Computer Methods in Applied Mechanics and Engineering 419 (2024) 116686

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Optimal surrogate boundary selection and scalability studies for the shifted boundary method on octree meshes

Cheng-Hau Yang [a,1], Kumar Saurabh [a,1], Guglielmo Scovazzi [b], Claudio Canuto [c], Adarsh Krishnamurthy [a,*], Baskar Ganapathysubramanian [a,*]

[a] Department of Mechanical Engineering, Iowa State University, Ames, IA, USA
[b] Department of Civil and Environmental Engineering, Duke University, Durham, NC 27708, USA
[c] Dipartimento di Scienze Matematiche, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

ARTICLE INFO

Keywords:
Immersed boundary method
Incomplete octree
Optimal surrogate boundary
Massively parallel algorithm

ABSTRACT

The accurate and efficient simulation of Partial Differential Equations (PDEs) in and around arbitrarily defined geometries is critical for many application domains. Immersed boundary methods (IBMs) alleviate the usually laborious and time-consuming process of creating body-fitted meshes around complex geometry models (described by CAD or other representations, e.g., STL, point clouds), especially when high levels of mesh adaptivity are required. In this work, we advance the field of IBM in the context of the recently developed Shifted Boundary Method (SBM). In the SBM, the location where boundary conditions are enforced is shifted from the actual boundary of the immersed object to a nearby surrogate boundary, and boundary conditions are corrected utilizing Taylor expansions. This approach allows choosing surrogate boundaries that conform to a Cartesian mesh without losing accuracy or stability. Our contributions in this work are as follows: (a) we show that the SBM numerical error can be greatly reduced by an optimal choice of the surrogate boundary, (b) we mathematically prove the optimal convergence of the SBM for this optimal choice of the surrogate boundary, (c) we deploy the SBM on massively parallel octree meshes, including algorithmic advances to handle incomplete octrees, and (d) we showcase the applicability of these approaches with a wide variety of simulations involving complex shapes, sharp corners, and different topologies. Specific emphasis is given to Poisson's equation and the linear elasticity equations.

1. Introduction

Accurate numerical solution of PDEs in and around complex objects has a significant impact on various problems in science and technology. Examples include structural analysis of complex architectures, thermal analysis over complex geometries in semiconductor electronics, and flow analysis over complex geometries in aerodynamics. Standard numerical approaches for solving these PDEs on complex geometries—finite difference method (FDM), finite element method (FEM), or finite volume method (FVM)–usually rely on the generation of body-fitted meshes. This is a major bottleneck, as creating an analysis-suitable body-fitted mesh with appropriate refinement around the complex geometry is usually time-consuming and labor-intensive. This issue is exacerbated

* Corresponding authors.
E-mail addresses: chenghau@iastate.edu (C.-H. Yang), maksbh@iastate.edu (K. Saurabh), guglielmo.scovazzi@duke.edu (G. Scovazzi), claudio.canuto@polito.it (C. Canuto), adarsh@iastate.edu (A. Krishnamurthy), baskarg@iastate.edu (B. Ganapathysubramanian).
[1] These author contributed equally.

https://doi.org/10.1016/j.cma.2023.116686
Received 1 September 2023; Received in revised form 17 November 2023; Accepted 5 December 2023
Available online 20 December 2023

in problems involving moving bodies or multiphysics couplings, for which deforming meshes or re-meshing is often required (sometimes at every time step).

Immersed boundary methods (IBM) alleviate the requirement of body-fitted meshes by relaxing the requirement that the mesh conforms to the object [1,2]. IBM allowed scalable mesh generation, such as a Cartesian grid or tree-based approaches (quadtree/octree), to be deployed for simulating PDEs in and around complex objects. In this work, we concentrate on IBM in the context of FEM-based discretizations. Two main flavors of IBMs exist in this FEM context: immersogeometric analysis (IMGA, an acronym that will also refer, in what follows, to cutFEMs, the Finite Cell Method, and related approaches) and the Shifted Boundary Method (SBM).

In immersogeometric analysis (IMGA), the boundary representation of the body (B-rep, NURBS, or STL) is immersed into a non-body-fitted spatial discretization. The Dirichlet boundary conditions are enforced weakly on the immersed boundary surfaces using Nitsche's method, which proved a flexible, robust and consistent approach. Interested readers are referred to [3–20] for a detailed discussion of the mathematical formulation and practical deployment of the IMGA. The IMGA has been deployed to solve several industrial-scale complex problems [7], but suffers from the following drawbacks:

- **Sliver cut-cells**: The presence of sliver cut-cells (i.e., elements intersected by the object boundary that contain a very small volume of the object) may significantly deteriorate the conditioning of the algebraic system of equations. Literature suggests removing these so-formed sliver cut cells from the global assembly can prevent such deterioration in conditioning. However, this comes at the cost of accuracy. Alternatively, there have been studies demonstrating the design of preconditioners to alleviate this issue [5]. But, this has been limited to simpler operators such as Poisson's and Stokes and requires the development of preconditioners for other PDEs. Sliver-cut cells can even produce a loss of numerical stability [14,15].
- **Load balancing**: The accuracy of the IMGA is strongly contingent upon the accurate integration of cut cells. Accurate integration is performed by increasing the number of quadrature points in the cut elements. However, this leads to the issue of load balancing when performing parallel (distributed memory) simulations, as different elements end up having different amounts of computations. Furthermore, it also invalidates the tensor structure of the basis function that can be exploited to optimize the matrix and vector assembly [7,21].

The Shifted Boundary Method (SBM) [22–30] alleviates the aforementioned IMGA issues. The central idea of SBM is to impose the boundary conditions *not on the true boundary* (Γ, see Fig. 1) *but rather on a surrogate boundary in proximity of the true boundary* ($\tilde{\Gamma}_h$, see again Fig. 1). The appropriate value of the applied boundary condition is determined by performing a Taylor series expansion. The surrogate boundary and associated shifted boundary conditions essentially transform the problem of solving the PDE in the complex original domain (denoted as Ω) into a body-fitted problem in the surrogate domain (denoted as $\tilde{\Omega}_h$). This strategy overcomes the challenges associated with IMGA approaches. The SBM differs from the IMGA in the following aspects:

- In IMGA, the volume integration is performed over Ω; whereas in SBM it is performed over $\tilde{\Omega}_h$. Therefore, IMGA requires a classification test (to classify if a Gauss quadrature point belongs to Ω or $\neg\Omega$) for each Gauss quadrature point in the cut elements. In contrast, SBM does not require any such test. The integration for SBM is done over all Gauss points that belong to elements within the surrogate domain $\tilde{\Omega}_h$.
- The integration over all Gauss points in SBM eliminates the poor conditioning of discrete operators due to the sliver cut cells arising in IMGA.
- Additionally, SBM requires no adaptive quadrature for maintaining accuracy. This obviates the need for special algorithmic treatments (like weighted partitioning [7]) to ensure load balancing. In addition, the tensor nature of the basis function is retained, which can be leveraged for performance enhancement using fast vector–matrix assembly.

SBM, therefore, appears to be a promising numerical method for solving PDEs over complex domains. In this work, we seek to address some of the relevant questions for the practical adoption of SBM – especially for simulations over complex CAD geometry domains – that are important and yet somewhat missing from the existing literature. Specifically, we address the following:

1. We extend the numerical analysis of SBM to cover cases when the true domain is a subset of the surrogate domain ($\Omega \subset \tilde{\Omega}_h$).
2. Different (cartesian aligned) surrogate boundaries can be constructed for a complex domain. However, some of these boundaries can be invalid, leading to disconnected surrogate domains. In this work, we codify the requirements for the set of edges/faces (in two/three dimensions) to form a valid surrogate boundary.
3. Among these possible candidate surrogate domains, we identify the optimal surrogate domain, with boundary $\tilde{\Gamma}_h$, that exhibits the best accuracy. We define a simple, scalable strategy to identify this optimal surrogate boundary.
4. We develop the data structures and algorithms required for the scalable deployment of SBM on adaptive, incomplete octree grids. We illustrate good scaling behavior of the framework and showcase the utility of the framework by simulating a wide variety of complex three-dimensional shapes.

The present work focuses on a particular class of PDEs, namely elliptic PDEs with applications involving diffusion (Poisson's equation) and structural mechanics (linear elasticity) problems. The remaining paper is organized as follows: In Section 2, we describe the mathematics of the SBM along with a description of the surrogate boundary. In Section 3 we outline the definition, approach, and algorithms for identifying the optimal surrogate boundary. In Section 4 we provide the details of the algorithms for the scalable deployment of SBM. In Section 5, we illustrate this framework with extensive numerical examples in two and three dimensions. We summarize conclusions in Section 6.

2. Mathematical formulations

2.1. The immersed variational formulation over the physical domain

Consider the non-homogeneous elliptic equation

$$\begin{aligned} -\Delta u &= f \quad \text{on} \quad \Omega\,, \\ u &= u_D \quad \text{on} \quad \Gamma_D\,, \end{aligned} \tag{1}$$

where we are interested in solving for the scalar field, u, over the (immersed) domain of interest Ω with boundary $\partial\Omega = \Gamma_D$. Defining the appropriate functional spaces for test and trial functions, the weak formulation for Poisson's problem can be written as:

$$(\nabla u_h, \nabla w_h)_{\Omega_h} = (f, w_h)_{\Omega_h} + \underbrace{\langle \nabla u_h \cdot \boldsymbol{n}\,,\, w_h \rangle_{\Gamma_{D,h}}}_{\text{Consistency term}} + \underbrace{\langle u_h - u_D\,,\, \nabla w_h \cdot \boldsymbol{n} \rangle_{\Gamma_{D,h}}}_{\text{Adjoint consistency term}} - \underbrace{\langle \alpha\, h^{-1}\,(u_h - u_D)\,,\, w_h \rangle_{\Gamma_{D,h}}}_{\text{Penalty term}}, \tag{2}$$

where α is the penalty parameter for the Dirichlet boundary condition of the Poisson's equation, $\boldsymbol{n}$ indicates the unit outward-pointing normal to the Γ, and h is the element size.

The last three terms in Eq. (2)—consistency, adjoint consistency, and penalty terms are the result of weakly applying the Dirichlet boundary condition as a surface integral. These extra terms result in surface integration over the true geometry, assuming that the finite element interpolation space can describe it exactly (otherwise a geometric discretization error may be introduced).

In addition to the scalar elliptic equation (Poisson equation), we also consider the equations of linear elasticity. Here, we are interested in solving for the displacement vector field, $\boldsymbol{u}$. There are three essential equations for static linear elasticity. First, the equilibrium equation (and associated boundary conditions):

$$\begin{aligned} \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{b} &= 0, \quad \text{on} \quad \Omega\,, \\ \boldsymbol{u} &= \boldsymbol{u}_D \quad \text{on} \quad \Gamma_D\,, \end{aligned} \tag{3}$$

where $\boldsymbol{\sigma}$ is the stress tensor, $\boldsymbol{b}$ is the body force, and again $\partial\Omega = \Gamma_D$. Second, the kinematics equation:

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) = \nabla^s \boldsymbol{u} = \frac{1}{2}(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T)\,, \tag{4}$$

where $\boldsymbol{\varepsilon}(\boldsymbol{u})$ is the strain tensor and $\boldsymbol{u}$ is the displacement vector. Third, the constitutive equation:

$$\boldsymbol{\sigma} = \boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u})\,, \tag{5}$$

where $\boldsymbol{C}$ is the elastic stiffness tensor. For isotropic materials, $\boldsymbol{C}$ can be written as a combination of Young's modulus E and Poisson's ratio ν. Integrating by parts and using Nitsche's method to weakly enforce the Dirichlet boundary conditions, the variational form of linear elasticity can be stated as:

$$(\boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u})\,,\, \nabla^s \boldsymbol{w}_h)_{\Omega_h} = (\boldsymbol{b}\,,\, \boldsymbol{w}_h)_{\Omega_h} + \underbrace{\langle (\boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u})) \cdot \boldsymbol{n}\,,\, \boldsymbol{w}_h \rangle_{\Gamma_{D,h}}}_{\text{Consistency term}} - \underbrace{\langle \boldsymbol{u}_h - \boldsymbol{u}_D\,,\, (\boldsymbol{C}\nabla^s \boldsymbol{w}_h) \cdot \boldsymbol{n} \rangle_{\Gamma_{D,h}}}_{\text{Adjoint consistency term}} - \underbrace{\langle \gamma\, h^{-1}\,(\boldsymbol{u}_h - \boldsymbol{u}_D)\,,\, \boldsymbol{w}_h \rangle_{\Gamma_{D,h}}}_{\text{Penalty term}}\,, \tag{6}$$

where the γ is the penalty parameter for the Dirichlet boundary condition of the linear elasticity, and h is the element size. The appropriate function spaces are used for the solution $\boldsymbol{u}$ and test function $\boldsymbol{w}$.

2.2. The variational formulation for the shifted boundary method over the surrogate domain

The SBM introduced in Main and Scovazzi [22] discretizes the governing equations on a surrogate domain $\tilde{\Omega}_h$ of boundary $\tilde{\Gamma}_h$ (rather than Ω and Γ), where $\tilde{\Omega}_h$ and $\tilde{\Gamma}_h$ do not contain any cut elements or cut element sides, respectively. For example, looking at the sketches in Fig. 1, Ω is enclosed by Γ (the black curve), while $\tilde{\Omega}_h$ is enclosed by $\tilde{\Gamma}_h$ (the red segmented curve). The SBM resorts to a Taylor expansion of the solution variable at the surrogate boundary to *shift* the value of the boundary condition from Γ to $\tilde{\Gamma}_h$. It is important to note that the choices of $\tilde{\Omega}_h$ and $\tilde{\Gamma}_h$ are not independent but must satisfy certain constraints, later discussed in Section 3.1. Enforcing the Dirichlet boundary condition $u = u_D$ on Γ_D through the SBM, we deduce the following Galerkin discretization of the Poisson equation as shown below. Here, V_h represents the appropriate function space, and subscript h represents the finite dimensional analogue of operators/domains after discretization with a tessellation of size h.

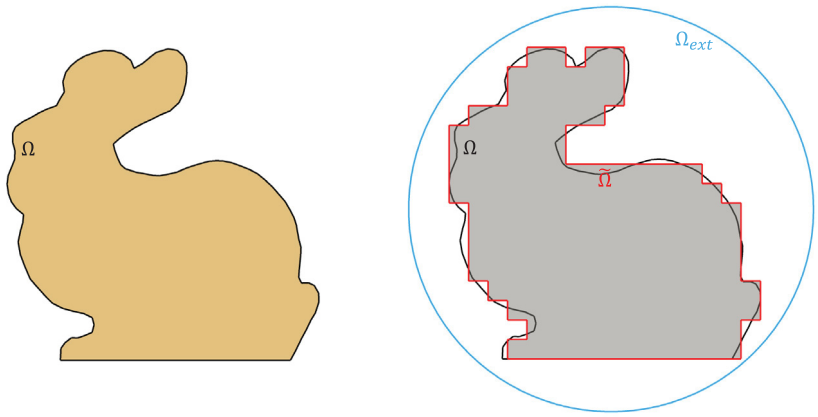
Find $u_h \in V_h(\tilde{\Omega}_h)$ such that, $\forall w_h \in V_h(\tilde{\Omega}_h)$

$$(\nabla u_h\,,\, \nabla w_h)_{\tilde{\Omega}_h} = (f\,,\, w_h)_{\tilde{\Omega}_h} + \underbrace{\langle \nabla u_h \cdot \tilde{\boldsymbol{n}}\,,\, w_h \rangle_{\tilde{\Gamma}_{D,h}}}_{\text{Consistency term}} + \underbrace{\langle \mathrm{S}_h u_h - u_D\,,\, \nabla w_h \cdot \tilde{\boldsymbol{n}} \rangle_{\tilde{\Gamma}_{D,h}}}_{\text{Adjoint consistency term}} \tag{7}$$

$$- \underbrace{\langle \alpha\, h^{-1}\,(\mathrm{S}_h u_h - u_D)\,,\, \mathrm{S}_h w_h \rangle_{\tilde{\Gamma}_{D,h}}}_{\text{Penalty term}}\,, \tag{8}$$

where $\tilde{\boldsymbol{n}}$ indicates the unit outward-pointing normal to the $\tilde{\Gamma}_h$, and $\mathrm{S}_h v$ is the boundary shift operator:

$$\mathrm{S}_h v := v + \nabla v \cdot \boldsymbol{d}\,, \qquad \text{on } \tilde{\Gamma}_{D,h}\,, \tag{9}$$



(a) Ω and $\Gamma = \partial\Omega$ **(b)** Ω, $\tilde{\Omega}_h$, and Ω_{ext}

Fig. 1. Domain definitions for the SBM numerical analysis. The domains are classified into three types, with a corresponding color scheme: (a) The physical (or true) domain Ω (■), enclosed by the physical (or true) boundary Γ (—); (b) The surrogate domain $\tilde{\Omega}_h$ (■), enclosed by the surrogate boundary $\tilde{\Gamma}_h$ (—); (c) The extended domain Ω_{ext}, enclosed by the blue circle (—). The extended domain $\Omega_{\text{ext}} \supset \Omega$ will only be used in mathematical proofs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where $\boldsymbol{d}$ is the distance function from Gauss points on surrogate boundary ($\tilde{\Gamma}_h$) to the true boundary (Γ).

Similarly, the SBM Galerkin discretization for static linear elasticity with Dirichlet boundary condition $\boldsymbol{u} = \boldsymbol{u}_D$ on Γ_D can be stated as:

$$\begin{aligned}
&\text{Find } \boldsymbol{u}_h \in V_h(\tilde{\Omega}_h) \text{ such that, } \forall \boldsymbol{w}_h \in V_h(\tilde{\Omega}_h) \\
&(\boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u}),\, \nabla^s \boldsymbol{w}_h)_{\tilde{\Omega}_h} = (\boldsymbol{b},\, \boldsymbol{w}_h)_{\tilde{\Omega}_h} + \underbrace{\langle (\boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u}))\cdot \tilde{\boldsymbol{n}},\, \boldsymbol{w}_h\rangle_{\tilde{\Gamma}_{D,h}}}_{\text{Consistency term}} - \underbrace{\langle \mathbf{S}_h \boldsymbol{u}_h - \boldsymbol{u}_D,\, (\boldsymbol{C}\nabla^s \boldsymbol{w}_h)\cdot \tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_{D,h}}}_{\text{Adjoint Consistency term}} \\
&\qquad - \underbrace{\langle \gamma\, h^{-1}\, (\mathbf{S}_h \boldsymbol{u}_h - \boldsymbol{u}_D),\, \mathbf{S}_h \boldsymbol{w}_h\rangle_{\tilde{\Gamma}_{D,h}}}_{\text{Penalty term}}.
\end{aligned} \tag{10}$$

The adjoint consistency and penalty terms in Eqs. (7) and (10) are modified in the SBM formulation using Taylor expansions. We refer the interested reader to the detailed derivation of the formulation by Atallah et al. [26].

Remark. We limit ourselves to problems with the Dirichlet boundary condition. However, it is conceptually straightforward to extend the formulation of SBM for Neumann boundary conditions. Suppose our flux boundary condition is $-\frac{\partial u}{\partial n} = t_N$ on Γ_N, the weak form of SBM for the Poisson equation is:

$$\begin{aligned}
&\text{Find } u_h \in V_h(\tilde{\Omega}_h) \text{ such that, } \forall w_h \in V_h(\tilde{\Omega}_h) \\
&(\nabla u_h,\, \nabla w_h)_{\tilde{\Omega}_h} = (f,\, w_h)_{\tilde{\Omega}_h} \\
&\qquad + \langle (\tilde{\boldsymbol{n}}\cdot \boldsymbol{n})(-t_N - \nabla u_h \cdot \boldsymbol{n}), w_h\rangle_{\tilde{\Gamma}_{N,h}} + \langle \nabla u_h \cdot \tilde{\boldsymbol{n}}, w_h\rangle_{\tilde{\Gamma}_{N,h}}.
\end{aligned} \tag{11}$$

Considering the use of linear basis functions for our FEM, we do not require any shift operator (S_h) as in the case of the Dirichlet boundary condition. The primary distinction between directly enforcing Neumann BC and using the Shifted Boundary Method with Neumann BC lies in the area correction term ($\tilde{\boldsymbol{n}}\cdot\boldsymbol{n}$).

Similarly, the SBM Galerkin discretization for static linear elasticity with Neumann (traction) boundary condition $\boldsymbol{\sigma}\boldsymbol{n} = \boldsymbol{t}_N$ on Γ_N can be written as:

$$\begin{aligned}
&\text{Find } \boldsymbol{u}_h \in V_h(\tilde{\Omega}_h) \text{ such that, } \forall \boldsymbol{w}_h \in V_h(\tilde{\Omega}_h) \\
&(\boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u}_h),\, \nabla^s \boldsymbol{w}_h)_{\tilde{\Omega}_h} = (\boldsymbol{b},\, \boldsymbol{w}_h)_{\tilde{\Omega}_h} \\
&\qquad + \langle (\tilde{\boldsymbol{n}}\cdot\boldsymbol{n})(\boldsymbol{t}_N - \boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u}_h)\boldsymbol{n}), \boldsymbol{w}_h\rangle_{\tilde{\Gamma}_{N,h}} \\
&\qquad + \langle \boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u}_h)\tilde{\boldsymbol{n}}, \boldsymbol{w}_h\rangle_{\tilde{\Gamma}_{N,h}}.
\end{aligned} \tag{12}$$

2.3. Numerical analysis of the shifted boundary method over extended surrogate domains

In order to identify the surrogate domain $\tilde{\Omega}_h$ that leads to the most accurate results, we need to first understand the behavior of the SBM approximation when the surrogate domain extends beyond the physical domain Ω, as shown, for example, in the sketch on the right of Fig. 1. As a starting point in the numerical analysis, we will need a number of definitions and assumptions.

The true domain Ω is assumed to have Lipschitz boundary $\Gamma = \partial\Omega$. The surrogate domain $\tilde{\Omega}_h$ – in contrast with previous versions of the SBM – is not necessarily contained in Ω, but may include elements that are cut by Γ (called intercepted elements in the sequel). Its boundary is indicated by $\tilde{\Gamma}_h$.

We then introduce two collections of elements: (a) the collection, $\tilde{\mathscr{T}}_h$, of all the elements T of the grid that are contained in $\tilde{\Omega}_h$; and (b) the collection, $\hat{\mathscr{T}}_h$, of all the elements T of the grid that are contained in $\hat{\Omega}_h$, where $\hat{\Omega}_h$ is the union of the elements cut by Ω or strictly contained in Ω. Hence, $\tilde{\Omega}_h \subseteq \hat{\Omega}_h$, but it is not necessarily true that $\tilde{\Omega}_h \equiv \hat{\Omega}_h$. Here, $\hat{\Omega}_h$ can be thought of as the circumscribing cartesian mesh of Ω. We next define a domain Ω_{ext} *with smooth boundary* and such that $\text{cl}(\hat{\Omega}_h) \subset \Omega_{\text{ext}}$, where $\text{cl}(\hat{\Omega}_h)$ indicates the *closure* of $\hat{\Omega}_h$. Observe that $\hat{\Omega}_h$ and Ω_{ext} are needed only in the mathematical analysis and are not needed in computations. For simplicity, the mathematical analysis will be developed only in the case of the Poisson problem, but conclusions similar to the ones outlined in what follows can be applied to the elasticity equations.

Consider the Poisson problem with non-homogeneous Dirichlet boundary conditions, that is, the problem of finding a $u \in H^1(\Omega)$ that solves Eq. (1) for a given $f \in L^2(\Omega)$. We assume that either f is defined directly over Ω_{ext} or that we can construct a linear continuous extension operator $E : L^2(\Omega) \to L^2(\Omega_{\text{ext}})$ such that $Ef_{|\Omega} = f$ and $\|Ef\|_{L^2(\Omega_{\text{ext}})} \leq C\,\|f\|_{L^2(\Omega)}$, for any $f \in L^2(\Omega)$. For example, f can be extended by zero outside Ω, but we use more advanced prolongation strategies in the numerical experiments. We denote by $\bar{f} = Ef$ the extension of f that we choose, and our goal is now to extend u to $\bar{u}$ in Ω_{ext}. The following result holds, the proof of which is provided in Appendix:

Proposition 1. *There exists an extension $\bar{u}$ of u in Ω_{ext}, such that:*

(a) $-\Delta\bar{u} = \bar{f}$ in Ω_{ext}; and
(b) if $u \in H^2(\Omega)$, then $\bar{u} \in H^2(\tilde{\Omega}_h)$, with $\|\bar{u}\|_{H^2(\tilde{\Omega}_h)} \leq C\,\|u\|_{H^2(\Omega)}$.

The importance of having extensions $\bar{u}$ and $\bar{f}$ of u and f that satisfy conditions (a) and (b) above is needed when studying the convergence of the SBM for a surrogate domain $\tilde{\Omega}_h$ that is not completely contained in the physical domain Ω. Observe that the numerical stability of the SBM is not affected by the particular choice of surrogate domain, as long as $\boldsymbol{d}$ goes to zero as the grid size h is refined. We state then the following result without proof, since the derivations will not differ from the ones already found in the existing literature on SBM [25].

Theorem 1 (*Coercivity*). *Consider the bilinear form $a_h(u_h, w_h)$ defined in (16a) and assume there exist constants $c_d > 0$ and $\zeta > 0$ such that*

$$\|\boldsymbol{d}(\tilde{\boldsymbol{x}})\| \leq c_d\, h_T\, \hat{h}_T^{\zeta} \qquad \forall \tilde{\boldsymbol{x}} \in \tilde{\Gamma}_h \cap T, \quad T \in \tilde{\mathscr{T}}_h\,, \tag{13}$$

where

$$\hat{h}_T = l(\tilde{\Omega}_h)^{-1}\, h_T. \tag{14}$$

Then, if the parameter α is sufficiently large and $\hat{h}_{\tilde{\Gamma}_h}$ sufficiently small, there exists a constant $C_a > 0$ independent of the mesh size, such that

$$a_h(u_h, u_h) \geq C_a\,\|u_h\|_a^2 \qquad \forall u_h \in V_h(\tilde{\Omega}_h)\,, \tag{15}$$

where $\|u_h\|_a^2 = \|\nabla u_h\|^2_{L^2(\tilde{\Omega}_h)} + \|h^{-1/2}\,\mathrm{S}_h u_h\|^2_{L^2(\tilde{\Gamma}_h)}$.

Remark. Condition (13) is just a technical condition for the proofs. In fact, we take $\|\boldsymbol{d}(\tilde{\boldsymbol{x}})\| \sim h_T$ in all computations presented in this work, that is, mesh refinement is obtained by just subdividing every edge of the discretization into two equal-size sub-edges.

The convergence analysis, which follows the same general strategy developed in [25,27,29], needs to be considered with more care. In particular a convergence proof is achieved using Strang's lemma, which in turn requires a result of asymptotic consistency of the SBM. Because most of the derivations are substantially similar to the ones in [25,27,29], we will only focus on the differences, notably the *asymptotic consistency estimate*. In the present case, recasting Eq. (7) as

$$a_h(u_h, w_h) = \ell_h(w_h)\,, \tag{16a}$$

with

$$a_h(u_h, w_h) = (\nabla u_h\,,\, \nabla w_h)_{\tilde{\Omega}_h} - \langle \nabla u_h \cdot \tilde{\boldsymbol{n}}\,,\, w_h\rangle_{\tilde{\Gamma}_h} - \langle \mathrm{S}_h u_h\,,\, \nabla w_h \cdot \tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_h} + \langle \alpha\, h^{-1}\, \mathrm{S}_h u_h\,,\, \mathrm{S}_h w_h\rangle_{\tilde{\Gamma}_h}\,, \tag{16b}$$

$$\ell_h(w_h) = (f\,,\, w_h)_{\tilde{\Omega}_h} - \langle u_D\,,\, \nabla w_h \cdot \tilde{\boldsymbol{n}}\rangle_{\tilde{\Gamma}_h} + \langle \alpha\, h^{-1}\, u_D\,,\, \mathrm{S}_h w_h\rangle_{\tilde{\Gamma}_h}\,, \tag{16c}$$

and replacing in Eq. (16) f with the extension $\bar{f}$ and u_h with the extension $\bar{u}$ of the exact solution u, we have:

$$a_h(\bar{u}, w_h) - \ell_h(w_h) = \underbrace{(-\Delta\bar{u} + \bar{f}}_{\equiv 0}, \nabla w_h)_{\tilde{\Omega}_h} - \underbrace{\langle \mathrm{S}_h\bar{u} - u_D}_{R_h\bar{u}}, \nabla w_h \cdot \tilde{\boldsymbol{n}} - \alpha\, h^{-1}\, \mathrm{S}_h w_h\rangle_{\tilde{\Gamma}_h} ,$$

where $R_h\bar{u} = \mathrm{S}_h\bar{u} - u_D$ denotes the *residual* of the Taylor expansion. From this, using appropriate trace inequalities, we deduce

$$|a_h(\bar{u}, w_h) - \ell_h(w_h)| \leq C\, \|R_h\bar{u}\|_{L^2(\tilde{\Omega}_h)}\, \|w_h\|_{V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)} ,$$

where

$$\|v\|^2_{V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)} = \|v\|^2_a + |hv|^2_{H^2(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)} \tag{17}$$

is the norm associated with the infinite dimensional space

$$V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h) = V_h(\tilde{\Omega}_h) + H^2(\tilde{\Omega}_h) \subset H^2(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h) . \tag{18}$$

Here $V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)$ is an extension of the finite dimensional space $V_h(\tilde{\Omega}_h)$ of globally continuous, piecewise-linear polynomials, which contains the extension of the exact solution $\bar{u}$, that is $\bar{u} \in V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)$. Here $H^2(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h) = \prod_{T\in\tilde{\mathscr{T}}_h} H^2(T)$ with 'broken' norm $\|\cdot\|_{H^2(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)} = \sum_{T\in\tilde{\mathscr{T}}_h} \|\cdot\|_{H^2(T)}$ and 'broken' seminorm $|\cdot|_{H^2(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)} = \sum_{T\in\tilde{\mathscr{T}}_h} |\cdot|_{H^2(T)}$. It is easily checked that the form $a_h(\cdot,\cdot)$ is well-defined also on the space $V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)\times V_h(\tilde{\Omega}_h)$.

If we assume $u \in H^2(\Omega)$, then according to Proposition 1, $\bar{u}$ has regularity H^2 around $\tilde{\Gamma}_h$ and the norm of the reminder $R_h\bar{u}$ can be estimated as in the standard case, in which $\tilde{\Omega}_h \subset \Omega$ (see, e.g., [25]), leading to:

Theorem 2 (*Optimal Convergence in the Natural Norm*)**.** *Assume that $u \in H^2(\Omega)$. Under the assumption of Theorem 1, and the condition that $h_{\tilde{\Gamma}_h}$ is sufficiently small, the numerical solution u_h satisfies the following error estimate:*

$$\|\bar{u} - u_h\|_{V(\tilde{\Omega}_h;\tilde{\mathscr{T}}_h)} \leq C\, h_{\tilde{\Omega}_h}\, \|\nabla(\nabla\bar{u})\|_{L^2(\Omega)} , \tag{19}$$

where $C > 0$ is a constant independent of the mesh size and the solution.

In addition, duality estimates can be derived to show that the $L^2(\tilde{\Omega}_h)$-error of the discrete solution converges with rate 3/2, which suboptimal by an order 1/2. Note however that optimal L^2-error convergence rates have been observed in all computations performed to date with the SBM, for a variety of problems and differential operators. This might indicate that the available L^2-error estimates are not sharp.

3. Optimal surrogate boundary

As already discussed at length, the key aspect of the SBM is the correction of the boundary conditions on the surrogate boundary, obtained by performing a Taylor series expansion. Previous literature [22,31] has shown that convergence in the L^2-norm reduces to first order, when using linear basis functions without this correction, for instance. In this section, we answer the question of constructing the *optimal* surrogate boundary, which gives minimal error while retaining all the expected properties of SBM.

It is rather intuitive to recognize that the surrogate boundary with minimum distance $\boldsymbol{d}$ (in some sense) from the true boundary should be optimal. Using a wide variety of canonical examples exhibiting complex shapes and topology, we show that solving the PDE using an optimal surrogate boundary (i.e., with minimal $\boldsymbol{d}$) can produce significantly more accurate solutions compared to a non-optimal surrogate. Given a background adaptive Cartesian mesh (octree or quadtree), identification of the optimal surrogate can be stated as an optimization problem, where the goal is to minimize the distance between the true boundary and the surrogate boundary (Eq. (20)):

$$\mathrm{argmin}_{\tilde{\Gamma}_h} \|\Gamma - \tilde{\Gamma}_h\| := \mathrm{argmin}_{\tilde{\Gamma}_h} \int_{\tilde{\Gamma}_h} |\boldsymbol{d}\cdot\tilde{\boldsymbol{n}}|\, \mathrm{d}\tilde{\Gamma}_h , \tag{20}$$

which corresponds to the measure of the gap between Γ and $\tilde{\Gamma}_h$. Performing a global optimization on the surrogate boundary is a non-trivial task. We recast this global optimization into a set of element-level optimization as:

$$\mathrm{argmin}_{\tilde{\Gamma}_h} \int_{\tilde{\Gamma}_h} |\boldsymbol{d}\cdot\tilde{\boldsymbol{n}}|\, \mathrm{d}\tilde{\Gamma}_h = \mathrm{argmin}_{\tilde{\Gamma}_h} \left(\sum_{T\in\tilde{\mathscr{T}}_h} \int_{\partial T\cap\tilde{\Gamma}_h} |\boldsymbol{d}\cdot\tilde{\boldsymbol{n}}|\, \mathrm{d}\tilde{\Gamma}_h \right) , \tag{21}$$

where we recall that $\tilde{\mathscr{T}}_h$ is the collection of elements in $\tilde{\Omega}_h$. Converting the global optimization into a set of element-level optimizations is algorithmically useful, both from a complexity standpoint and a communication/data structure standpoint. However, performing the optimization at the local elemental level does not guarantee the satisfaction of constraints of the surrogate boundary (described in detail in Section 3.1). To alleviate this issue, we modify the problem represented by Eq. (20): Instead of asking the question *"how close is the surrogate boundary to the true boundary?"* we ask the question *"how close is the surrogate volume to the true volume?"* Basically we approximate Eq. (20) as:

$$\mathrm{argmin}_{\tilde{\Gamma}_h} \int_{\tilde{\Gamma}_h} |\boldsymbol{d}\cdot\tilde{\boldsymbol{n}}|\, \mathrm{d}\tilde{\Gamma}_h \approx \mathrm{argmin}_{\tilde{\Omega}_h} |(\Omega \setminus \tilde{\Omega}_h) \cup (\tilde{\Omega}_h \setminus \Omega)|. \tag{22}$$

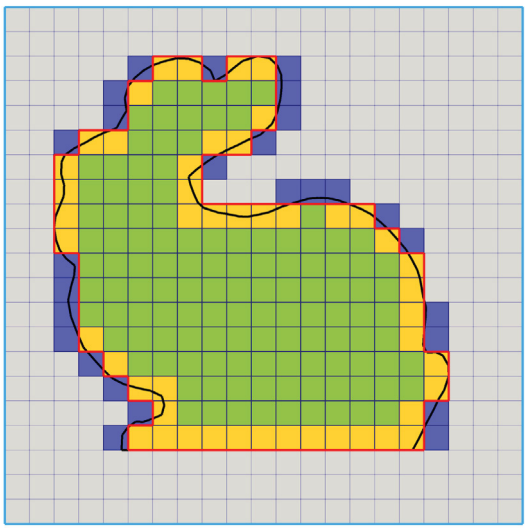


Fig. 2. Color scheme depicting four different types of elements in the SBM algorithms and the four types of associated domains: INTERIOR elements (■), TRUEINTERCEPTED elements (■), FALSEINTERCEPTED elements (■), and EXTERIOR elements (■). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.1. Algorithmic description of the surrogate domain and its boundary

Main and Scovazzi [22] proposed to define the surrogate boundary as the closest projection of the true boundary. However, the surrogate domain ($\tilde{\Omega}_h$) was constructed using only elements that are completely contained in the true domain. We formulate the requirements of the surrogate domain ($\tilde{\Omega}_h$) and surrogate boundary ($\tilde{\Gamma}_h$) more formally, and limit our discussion to quadrilateral/hexahedral elements. This is motivated by the fact that scalable adaptive algorithms exist for creating quad/octree meshes [32–35]. Simulation strategies based on quad/octree have been very successful in modeling complex flow and multi-physics phenomena [7,21,36–41].

We start with some terminology that we will use throughout the manuscript. We encourage the reader to familiarize with Fig. 2 before moving on to the definitions below. The figure illustrates all the domains defined in earlier sections (true, surrogate, circumscribing, extension) and relates them to the corresponding mesh elements. Namely:

- INTERIOR elements (■): the elements whose nodes are inside the physical boundary (Γ).
- EXTERIOR elements (■): the elements whose four nodal points are outside the physical boundary (Γ).
- INTERCEPTED elements (■ ∪ ■): elements whose nodal points are partially within and partially outside of the physical boundary Γ. We further subdivide INTERCEPTED elements into two sub-categories, by means of an optimal strategy that will be presented later in this section:
  - TRUEINTERCEPTED elements (■): the INTERCEPTED elements that are inside the surrogate domain $\tilde{\Omega}_h$ and are part of the SBM calculation.
  - FALSEINTERCEPTED elements (■): the INTERCEPTED elements that are outside the surrogate domain $\tilde{\Omega}_h$ and are not part of the SBM calculation.

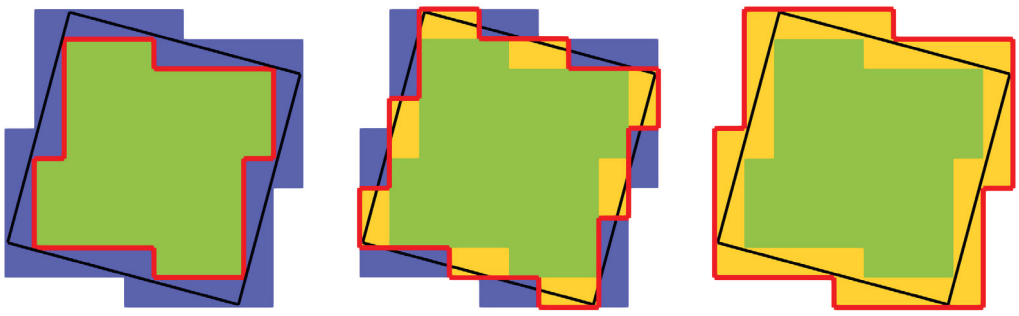
The sketch also shows the three different domains considered in what follows:

- The physical (or true) domain Ω: the domain enclosed by the physical (or true) boundary Γ (—).
- The surrogate domain $\tilde{\Omega}$ = ■ ∪ ■: the domain enclosed by the surrogate boundary $\tilde{\Gamma}$ (—), that is the union of the INTERIOR elements (■) and the TRUEINTERCEPTED elements (■). This is the domain over which the SBM calculations are performed.
- The extended domain Ω_{ext}: the domain enclosed by the blue square (—). This domain contains INTERIOR elements (■), TRUEINTERCEPTED elements (■), FALSEINTERCEPTED elements (■), and EXTERIOR elements (■).

We refer the reader to Section 4, which contains algorithmic details of how to perform the classification of the various element types. We next state formal definitions that allow rigorous algorithmic developments of scalable strategies for constructing these optimal surrogate domains:

Definition 3.1 (*Node*)**.** A node is defined as a point $\vec{x} \in \mathscr{R}^{dim}$, where dim is the domain dimensionality (2, or 3).

Definition 3.2 (*Node Classification*)**.** A node is classified as INTERIOR if it lies within the true domain Ω, otherwise is classified as EXTERIOR



(a) $\lambda = 0$ **(b)** $\lambda = 0.5$ **(c)** $\lambda = 1$

Fig. 3. Surrogate boundary and the marker with varying λ: Figure showing the classification of (a) INTERIOR (■); (b) INTERCEPTED (■); (c) FALSEINTERCEPTED (■) with varying λ. INACTIVE elements do not form the incomplete octree and are not present. Note that (—) indicates the surrogate boundary whereas (—-) denotes the true boundary.

Definition 3.3 (*Element Node Relation*)**.** Each octant or element in the mesh comprises a certain number of nodes. The actual number of nodes that comprise an element depends on the order of the basis function and the dimension and varies as $(p+1)^{dim}$, where p is the basis function order, and dim is the dimensionality.

Definition 3.4 (*Element Classification*)**.** The elements/octants of the octree are categorized into three categories: INTERIOR, EXTERIOR, INTERCEPTED. An element is classified as EXTERIOR if all the nodes of the element are classified as EXTERIOR. Similarly, the element is classified as INTERIOR if all the nodes of the element are INTERIOR. When the nodes of the element have some nodes classified as INTERIOR and some as EXTERIOR, the elements are classified as INTERCEPTED.

Remark. We note that Definition 3.4 is consistent. Given our ability to adaptively refine, we assume that the elements along the domain boundary (i.e. the intersected elements) are sufficiently refined to prevent the pathological case of a boundary that crosses an edge (in 2D) or face (in 3D) of an element multiple times. These are cases that are fundamentally due to under-resolution of the grid with respect to the geometrical details of Γ. These cases can generally be resolved by refining the grids in regions where Γ has high curvature.

Remark. The classification of INTERIOR and EXTERIOR regions depend on the domain of interest for the PDEs. It can be the inside or outside of the enclosed geometry. For instance, if one is interested in the effect of inclusions/voids then the domain of interest is the outside of the geometry defining these voids.

In practical terms, Eq. (22) boils down to looping over all INTERCEPTED elements and deciding whether that INTERCEPTED element should be retained in the surrogate domain ($\tilde{\Omega}_h$). A simple and effective strategy is to retain an INTERCEPTED element if it encloses enough of the true domain. To formalize this, we define an additional classification of an element, FALSEINTERCEPTED.

Definition 3.5 (FALSEINTERCEPTED)**.** An INTERCEPTED element is classified as FALSEINTERCEPTED if the ratio of the element volume exterior to Ω to the total element volume is greater (>) than the threshold factor λ.

We note that the classification of the element as FALSEINTERCEPTED is contingent on the choice of the user-defined parameter λ. When we choose $\lambda = 0$, all the INTERCEPTED elements are classified as FALSEINTERCEPTED, which produces a surrogate domain that fully inscribes (i.e., is inside) the true domain. On the other hand, choosing $\lambda = 1$ leads to the inclusion of all the INTERCEPTED elements producing a surrogate domain that fully circumscribes the true domain. Fig. 3 illustrates various surrogate boundaries as a function of varying λ. Intuitively, $\lambda = 0.5$ produces an optimal surrogate that minimizes Eq. (20).

The *surrogate domain* $\tilde{\Omega}_h$ of size h is defined as a set of elements with element size $\|\Delta \boldsymbol{x}\| \leq h$ such that when any extra element (of size $\|\Delta \boldsymbol{x}\| \leq h$) that belongs to the complement $\tilde{\Omega}_h^c$ of $\tilde{\Omega}_h$ is added, it must be classified as either EXTERIOR or FALSEINTERCEPTED. A surrogate domain can be constructed as circumscribing ($\lambda = 1$) or inscribing ($\lambda = 0$) the true domain, or "something in between" ($\lambda \in (0, 1)$) these two extreme cases. A *surrogate boundary* is the set of faces/edges that traverses the surrogate domain $\tilde{\Omega}_h$. Fig. 3 illustrates a variety of surrogate domains and associated surrogate boundaries for a given geometry. We design algorithms such that the surrogate domain satisfies the following conditions to ensure correct computations:

- **Watertightness**: Ideally, the true boundary must be watertight or 2-manifold as nothing can enter or leave the domain. In practice, however, the SBM approach is robust to small gaps/overlaps.

- **Single-cycle condition**: The set of edges or faces that form the surrogate boundary must form one and exactly one cycle that traverses the surrogate domain $\tilde{\Omega}_h$. In other words, there should not be any self-intersections in the surrogate boundary.

In the next section, we describe the algorithms to construct the surrogate boundary for arbitrary choices of λ in a massively parallel environment.

4. Algorithms and implementation details

4.1. Algorithms

To start the discussion of the algorithms for the efficient and accurate construction of surrogate domain and surrogate boundary for the SBM computations, we clarify some assumptions and motivations behind the choices described in what follows.

4.1.1. Assumptions regarding meshes

Before proceeding to the algorithm sections, we make the following assumption regarding the data structures.

1. **No neighbor information:** We assume that the mesh elements do not have access neighborhood information. Tagging neighbors is particularly challenging with unstructured meshes, as elements can have varying neighbors with no plausible upper limits.
2. **Partitioned from get-go :** The octree-based mesh data structure is partitioned right from the construction stage using distributed memory parallelism. This aspect has made octrees possible to scale to thousands of processors. This is in contrast to the traditional unstructured mesh generation, where the mesh is first generated on a single processor and later partitioned through a graph partitioning library such as ParMetis. This is an important aspect to consider while developing algorithms that retain the scalability of octree meshes.
3. **Massively parallel environment:** The algorithm proposed should scale to thousands of processors. We are not only interested in the accurate solution of PDEs but also in an efficient and scalable solution.
4. **Different element sizes:** Octrees can have different element sizes. We consider 2:1 balanced octrees during our algorithmic development [42]. Additionally, we assume that the Intercepted elements and Interior elements that are neighbors of the Intercepted elements (elements that share at least one node of the Intercepted elements) are at the same level. This is done to ensure that there are no hanging nodes, which retains the simplicity of algorithms without too much extra computational cost.

4.1.2. Algorithm for determining surrogate boundary for arbitrary boundary

With the above assumption in Section 4.1.1, we can define the algorithms for determining the surrogate boundary for any arbitrary choice of λ. The basic idea of the proposed algorithm is to rely on the connection between the elements through the nodes. We note that the nodes are shared across the elements in the Continuous Galerkin (CG) Finite element method. Other researchers have leveraged this to implement several graph-based algorithms for unstructured meshes, even without any neighbor information stored in the mesh data structure [43]. This can be efficiently performed as a series of matvec operations—a key component in FEM libraries and can be performed in a highly efficient and scalable manner [31,32,34,44]. matvec operation typically involves looping over the local elements locally owned by the processors followed by sharing the nodal values at the processor boundaries – typically known as ghost exchange. First, the ghosted values from each of the processors are duplicated (GhostRead) so that each processor can now independently execute matvec operations. Finally, matvec ghosted values are copied to the processor that owns the node at the processor boundaries (GhostWrite). We would refer the interested reader to our prior papers [7,21,33,34,45] for additional details on ghost exchange.

Algorithm 1 briefs the major step required to identify the surrogate boundary. We begin with identifying markers for each element (Algorithm 2). At this stage, each element is classified as Interior, Exterior, or Intercepted. Next, each Intercepted element is classified as FalseIntercepted depending on the value of λ. For accurate evaluation of the volume term within Ω, we use 5^{dim} Gauss–Legendre points; i.e., each element is filled with 5 Gauss points in each dimension. This computational choice works well for all our results but can be easily changed at compile time.

Removing FalseIntercepted elements from the domain requires a change of surrogate boundary. To identify the surrogate boundary, we generate the markers for neighbors of FalseIntercepted. Without the neighbor information within the mesh data

Algorithm 1 SurrogateBoundaryIdentification: Identify the surrogate boundary

Require: Incomplete octree mesh $\mathcal{O}$,Threshold factor λ
Ensure: Surrogate boundary ($\tilde{\Gamma}$)

1: Marker $\mathcal{M}_1 \leftarrow$ GenerateMarkers($\mathcal{O}, \lambda$) ▷ Algorithm 2.
2: Marker $\mathcal{M}_2$, FalseIntercepted Nodes ($\mathcal{N}$) $\leftarrow$ GenerateNeighborsOfFalseIntercepted($\mathcal{O}$,$\mathcal{M}_1$) ▷ Algorithm 3
3: Boundary $\tilde{\Gamma}$, Marker $\mathcal{M}_3 \leftarrow$ GetBoundary($\mathcal{O}, \mathcal{M}_2$) ▷ Algorithm 4
return ($\tilde{\Gamma}$)

Algorithm 2 GenerateMarkers: Generate Marker classification :Exterior, Interior, Intercepted, FalseIntercepted

Require: Octree $\mathscr{O}$,Threshold factor λ
Ensure: Marker M

$\mathscr{M} \leftarrow []$
for element $\in \mathscr{O}$ **do** ▷ Loop over the elements of octree
  count $\leftarrow 0$
  for node $\in$ element **do** ▷ Loop over the nodes of each element
    if node == Interior **then** ▷ Classify nodes as Exterior or Interior
      count++ ▷ Increment for Interior nodes of element
  if count == num_nodes **then**
    $\mathscr{M}$[element] $\leftarrow$ Interior
  else if count == 0 **then**
    $\mathscr{M}$[element] $\leftarrow$ Exterior
  else
    $\mathscr{M}$[element] $\leftarrow$ Intercepted
    count_gp $\leftarrow 0$ ▷ Counter for number of Gauss points that are Interior
    for gp $\in$ GaussPoints **do**
      if gp $\leftarrow$ Interior **then**
        count_gp ++
    $\lambda_c \leftarrow$ count_gp/num_gp ▷ Fraction of Gauss point that are Interior
    if $\lambda_c \geq \lambda$ **then** ▷ Classify elements on the basis of the threshold λ
      $\mathscr{M}$[element] $\leftarrow$ FalseIntercepted
return ($\mathscr{M}$)

Algorithm 3 GenerateNeighborsOfFalseIntercepted: Generate neighbors of FalseIntercepted

Require: Octree $\mathscr{O}$, Element marker $\mathscr{M}$
Ensure: Marker M with element marker for NeighborsFalseIntercepted, Nodal False Intercepted nodes $\mathscr{N}$

$\mathscr{N}$_ghosted $\leftarrow$ [0] ▷ Vector of zeros with size of ghosted nodal vectors
for element $\in \mathscr{O}$ **do**
  if element == FalseIntercepted **then**
    for nodes $\in$ element **do**
      $\mathscr{N}$_ghosted[nodes] $\leftarrow 1$ ▷ Assign value of 1 to all the nodes of the element tagged as FalseIntercepted
$\mathscr{N} \leftarrow$ GhostWrite($\mathscr{N}$_ghosted) ▷ Ghost write
$\mathscr{N}$_ghosted $\leftarrow$ GhostRead($\mathscr{N}$) ▷ Ghost read
for element $\in \mathscr{O}$ and $\mathscr{M} \in$ Interior **do**
  if anyof(nodes) $\in$ element == 1 **then** ▷ If any of the nodes is marked as 1
    $\mathscr{M}$[element] $\leftarrow$ NeighborsFalseIntercepted
return ($\mathscr{M}$,$\mathscr{N}$)

structure, we rely on the efficient matvec computation to achieve this task (Algorithm 3). This step can also be considered a *scatters-to-gather* transformation. Each FalseIntercepted element scatter the information by assigning the value of 1 to the incident nodes on that given element. In the second pass, each element gathers the data by looking into the values of the nodes that are incident on it. Once we have the nodal and elemental information about the NeighborsFalseIntercepted, we can compute the faces that form the new $\tilde{\Gamma}_h$. Two distinct cases exist:

1. If an element is marked as Intercepted, we proceed as usual. The face(s) of the element with all the nodes marked as Exterior is added to the surrogate boundary faces.
2. For the element marked NeighborsFalseIntercepted, we loop through the faces of the element. If all the nodes on a given face are either FalseIntercepted or Exterior, then and only then, they form the part of $\tilde{\Gamma}_h$.

In some cases, we observe a cycle being formed where the opposite faces ($X_i^-, X_i^+, i = 1 \cdots$ dim) of a given element are both chosen to be part of the surrogate boundary. This violates the second condition of the surrogate boundary described in Section 3.1. We mark such elements as FalseIntercepted to resolve this issue. This ensures that only one of the sides X_i^- or X_i^+ is selected to be part of the surrogate boundary depending on the side of NeighborsFalseIntercepted (Algorithm 4). Fig. 3 shows the variation of the surrogate domain and surrogate boundary for different choices of λ. We note that all the steps in Algorithm 1 can be done efficiently in $\mathscr{O}(N)$ steps and require a small number of passes over the elements of the octree.

4.1.3. SBM computation

Once we set up the surrogate domain ($\tilde{\Omega}_h$) and resultant surrogate boundary ($\tilde{\Gamma}_h$) along with the associated markers, we proceed with the steps for the deployment of the SBM. Algorithm 5 briefs the major step for the SBM computation. We note that the elements marked as FalseIntercepted (and Exterior, if not using incomplete octree) are skipped over, whereas the volume integration is performed on other elements. Each face of the Intercepted and NeighborsFalseIntercepted element is checked to see if it belongs to the surrogate boundary $\tilde{\Gamma}_h$. If a given face belongs to $\tilde{\Gamma}_h$, the required surface integration (as given in Eq. (7)) computation is

Algorithm 4 GetBoundary: Get surrogate boundary

Require: Octree $\mathscr{O}$, Element marker $\mathscr{M}$
Ensure: Surrogate boundary $\tilde{\Gamma}$, Marker $\mathscr{M}$

for element $\in \mathscr{O}$ **do**
  FaceBits $\leftarrow$ [false]
  for face $\in$ element **do** ▷ Loop over the faces of the element
    if $\mathscr{M}$[element] == Intercepted and `allof`(nodes $\in$ face == Exterior) **then** ▷ Condition for Interceptedelement
      FaceBits[face] $\leftarrow$ True
    else if $\mathscr{M}$[element] == NeighborsFalseIntercepted **then** ▷ Condition for NeighborsFalseInterceptedelements
      BoundaryFace $\leftarrow$ True
      for nodes $\in$ face **do**
        if (nodes == Exterior) or (nodes == FalseInterceptedNode) **then** ▷ Each node should be either Exterior or FalseIntercepted
          continue
        else
          BoundaryFace $\leftarrow$ False
          break
  if cycle(faceBits) **then** ▷ If opposite face of the same element forms the part of $\tilde{\Gamma}$
    $\mathscr{M}$[element] $\leftarrow$ FalseIntercepted
  else
    for face $\in$ element **do** ▷ Add faces to the surrogate domain $\tilde{\Gamma}$
      if faceBits[face] $\leftarrow$ true **then**
        $\tilde{\Gamma} \leftarrow \tilde{\Gamma}$.`push_back`(face)
return ($\tilde{\Gamma}$,$\mathscr{M}$)

Algorithm 5 SBM: Shifted Boundary method

Require: Octree $\mathscr{O}$, Element marker $\mathscr{M}$,Surrogate boundary $\tilde{\Gamma}$, PDE ($\mathscr{L}(u) = f$)
Ensure: Solution of PDE: u^h

for element $\in \mathscr{O}$ **do**
  if element == FalseIntercepted $\cup$ Exterior **then**
    continue
  else
    Assemble matrix ($\mathscr{A}$) and right hand side vector (b)
    if (element == Intercepted) or (element == NeighborsFalseIntercepted) **then**
      for face $\in$ element **do**
        if face $\in \tilde{\Gamma}$ **then**
          Assemble surface contribution to matrix ($\mathscr{A}$) and right hand side vector (b)
Solve $\mathscr{A}u^h = b$ ▷ Solve system of linear equation
return u^h

Algorithm 6 Overview: Calculation of distance functions

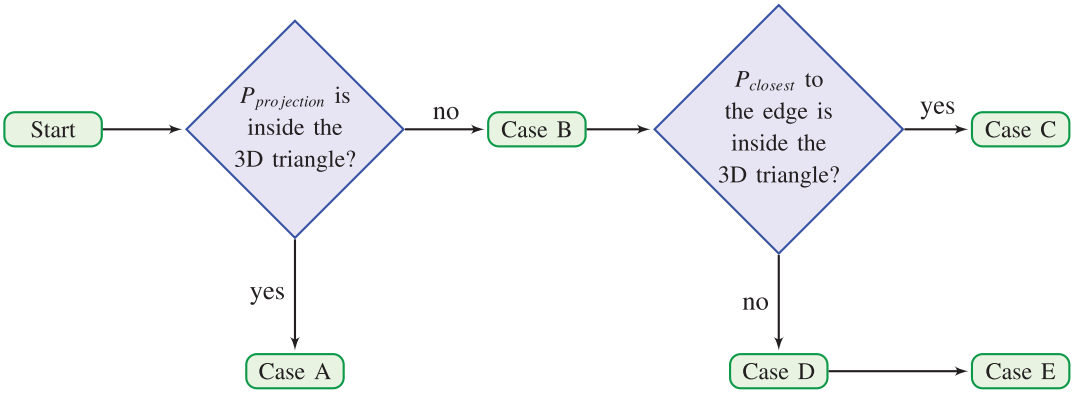
Require: Gauss point positions on the surrogate boundary (GaussPoints), geometry, and mapping between Gauss points and distance functions
Ensure: Distance function corresponding to each Gauss point

for $P \in$ GaussPoints **do**
  if P is already in the mapping **then**
    Use the precomputed distance function for P
  else
    Find the distance function from P to the nearest triangle using NormalDistCalc ▷ Algorithm 7
    Save the mapping between P and its distance function
return distance functions for all Gauss points

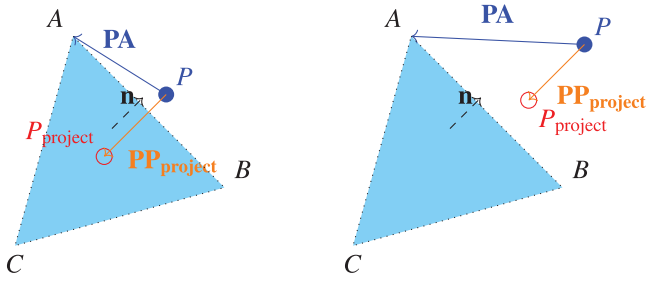
performed and assembled to the global matrix and vector. Once the global matrix $\mathscr{A}$ and global vector b are assembled, we solve the system of equations to obtain the solution u^h.

4.2. Distance function calculation

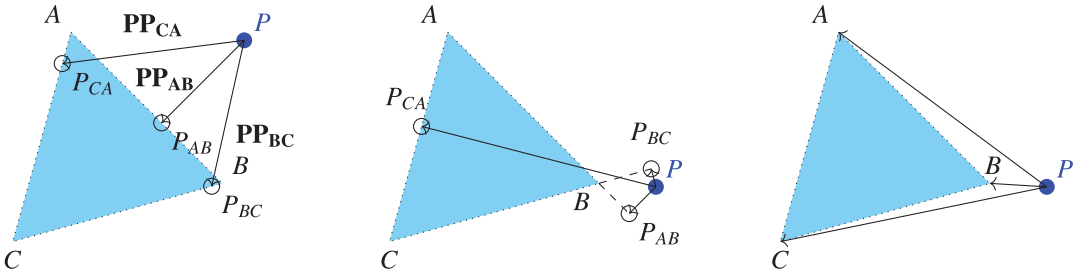
Note that the SBM computations require evaluating the distance, $\boldsymbol{d}$, of Gauss points on the surrogate boundary to the closest point on the true boundary. This section focuses on calculating distance functions for intricate three-dimensional geometries. We consider the geometries to be represented in STL files, which are, in turn, represented by sets of triangles. SBM computation requires computing the distance function by finding the normal distance from the Gauss point to the nearest triangle. We store information about Gauss points and their corresponding distance functions to avoid repeated distance function calculations. This is particularly important for time-dependent problems on the static mesh as it prevents repetitive computation at the cost of extra memory.



(a) Flowchart of distance calculation procedure.



(b) Case A:$P_{projection}$ falls inside the 3D triangle. **(c)** Case B:$P_{projection}$ lies outside the 3D triangle.



(d) Case C:The shortest distance from the point to the triangle edges.

(e) Case D:The shortest distance from the point to the triangle edges, and the closest point ($P_{closest}$) fall outside of the triangle.

(f) Case E:The shortest distance from the point to the triangle vertices.

Fig. 4. Distance function calculation. As shown in the flowchart in (a), the projection of the point into the plane of the triangle can be inside or outside. If it falls outside, the point is projected to the three edges. If the closest of the edge projections is outside the triangle, then the closest point is computed to the vertices.

Algorithm 7 NormalDistCalc: Calculation of normal distance to nearest triangle

Require: k-d tree built from triangle centroids, Gauss point position (P), and geometry information
Ensure: Distance function corresponding to the Gauss point

1: Use the k-d tree to find the ID of the nearest triangle to the Gauss point P
2: Calculate the normal vector from P to the nearest triangle by projecting the vector **PA** onto the triangle's normal vector **n**: $(\frac{\mathbf{PA}\cdot\mathbf{n}}{|\mathbf{n}|^2})\cdot\mathbf{n}$, where A is one vertex position of the triangle
3: project point $\leftarrow P+(\frac{\mathbf{PA}\cdot\mathbf{n}}{|\mathbf{n}|^2})\cdot\mathbf{n}$
4: **if** the projection point is inside the 3D triangle **then** ▷ Algorithm 8
5: Set the distance function as the normal vector to the nearest triangle
6: **else**
7: Calculate the shortest distance from P to the nearest triangle edge as the distance function ▷ Algorithm 9
8: **return** distance function

Algorithm 8 CheckInside3DTriangle: Check if point is inside 3D triangle

Require: Projection point position ($P_{project}$) and vertex positions of triangle (A, B, C)
Ensure: Whether the projection point is inside the 3D triangle

1: Calculate cross products $\mathbf{u} = \mathbf{AB} \times \mathbf{AP}_{\mathbf{project}}$, $\mathbf{v} = \mathbf{BC} \times \mathbf{BP}_{\mathbf{project}}$, $\mathbf{w} = \mathbf{CA} \times \mathbf{CP}_{\mathbf{project}}$
2: **if** $\mathbf{u} \cdot \mathbf{v} < 0$ or $\mathbf{u} \cdot \mathbf{w} < 0$ **then**
3: **return** false
4: **else**
5: **return** true

Algorithm 9 ShortestDist2TriEdge: Calculate the shortest distance to a triangle edge

Require: Gauss point position P, and vertices of a triangle A, B, and C
Ensure: Shortest distance vector from P to a triangle edge

1: Compute the projection of P onto each triangle edge, i.e., $\mathbf{PP}_{\mathbf{AB}} = \frac{\mathbf{AP}\cdot\mathbf{AB}}{|\mathbf{AB}|^2}\mathbf{AB} - \mathbf{AP}$, $\mathbf{PP}_{\mathbf{BC}} = \frac{\mathbf{BP}\cdot\mathbf{BC}}{|\mathbf{BC}|^2}\mathbf{BC} - \mathbf{BP}$, $\mathbf{PP}_{\mathbf{CA}} = \frac{\mathbf{CP}\cdot\mathbf{CA}}{|\mathbf{CA}|^2}\mathbf{CA} - \mathbf{CP}$, where P_{AB}, P_{BC}, P_{CA} denote the closest points on edges $\overline{AB}$, $\overline{BC}$, $\overline{CA}$ to P
2: Compute the minimum distance from P to each edge, i.e., $|\mathbf{PP}_{\mathbf{AB}}|$, $|\mathbf{PP}_{\mathbf{BC}}|$, $|\mathbf{PP}_{\mathbf{CA}}|$ and find the closest point ($P_{closest}$) to P within P_{AB}, P_{BC}, P_{CA}
3: **if** the $P_{closest}$ is inside the 3D triangle **then** ▷ Algorithm 8
4: Set the distance function as the $\mathbf{PP}_{\mathbf{closest}}$
5: **else**
6: Calculate the shortest distance function from P to the nearest triangle vertex, $\mathbf{PP}_{\mathbf{vertex}}$
7: **return** distance function

The procedure for calculating the distance function is outlined in Algorithm 6 and Fig. 4(a). We utilize the algorithm presented in Algorithm 7 to compute the normal distance between Gauss points and the nearest triangle, as depicted in Fig. 4(b). To expedite this process, we leverage a k-d tree to efficiently search the closest triangle using `nanoflann` library [46]. Constructing the k-d tree data structure is not compute-intensive. Even for intricate 3D scenarios (like the 3D Stanford bunny model), the time required for building the k-d tree is $\ll 1$ s (0.01 s). Additionally, we use memory-efficient C++ constructs (reference and pointer) to access the data. In our implementation, every processor undertakes the k-d tree construction independently and in parallel, ensuring this process needs zero communication and, thus, does not become a performance bottleneck.

The distance calculation process is embedded within the vector assembly procedure shown in Section 5.5. Using a k-d tree over the conventional method of looping through the surface triangles in the STL file, we noticed a significant speed enhancement, nearly tenfold, in the 3D Stanford bunny example shown in Section 5.3. This significant boost in efficiency underscores our choice of the k-d tree data structure in this context. Specifically, the k-d tree is constructed from the triangle centroids, and during the Gauss point iteration, the k-d tree assists in identifying the nearest triangle and obtaining its ID for each Gauss point. Nonetheless, in certain instances, the projection points from Gauss points to triangles may fall outside the triangle, as illustrated in Fig. 4(c). To handle such situations, we have incorporated two additional procedures. Firstly, Algorithm 8 verifies whether the projection points lie within the nearest triangle. Secondly, Algorithm 9 determines the shortest distance between the point and the edges of the triangle, as shown in Fig. 4(d). It is worth noting that the projection method we use to find the closest point on the triangle edges to the Gauss point may not always result in a point inside the triangle in three dimensions, as shown in Fig. 4(e). In such cases, we search for the closest vertex of the triangle and calculate the distance function based on it (see Fig. 4(f)) and Algorithm 9.

5. Numerical results

This section presents numerical results for the simulation of Poisson equation and the equations of linear elasticity, over domains of complex geometry. We note the following important points for the readers to interpret the results:

- The integration for the weak form is performed using standard $(p+1)^{dim}$ Gauss quadrature points in all the elements, where p is the order of the polynomial finite element interpolation basis. We use the linear basis function ($p = 1$) in all reported results.
- We report accuracy by comparing the numerical solution against analytical solutions. This post-processing operation to compute the L^2-error is performed on each element using five Gauss points per dimension. The reported L^2-error is computed as

$$\|e\|_{L^2(\Omega)} = \|u^h - u_{exact}\|_{L^2(\Omega)} = \sqrt{\int_\Omega (u^h - u_{exact})^2 d\Omega}\,.$$

Note that the error is computed on the true domain Ω. In particular, the SBM solution is smoothly extended over elements that intersect Ω but are not part of the SBM active domain $\tilde{\Omega}_h$.
Besides L^2-error, we also report H^1-error in some test cases. H^1-error is computed as

$$\|e\|_{H^1(\Omega)} = \|u^h - u_{exact}\|_{H^1(\Omega)} = \sqrt{\int_\Omega \left[(u^h - u_{exact})^2 + L^2(\nabla u - \nabla u_{exact}) \cdot (\nabla u - \nabla u_{exact})\right] d\Omega}\,,$$

where L represents the largest dimension of the bounding box of $\tilde{\Gamma}_h$.
Given that the true domain Ω is embedded within a Cartesian grid, we have devised a method to accurately compute the integrals over Ω. We integrate over $\tilde{\Omega}$ and implement an In–Out test for the Gauss points within the intercepted elements. This ensures that only the contributions from the quadrature points inside Ω are considered. We utilize five quadrature points per direction for these elements, translating to 25 points for 2D scenarios and 125 for 3D. Please note that when $\lambda = 0$, $\tilde{\Omega}$ is invariably smaller than Ω; only the $\lambda = 1$ scenario can accurately integrate over Ω. This prompts us to introduce our specialized L_{2N} and H_{1N} error norms, as the integration areas vary between λs.

- In order to compare errors from different simulations, for which the sizes of the domains Ω may differ, we report the normalized error, defined as:

$$L_{2N}(\Omega) = \frac{\|e\|_{L^2(\Omega)}}{\sqrt{\int_\Omega d\Omega}}.$$

Furthermore, the normalized H^1-error is given by:

$$H_{1N}(\Omega) = \frac{\|e\|_{H^1(\Omega)}}{\sqrt{\int_\Omega d\Omega}}.$$

- We define the improvement factor $I_{2\lambda}$ as the metric for comparison between different surrogates with respect to the case in which $\lambda = 1$. We recall that $\lambda = 1$ corresponds to the case in which $\tilde{\Omega}_h \supset \Omega$, that is $\tilde{\Omega}_h$ circumscribes Ω. When comparing simulations performed with different values of λ, a lower value of $I_{2\lambda}$, corresponds to better solutions:

$$I_{2\lambda} = \frac{L_{2N}(\Omega;\lambda)}{L_{2N}(\Omega;\lambda=1)}.$$

- We recall the definition of $\lambda, (0 \le \lambda \le 1)$ as the elemental volume fraction of the domain Ω. More specifically, λ is a threshold used to check whether an Intercepted element needs also to be classified as FalseIntercepted element.

5.1. Identifying the optimal surrogate boundary

Recall that our goal is to construct the *optimal* surrogate boundary that minimizes the distance $\boldsymbol{d}$, the distance between Γ and $\tilde{\Gamma}_h$. Our approach yields that $\lambda = 0.5$ minimizes $\boldsymbol{d}$. We illustrate this result using a canonical test example. Consider a rotated rectangle inclined at a 15° angle with respect to the background octree mesh. We compute the distance (the RMS distance) between the surrogate boundary and true boundary for a range of surrogate boundary choices by varying $\lambda \in [0, 1]$. To be more concise, the RMS (Root Mean Square) distance is defined as:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Gamma_i - \widetilde{\Gamma}_i)^2}, \tag{23}$$

where N is the total number of surface Gauss points, and $\Gamma_i - \widetilde{\Gamma}_i$ represents the distance between Gauss points and the true boundary.
Fig. 5 compares the value of RMS distance for different λ, for different mesh resolutions (identified by the level of refinement). Observe that $\lambda = 0.5$ yields the minimal RMS distance, irrespective of the level of the mesh resolution. In addition to conducting tests at a 15° angle, we also carried out experiments at 10°, 20°, 30°, and 40° angles, paired with varying mesh refinement levels. The results consistently pointed towards $\lambda = 0.5$ as the value that assures the least RMS distance. Fig. 6 further demonstrates these conclusions for various geometries defined by rotating a square at various angles. Thus, choosing $\lambda = 0.5$ provides a simple and effective strategy for constructing the surrogate boundary. In the subsequent sections, we analyze improvement in solution accuracy resulting from constructing an *optimal* surrogate boundary ($\lambda = 0.5$) as compared to a non-optimal surrogate (usually $\lambda = 0, \lambda = 1$).

5.2. Solving Poisson's equation on disk

We next utilize the optimal surrogate construction approach to solve Poisson's equation when Ω has a simple shape: a circular disk. This is an exact geometry, and several ways exist to construct an axis-aligned surrogate boundary for a circle. Consider a disk with a radius $R = 0.5$, centered at $(x_0 = 0.5, y_0 = 0.5)$. We solve Poisson's equation on this geometry with a Dirichlet boundary of $u_0 = 0.01$ on the boundary and forcing term $f = 1$. We choose the penalty parameter α to be 400. This problem has an analytical solution given by: $u(r) = 0.25(R^2 - r^2) + u_0$, where r is the distance from the center ($r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$).
Fig. 7 shows the mesh convergence plot for different choices of λ. We observe second-order convergence in normalized error L_{2N} (Fig. 7(a)), irrespective of the choice of λ. However, the optimal choice of λ, and thereby of surrogate boundary, can significantly reduce the magnitude of the error. We observe that choosing $\lambda = 0.5$ yields minimal error that is almost an order of magnitude lower when compared to the previously reported surrogate boundary choices of $\lambda = 0$ [22] or $\lambda = 1$ [31]. Fig. 7(b) compares this improvement factor for the cases of $\lambda = 0.0$ and $\lambda = 0.5$ with respect to case of $\lambda = 1$ (note that $I_{2\lambda} = 1$ for $\lambda = 1$). The choice $\lambda = 0.5$ produces errors at least three times lower than the case $\lambda = 0$ or $\lambda = 1$ across all mesh resolutions. This improvement is significant, considering that the computational effort in identifying the optimal surrogate is minimal.

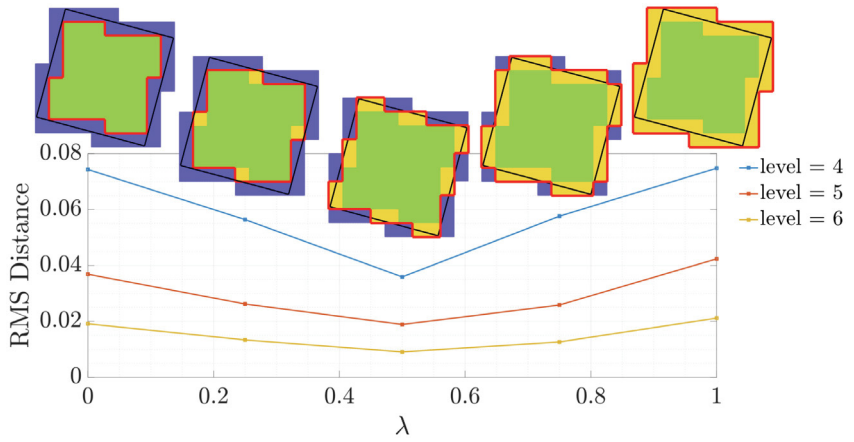


Fig. 5. Constructing the surrogate boundary for a rotated square on a Cartesian mesh. The figure shows the RMS distance with varying λ values for different octree levels. In each inset, the solid red line represents the surrogate surface, while the solid black line represents the true boundary. FalseIntercepted elements are marked by ■, whereas TrueIntercepted elements are marked by ■ and Interior elements by ■.

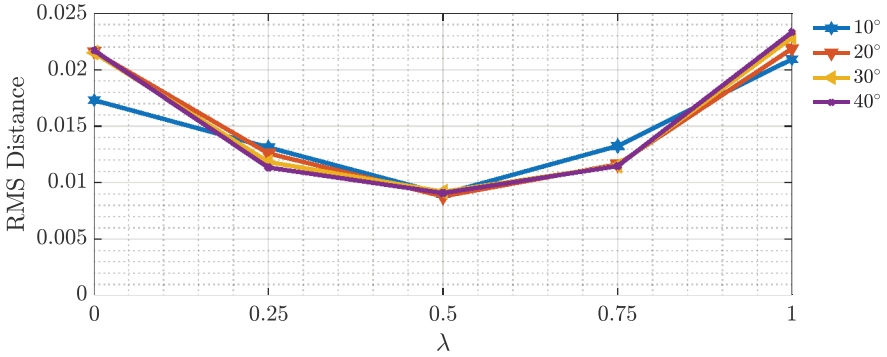
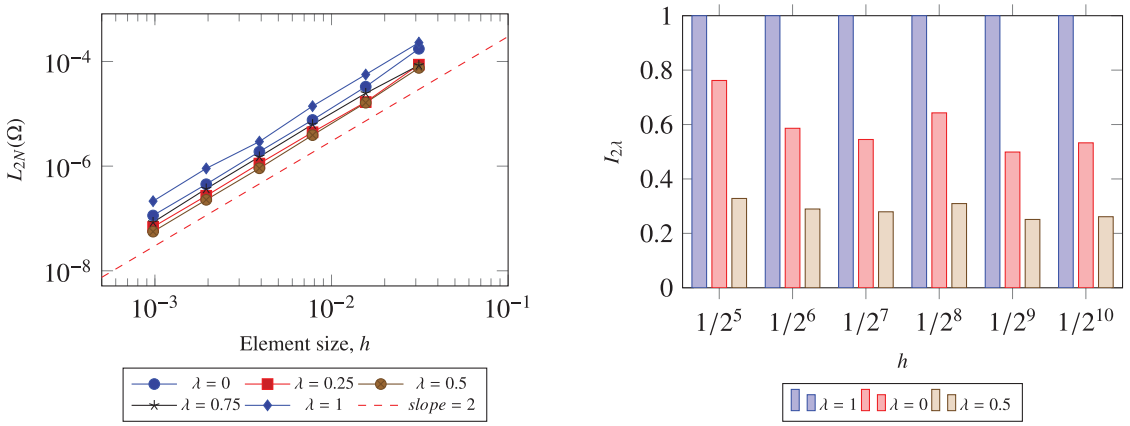


Fig. 6. RMS of the distance between the true and surrogate boundaries, for a square rotated by various angles (10°, 20°, 30°, 40°), and grid refinement level equal to 6.



(a) Normalized L_2 error convergence

(b) Improvement factors, $I_{2\lambda}$. (The lower the better)

Fig. 7. Left: Mesh convergence plot solving the Poisson's equation on a disk. Each line represents a convergence plot with a specific surrogate boundary constructed using a λ. Notice that while all surrogate choices exhibit the expected slope, the $\lambda = 0.5$ surrogate produces the lowest error for any mesh size. Right: The improvement plot shows that the optimal surrogate produces more accurate solutions.

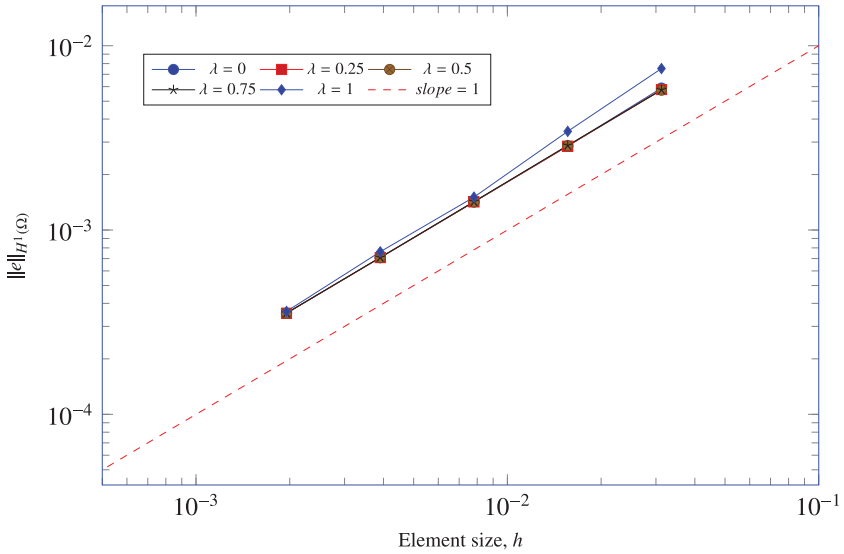


Fig. 8. $\|e\|_{H^1(\Omega)}$ convergence.

Table 1
Results for different values of λ and $H_{1N}(\Omega)$.

λ	$h = \frac{1}{32}$	$h = \frac{1}{64}$	$h = \frac{1}{128}$	$h = \frac{1}{256}$	$h = \frac{1}{512}$
0	$6.93172 \cdot 10^{-3}$	$3.28485 \cdot 10^{-3}$	$1.61935 \cdot 10^{-3}$	$8.03201 \cdot 10^{-4}$	$4.00104 \cdot 10^{-4}$
0.25	$6.64218 \cdot 10^{-3}$	$3.24276 \cdot 10^{-3}$	$1.60941 \cdot 10^{-3}$	$8.01324 \cdot 10^{-4}$	$3.99658 \cdot 10^{-4}$
0.5	$6.50498 \cdot 10^{-3}$	$3.24180 \cdot 10^{-3}$	$1.60427 \cdot 10^{-3}$	$8.00379 \cdot 10^{-4}$	$3.99427 \cdot 10^{-4}$
0.75	$6.51784 \cdot 10^{-3}$	$3.25060 \cdot 10^{-3}$	$1.61003 \cdot 10^{-3}$	$8.01176 \cdot 10^{-4}$	$3.99775 \cdot 10^{-4}$
1	$8.48712 \cdot 10^{-3}$	$3.87137 \cdot 10^{-3}$	$1.70219 \cdot 10^{-3}$	$8.60006 \cdot 10^{-4}$	$4.08395 \cdot 10^{-4}$

Besides the L^2-error evaluation, we have also computed the H^1-error. Referring to Fig. 8 for the H^1-error mesh convergence analysis, we observe an almost perfect order of accuracy, close to 1, across simulations with various λ values. To further comprehend the impact of different λ selections, we present the $H_{1N}(\Omega)$ values in Table 1. Notably, a $\lambda = 0.5$ choice consistently ensures the smallest $H_{1N}(\Omega)$ for a given grid size.

5.3. Complex geometries

We next showcase the ability of the SBM to accurately solve the Poisson's equation over three-dimensional objects. We consider three standard benchmarks exhibiting complex geometries and sharp corners: the Stanford Bunny, the Moai, and the Armadillo. For all three-dimensional cases, the penalty parameter α is set to 400. We use the method of manufactured solutions to construct an analytical solution against which to compare the SBM results. This allows rigorous comparison across different values of λ. The analytical solution for each of the cases is given as:

$$u(x, y, z) = \begin{cases} \cos(\pi x)\, y\, \sin(\pi z), & \text{Stanford bunny;} \\ (1 - x)\,(1 - y)\, \cos(3\pi z), & \text{Moai;} \\ \cos(\pi x)\,(1 - y)\, \sin(\pi z), & \text{Armadillo.} \end{cases} \tag{24}$$

For each object, we perform a mesh convergence analysis by solving the Poisson's equation using the optimal surrogate ($\lambda = 0.5$) and compare the accuracy against the choice of surrogates with $\lambda = 1$ and $\lambda = 0$. Fig. 9 compares $I_{2\lambda}$ for various values of λ and different mesh sizes. We can see that the value of $\lambda = 0.5$ consistently outperforms $\lambda = 0$ and $\lambda = 1$ in terms of accuracy. This demonstrates both the importance of choosing an optimal λ as well as the robustness of the proposed algorithm for solving PDEs on complex geometries. Notably, the errors observed with $\lambda = 0$ are larger than those with $\lambda = 1$ and $\lambda = 0.5$. This discrepancy can be attributed to the intricate shapes of our 3D test geometries, which comprise numerous concave and convex regions. Using $\lambda = 0$ equals to eliminate all the Intercepted elements. This omission makes it challenging for simulations to fully capture the complexity of these shapes. The closest point searching algorithm often selects distant projection points in this scenario. This results in the application of inaccurate Dirichlet conditions (due to the deterioration of the Taylor series expansion with distance) on the surrogate boundary, contributing to the increased error. Retaining more Intercepted elements with $\lambda = 1$ and $\lambda = 0.5$ is a more effective approach, as the closest point searching algorithm finds closer projection points.

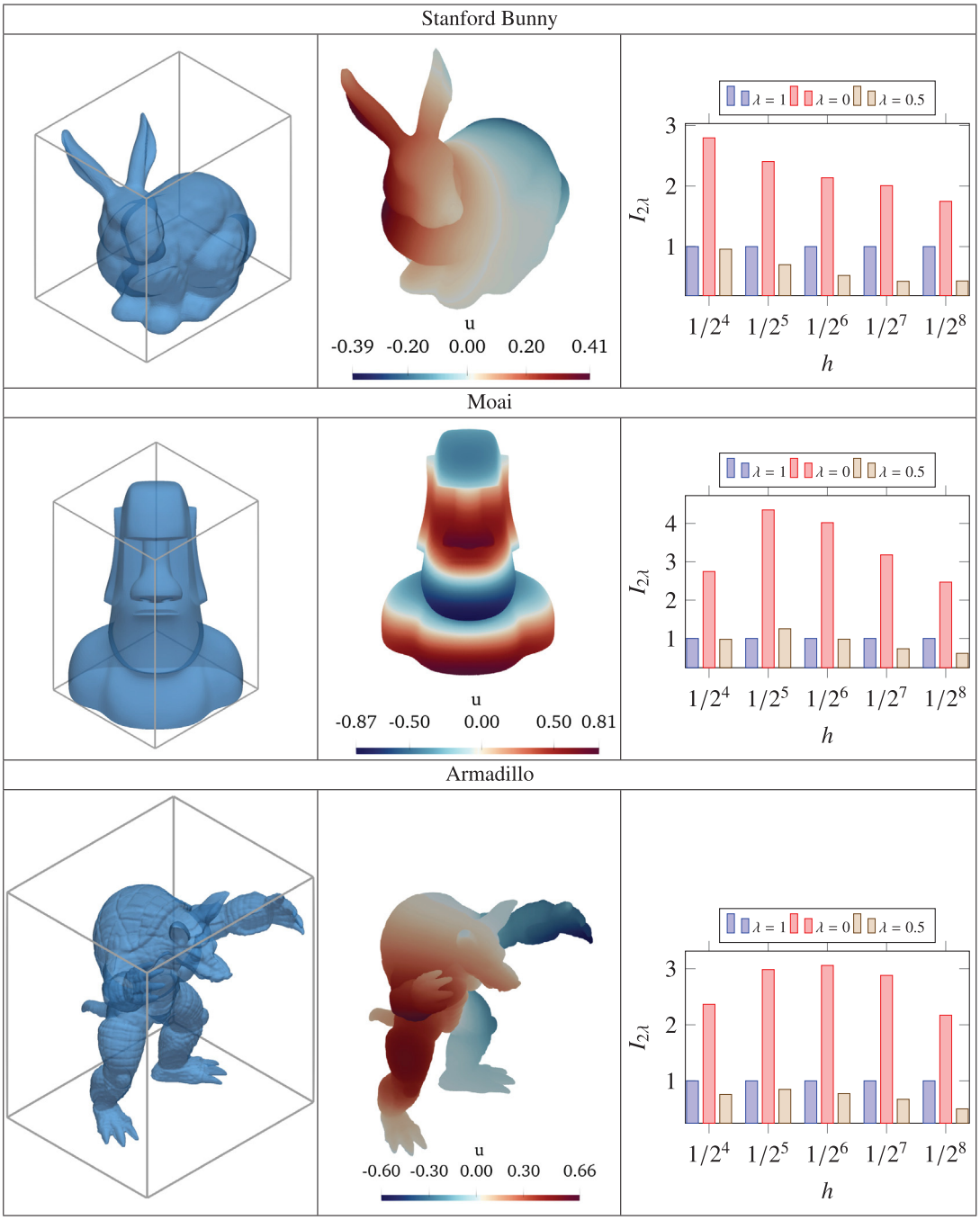
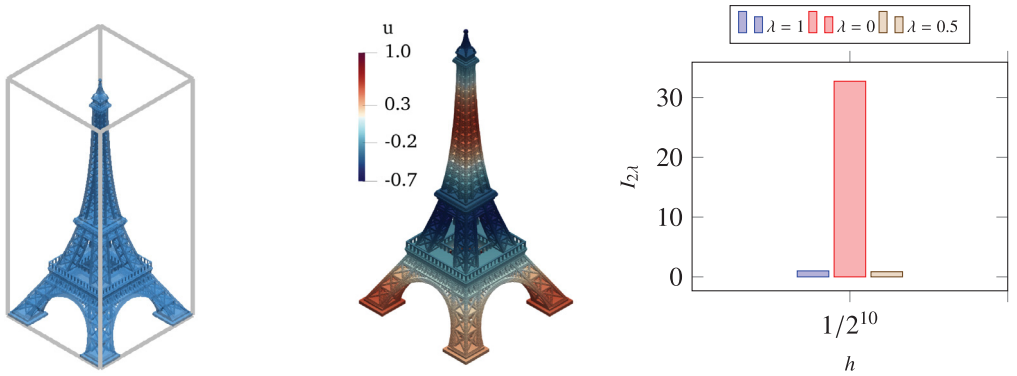


Fig. 9. Solving Poisson's equations on complex 3D domains using the optimized surrogate boundary.

Next, to test the algorithm's robustness, we explore its performance on a geometry exhibiting an extremely complex topology. We consider a simulation on a three-dimensional model of the Eiffel Tower, in STL format. Fig. 10(c) shows the surface representation of the geometry, with a very large number of small holes and sharp corners. We choose a manufactured solution of the form:

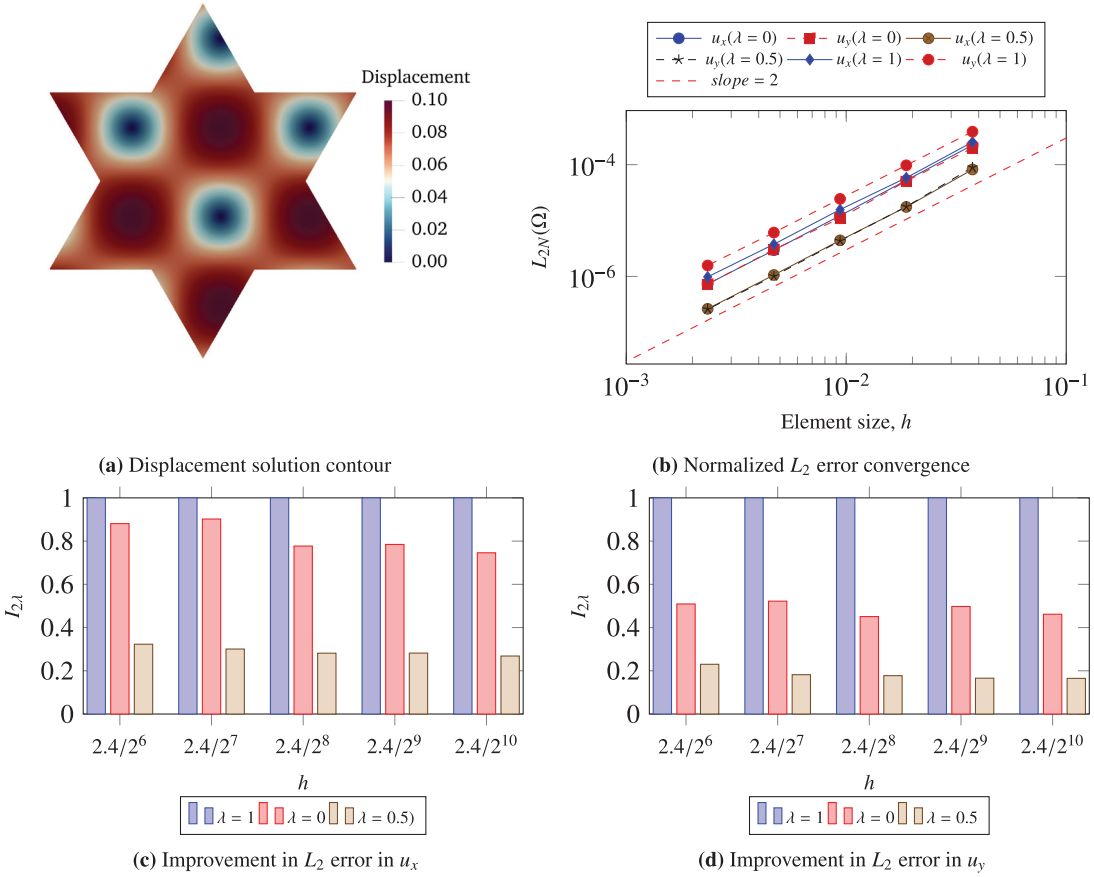
$$u(x, y, z) = (1 - x)(1 - y)\cos(3\pi z). \tag{25}$$

Fig. 10(b) shows the resulting solution field. Finally, we notice the same trend in L^2-error, with surrogates defined by $\lambda = 0.5$ outperforming the other choices (Fig. 10(a)).



(a) STL file **(b)** Solution contour **(c)** Error in the L_2-norm

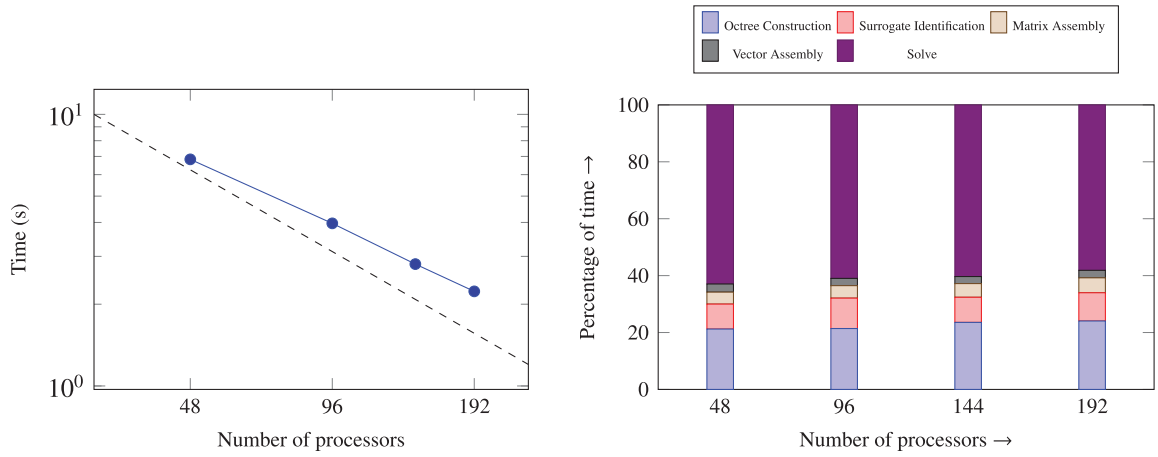
Fig. 10. Solving Poisson's equation on the Eiffel tower model using SBM: (*a*) The STL file of the Eiffel tower immersed into the background mesh and the bounding box. (*b*) Solution contour of Eiffel Tower with mesh size equal to $1/2^{10}$. (*c*) Improvement plot of $L_{2\lambda}$ and the results show that the optimal surrogate boundary can reduce the L_{2N} error.



(a) Displacement solution contour **(b)** Normalized L_2 error convergence

(c) Improvement in L_2 error in u_x **(d)** Improvement in L_2 error in u_y

Fig. 11. Solving linear elasticity on a star shape using SBM: Fig. 11(a) shows the contour for the displacement magnitude contour for $h = 2.4/2^{10}$. Fig. 11(b) shows the convergence plot of L_{2N} error for u_x and u_y for different choices of λ. Fig. 11(c) and Fig. 11(d) show the improvement in L_{2N} error when using the optimal surrogate, $\lambda = 0.5$, for u_x and u_y.



(a) Total solve time with increase in the number of processors.

(b) Percentage of time for individual steps.

Fig. 12. Scaling behavior and percentage of time at different stages of the SBM computation in 2D on TACC Stampede2.

5.4. Linear elasticity

Our final example showcases the SBM approach for solving linear elasticity equations. As shown in Fig. 11(a), we consider a star-shaped domain. The sharp corners and non-convex geometry makes this a challenging case. We set Young's modulus E to be 1 and Poisson's ratio ν to be 0.3. The elastic tensor $\boldsymbol{C}$ is for plain stress. The penalty parameter γ is equal to 400. We consider a manufactured solution of the form:

$$\begin{cases} u_x(x,y) = \dfrac{\sin(\pi x)\cos(\pi y)}{10} \\ u_y(x,y) = \dfrac{\cos(\pi x)\sin(\pi y)}{10}. \end{cases} \tag{26}$$

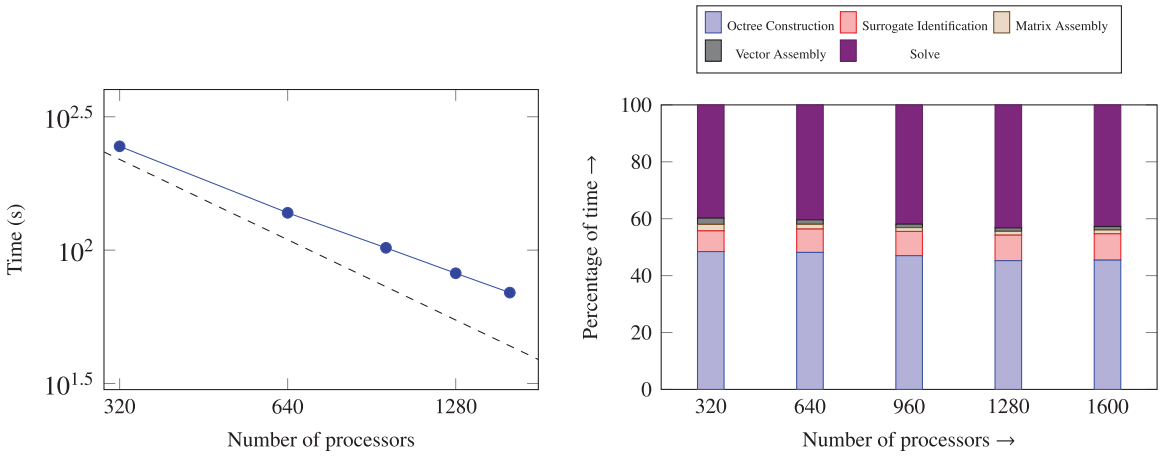
Fig. 11(a) shows the displacement solution contour, while Fig. 11(b) shows the convergence of L_{2N} error. Similar to Poisson's case, we observe a second-order convergence in L^2-error. We see a significant improvement in L^2-error by choosing a value of $\lambda = 0.5$ when compared to $\lambda = 0$ or $\lambda = 1$ both for the x-displacement, u_x (Fig. 11(c)) and y-displacement, u_y(Fig. 11(d)). This improved performance is seen across all mesh refinements.

5.5. Parallel computing scaling

Finally, we present some scaling results of our framework on TACC Stampede2 SKX and ICX nodes. To perform the scaling test, we consider the problem described in Section 5.2. We choose λ of 0.5 for the scaling test as it conforms to the *optimal* surrogate. The strong scaling test considers octree at a level of 10 ($h = 1/2^{10}$). We employ the BiCGStab (Biconjugate Gradient Stabilized) method, a Krylov subspace iterative solver using stopping convergence criteria of 10^{-8}. Concurrently, we apply a Jacobi preconditioner to enhance the convergence properties of the solver. Fig. 12(a) shows the variation of total solve time with increasing the number of processors; with a near-ideal scaling behavior. Fig. 12(b) shows the percentage of time taken by the different stages. We observe that the extra step for constructing the surrogate amounts to almost 10% of the total time. We note that this step, along with octree construction, needs to be performed only once for a static mesh and will be amortized to a smaller fraction for transient problems. As expected, we observe that the overall runtime is dominated by linear algebra solve.

In addition to the 2D case, we conducted a scaling study on a three-dimensional Stanford Bunny, as presented in Section 5.5. For the strong scaling test, we utilized an octree level of 9 ($h = 1/2^9$). The total solving time with respect to the number of processors is shown in Fig. 13(a), while Fig. 13(b) illustrates the percentage of time taken by each stage. Unlike the 2D case, the incomplete octree construction time became the problem's bottleneck. However, the octree construction procedure is a one-time event if we encounter time-dependent problems. Notably, the matrix assembly time is similar to the vector assembly time for this three-dimensional case, given that we calculate and store distance function information during vector assembly and reuse it during matrix assembly to minimize the distance function calculation time. Furthermore, we implemented k-D tree to improve the time required to calculate the distance function in three-dimensional complex shapes. Thus, our algorithms and implementation ensure good scalability, making this approach a practical strategy for solving PDEs in complex domains using conceptually simple octree meshes.

Remark 1. The current implementation of the surrogate boundary identification suggests that the surrogate boundary identification takes up to 8%–10% of the total time (see Figs. 12(b) and 13(b)). Also, we note that the percentage of time is almost independent of the number of processors, indicating a good parallel implementation. The percentage of the time reported is for a Poisson solve.



(a) Total solve time with increase in the number of processors.

(b) Percentage of time for individual steps.

Fig. 13. Scaling behavior and percentage of time at different stages of the SBM computation for 3D Stanford Bunny on TACC Stampede2.

We anticipate that for complex FSI problems, the solve time would dominate even more when considering a Navier–Stokes solver, and therefore, the additional cost of identifying an optimal surrogate is unlikely to become a bottleneck.

6. Conclusions and future work

By shifting the enforcement of boundary conditions from the actual boundary to a surrogate boundary, the SBM allows for the use of Cartesian meshes, eliminating the need for laborious and time-consuming body-fitted meshes around complex geometries. In this work, we answer some key questions regarding the scalable and accurate deployment of the Shifted Boundary Method. The key findings of this work are as follows: (a) identification of an optimal surrogate boundary parameter that greatly reduces numerical error in the SBM, (b) rigorous theoretical analysis demonstrating the optimal convergence of SBM on extended surrogate domains, (c) successful deployment of the SBM on massively parallel octree meshes, including handling of incomplete octrees, and (d) successful application of the SBM to various simulations involving complex shapes, including those with sharp corners and different topologies, with a focus on Poisson's equation and linear elasticity equations. This work sets the stage for a massively parallel, octree-based, general-purpose solution framework – using the SBM – for solving PDEs on arbitrarily complex geometries.

There are several avenues for future developments. One avenue we are actively exploring involves extending the SBM on the octree framework to multi-physics and coupled PDEs, including Navier–Stokes, Cahn-Hilliard Navier–Stokes, and the Poisson–Nernst–Planck equations. Another avenue is to extend the framework to account for moving boundaries, with a natural extension to efficiently model fluid–structure interaction problems across complex geometries. We plan to extend the Weighted Shifted Boundary Method (W-SBM) [28] to the octree framework to manage the variations in fluid volume over time and ensure an accurate capture of the pressure field. Another active avenue of research is to develop robust preconditioners and architecture-aware solvers (for example, GPU-accelerated multigrid methods) for such SBM on octree approaches. A final straightforward extension is to explore the utility of higher-order basis functions and their tradeoff on error vs. time-to-solve.

CRediT authorship contribution statement

Cheng-Hau Yang: Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Kumar Saurabh:** Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Claudio Canuto:** Formal analysis, Writing – review & editing. **Adarsh Krishnamurthy:** Conceptualization, Methodology, Project administration, Resources, Supervision, Writing – review & editing. **Baskar Ganapathysubramanian:** Conceptualization, Funding acquisition, Methodology, Project administration, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was partly supported by the National Science Foundation under the grants NSF LEAP-HI 2053760, NSF CNS 1954556, and NSF OAC 1750865. BG, AK, KS, and CHY are supported in part by AI Research Institutes program supported by the NSF and USDA-NIFA under AI Institute for Resilient Agriculture (2021-67021-35329). Guglielmo Scovazzi has been supported by the National Science Foundation Division of Mathematical Sciences (DMS) under Grant NSF DMS 2207164.

Appendix. Proof of Proposition 1

Proposition. *There exists an extension $\bar{u}$ of u in Ω_{ext}, such that:*

(a) $-\Delta\bar{u} = \bar{f}$ *in* Ω_{ext}*; and*
(b) if $u \in H^2(\Omega)$*, then* $\bar{u} \in H^2(\bar{\Omega}_h)$*, with* $\|\bar{u}\|_{H^2(\bar{\Omega}_h)} \leq C\ \|u\|_{H^2(\Omega)}$.

Proof. To fulfill (a) and (b) above, let $\Omega_\Delta = \Omega_{\text{ext}} \setminus \text{cl}(\Omega)$, where $\text{cl}(\Omega)$ is the closure of Ω. The boundary $\partial\Omega_\Delta$ of the set Ω_Δ is decomposed as the union of two disjoint boundaries, namely $\partial\Omega_\Delta = \Gamma_0 \cup \Gamma_1$, with $\Gamma_0 = \partial\Omega$ and $\Gamma_1 = \partial\Omega_{\text{ext}}$. Then we define

$$\bar{u} = \begin{cases} u & \text{in cl}(\Omega)\,, \\ \hat{u} & \text{in } \Omega_\Delta\,, \end{cases} \tag{A.1}$$

where $\hat{u}$ is the solution of

$$\begin{cases} -\Delta\hat{u} = \bar{f} & \text{in } \Omega_\Delta\,, \\ \hat{u} = u_D & \text{on } \Gamma_0\,, \\ \hat{u} = \varphi & \text{on } \Gamma_1\,, \end{cases} \tag{A.2}$$

and φ is chosen so that

$$\frac{\partial\hat{u}}{\partial n} = \frac{\partial u}{\partial n} \quad \text{on } \Gamma_0\,, \tag{A.3}$$

with $\partial(\cdot)/\partial n$ the normal derivative along $\boldsymbol{n}$ (i.e. along the unit normal to Γ_0 pointing outside of Ω). To motivate Eq. (A.3), observe that Γ_0 and Γ_1 are smooth curves and that $u_D \in H^{3/2}(\Gamma_0)$. Since $f \in L^2(\Omega)$, then, by the regularity theorem of elliptic problems, we have $u \in H^2(\Omega)$ and consequently $\frac{\partial u}{\partial n} \in H^{1/2}(\Gamma_0)$. Similarly, if $\varphi \in H^{3/2}(\Gamma_1)$, then from $\bar{f} \in L^2(\Omega_\Delta)$ and $u_D \in H^{3/2}(\Gamma_0)$ we deduce $\hat{u} \in H^2(\Omega_\Delta)$ and consequently $\frac{\partial\hat{u}}{\partial n} \in H^{1/2}(\Gamma_0)$. In order to deduce $\bar{u} \in H^2(\Omega_{\text{ext}})$, we need to have

$$u_{|\Gamma_0} = \hat{u}_{|\Gamma_0}\,,$$

which is true since $u_{|\Gamma_0} = \hat{u}_{|\Gamma_0} = u_D$ on Γ_0, and

$$\nabla u_{|\Gamma_0} = \nabla\hat{u}_{|\Gamma_0}\,.$$

This last condition can be decomposed into the matching of the component of the gradient normal to Γ_0, which is precisely Eq. (A.3), and the matching of the component of the gradient tangent to Γ_0, namely

$$\frac{\partial\hat{u}}{\partial\tau} = \frac{\partial u}{\partial\tau} \quad \text{on } \Gamma_0\,, \tag{A.4}$$

which is true since they both coincide with $\frac{\partial u_D}{\partial\tau}$ on Γ_0.

Next, we verify the existence of φ in Eq. (A.2) such that Eq. (A.3) is verified. The solution of Eq. (A.2) is equivalent to the solution of the two sub-problems

$$\begin{cases} -\Delta\hat{u}_1 = \bar{f} & \text{in } \Omega_\Delta\,, \\ \hat{u}_1 = u_D & \text{on } \Gamma_0\,, \\ \hat{u}_1 = 0 & \text{on } \Gamma_1\,, \end{cases} \tag{A.5a}$$

$$\begin{cases} -\Delta\hat{u}_0 = 0 & \text{in } \Omega_\Delta\,, \\ \hat{u}_0 = 0 & \text{on } \Gamma_0\,, \\ \hat{u}_0 = \varphi & \text{on } \Gamma_1\,, \end{cases} \tag{A.5b}$$

when we set $\hat{u} = \hat{u}_1 + \hat{u}_0$, so that condition Eq. (A.3) becomes

$$\frac{\partial\hat{u}_0}{\partial n} = \frac{\partial u}{\partial n} - \frac{\partial\hat{u}_1}{\partial n} \in H^{1/2}(\Gamma_0). \tag{A.6}$$

Let us then introduce the operator

$$\mathscr{T} : H^{1/2}(\Gamma_1) \to H^{-1/2}(\Gamma_0)$$

$$\varphi \mapsto \left.\frac{\partial \hat{u}_0}{\partial n}\right|_{\Gamma_0}$$

and check that $\mathscr{T}$ is *onto* (i.e., *surjective*), namely that $\mathrm{Im}(\mathscr{T}) = \left\{\mathscr{T}\varphi : \varphi \in H^{1/2}(\Gamma_1)\right\}$ coincides with $H^{-1/2}(\Gamma_0)$. We argue by contradiction: since $\mathrm{Im}(\mathscr{T})$ is a closed subspace of $H^{-1/2}(\Gamma_0)$, assume that its orthogonal space contains elements $\eta \neq 0$. Any such $\eta \in H^{-1/2}(\Gamma_0)$ then satisfies

$$(\eta, \mathscr{T}\varphi)_{H^{-1/2}(\Gamma_0)} = 0\,, \qquad \forall \varphi \in H^{-1/2}(\Gamma_0)\,.$$

Note that the inner product in $H^{-1/2}(\Gamma_0)$ is the duality pairing

$${}_{H^{-1/2}(\Gamma_0)}\langle \mathscr{T}\varphi\,,\, R\eta\rangle_{H^{1/2}(\Gamma_0)}\,,$$

where $R\eta = z_{|\Gamma_1}$ with z the solution of the problem

$$\begin{cases} -\Delta z = 0 & \text{in } \Omega_\Delta\,, \\ z = 0 & \text{on } \Gamma_1\,, \\ \dfrac{\partial z}{\partial n} = \eta & \text{on } \Gamma_0. \end{cases} \tag{A.7}$$

Hence, for any $\varphi \in H^{1/2}(\Gamma_1)$, we have

$$\begin{aligned}
0 = {}& {}_{H^{-1/2}(\Gamma_0)}\langle \frac{\partial \hat{u}_0}{\partial n}\,,\, z\rangle_{H^{1/2}(\Gamma_0)} \\
= {}& {}_{H^{-1/2}(\Gamma_0)}\langle \frac{\partial \hat{u}_0}{\partial n}\,,\, z\rangle_{H^{1/2}(\Gamma_0)} + \underbrace{{}_{H^{-1/2}(\Gamma_1)}\langle \frac{\partial \hat{u}_0}{\partial n}\,,\, z\rangle_{H^{1/2}(\Gamma_1)}}_{=0 \text{ by Eq.(A.7)}} \\
= {}& {}_{H^{-1/2}(\partial\Omega_\Delta)}\langle \frac{\partial \hat{u}_0}{\partial n}\,,\, z\rangle_{H^{1/2}(\partial\Omega_\Delta)} \\
= {}& \int_{\Omega_\Delta} \underbrace{\Delta \hat{u}_0}_{=0 \text{ by Eq.(A.5b)}}\, z + \int_{\Omega_\Delta} \nabla \hat{u}_0 \cdot \nabla z \qquad \text{(using integration by parts)} \\
= {}& \int_{\Omega_\Delta} \nabla \hat{u}_0 \cdot \nabla z \\
= {}& \int_{\Omega_\Delta} \underbrace{\Delta z}_{=0 \text{ by Eq.(A.7)}}\, \hat{u}_0 + {}_{H^{-1/2}(\partial\Omega_\Delta)}\langle \frac{\partial z}{\partial n}\,,\, \hat{u}_0\rangle_{H^{1/2}(\partial\Omega_\Delta)} \qquad \text{(using integration by parts)} \\
= {}& \underbrace{{}_{H^{-1/2}(\Gamma_0)}\langle \frac{\partial z}{\partial n}\,,\, \hat{u}_0\rangle_{H^{1/2}(\Gamma_0)}}_{=0 \text{ by Eq.(A.5b)}} + {}_{H^{-1/2}(\Gamma_1)}\langle \frac{\partial z}{\partial n}\,,\, \hat{u}_0\rangle_{H^{1/2}(\Gamma_1)} \\
= {}& {}_{H^{-1/2}(\Gamma_1)}\langle \eta\,,\, \varphi\rangle_{H^{1/2}(\Gamma_1)}.
\end{aligned} \tag{A.8}$$

We conclude then that η is the null linear form on $H^{1/2}(\Gamma_0)$, that is $\eta = 0$. As a consequence of $\mathscr{T}$ being *onto*, picking

$$\eta = \frac{\partial u}{\partial n} - \frac{\partial \hat{u}_1}{\partial n}\,,$$

there exists $\phi \in H^{1/2}(\Gamma_1)$ such that the solution of Eq. (A.5b) satisfies

$$\frac{\partial \hat{u}_0}{\partial n} = \eta\,.$$

In addition, since $\eta \in H^{1/2}(\Gamma_0)$, we get $\hat{u}_0 \in H^2(\Omega_\Delta)$ and $\varphi = (\hat{u}_0)_{|\Gamma_1} \in H^{3/2}(\Gamma_1)$, whence $\hat{u} \in H^2(\Omega_\Delta)$ as desired. □

References

[1] R. Mittal, G. Iaccarino, Immersed boundary methods, Annu. Rev. Fluid Mech. 37 (2005) 239–261.
[2] C.S. Peskin, Flow patterns around heart valves: a numerical method, J. Comput. Phys. 10 (2) (1972) 252–271.
[3] F. Xu, D. Schillinger, D. Kamensky, V. Varduhn, C. Wang, M.-C. Hsu, The tetrahedral finite cell method for fluids: Immersogeometric analysis of turbulent flow around complex geometries, Comput. & Fluids 141 (2016) 135–154.
[4] T. Hoang, C.V. Verhoosel, C.-Z. Qin, F. Auricchio, A. Reali, E.H. van Brummelen, Skeleton-stabilized immersogeometric analysis for incompressible viscous flow problems, Comput. Methods Appl. Mech. Engrg. 344 (2019) 421–450.
[5] F. de Prenter, C. Verhoosel, E. van Brummelen, Preconditioning immersed isogeometric finite element methods with application to flow problems, Comput. Methods Appl. Mech. Engrg. 348 (2019) 604–631.
[6] Q. Zhu, F. Xu, S. Xu, M.-C. Hsu, J. Yan, An immersogeometric formulation for free-surface flows with application to marine engineering problems, Comput. Methods Appl. Mech. Engrg. 361 (2019) 112748.

[7] K. Saurabh, B. Gao, M. Fernando, S. Xu, M.A. Khanwale, B. Khara, M.-C. Hsu, A. Krishnamurthy, H. Sundar, B. Ganapathysubramanian, Industrial scale large eddy simulations with adaptive octree meshes using immersogeometric analysis, Comput. Math. Appl. 97 (2021) 28–44.

[8] M.-C. Hsu, C. Wang, F. Xu, A.J. Herrema, A. Krishnamurthy, Direct immersogeometric fluid flow analysis using B-rep CAD models, Comput. Aided Geom. Design 43 (2016) 143–158.

[9] C. Wang, F. Xu, M.-C. Hsu, A. Krishnamurthy, Rapid B-rep model preprocessing for immersogeometric analysis using analytic surfaces, Comput. Aided Geom. Design 52 (2017) 190–204.

[10] A. Balu, M.R. Rajanna, J. Khristy, F. Xu, A. Krishnamurthy, M.-C. Hsu, Direct immersogeometric fluid flow and heat transfer analysis of objects represented by point clouds, Comput. Methods Appl. Mech. Engrg. 404 (2023) 115742.

[11] F. Xu, E.L. Johnson, C. Wang, A. Jafari, C.-H. Yang, M.S. Sacks, A. Krishnamurthy, M.-C. Hsu, Computational investigation of left ventricular hemodynamics following bioprosthetic aortic and mitral valve replacement, Mech. Res. Commun. (ISSN: 0093-6413) 112 (2021) 103604.

[12] J. Parvizian, A. Düster, E. Rank, Finite cell method: *h*- and *p*- extension for embedded domain methods in solid mechanics, Comput. Mech. 41 (2007) 122–133.

[13] A. Massing, M. Larson, A. Logg, M. Rognes, A Nitsche-based cut finite element method for a fluid-structure interaction problem, Commun. Appl. Math. Comput. Sci. 10 (2) (2015) 97–120.

[14] E. Burman, S. Claus, P. Hansbo, M.G. Larson, A. Massing, CutFEM: Discretizing geometry and partial differential equations, Internat. J. Numer. Methods Engrg. 104 (7) (2015) 472–501.

[15] E. Burman, Ghost penalty, C. R. Math. 348 (21–22) (2010) 1217–1220.

[16] E. Burman, P. Hansbo, Fictitious domain methods using cut elements: III. A stabilized nitsche method for Stokes' problem, ESAIM Math. Model. Numer. Anal. 48 (3) (2014) 859–874.

[17] B. Schott, U. Rasthofer, V. Gravemeier, W. Wall, A face-oriented stabilized nitsche-type extended variational multiscale method for incompressible two-phase flow, Internat. J. Numer. Methods Engrg. 104 (7) (2015) 721–748.

[18] B. Schott, W. Wall, A new face-oriented stabilized XFEM approach for 2D and 3D incompressible Navier–Stokes equations, Comput. Methods Appl. Mech. Engrg. 276 (2014) 233–265.

[19] E. Burman, P. Hansbo, Fictitious domain finite element methods using cut elements: II. A stabilized Nitsche method, Appl. Numer. Math. 62 (4) (2012) 328–341.

[20] E. Burman, M.A. Fernández, An unfitted nitsche method for incompressible fluid–structure interaction using overlapping meshes, Comput. Methods Appl. Mech. Engrg. 279 (2014) 497–514.

[21] K. Saurabh, M. Ishii, M.A. Khanwale, H. Sundar, B. Ganapathysubramanian, Scalable adaptive algorithms for next-generation multiphase simulations, 2022, arXiv preprint arXiv:2209.12130.

[22] A. Main, G. Scovazzi, The shifted boundary method for embedded domain computations. Part I: Poisson and Stokes problems, J. Comput. Phys. 372 (2018) 972–995.

[23] A. Main, G. Scovazzi, The shifted boundary method for embedded domain computations. Part II: Linear advection-diffusion and incompressible Navier-Stokes equations, J. Comput. Phys. 372 (2018) 996–1026.

[24] E.N. Karatzas, G. Stabile, L. Nouveau, G. Scovazzi, G. Rozza, A reduced-order shifted boundary method for parametrized incompressible Navier-Stokes equations, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 370 (2020) 113273.

[25] N.M. Atallah, C. Canuto, G. Scovazzi, The second-generation shifted boundary method and its numerical analysis, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113341.

[26] N. Atallah, C. Canuto, G. Scovazzi, The shifted boundary method for solid mechanics, Internat. J. Numer. Methods Engrg. 122 (20) (2021) 5935–5970.

[27] N. Atallah, C. Canuto, G. Scovazzi, Analysis of the Shifted Boundary Method for the Poisson problem in domains with corners, Math. Comp. 90 (2021) 2041–2069.

[28] O. Colomés, A. Main, L. Nouveau, G. Scovazzi, A weighted shifted boundary method for free surface flow problems, J. Comput. Phys. 424 (2021) 109837.

[29] N.M. Atallah, C. Canuto, G. Scovazzi, The high-order shifted boundary method and its analysis, Comput. Methods Appl. Mech. Engrg. 394 (2022) 114885.

[30] X. Zeng, G. Stabile, E.N. Karatzas, G. Scovazzi, G. Rozza, Embedded domain reduced basis models for the shallow water hyperbolic equations with the shifted boundary method, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 398 (2022) 115143.

[31] K. Saurabh, M. Ishii, M. Fernando, B. Gao, K. Tan, M.-C. Hsu, A. Krishnamurthy, H. Sundar, B. Ganapathysubramanian, Scalable adaptive PDE solvers in arbitrary domains, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–15.

[32] C. Burstedde, L.C. Wilcox, O. Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM J. Sci. Comput. 33 (3) (2011) 1103–1133.

[33] H. Sundar, R.S. Sampath, G. Biros, Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel, SIAM J. Sci. Comput. 30 (5) (2008) 2675–2708.

[34] M. Ishii, M. Fernando, K. Saurabh, B. Khara, B. Ganapathysubramanian, H. Sundar, Solving PDEs in space-time: 4D tree-based adaptivity, mesh-free and matrix-free approaches, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019, pp. 1–61.

[35] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The `deal.II` library, version 9.0, J. Numer. Math. 26 (4) (2018) 173–183.

[36] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries, J. Comput. Phys. 190 (2) (2003) 572–600.

[37] S. Popinet, An accurate adaptive solver for surface-tension-driven interfacial flows, J. Comput. Phys. 228 (16) (2009) 5838–5866.

[38] S. Popinet, G. Rickard, A tree-based solver for adaptive ocean modelling, Ocean Model. 16 (3–4) (2007) 224–249.

[39] C. Min, F. Gibou, A second order accurate level set method on non-graded adaptive cartesian grids, J. Comput. Phys. 225 (1) (2007) 300–321.

[40] A. Guittet, M. Theillard, F. Gibou, A stable projection method for the incompressible Navier–Stokes equations on arbitrary geometries and adaptive quad/octrees, J. Comput. Phys. 292 (2015) 215–238.

[41] M. Mirzadeh, A. Guittet, C. Burstedde, F. Gibou, Parallel level-set methods on adaptive tree-based grids, J. Comput. Phys. 322 (2016) 345–364.

[42] H. Sundar, R. Sampath, G. Biros, Bottom-up construction and 2:1 balance refinement of linear octrees in parallel, SIAM J. Sci. Comput. 30 (5) (2008) 2675–2708, http://dx.doi.org/10.1137/070681727.

[43] I. Bogle, K. Devine, M. Perego, S. Rajamanickam, G.M. Slota, A parallel graph algorithm for detecting mesh singularities in distributed memory ice sheet simulations, in: Proceedings of the 48th International Conference on Parallel Processing, 2019, pp. 1–10.

[44] M. Fernando, D. Duplyakin, H. Sundar, Machine and application aware partitioning for adaptive mesh refinement applications, in: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, 2017, pp. 231–242.

[45] M. Fernando, D. Neilsen, H. Lim, E. Hirschmann, H. Sundar, Massively parallel simulations of binary black hole intermediate-mass-ratio inspirals, SIAM J. Sci. Comput. 41 (2) (2019) C97–C138, http://dx.doi.org/10.1137/18M1196972.

[46] J.L. Blanco, P.K. Rai, nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with KD-trees, 2014, https://github.com/jlblancoc/nanoflann.