrapolation could further improve accuracy. Placing several sets of check points for decreasing values of r and evaluating them with a single FMM call would allow for better approximate extrapolation by merging the individual results. Several algorithms in Section 2.3 can be improved. The closest point algorithm in Appendix A.2 can be dramatically improved by leveraging subdivision properties of the Bézier surface representation. We can compute the closest point to the control points of a patch, subdivide the patch, and recursively repeat the process to arrive at more accurate initial guess for the closest point for the 2D Newton method in Appendix A.2. This guess can further cull the 1D optimizations based on the quadrant of the initial guess. Preliminary investigation shows that this outperforms the method detailed in Section 2.3.6.

The point marking algorithm in Section 2.3.6 and the upsampling algorithm in Section 2.3.5

are based on near-zone bounding boxes. A proper quadrature error heuristic similar to [af Klinteberg and Tornberg 2017] would dramatically reduce the amount of upsampling required to guarantee the accuracy of hedgehog. This would improve the third plot in Fig. 2.11 by more accurately determine which points require evaluation via hedgehog, both of which will reduce overall cost. The approach taken in [af Klinteberg et al. 2020] shows exceptional promise toward this end.

The refinement algorithms in Sections 2.3.4 and 2.3.5 require parallelization for hedgehog to be a useful computational tool. This requires minor changes to the parallel closest point algorithm in Section 3.3.3 and the parallel near-pair algorithm in Section 3.4.

Recent work [Wang et al. 2021] has demonstrated that the collision detection scheme in [Harmon et al. 2011] used in Chapter 3 and [Lu et al. 2019, 2018] seems to miss collisions with large separation distances. It appears from [Wang et al. 2021, Section 6] that [Harmon et al. 2011] entirely misses a relatively large number of collisions across all datasets. The collision geometries in Chapter 3 are quite benign compared to the datasets in [Wang et al. 2021], since RBCs and vessels are represented by spherical harmonics and high order polynomials, respectively. Though we verify a collision-free state at each time step in Chapter 3, addressing this shortcoming is crucial. [Li et al. 2020; Ferguson et al. 2021] presents a viable approach, but currently only operates on a single compute node. Implementing a scalable parallel version of [Ferguson et al. 2021] is a key step in simulating more complex geometries or sharp rigid particulates.

Finally, incorporating in-plane shear forces into the RBC model will dramatically improve the overall model accuracy in Chapter 3. Recent optical tweezer exerpiments [Mills et al. 2004; Li et al. 2005] have shown that the cytoskeletal structure of RBCs can withstand surprisingly large amounts of shear force in extreme circumstances. [Fai et al. 2017] could serve as an effective approach. Moreover, quantifying the impact of cytoskeletal modeling on overall RBC flows would be of great interest.

A | Appendix

A.1 Kernels

Here we list the elliptic PDE's investigated in this work along with the associated kernels for their single- and double-layer potentials. In this section, $\boldsymbol{x}$ and $\boldsymbol{y}$ are in $\mathbb{R}^3$, $\boldsymbol{x}$ is the point of evaluation and $\boldsymbol{y}$ is a point on the boundary and $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{y}$. Recall that $\boldsymbol{n}$ is the outward pointing unit normal at $\boldsymbol{y}$ to the domain boundary Γ. We denote the single layer kernel, also known as the *fundamental solution* or *Green's function* of the PDE, by S and the double layer kernel by D.

1. *Laplace equation*:

$$\Delta u = 0$$
$$S(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{4\pi}\frac{1}{\|\boldsymbol{r}\|}, \quad D(\boldsymbol{x},\boldsymbol{y}) = -\frac{1}{4\pi}\frac{\boldsymbol{r}\cdot\mathbf{n}}{\|\boldsymbol{r}\|^3}$$

2. *Stokes equation*:

$$\mu\Delta u - \nabla p = 0, \;\; \nabla \cdot u = 0$$
$$S(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{8\pi\mu}\left(\frac{1}{\|\boldsymbol{r}\|} + \frac{\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^3}\right), \quad D(\boldsymbol{x},\boldsymbol{y}) = -\frac{3}{4\mu\pi}\frac{\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^5}(\boldsymbol{r}\cdot\mathbf{n})$$

3. *Elasticity equation*:

$$\begin{aligned}
&\mu\Delta u - \frac{\mu}{1-2\nu}\nabla(\nabla\cdot u) = 0\\
&S(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{16\pi\mu(1-\nu)}\left(\frac{3-4\nu}{\|\boldsymbol{r}\|} + \frac{\boldsymbol{r}\otimes\boldsymbol{r}}{\|\boldsymbol{r}\|^3}\right),\\
&D(\boldsymbol{x},\boldsymbol{y}) = -\frac{1-2\nu}{8\mu(1-\nu)}\left(\frac{1}{\|\boldsymbol{r}\|^3}\left(\boldsymbol{r}\otimes\boldsymbol{n} - (\boldsymbol{r}\cdot\boldsymbol{n})I - \boldsymbol{n}\otimes\boldsymbol{r}\right) - \frac{3}{1-2\nu}\frac{(\boldsymbol{r}\cdot\boldsymbol{n})(\boldsymbol{r}\otimes\boldsymbol{r})}{\|\boldsymbol{r}\|^5}\right)
\end{aligned}$$

A.2 Computing the closest point on a patch

We include our algorithm to find the closest point $\boldsymbol{y}$ on a patch $\boldsymbol{P}$ to a point $\boldsymbol{x}\in\mathbb{R}^3$ in the section for completeness. For a surface or quadrature patch $\boldsymbol{P}$ and point $\boldsymbol{x}\in\mathbb{R}^3$, we need to compute a point $\boldsymbol{y} = \boldsymbol{P}(s^*,t^*)$ such that

$$(s^*,t^*) = \underset{(s,t)\in[-1,1]^2}{\arg\min}\ \|\boldsymbol{x} - \boldsymbol{P}(s,t)\|_2^2 = \underset{(s,t)\in[-1,1]^2}{\arg\min}\ \boldsymbol{r}(s,t)\cdot\boldsymbol{r}(s,t) \tag{A.1}$$

where $\boldsymbol{r} = \boldsymbol{r}(s,t) = \boldsymbol{x} - \boldsymbol{P}(s,t)$; let $g(s,t) = \boldsymbol{r}\cdot\boldsymbol{r}$. We first consider the unconstrained problem

$$(s^*,t^*) = \underset{(s,t)\in\mathbb{R}^2}{\arg\min}\,\|\boldsymbol{x} - \boldsymbol{P}(s,t)\|_2^2 = \underset{(s,t)\in\mathbb{R}^2}{\arg\min}\,\psi(s,t) \tag{A.2}$$

We solve this optimization problem with Newton's method. The first and second derivatives of ψ can be evaluated efficiently, since they are polynomials of fixed order. The gradient and Hessian of the objective function are:

$$\nabla\psi = \begin{pmatrix} -\boldsymbol{P}_s\cdot\boldsymbol{r}\\ -\boldsymbol{P}_t\cdot\boldsymbol{r}\end{pmatrix},\quad \nabla^2\psi = \begin{pmatrix} \boldsymbol{P}_s\cdot\boldsymbol{P}_s - \boldsymbol{r}\cdot\boldsymbol{P}_{ss} & \boldsymbol{P}_s\cdot\boldsymbol{P}_t - \boldsymbol{r}\cdot\boldsymbol{P}_{st}\\ \boldsymbol{P}_s\cdot\boldsymbol{P}_t - \boldsymbol{r}\cdot\boldsymbol{P}_{st} & \boldsymbol{P}_t\cdot\boldsymbol{P}_t - \boldsymbol{r}\cdot\boldsymbol{P}_{tt}\end{pmatrix}. \tag{A.3}$$

The optimality conditions are

$$\boldsymbol{P}_s^* \cdot \boldsymbol{r}^* = 0, \quad \boldsymbol{P}_t^* \cdot \boldsymbol{r}^* = 0, \quad (u, v) = (s^*, t^*). \tag{A.4}$$

at a local optimum (s^*, t^*).

Let $\psi_i = \psi(s_i, t_i)$, where (s_i, t_i) is the value of the solution during the ith iteration of Newton's method. To solve for the descent direction in Newton's method, we need to solve

$$\nabla^2 \psi_i \, \eta_i = -\nabla \psi_i \tag{A.5}$$

where $\eta_i = (\Delta s_i, \Delta t_i)$ is the ith Newton update to (s_i, t_i) such that

$$s_{i+1} = \alpha_i \Delta s_i + s_i, \quad t_{i+1} = \alpha_i \Delta t_i + t_i \tag{A.6}$$

We use four iterations of a backtracking line search with an Armijo condition to compute the step length α_i to ensure an appropriate size step is taken in case the initial guess is outside the region of quadratic convergence. We compute the solution (s^*, t^*) by iterating

$$(s_n, t_n) = (s_{n-1}, t_{n-1}) + \alpha_{n-1} \eta_{n-1}, \qquad \text{while } \boldsymbol{P}_s \cdot \boldsymbol{r} > \varepsilon_{\text{opt}}, \quad \boldsymbol{P}_t \cdot \boldsymbol{r} > \varepsilon_{\text{opt}}, \tag{A.7}$$

until convergence, i.e., $\psi_i \approx \varepsilon_{\text{opt}}$, $\boldsymbol{r} \approx \boldsymbol{n}(\boldsymbol{y})$.

If $(s^*, t^*) \in (-1, 1)^2$, then the solution to the unconstrained problem is also the solution to the constrained problem. However, if the closest point lies in $\mathbb{R} \setminus [-1, 1]^2$, we need to ensure the inequality constraints are satisfied. Additionally, if (s^*, t^*) is on the boundary of $[-1, 1]^2$, either s^* or t^* should be exactly zero; with the optimization scheme above, we can only claim that $|s^*| < \varepsilon_{\text{opt}}$ (similarly for t^*). To address both of these troubles, we can solve a one-dimensional projection of Eq. (A.5) on to the boundary of $[-1, 1]^2$. For example, to find the closest point along the edge $v = 0$, the Newton iteration becomes

$$s_n = s_{n-1} + \alpha_{n-1} \frac{-\boldsymbol{P}_s \cdot \boldsymbol{r}}{\boldsymbol{P}_s \cdot \boldsymbol{P}_s - \boldsymbol{r} \cdot \boldsymbol{P}_{ss}}, \tag{A.8}$$

where $\boldsymbol{P}_s, \boldsymbol{P}_{ss}$ and $\boldsymbol{r}$ are evaluated at s_{n-1}. Since the boundary is composed of $[-1, t], [1, t], [s, -1], [s, 1]$ for $s, t \in [-1, 1]$, we solve Eq. (A.8) once for each interval.

This final algorithm to compute the closest point is as follows:

1. We solve Eq. (A.5) on an extended parameter domain $[-1 - c, 1 + c]^2$, and terminate the Newton iteration if (s_i, t_i) walks outside this boundary. If the Newton iteration terminates inside $[-1, 1]^2$, then we've found the closest point. We typically choose $c = .2$.

2. If the solution is outside $[-1, 1]^2$, we solve Eq. (A.5) along each component of the boundary of $[-1, 1]^2$, also on an extended parameter domain $[-1-c, 1+c]$, by choosing an initial guess contained within the interval. The solution to these four problems that yields a minimal distance to $\boldsymbol{x}$ to used as the closest point, if the solution is inside $[-1, 1]$.

3. If the closest point on the boundary is still outside of $[-1, 1]^2$, the closest point to $\boldsymbol{x}$ is chosen from $\boldsymbol{P}(-1, -1), \boldsymbol{P}(-1, 1), \boldsymbol{P}(1, -1)$, and $\boldsymbol{P}(1, 1)$ closest to $\boldsymbol{x}$.

This gives us an algorithm to compute the closest point on a quadrature patch $\boldsymbol{P}$ to $\boldsymbol{x}$. The 1D and 2D Newton minimizations converge in ten iterations on average.

A.3 Comparison with [Ying et al. 2006b]

To understand the performance of [Ying et al. 2006b] and hedgehog and see the implications of this complexity difference in practice, we now compare the performance of hedgehog with that of [Ying et al. 2006b] on several concrete numerical examples. The metric we are interested is *cost for a given relative error.* Assuming the surface discretization is $O(N)$, we measure the *cost* of a method as its total wall time during execution T divided by the total wall time of an FMM evaluation on the same $O(N)$ discretization, T_{FMM}. By normalizing by the FMM evaluation cost, we minimize the dependence of the cost on machine- and implementation-dependent machine-dependent parameters, such as clock speed, cache size, performance optimizations, etc. We run the tests in this section on the sphere geometry shown in [Morse et al. 2020a, Figure 8-left] and continue to focus on the singular quadrature scheme of [Ying et al. 2006b] as described in [Morse et al. 2020a, Section 6.2].

A.3.1 Complexity comparison

The algorithm of [Ying et al. 2006b] substantially differs from hedgehog in two main ways. First, on-surface singular integral evaluation is computed in [Ying et al. 2006b] by subtracting the inaccurate part of the FMM -accelerated smooth quadrature rule using a partition-of-unity (POU) function, then adding an accurately computed part singular integral close to singularity via polar quadrature. Second, [Ying et al. 2006b] sets more algorithms parameters *a priori* rather than determining them adaptively. Specific choices used in [Ying et al. 2006b] may be considered optimal for the uniform volume point distribution described in Section 2.5.4, but need to be adjusted based on additional analysis for other distribution types. Additionally, [Ying et al. 2006b] has a trade-off between accuracy and complexity proportional to the POU radius d_P, which hedgehog does not have.

The intermediate and far zone complexity estimates are similar for both hedgehog and [Ying et al. 2006b]. The near-zone complexity for the algorithm of [Ying et al. 2006b] has an additional term of the form $O(Nd_P^2/L_{\max}^2)$, where d_P is the radius of the POU function. For simplicity, we use $L_{\max}$ as a measure of surface sampling density as in Sections 2.5.1 and 2.5.2, since $L_{\max}$ and the h from [Ying et al. 2006b] differ by a constant.

The error of [Ying et al. 2006b]'s singular evaluation is $O(d_P^{-2q-1}L_{\max}^{2q})$, for an optimally chosen local quadrature rule. We note that the factor d_P^{-2q-1} is entirely an artifact of using a compactly supported POU function to localize the singular integral computation. As observed in [Ying et al. 2006b], to achieve optimal convergence as the surface is refined, d_P needs to decrease slower than $L_{\max}$, i.e., slower than $N^{-1/2}$, under the assumptions on point distribution in Ω from Section 2.5.4. In [Ying et al. 2006b], $d_P = O(N^{-1/2(1+\gamma)})$ is suggested. As a result, the overall complexity is $O(N^{1+\gamma})$ and grows faster than N.

By choosing $\gamma = \frac{1}{2}$, [Ying et al. 2006b]'s final complexity becomes $O(N^{3/2})$ in order to produce an error proportional to $N^{(-2q+1)/4}$. In other words, the work needed for an error ε is proportional to $\varepsilon^{-6/(2q-1)}$, which is asymptotically higher than hedgehog (with ε from Section 2.5.2). On the other hand, our method has the disadvantage of requiring p check point evaluations for every sample point in $\mathcal{N}_{\text{near}}$. This requires an FMM call that is $(m + p)$-times larger than [Ying et al. 2006b]. In common use cases, such as solving [Morse et al. 2020a, Equation 5] via GMRES, repeated hedgehog evaluations through a more expensive FMM can require more work in practice for lower accuracy than [Ying et al. 2006b].

A.3.2 Experimental comparison.

To understand the performance of these two methods and see the implications of this complexity difference in practice, we now compare the performance of hedgehog with that of [Ying et al. 2006b] on several concrete numerical examples. The metric we are interested is *cost for a given relative error.* Assuming the surface discretization is $O(N)$, we measure the *cost* of a method as its total wall time during execution T divided by the total wall time of an FMM evaluation on the same $O(N)$ discretization, T_{FMM}. By normalizing by the FMM evaluation cost, we minimize the dependence of the cost on machine- and implementation-dependent machine-dependent parameters, such as clock speed, cache size, performance optimizations, etc.

Comparison on C^∞ surface of [Ying and Zorin 2004] An important contribution of [Ying et al. 2006b] was the use of a C^∞ surface representation, first introduced in [Ying and Zorin

2004], allowing for exponential accuracy via the trapezoidal rule, and easy resampling for singular quadrature. To fairly compare the two quadrature methods, we have implemented a modified version of hedgehog on the surface representation of [Ying and Zorin 2004]. The algorithm of [Morse et al. 2020a, Section 3.1] has the following modifications: (i) we discretize the vertex-centered patches of [Ying and Zorin 2004] with the tensor-product trapezoidal rule for compactly supported functions with spacing parameter h, as in [Ying et al. 2006b]; (ii) the upsampled quadrature rule uses a trapezoidal rule with spacing $h/4$; (iii) density interpolation is computed with FFT 's, as in [Ying et al. 2006b]; the rest of the algorithm proceeds unchanged. This essentially matches [Morse et al. 2020a, Section 3.1] but uses the discretization scheme of [Ying et al. 2006b] instead of Clenshaw-Curtis.

For each of the tests in this section, we choose some initial spacing parameter h_0 to discretize the surface of [Ying and Zorin 2004] as in [Ying et al. 2006b] and use the same 16× upsampled grid to evaluate both hedgehog and [Ying et al. 2006b]. We apply the modified hedgehog algorithm and the scheme of [Ying et al. 2006b] with spacing h_0 and compute the relative error and collect timing statistics. We repeat this test with $h_0/2^i$ for $i = 1, \dots 4$ and plot the results. This ensures that the smooth quadrature rule used by both methods have the same resolution.

We choose the floating partition of unity size in [Ying et al. 2006b] to be $\sqrt{h}$ as in the original work. As in the previous section, we choose the parameters r and R of hedgehog to be $O(\sqrt{h})$ to observe standard convergence behavior. For both quadrature methods, we use a multipole order of 16 for PVFMM with at most 250 points in each leaf box and with the same initial spacing.

In Figs. A.1 and A.2, we summarize our results for two test cases. In Fig. A.1, we evaluate [Morse et al. 2020a, Equation 8] using one-sided hedgehog and the singular quadrature method of [Ying et al. 2006b] with the density $\phi = 1$, in order to demonstrate their behavior without interaction with GMRES. In Fig. A.2, we construct a boundary condition using [Morse et al. 2020a, Equation 25] with random charge values and solve [Morse et al. 2020a, Equation 5] using two-sided hedgehog and with the singular quadrature method of [Ying et al. 2006b] inside of GMRES.

We then evaluate the singular integral at a finer discretization of the surface using either one-sided hedgehog or [Ying et al. 2006b], respectively. From left to right, each plot details the total cost of each scheme, the cost of each subroutine for hedgehog (denoted HH) and the singular quadrature scheme of [Ying et al. 2006b] (denoted POU), and the relative error as a function of h. Each data point in the plots, from right to left, is the result of running the method on a discretization with spacing $h_0/2^i$ for $i = 0, \dots, 4$. We plot the cost of both schemes the cost of each algorithmic step as a function of their computed relative error. In each figure, we present results for a Laplace problem (top) and an elasticity problem (bottom), to highlight the difference in performance between scalar and vector kernels.

As expected, the hedgehog total cost curves lie somewhere between 1 and 10, since the required FMM evaluation is $(m + p)$-times larger than N. This step is the dominant cost: the next most expensive step is density interpolation, which is two orders of magnitude faster. Initially, the main cost of [Ying et al. 2006b] is FMM evaluation time, but eventually the local correction cost begins to dominant. Note that the hedgehog and [Ying et al. 2006b]-FMM curves are not quite flat, due to the initial quadratic complexity of a shallow FMM tree.

From Figs. A.1 and A.2, we observe a higher convergence rate for hedgehog than [Ying et al. 2006b], except for the elasticity solve in Fig. A.2-bottom where the methods perform about equally. This allows the cost of hedgehog to decrease below [Ying et al. 2006b] for errors less than 10^{-7} for Laplace problems. More importantly, however, [Ying et al. 2006b] outperforms hedgehog for elasticity problems for all tested discretizations, and also for low and moderate accuracy Laplace problems. This is due to the greater cost of a vector FMM evaluation compared to a scalar one: the $m + p$ factor saved in the FMM evaluation of [Ying et al. 2006b] can be accelerated more efficiently with the method's small dense linear algebra computations. This means that a local singular quadrature method of *worse* complexity can beat a global method, simply by virtue of reducing the FMM size. Moreover, our implementation of [Ying et al. 2006b] is not highly optimized, so we can expect a well-engineered POU singular quadrature implementation such as [Malhotra

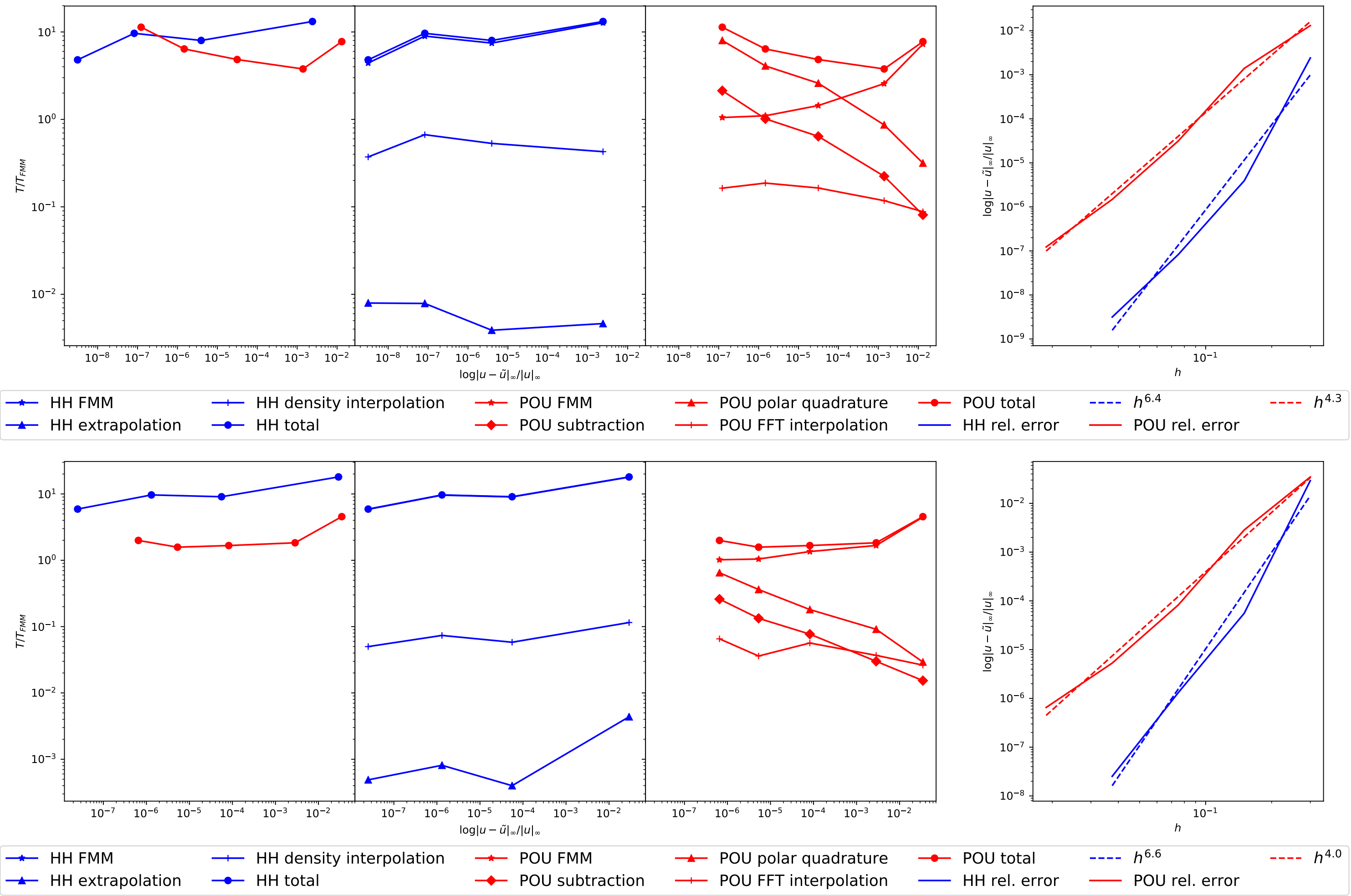


Figure A.1: Comparison of hedgehog (HH) versus [Ying et al. 2006b] (POU) on the surface representation of [Ying and Zorin 2004] evaluating double-layer potential with $\phi = 1$. Laplace (top) and elasticity (bottom) problems solved on the sphere shown in [Morse et al. 2020a, Figure 8-left]. From left to right, we plot the total cost of each scheme, the cost of each subroutine for hedgehog (blue) and the singular quadrature scheme of [Ying et al. 2006b] (red), and the relative error as a function of h. The plots show the cost and relative error for $h_0 = .3$ representing the right-most data point and each point to the left corresponding to a spacing of $h_i = h_0/2^i$. For the Laplace problem, we choose $r = .186\sqrt{h}$, $R = 1.12\sqrt{h}$ and $p = 6$ for hedgehog parameters; for the elasticity problem, we choose $r = .133\sqrt{h}$, $R = .8\sqrt{h}$ and $p = 6$. The initial spacing parameter is $h_0 = .3$.

et al. 2019] to widen this gap. By noting the large difference between the hedgehog FMM cost and the hedgehog density interpolation, we can reasonably infer that a local hedgehog scheme should narrow this gap and outperform [Ying et al. 2006b], assuming that this transition does not

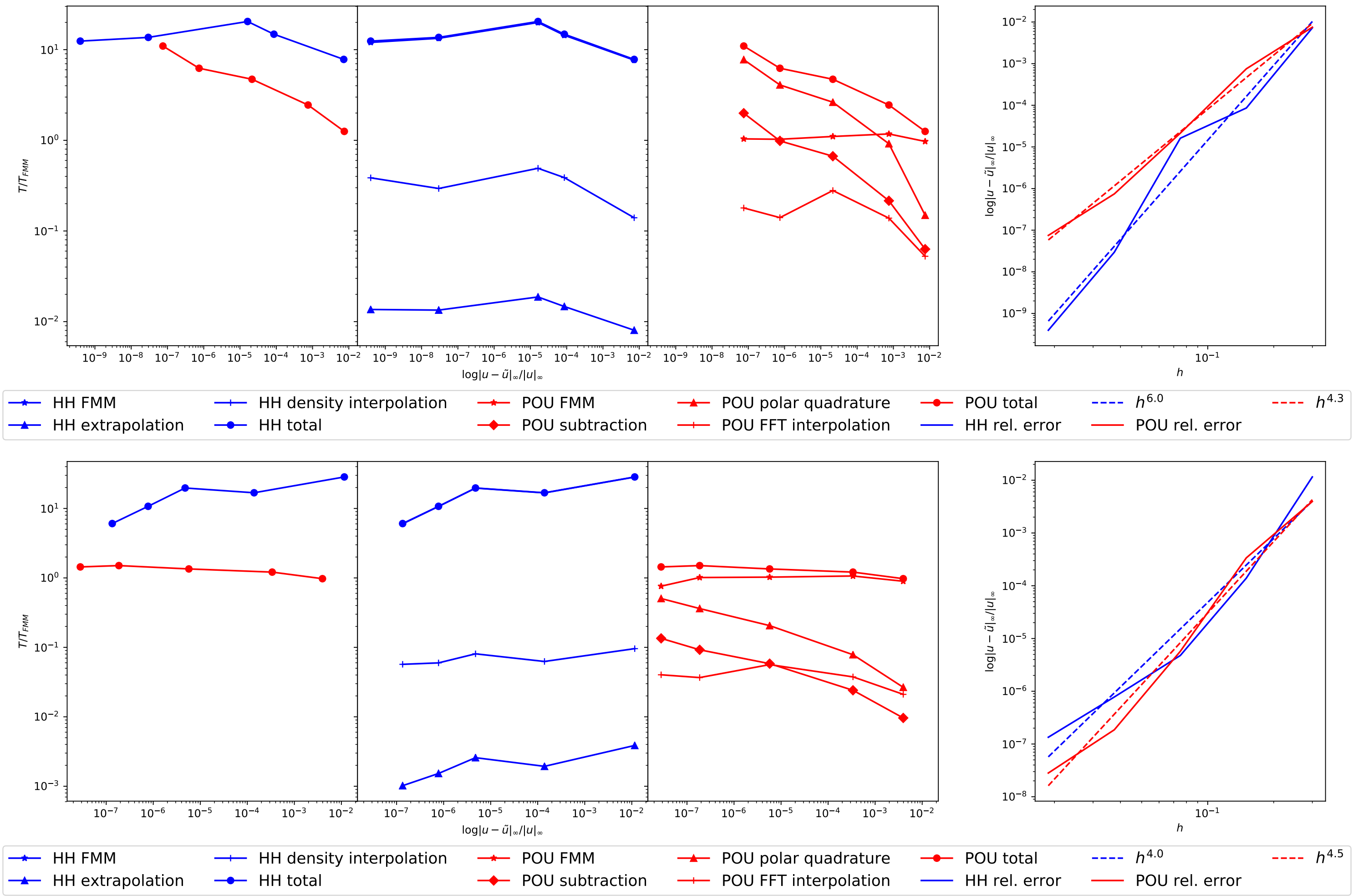


Figure A.2: Comparison of hedgehog versus [Ying et al. 2006b] on the surface representation of [Ying and Zorin 2004] solving via GMRES for u_c. This figure's format is similar to Fig. A.1. For the Laplace problem, we choose $r = .028\sqrt{h}$, $R = .172\sqrt{h}$ and $p = 6$ for hedgehog parameters; for the elasticity problem, we choose $r = .042\sqrt{h}$, $R = .253\sqrt{h}$ and $p = 6$. The initial spacing parameter is $h_0 = .3$.

dramatically affect error convergence.

A.4 Derivation of Heuristic 2.1

We are interested in computing the error incurred when approximating a 2D surface integral with an interpolatory quadrature rule. In 1D on the interval $[-1,1]$, we're interested in the quantity

$$R_q[f] = I[f] - Q_q[f] \tag{A.9}$$

where

$$I[f] = \int_{-1}^{1} f(x)dx \tag{A.10}$$

$$Q_q[f] = \sum_{i=0}^{q} f(x_i)w_i, \tag{A.11}$$

$$\tag{A.12}$$

for quadrature weights w_i for a q-point quadrature rule. For a 2D double integral, we define a similar relationship between the remainder, the exact integral and the qth order quadrature rule:

$$R_q^{(2)}[f] = I^{(2)}[f] - Q_q^{(2)}[f] \tag{A.13}$$

where

$$I^{(2)}[f] = \int_{-1}^{1}\int_{-1}^{1} f(s,t)dsdt \tag{A.14}$$

$$Q_q^{(2)}[f] = \sum_{j=0}^{q}\sum_{i=0}^{q} f(s_i,t_j)w_iw_j, \tag{A.15}$$

For a function of two variables $f(s,t)$, we will denote $I_s[f] = \int_{-1}^{1} f(s,\cdot)ds$ as integration with respect to the s variable only, which produces a function of t. The same subscript notation applies to $R_{q,s}[f]$ and $Q_{q,s}[f]$ and use similar notation for t: we apply the 1D functional to the variable in the subscript, producing a 1D function in the remaining variable. We observe that

$$I^{(2)}[f] = \int_{-1}^{1} \left(\int_{-1}^{1} f(s,t) ds \right) dt = \int_{-1}^{1} I_s[f] dt = I_t[I_s[f]] \tag{A.16}$$

Following the discussion in [af Klinteberg and Tornberg 2017], we substitute into Eq. (A.16) and have

$$\begin{aligned} I^{(2)}[f] &= I_t[R_{q,s}[f] + Q_{q,s}[f]] && \text{(A.17)} \\ &= R_{q,t}[R_{q,s}[f] + Q_{q,s}[f]] + Q_{q,t}[R_{q,s}[f] + Q_{q,s}[f]] && \text{(A.18)} \\ &= R_{q,t}[R_{q,s}[f]] + Q_{q,s}[R_{q,t}[f]] + Q_{q,t}[R_{q,s}[f]] + Q_{q,t}[Q_{q,s}[f]] && \text{(A.19)} \end{aligned}$$

We assume that the higher-order "remainder of remainder" term contributes negligibly to the error. Although it has been shown that this term has a non-trivial contribution to a tight error estimate [Elliott et al. 2015], we are able to provide a sufficiently tight upper bound. For large q, the quadrature rule approaches the value of the integral, i.e., $Q_{q,\beta} \approx I_\beta$ for $\beta = s, t$, we're left with:

$$I^{(2)}[f] \approx I_s[R_{q,t}[f]] + I_t[R_{q,s}[f]] + Q_q^{(2)}[f], \tag{A.20}$$

and hence:

$$R_q^{(2)}[f] \lesssim I_s[R_{q,t}[f]] + I_t[R_{q,s}[f]], \tag{A.21}$$

where $\lesssim$ means "approximately less than or equal to." From [Trefethen 2008, Theorem 5.1], we recall that for a 1D function θ defined on $[-1, 1]$, if $Q_q[\theta]$ is computed with Clenshaw-Curtis quadrature, θ is C^k and $\|\theta^{(k)}\|_T < V$ on $[-1, 1]$ for real finite V, then for sufficiently large q, the following inequality holds

$$R_q[\theta] \le \frac{32V}{15\pi k(2q + 1 - k)^k}, \tag{A.22}$$

where $\|\alpha(x)\|_T = \|\alpha'/\sqrt{1-x^2}\|_1$. We're interested in integrating a function $\tilde{\theta}$ over an interval

$[-h, h]$ for various h. If $\tilde{\theta}$ is C^k and $\|\tilde{\theta}\|_T < V'$ on $[-h, h]$ for a real constant V' independent of h, then we can define $\theta(x) = \tilde{\theta}(hx)$ on $[-1, 1]$ and apply Eq. (A.22):

$$R_q[\tilde{\theta}] \leq \frac{32h^{k+1}V'}{15\pi k(2q+1-k)^k}. \tag{A.23}$$

This follows directly from the proof of [Trefethen 2008, Theorem 4.2] applied to θ by replacing θ with $\tilde{\theta}(hx)$ and noting that $\theta^{(k)}(x) = h^k \tilde{\theta}^{(k)}(hx)$. The change of variables produces the first power of h, while each of the k integration by parts produces an additional power of h. In the context of hedgehog, the size of h is proportional to the edge length of the subdomain D_i outlined in Section 2.2.2.

Applying Eq. (A.23) to Eq. (A.21), and again letting $f(s, t) = \Theta(hs, ht)$, gives us

$$R_q^{(2)}[f] \lesssim \frac{32h^{k+1}}{15\pi k(2q+1-k)^k} \left[I_s[V_t'(s)] + I_t[V_s'(t)] \right] \tag{A.24}$$

where $V_t'(s) = \max_t \|\Theta^{(k)}(hs, ht)\|_T$ and $V_s'(t) = \max_s \|\Theta^{(k)}(hs, ht)\|_T$ for fixed values of s, t. If we can choose a $\tilde{V}$ that is strictly greater than $V_s'(t)$ and $V_t'(s)$ for any s, t in $\mathcal{I}^{(2)}$, we are left with

$$R_q^{(2)}[f] \lesssim \frac{128h^{k+1}\tilde{V}}{15\pi k(2q+1-k)^k}. \tag{A.25}$$

Applying this to the integration of double layer potentials, we can simply let $\tilde{V}$ be the largest variation of the kth partial derivatives of the integrand of any single patch in Eq. (2.7). In fact, we know that this value is achieved at the projection of $\boldsymbol{x}$ on the patch P_i closest to $\boldsymbol{x}$, i.e., $(s^*, t^*) = \operatorname{argmin}_{\mathcal{I}^{(2)}} \|\boldsymbol{x} - P_i(s, t)\|_2$. We can also choose $h = \max_i h_i$ to observe standard high-order convergence as a function of patch domain size, which we summarize in the following theorem. The smoothness and bounded variation assumptions required to apply Eq. (A.22) to our layer potential follow directly from the smoothness of $u(\boldsymbol{x})$ in Ω. Our heuristic directly follows.

Bibliography

Abduljabbar, M., Farhan, M. A., Al-Harthi, N., Chen, R., Yokota, R., Bagci, H., and Keyes, D. (2019). Extreme scale FMM-accelerated boundary integral equation solver for wave scattering. *SIAM Journal on Scientific Computing*, 41(3):C245–C268.

af Klinteberg, L. and Barnett, A. H. (2019). Accurate quadrature of nearly singular line integrals in two and three dimensions by singularity swapping. *arXiv preprint arXiv:1910.09899*.

af Klinteberg, L., Sorgentone, C., and Tornberg, A.-K. (2020). Quadrature error estimates for layer potentials evaluated near curved surfaces in three dimensions. *arXiv preprint arXiv:2012.06870*.

af Klinteberg, L. and Tornberg, A.-K. (2016). A fast integral equation method for solid particles in viscous flow using quadrature by expansion. *Journal of Computational Physics*, 326:420–445.

af Klinteberg, L. and Tornberg, A.-K. (2017). Error estimation for Quadrature by Expansion in layer potential evaluation. *Advances in Computational Mathematics*, 43(1):195–234.

af Klinteberg, L. and Tornberg, A.-K. (2018). Adaptive Quadrature by Expansion for layer potential evaluation in two dimensions. *SIAM Journal on Scientific Computing*, 40(3):A1225–A1249.

Al Quddus, N., Moussa, W. A., and Bhattacharjee, S. (2008). Motion of a spherical particle in a cylindrical channel using arbitrary lagrangian–eulerian method. *Journal of colloid and interface science*, 317(2):620–630.

Alpert, B. K. (1999). Hybrid Gauss-trapezoidal quadrature rules. *SIAM Journal on Scientific Computing*, 20(5):1551–1584.

Atkinson, K. and Han, W. (2009). Numerical solution of Fredholm integral equations of the second kind. In *Theoretical Numerical Analysis*, pages 473–549. Springer.

Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W., Kaushik, D., et al. (2017). Petsc users manual revision 3.8. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States).

Balogh, P. and Bagchi, P. (2017a). A computational approach to modeling cellular-scale blood flow in complex geometry. *Journal of Computational Physics*, 334:280–307.

Balogh, P. and Bagchi, P. (2017b). Direct numerical simulation of cellular-scale blood flow in 3d microvascular networks. *Biophysical journal*, 113(12):2815–2826.

Barenholz, Y., Moore, N. F., and Wagner, R. R. (1976). Enveloped viruses as model membrane systems: microviscosity of vesicular stomatitis virus and host cell membranes. *Biochemistry*, 15(16):3563–3570.

Barnett, A. H. (2014). Evaluation of layer potentials close to the boundary for Laplace and Helmholtz problems on analytic planar domains. *SIAM Journal on Scientific Computing*, 36(2):A427–A451.

Barnett, A. H. and Betcke, T. (2008). Stability and convergence of the method of fundamental solutions for Helmholtz problems on analytic domains. *Journal of Computational Physics*, 227(14):7003–7026.

Beale, J. T. (2004). A grid-based boundary integral method for elliptic problems in three dimensions. *SIAM Journal on Numerical Analysis*, 42(2):599–620.

Beale, J. T., Ying, W., and Wilson, J. R. (2016). A simple method for computing singular or nearly singular integrals on closed surfaces. *Communications in Computational Physics*, 20(3):733–753.

Berrut, J.-P. and Trefethen, L. N. (2004). Barycentric lagrange interpolation. *Siam Review*, 46(3):501–517.

Bespalov, A., Betcke, T., Haberl, A., and Praetorius, D. (2019). Adaptive BEM with optimal convergence rates for the Helmholtz equation. *Computer Methods in Applied Mechanics and Engineering*, 346:260–287.

Betcke, T., Haberl, A., and Praetorius, D. (2019). Adaptive boundary element methods for the computation of the electrostatic capacity on complex polyhedra. *arXiv preprint arXiv:1901.08393*.

Billett, H. H. (1990). Hemoglobin and hematocrit. In *Clinical Methods: The History, Physical, and Laboratory Examinations. 3rd edition*. Butterworths.

Bremer, J. and Gimbutas, Z. (2012). A Nyström method for weakly singular integral operators on surfaces. *Journal of computational physics*, 231(14):4885–4903.

Bremer, J. and Gimbutas, Z. (2013). On the numerical evaluation of the singular integrals of scattering theory. *Journal of Computational Physics*, 251:327–343.

Bruno, O. P. and Kunyansky, L. A. (2001). A fast, high-order algorithm for the solution of surface scattering problems: basic implementation, tests, and applications. *Journal of Computational Physics*, 169(1):80–110.

Bruno, O. P. and Lintner, S. K. (2013). A high-order integral solver for scalar problems of diffraction by screens and apertures in three-dimensional space. *Journal of Computational Physics*, 252:250–274.

Burstedde, C., Wilcox, L. C., and Ghattas, O. (2011). `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133.

Canham, P. B. (1970). The minimum energy of bending as a possible explanation of the biconcave shape of the human red blood cell. *Journal of theoretical biology*, 26(1):61–81.

Caro, C. G., Pedley, T., Schroter, R., and Seed, W. (2012). *The mechanics of the circulation.* Cambridge University Press.

Carreira, A. C., de Almeida, R. F., and Silva, L. C. (2017). Development of lysosome-mimicking vesicles to study the effect of abnormal accumulation of sphingosine on membrane properties. *Scientific reports*, 7(1):1–16.

Carvalho, C., Khatri, S., and Kim, A. D. (2018a). Asymptotic analysis for close evaluation of layer potentials. *Journal of Computational Physics*, 355:327–341.

Carvalho, C., Khatri, S., and Kim, A. D. (2018b). Asymptotic approximations for the close evaluation of double-layer potentials. *arXiv preprint arXiv:1810.02483*.

Chaillat, S., Darbas, M., and Le Louër, F. (2017a). Fast iterative boundary element methods for high-frequency scattering problems in 3D elastodynamics. *Journal of Computational Physics*, 341:429–446.

Chaillat, S., Desiderio, L., and Ciarlet, P. (2017b). Theory and implementation of H-matrix based iterative and direct solvers for Helmholtz and elastodynamic oscillatory kernels. *Journal of Computational physics*, 351:165–186.

Cortinovis, A., Crippa, A., Cavalli, R., Corti, M., and Cattaneo, L. (2006). Capillary blood viscosity in microcirculation. *Clinical hemorheology and microcirculation*, 35(1-2):183–192.

Demanet, L. and Townsend, A. (2016). Stable extrapolation of analytic functions. *arXiv preprint arXiv:1605.09601*.

Du, P., Zhao, J., Cao, W., and Wang, Y. (2017). Dccd: Distributed n-body rigid continuous collision detection for large-scale virtual environments. *Arabian Journal for Science and Engineering*, 42(8):3141–3147.

Elliott, D., Johnston, B. M., and Johnston, P. R. (2008). Clenshaw–Curtis and Gauss–Legendre quadrature for certain boundary element integrals. *SIAM Journal on Scientific Computing*, 31(1):510–530.

Elliott, D., Johnston, B. M., and Johnston, P. R. (2015). A complete error analysis for the evaluation of a two-dimensional nearly singular boundary element integral. *Journal of Computational and Applied Mathematics*, 279:261–276.

Epstein, C. L., Greengard, L., and Klockner, A. (2013). On the convergence of local expansions of layer potentials. *SIAM Journal on Numerical Analysis*, 51(5):2660–2679.

Fai, T. G., Leo-Macias, A., Stokes, D. L., and Peskin, C. S. (2017). Image-based model of the spectrin cytoskeleton for red blood cell simulation. *PLoS computational biology*, 13(10):e1005790.

Fang, S. (1984). A linearization method for generalized complementarity problems. *IEEE transactions on automatic control*, 29(10):930–933.

Farin, G. (1988). *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Professional, Inc., San Diego, CA, USA.

Ferguson, Z., Li, M., Schneider, T., Gil-Ureta, F., Langlois, T., Jiang, C., Zorin, D., Kaufman, D. M., and Panozzo, D. (2021). Intersection-free rigid body dynamics. *ACM Transactions on Graphics (SIGGRAPH)*, 40(4).

Freund, J. B. (2014). Numerical simulation of flowing blood cells. *Annual review of fluid mechanics*, 46:67–95.

Gan, Q. and Watanabe, S. (2018). Synaptic vesicle endocytosis in different model systems. *Frontiers in cellular neuroscience*, 12:171.

Ganesh, M. and Graham, I. G. (2004). A high-order algorithm for obstacle scattering in three dimensions. *Journal of Computational Physics*, 198(1):211–242.

Ghigliotti, G., Rahimian, A., Biros, G., and Misbah, C. (2011). Vesicle migration and spatial organization driven by flow line curvature. *Physical Review Letters*, 106(2):028101.

Gopal, A. and Trefethen, L. N. (2019). Solving Laplace problems with corner singularities via rational functions. *arXiv preprint arXiv:1905.02960*.

Gounley, J., Vardhan, M., and Randles, A. (2017). A computational framework to assess the influence of changes in vascular geometry on blood flow. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, page 2. ACM.

Graham, I. G. and Sloan, I. H. (2002). Fully discrete spectral boundary integral methods for helmholtz problems on smooth closed surfaces in $\mathbb{R}^3$. *Numerische Mathematik*, 92(2):289–323.

Greengard, L., O'Neil, M., Rachh, M., and Vico, F. (2021). Fast multipole methods for the evaluation of layer potentials with locally-corrected quadratures. *Journal of Computational Physics: X*, 10:100092.

Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348.

Griffith, B. E. (2012). Immersed boundary model of aortic heart valve dynamics with physiological driving and loading conditions. *International Journal for Numerical Methods in Biomedical Engineering*, 28(3):317–345.

Grinberg, L., Insley, J. A., Morozov, V., Papka, M. E., Karniadakis, G. E., Fedosov, D., and Kumaran, K. (2011). A new computational paradigm in multiscale simulations: Application to brain blood flow. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 5. ACM.

Harmon, D., Panozzo, D., Sorkine, O., and Zorin, D. (2011). Interference-aware geometric modeling. *ACM Transactions on Graphics*, 30(6):1.

Helfrich, W. (1973). Elastic properties of lipid bilayers: theory and possible experiments. *Zeitschrift für Naturforschung C*, 28(11-12):693–703.

Helsing, J. and Ojala, R. (2008). On the evaluation of layer potentials close to their sources. *Journal of Computational Physics*, 227(5):2899–2921.

Horobin, J. T., Sabapathy, S., and Simmonds, M. J. (2019). Red blood cell tolerance to shear stress above and below the subhemolytic threshold. *Biomechanics and modeling in mechanobiology*, pages 1–10.

Hoskins, J. G., Rokhlin, V., and Serkh, K. (2019). On the numerical solution of elliptic partial differential equations on polygonal domains. *SIAM Journal on Scientific Computing*, 41(4):A2552–A2578.

Hughes, T. J., Cottrell, J. A., and Bazilevs, Y. (2005). Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39-41):4135–4195.

Huisjes, R., Bogdanova, A., van Solinge, W. W., Schiffelers, R. M., Kaestner, L., and Van Wijk, R. (2018). Squeezing for life–properties of red blood cell deformability. *Frontiers in physiology*, 9:656.

Iglberger, K. and Rüde, U. (2009). A parallel rigid body dynamics algorithm. In Sips, H., Epema, D., and Lin, H.-X., editors, *Euro-Par 2009 Parallel Processing*, pages 760–771, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jacobson, A., Kavan, L., and Sorkine-Hornung, O. (2013). Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32(4):33.

Järvenpää, S., Taskinen, M., and Ylä-Oijala, P. (2003). Singularity extraction technique for integral equation methods with higher order basis functions on plane triangles and tetrahedra. *International journal for numerical methods in engineering*, 58(8):1149–1165.

Jarvenpaa, S., Taskinen, M., and Ylä-Oijala, P. (2006). Singularity subtraction technique for high-order polynomial vector basis functions on planar triangles. *IEEE transactions on antennas and propagation*, 54(1):42–49.

Kabacaoğlu, G. and Biros, G. (2019). Sorting same-size red blood cells in deep deterministic lateral displacement devices. *Journal of Fluid Mechanics*, 859:433–475.

Kapur, S. and Rokhlin, V. (1997). High-order corrected trapezoidal quadrature rules for singular functions. *SIAM Journal on Numerical Analysis*, 34(4):1331–1356.

Khatri, S., Kim, A. D., Cortez, R., and Carvalho, C. (2020). Close evaluation of layer potentials in three dimensions. *Journal of Computational Physics*, 423:109798.

Kim, D., Heo, J.-P., Huh, J., Kim, J., and Yoon, S.-e. (2009). HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs. *Computer Graphics Forum*.

Klöckner, A., Barnett, A., Greengard, L., and O'Neil, M. (2013a). Quadrature by Expansion: A new method for the evaluation of layer potentials. *Journal of Computational Physics*, 252:332–349.

Klöckner, A., Barnett, A., Greengard, L., and O'Neil, M. (2013b). Quadrature by Expansion: A new method for the evaluation of layer potentials. *Journal of Computational Physics*, 252:332–349.

Klöckner, A., Barnett, A., Greengard, L., and O'Neil, M. (2013c). Quadrature by expansion: A new method for the evaluation of layer potentials. *Journal of Computational Physics*, 252:332–349.

Kraus, M., Wintz, W., Seifert, U., and Lipowsky, R. (1996). Fluid vesicles in shear flow. *Physical review letters*, 77(17):3685.

Kress, R. (1999). Linear integral equations, volume 82 of applied mathematical sciences.

Lee, J. and Smith, N. P. (2008). Theoretical modeling in hemodynamics of microcirculation. *Microcirculation*, 15(8):699–714.

Lévy, B. (2015). Geogram.

Li, J., Dao, M., Lim, C., and Suresh, S. (2005). Spectrin-level modeling of the cytoskeleton and optical tweezers stretching of the erythrocyte. *Biophysical journal*, 88(5):3707–3719.

Li, M., Ferguson, Z., Schneider, T., Langlois, T., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. (2020). Incremental potential contact: Intersection-and inversion-free, large-deformation dynamics. *ACM transactions on graphics*.

Linden, M., Ward, J. M., and Cherian, S. (2012). Hematopoietic and lymphoid tissues. In *Comparative Anatomy and Histology*, pages 309–338. Elsevier.

Liu, F., Harada, T., Lee, Y., and Kim, Y. J. (2010). Real-time collision culling of a million bodies on graphics processing units. In *ACM SIGGRAPH Asia 2010 Papers*, SIGGRAPH ASIA '10, pages 154:1–154:8, New York, NY, USA. ACM.

Liu, Y. and Barnett, A. H. (2016). Efficient numerical solution of acoustic scattering from doubly-periodic arrays of axisymmetric objects. *Journal of Computational Physics*, 324:226–245.

Lourith, N. and Kanlayavattanakul, M. (2009). Natural surfactants used in cosmetics: glycolipids. *International journal of cosmetic science*, 31(4):255–261.

Lu, L. (2019). *Parallel Contact-Aware Algorithms for Large-Scale Direct Blood Flow Simulations*. PhD thesis, New York University.

Lu, L., Morse, M. J., Rahimian, A., Stadler, G., and Zorin, D. (2019). Scalable simulation of realistic volume fraction red blood cell flows through vascular networks. *arXiv preprint arXiv:1909.11085*.

Lu, L., Rahimian, A., and Zorin, D. (2017). Contact-aware simulations of particulate Stokesian suspensions. *Journal of Computational Physics*, 347C:160–182.

Lu, L., Rahimian, A., and Zorin, D. (2018). Parallel contact-aware simulations of deformable particles in 3d stokes flow. *arXiv preprint arXiv:1812.04719*.

Malhotra, D. and Biros, G. (2015a). PVFMM: A parallel kernel independent FMM for particle and volume potentials. *Communications in Computational Physics*, 18(3):808–830.

Malhotra, D. and Biros, G. (2015b). PVFMM: A Parallel Kernel Independent FMM for Particle and Volume Potentials. *Communications in Computational Physics*, 18:808–830.

Malhotra, D. and Biros, G. (2016). Algorithm 967: A distributed-memory fast multipole method for volume potentials. *ACM Transactions on Mathematical Software (TOMS)*, 43(2):17.

Malhotra, D., Cerfon, A., Imbert-Gérard, L.-M., and O'Neil, M. (2019). Taylor states in stellarators: A fast high-order boundary integral solver. *arXiv preprint arXiv:1902.01205*.

Malhotra, D., Rahimian, A., Zorin, D., and Biros, G. (2017). A parallel algorithm for long-timescale simulation of concentrated vesicle suspensions in three dimensions.

Mazhar, H., Heyn, T., and Negrut, D. (2011). A scalable parallel method for large collision detection problems. 26:37–55.

McGrath, J., Jimenez, M., and Bridle, H. (2014). Deterministic lateral displacement for particle separation: a review. *Lab on a Chip*, 14(21):4139–4158.

Mikhlin, S. G. (2014). *Integral equations: and their applications to certain problems in mechanics, mathematical physics and technology*, volume 4. Elsevier.

Mills, J., Qie, L., Dao, M., Lim, C., and Suresh, S. (2004). Nonlinear elastic and viscoelastic deformation of the human red blood cell with optical tweezers. *Molecular & Cellular Biomechanics*, 1(3):169.

Morse, M. J., Rahimian, A., and Zorin, D. (2020a). A robust solver for elliptic pdes in 3d complex geometries. *arXiv preprint arXiv:2002.04143*.

Morse, M. J., Rahimian, A., and Zorin, D. (2020b). Supplementary material for: A robust solver for elliptic PDEs in 3D complex geometries. *https://cims.nyu.edu/gcl/papers/2020-qbkix3d-supplementary.pdf*.

Nair, N., Pray, A., Villa-Giron, J., Shanker, B., and Wilton, D. (2013). A singularity cancellation technique for weakly singular integrals on higher order surface descriptions. *IEEE Transactions on Antennas and Propagation*, 61(4):2347–2352.

Nazockdast, E., Rahimian, A., Needleman, D., and Shelley, M. (2017). Cytoplasmic flows as signatures for the mechanics of mitotic positioning. *Molecular biology of the cell*, 28(23):3261–3270.

Nazockdast, E., Rahimian, A., Zorin, D., and Shelley, M. (2015). Fast and high-order methods for simulating fiber suspensions applied to cellular mechanics. preprint.

Pabst, S., Koch, A., and Strasser, W. (2010). Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces. *Computer Graphics Forum*.

Parton, V. Z. and Perlin, P. I. (1982). *Integral equations in elasticity*. Imported Pubn.

Perdikaris, P., Grinberg, L., and Karniadakis, G. E. (2015). An effective fractal-tree closure model for simulating blood flow in large arterial networks. *Annals of biomedical engineering*, 43(6):1432–1442.

Perdikaris, P., Grinberg, L., and Karniadakis, G. E. (2016). Multiscale modeling and simulation of brain blood flow. *Physics of Fluids*, 28(2):021304.

Peskin, C. S. (1977). Numerical analysis of blood flow in the heart. *Journal of computational physics*, 25(3):220–252.

Peyrounette, M., Davit, Y., Quintard, M., and Lorthois, S. (2018). Multiscale modelling of blood flow in cerebral microcirculation: Details at capillary scale control accuracy at the level of the cortex. *PloS one*, 13(1):e0189474.

Pfeifer, F. (2012). Distribution, formation and regulation of gas vesicles. *Nature Reviews Microbiology*, 10(10):705–715.

Pironneau, O. (1982). Optimal shape design for elliptic systems. In *System Modeling and Optimization*, pages 42–66. Springer.

Platte, R. B., Trefethen, L. N., and Kuijlaars, A. B. (2011). Impossibility of fast stable approximation of analytic functions from equispaced samples. *SIAM review*, 53(2):308–318.

Potter, R. and Groom, A. (1983). Capillary diameter and geometry in cardiac and skeletal muscle studied by means of corrosion casts. *Microvascular research*, 25(1):68–84.

Power, H. and Miranda, G. (1987). Second kind integral equation formulation of Stokes' flows past a particle of arbitrary shape. *SIAM Journal on Applied Mathematics*, 47(4):689.

Pozrikidis, C. (1992a). *Boundary integral and singularity methods for linearized viscous flow*. Cambridge University Press.

Pozrikidis, C. (1992b). *Boundary integral and singularity methods for linearized viscous flow*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge.

Quaife, B. and Biros, G. (2014). High-volume fraction simulations of two-dimensional vesicle suspensions. *Journal of Computational Physics*, 274:245–267.

Rachh, M., Klöckner, A., and O'Neil, M. (2017). Fast algorithms for Quadrature by Expansion I: Globally valid expansions. *Journal of Computational Physics*, 345:706–731.

Rachh, M. and Serkh, K. (2017). On the solution of Stokes equation on regions with corners. *arXiv preprint arXiv:1711.04072*.

Rahimian, A., Barnett, A., and Zorin, D. (2018). Ubiquitous evaluation of layer potentials using Quadrature by Kernel-Independent Expansion. *BIT Numerical Mathematics*, 58(2):423–456.

Rahimian, A., Lashuk, I., Veerapaneni, S., Chandramowlishwaran, A., Malhotra, D., Moon, L., Sampath, R., Shringarpure, A., Vetter, J., Vuduc, R., et al. (2010). Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society.

Rahimian, A., Veerapaneni, S. K., Zorin, D., and Biros, G. (2015). Boundary integral method for the flow of vesicles with viscosity contrast in three dimensions. *Journal of Computational Physics*, 298:766–786.

Randles, A., Draeger, E. W., Oppelstrup, T., Krauss, L., and Gunnels, J. A. (2015). Massively parallel models of the human circulatory system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 1. ACM.

Rossinelli, D., Tang, Y.-H., Lykov, K., Alexeev, D., Bernaschi, M., Hadjidoukas, P., Bisson, M., Joubert, W., Conti, C., Karniadakis, G., et al. (2015). The in-silico lab-on-a-chip: petascale and high-throughput simulations of microfluidics at cell resolution. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 2. ACM.

Saadat, A., Guido, C. J., Iaccarino, G., and Shaqfeh, E. S. (2018). Immersed-finite-element method for deformable particle suspensions in viscous and viscoelastic media. *Physical Review E*, 98(6):063316.

Saadat, A., Guido, C. J., and Shaqfeh, E. S. (2019). Simulation of red blood cell migration in small arterioles: Effect of cytoplasmic viscosity. *bioRxiv*, page 572933.

Sackmann, E. (1996). Supported membranes: scientific and practical applications. *Science*, 271(5245):43–48.

Samet, H. (2006). *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.

Schleicher, D. and Stoll, R. (2017). Newton's method in practice: Finding all roots of polynomials of degree one million efficiently. *Theoretical Computer Science*, 681:146–166.

Schramm, L. L. (2000). *Surfactants: fundamentals and applications in the petroleum industry*. Cambridge university press.

Schramm, L. L., Stasiuk, E. N., and Marangoni, D. G. (2003). Surfactants and their applications. *Annual Reports Section" C"(Physical Chemistry)*, 99:3–48.

Serkh, K. (2017). On the solution of elliptic partial differential equations on regions with corners II: Detailed analysis. *Applied and Computational Harmonic Analysis*.

Serkh, K. (2018). On the solution of elliptic partial differential equations on regions with corners III: curved boundaries. *Manuscript in preparation*.

Serkh, K. and Rokhlin, V. (2016a). On the solution of elliptic partial differential equations on regions with corners. *Journal of Computational Physics*, 305:150–171.

Serkh, K. and Rokhlin, V. (2016b). On the solution of the Helmholtz equation on regions with corners. *Proceedings of the National Academy of Sciences*, 113(33):9171–9176.

Siegel, M. and Tornberg, A.-K. (2018). A local target specific quadrature by expansion method for evaluation of layer potentials in 3D. *Journal of Computational Physics*, 364:365–392.

Śmigaj, W., Betcke, T., Arridge, S., Phillips, J., and Schweiger, M. (2015). Solving boundary integral problems with BEM++. *ACM Transactions on Mathematical Software (TOMS)*, 41(2):6.

Sorgentone, C. and Tornberg, A.-K. (2018). A highly accurate boundary integral equation method for surfactant-laden drops in 3d. *Journal of Computational Physics*, 360:167–191.

Sorgentone, C., Tornberg, A.-K., and Vlahovska, P. M. (2019). A 3d boundary integral method for the electrohydrodynamics of surfactant-covered drops. *Journal of Computational Physics*.

Steinbach, O. (2007). *Numerical approximation methods for elliptic boundary value problems: finite and boundary elements*. Springer Science & Business Media.

Sundar, H., Malhotra, D., and Biros, G. (2013). Hyksort: A new variant of hypercube quicksort on distributed memory architectures. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 293–302, New York, NY, USA. ACM.

Taus, M., Rodin, G. J., and Hughes, T. J. (2016). Isogeometric analysis of boundary integral equations: High-order collocation methods for the singular and hyper-singular equations. *Mathematical Models and Methods in Applied Sciences*, 26(08):1447–1480.

Tlupova, S. and Beale, J. T. (2019). Regularized single and double layer integrals in 3D Stokes flow. *Journal of Computational Physics*.

Trefethen, L. N. (2008). Is Gauss quadrature better than Clenshaw–Curtis? *SIAM review*, 50(1):67–87.

Trefethen, L. N. (2013). *Approximation theory and approximation practice*, volume 128. Siam.

Trefethen, L. N. and Weideman, J. (1991). Two results on polynomial interpolation in equally spaced points. *Journal of Approximation Theory*, 65(3):247–260.

Uzoigwe, C. (2006). The human erythrocyte has developed the biconcave disc shape to optimise the flow properties of the blood in the large vessels. *Medical hypotheses*, 67(5):1159–1163.

Veerapaneni, S. K., Gueyffier, D., Biros, G., and Zorin, D. (2009a). A numerical method for simulating the dynamics of 3D axisymmetric vesicles suspended in viscous flows. *Journal of Computational Physics*, 228(19):7233–7249.

Veerapaneni, S. K., Gueyffier, D., Zorin, D., and Biros, G. (2009b). A boundary integral method for simulating the dynamics of inextensible vesicles suspended in a viscous fluid in 2D. *Journal of Computational Physics*, 228(7):2334–2353.

Veerapaneni, S. K., Rahimian, A., Biros, G., and Zorin, D. (2011). A fast algorithm for simulating vesicle flows in three dimensions. *Journal of Computational Physics*, 230(14):5610–5634.

Vigmond, E. J., Clements, C., McQueen, D. M., and Peskin, C. S. (2008). Effect of bundle branch block on cardiac output: a whole heart simulation study. *Progress in biophysics and molecular biology*, 97(2-3):520–542.

Wala, M. and Klöckner, A. (2018a). A fast algorithm with error bounds for quadrature by expansion. *arXiv preprint arXiv:1801.04070*.

Wala, M. and Klöckner, A. (2018b). A fast algorithm with error bounds for Quadrature by Expansion. *Journal of Computational Physics*, 374:135–162.

Wala, M. and Klöckner, A. (2018c). Optimization of fast algorithms for global quadrature by expansion using target-specific expansions. *arXiv preprint arXiv:1811.01110*.

Wala, M. and Klöckner, A. (2019a). A fast algorithm for Quadrature by Expansion in three dimensions. *Journal of Computational Physics*, 388:655–689.

Wala, M. and Klöckner, A. (2019b). Optimization of fast algorithms for global Quadrature by Expansion using target-specific expansions. *Journal of Computational Physics*, page 108976.

Wala, M. and Klöckner, A. (2020). On the approximation of local expansions of laplace potentials by the fast multipole method. *arXiv preprint arXiv:2008.00653*.

Wang, B., Ferguson, Z., Schneider, T., Jiang, X., Attene, M., and Panozzo, D. (2021). A large scale benchmark and an inclusion-based algorithm for continuous collision detection. *ACM Transactions on Graphics*.

Wang, W., Diacovo, T. G., Chen, J., Freund, J. B., and King, M. R. (2013). Simulation of platelet, thrombus and erythrocyte hydrodynamic interactions in a 3d arteriole with in vivo comparison. *PLoS One*, 8(10):e76949.

Webb, M., Trefethen, L. N., and Gonnet, P. (2012). Stability of barycentric interpolation formulas for extrapolation. *SIAM Journal on Scientific Computing*, 34(6):A3009–A3015.

Wu, B., Zhu, H., Barnett, A., and Veerapaneni, S. (2020). Solution of stokes flow in complex nonsmooth 2d geometries via a linear-scaling high-order adaptive integral equation scheme. *Journal of Computational Physics*, page 109361.

Xiao, H. and Gimbutas, Z. (2010). A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions. *Computers & mathematics with applications*, 59(2):663–676.

Xu, D., Kaliviotis, E., Munjiza, A., Avital, E., Ji, C., and Williams, J. (2013). Large scale simulation of red blood cell aggregation in shear flows. *Journal of Biomechanics*, 46(11):1810–1817.

Yan, W., Zhang, H., and Shelley, M. J. (2019). Computing collision stress in assemblies of active spherocylinders: Applications of a fast and generic geometric method. *The Journal of chemical physics*, 150(6):064109.

Ye, T., Peng, L., and Li, Y. (2018). Three-dimensional motion and deformation of a red blood cell in bifurcated microvessels. *Journal of Applied Physics*, 123(6):064701.

Ye, T., Phan-Thien, N., and Lim, C. T. (2016). Particle-based simulations of red blood cells—a review. *Journal of biomechanics*, 49(11):2255–2266.

Ye, T., Phan-Thien, N., Lim, C. T., Peng, L., and Shi, H. (2017). Hybrid smoothed dissipative particle dynamics and immersed boundary method for simulation of red blood cells in flows. *Physical Review E*, 95(6):063314.

Ying, L., Biros, G., and Zorin, D. (2004). A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626.

Ying, L., Biros, G., and Zorin, D. (2006a). A high-order 3D boundary integral equation solver for elliptic PDEs in smooth domains. *Journal of Computational Physics*, 219(1):247–275.

Ying, L., Biros, G., and Zorin, D. (2006b). A high-order 3D boundary integral equation solver for elliptic PDEs in smooth domains. *Journal of Computational Physics*, 219(1):247–275.

Ying, L. and Zorin, D. (2004). A simple manifold-based construction of surfaces of arbitrary smoothness. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 271–275. ACM.

Yu, C.-H., Langowitz, N., Wu, H.-Y., Farhadifar, R., Brugues, J., Yoo, T. Y., and Needleman, D. (2014). Measuring microtubule polarity in spindles with second-harmonic generation. *Biophysical journal*, 106(8):1578–1587.

Zechner, J., Marussig, B., Beer, G., and Fries, T.-P. (2016). The isogeometric Nyström method. *Computer methods in applied mechanics and engineering*, 308:212–237.

Zhang, Z.-W., Cheng, J., Xu, F., Chen, Y.-E., Du, J.-B., Yuan, M., Zhu, F., Xu, X.-C., and Yuan, S. (2011). Red blood cell extrudes nucleus and mitochondria against oxidative stress. *IUBMB life*, 63(7):560–565.

Zhao, H., Isfahani, A. H., Olson, L. N., and Freund, J. B. (2010). A spectral boundary integral method for flowing blood cells. *Journal of Computational Physics*, 229(10):3726–3744.

Zhong, M.-C., Wei, X.-B., Zhou, J.-H., Wang, Z.-Q., and Li, Y.-M. (2013). Trapping red blood cells in living animals using optical tweezers. *Nature communications*, 4(1):1–7.