SIAM J. SCI. COMPUT.
Vol. 40, No. 4, pp. A2584–A2608

# ROBUST NODE GENERATION FOR MESH-FREE DISCRETIZATIONS ON IRREGULAR DOMAINS AND SURFACES[*]

VARUN SHANKAR[†], ROBERT M. KIRBY[‡], AND AARON L. FOGELSON[§]

**Abstract.** We present a new algorithm for the automatic one-shot generation of scattered node sets on irregular two dimensional (2D) and three dimensional (3D) domains using Poisson disk sampling coupled to novel parameter-free, high-order parametric spherical-radial-basis-function-based geometric modeling of irregular domain boundaries. Our algorithm also automatically modifies the scattered node sets *locally* for time-varying embedded boundaries in the domain interior. We derive complexity estimates for our node generator in 2 and 3 dimensions that establish its scalability, and verify these estimates with timing experiments. We explore the influence of Poisson disk sampling parameters on both quasi-uniformity in the node sets and errors in a radial-basis-function-based–finite-difference discretization of the heat equation. In all cases, our framework requires only a small number of "seed" nodes on domain boundaries. The entire framework exhibits $O(N)$ complexity in both 2 and 3 dimensions.

**Key words.** node generation, geometric modeling, radial basis functions, poisson disk sampling, embedded boundaries

**AMS subject classifications.** 68Q25, 68U05

**DOI.** 10.1137/17M114090X

## 1. Introduction.

Many engineering applications require solving partial differential equations (PDEs) on both surfaces and their enclosed volumes. A typical pipeline is that data drive one's understanding of the geometry, and from that initial data a discretization of the PDE is built—either via mesh-based methods (e.g., finite element methods (FEMs)) or from the class of mesh-free methods (e.g., radial basis function (RBF) methods). From the latter class, RBF-based–finite-difference (RBF-FD) methods have been used for over a decade to solve PDEs [2, 7, 4, 41, 5, 14, 15, 19, 13, 26, 27, 21, 37], and recent work by Flyer and coworkers [12, 11, 1] has helped overcome the traditional problems of RBF interpolants. Thus, for the remainder of this article, we will assume PDE discretizations based on RBF-FD methods.

One motivation for developing the methods described in this paper is our own work in simulating blood clotting, which involves the formation of an aggregate of platelets. We treat each platelet as a discrete object (reconstructed from a set of discrete points) moving with the fluid in which it is immersed [16, 35, 34]. Proteins and other molecules that are secreted into the fluid by platelets mediate interplatelet interactions. The fluid concentrations of these molecules each satisfy an advection-diffusion-reaction equation in the fluid domain, with Robin-type boundary conditions satisfied on each platelet's surface. There may be several hundred platelets in the

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section July 27, 2017; accepted for publication (in revised form) June 7, 2018; published electronically August 21, 2018.
http://www.siam.org/journals/sisc/40-4/M114090.html
**Funding:** This work was supported by NSF DMS-1521748. The first author's work was supported by DMS-1160432 and DMS-1714844.
[†]Corresponding author. Department of Mathematics, University of Utah, Salt Lake City, UT 84112 (vshankar@math.utah.edu, http://www.math.utah.edu/~vshankar/).
[‡]School of Computing and Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112 (kirby@cs.utah.edu, http://www.cs.utah.edu/~kirby/).
[§]Departments of Mathematics and Bioengineering, University of Utah, Salt Lake City, UT 84112 (fogelson@math.utah.edu, http://www.math.utah.edu/~fogelson/).

overall domain, with their numbers changing over time as new platelets enter and exit the domain; the fluid domain (that part of the overall domain external to all platelets) itself changes as the platelets' positions change. In order to apply RBF-FD to this problem, a rapid and *local* node generation and modification algorithm is required. The methods presented in this paper may also be valuable in constructing geometry and node sets for solving PDEs from medical images; such images are used to define the geometry of a vascular bed for simulations of blood flow in that vascular bed [38, 39]. While current methods utilize meshing and an FEM-based solution framework, a rapid node generation and modification algorithm would allow direct solution of PDEs on the point-cloud data obtained from these images using RBF-FD. Another application, which advances in cellular imaging should make possible soon, is constructing intracellular domains, accounting for the presence of cellular organelles, and generating node sets within them for the solution of fluid-flow or advection-diffusion-reaction equations for intracellular processes [23, 29].

With the above applications in mind and from previous experience with RBF-FD [31], we attempted to generate node sets using existing tools such as Gmsh [22] and Distmesh [25], with the idea that we would generate surface and volumetric meshes, disregard the edge information, and use the remaining node set for building differential operators. However, numerical differential operators built on these node sets were unstable on irregular domains (e.g., approximations of second-order differential operators that had positive real eigenvalues). We then turned to the graphics community for inspiration on point-based sampling techniques, and tried Poisson disk sampling [3, 42]. A naive implementation (without careful tuning) of Poisson disk sampling also resulted in point sets that were bunched at curved boundaries and/or that generated poorly conditioned numerical operators.

To overcome these challenges, we designed node generation algorithms (based on Poisson disk sampling) for irregular domains using only a small set of locations on domain boundaries/surfaces as a starting point. The algorithms presented in this paper attempt to satisfy four criteria. First, since RBF-FD is our chosen high-order discretization method, we require node sets on domain boundaries and the enclosing volume that approximately respect a user-specified node spacing $h$ on the surface and the volume; convergence estimates for RBF-FD methods are typically specified in terms of such measures of node spacing [8, 10]. Second, we wish to eliminate free or tuning parameters (other than $h$) for generating these node sets in order to provide mathematical modelers with robust and automatic tools for solving their model equations. Third, in order to rapidly generate node sets on time-varying domains, we desire *one-shot* node generation algorithms with the ability to modify node sets *locally*, in contrast to existing (iterative) repulsion-based approaches [18]. Fourth, in order to reduce time-to-discovery, we want our methods to be (provably) computationally efficient and scalable with regards to the number of points needed to discretize the problem, regardless of the order of the RBF-FD method used. Our focus in this work is not on generating a suitable parametrization for an arbitrary point cloud, which is a problem that has been tackled by others (e.g., [6]). We restrict ourselves to irregular domains with boundaries that are easily parametrizable, of genus-0, and at least homeomorphic to the sphere $\mathbb{S}^{d-1} \subset \mathbb{R}^d$.

The remainder of our paper is structured as follows. In section 2, we present a robust approach to geometric modeling using spherical RBF (SBF) interpolation, and confirm its high-order convergence rates with numerical experiments. In section 3, we present a node generation pipeline that utilizes this new geometric modeling technique and Poisson disk sampling to generate scattered node sets on irregular domains and

TABLE 1
*Table of symbols.*

| Symbol | Meaning |
|---|---|
| $N$ | Total number of domain nodes |
| $N_i$ | Number of interior nodes in the domain |
| $N_b$ | Number of boundary nodes |
| $N_{obb}$ | Number of nodes in the domain bounding box |
| $N_d$ | Number of "seed nodes" and data sites for geometric model |
| $q$ | Length/perimeter of the domain boundary in 2 dimensions |
| $a$ | Area of the domain in 2 dimensions |
| $a_{obb}$ | Area of the domain bounding box in 2 dimensions |
| $s$ | Surface area of the domain boundary in 3 dimensions |
| $v$ | Volume of the domain in 3 dimensions |
| $v_{obb}$ | Volume of the domain bounding box in 3 dimensions |
| $h$ | Approximate node spacing in domain |
| $h_i$ | Approximate node spacing in domain interior |
| $h_b$ | Approximate node spacing on domain boundary |
| $N_p$ | Number of embedded inner boundaries |
| $N_b^\Gamma$ | Number of nodes on each embedded boundary |
| $N_d^\Gamma$ | Number of seed nodes on each embedded boundary |
| $q^\Gamma$ | Length/perimeter of each embedded boundary in 2 dimensions |
| $s^\Gamma$ | Surface area of each embedded boundary in 3 dimensions |
| $h_\Gamma$ | Node spacing on each embedded boundary |

surfaces while satisfying our four design criteria. We also verify the quasi-uniformity of these node sets via histograms, and test their influence on stability and errors in high-order RBF-FD discretizations. In section 4, we derive complexity estimates that show the scalability of our proposed scheme, and verify these estimates with timing experiments. We conclude in section 5 by discussing the trade-offs of our work and possible future directions.

**A note on symbols.** For the readers' convenience, we have listed and defined the important symbols used in this article in Table 1. This table is not exhaustive, and other symbols will be defined in the text where required.

## 2. Geometric modeling with RBFs.

The foundation of our node generation algorithm for irregular domains is a method for reconstructing the domain boundary from a set of "seed" nodes obtained from an application. One approach to doing this reconstruction is to form a parametric geometric model of the domain boundary. The authors have previously developed parametric geometric models based on *global* SBF interpolants that have been used for modeling/reconstructing irregular surfaces in several applications [35, 36, 34, 33, 21, 20]. However, these methods require the user to tune a shape parameter. We now present a modification of the geometric models for closed surfaces developed in [35] that eliminates this parameter. We assume the closed surface $\mathbb{M} \subset \mathbb{R}^d$ is homeomorphic to the unit sphere $\mathbb{S}^{d-1}$, where $d$ is the dimension of the embedding space. Let $\chi_d = \{\boldsymbol{X}_k\}_{k=1}^{N_d}$ be a set of *data sites* on $\mathbb{M}$. For the following discussion, we assume that $\Lambda = \{\boldsymbol{\lambda}_k\}_{k=1}^{N_d} = \{\lambda_k^1, \lambda_k^2, \ldots, \lambda_k^{d-1}\}_{k=1}^{N_d}$ is the set of parameter values for the data sites $\chi_d$.

### 2.1. Modeling closed surfaces.

We focus on the problem of modeling a closed surface $\mathbb{M} \subset \mathbb{R}^d$. Let $\boldsymbol{\xi}(\boldsymbol{\lambda}) \in \mathbb{S}^{d-1}$. Furthermore, assume we are given $N_d$ points $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_d}$ on $\mathbb{M}$, where $\boldsymbol{X}_k = \{X_k^1, \ldots, X_k^d\}$. To model the surface, we use an SBF interpolant generated from a polyharmonic spline (PHS) kernel. Interpolating

$X^j$ with this new PHS SBF interpolant, we have

$$s^j(\boldsymbol{\lambda}) = \sum_{k=1}^{N_d} c_k^j \phi\left(\sqrt{2(1-\xi(\boldsymbol{\lambda})\cdot\xi(\boldsymbol{\lambda}_k))}\right), \tag{1}$$

where $\phi(r) = r^m$ on $\mathbb{S}^1$ ($m$ is odd), and $\phi(r) = r^m \log(r)$ on $\mathbb{S}^2$ ($m$ is even). For $\mathbb{S}^1$ (the unit circle), $\boldsymbol{\lambda} = \lambda^1 = \lambda$, allowing us to write (1) as

$$s^j(\lambda) = \sum_{k=1}^{N_d} c_k^j \left(2 - 2\cos\left(\lambda - \lambda_k\right)\right)^{\frac{m}{2}}, \quad 0 < m \notin 2\mathbb{N}. \tag{2}$$

A similar (albeit more complicated) expression can be derived for an interpolant on $\mathbb{S}^2$ (the unit sphere). To find the coefficients, we solve the following linear systems for $j = 1, \ldots, d$:

(3)
$$\begin{bmatrix} (2(1-\xi(\boldsymbol{\lambda}_1)\cdot\xi(\boldsymbol{\lambda}_1)))^{\frac{m}{2}} & \ldots & (2(1-\xi(\boldsymbol{\lambda}_1)\cdot\xi(\boldsymbol{\lambda}_{N_d})))^{\frac{m}{2}} \\ (2(1-\xi(\boldsymbol{\lambda}_2)\cdot\xi(\boldsymbol{\lambda}_1)))^{\frac{m}{2}} & \ldots & (2(1-\xi(\boldsymbol{\lambda}_2)\cdot\xi(\boldsymbol{\lambda}_{N_d})))^{\frac{m}{2}} \\ \vdots & \ddots & \vdots \\ (2(1-\xi(\boldsymbol{\lambda}_{N_d})\cdot\xi(\boldsymbol{\lambda}_1)))^{\frac{m}{2}} & \ldots & (2(1-\xi(\boldsymbol{\lambda}_{N_d})\cdot\xi(\boldsymbol{\lambda}_{N_d})))^{\frac{m}{2}} \end{bmatrix} \begin{bmatrix} c_1^j \\ \vdots \\ c_{N_d}^j \end{bmatrix} = \begin{bmatrix} X_1^j \\ \vdots \\ X_{N_d}^j \end{bmatrix}.$$

In the case of standard PHS RBF interpolants, it is important to augment the RBFs with polynomials of degree $\lfloor \frac{m-1}{2} \rfloor$ (or higher) to regularize the end conditions [17]. However, in the case of PHS SBFs, this is unnecessary due to periodicity. More important is the question of the invertibility of (3). It is well known that the inclusion of polynomials is required to show unisolvency of the interpolant for conditionally positive-definite kernels such as the PHS kernel [10]. However, in our experiments, we found that we were able to solve the linear system in (3) in all cases with simple Gaussian elimination even without the inclusion of trigonometric polynomials.

It is important to note that, unlike in [35] where infinitely smooth SBFs with shape parameters were used, we use the piecewise-smooth PHS SBF with a fixed $m$, eliminating the need for tuning the shape parameter; we remark on the choice of $m$ in section 2.2. We demonstrate in section 2.2 that this setting allows for spectral convergence rates when recovering smooth functions, despite the use of a piecewise-smooth SBF. The interpolation matrix in (3) can be decomposed for a cost of $O(N_d^3)$, with subsequent coefficient calculations for all functions of the parametric nodes costing $O(N_d^2)$ operations. We will discuss the computational complexity of this technique in the context of node generation in a later section.

#### 2.1.1. Parametrization and interpolation nodes.

We now discuss the parametrization of the seed nodes on the domain boundary. This parametrization will be used within the SBF interpolant (1). Clearly, since the SBF interpolant assumes parametrization on $\mathbb{S}^{d-1}$, we must parametrize the seed nodes on the same set. However, as mentioned in section 1, this is, in general, a nontrivial task in 3 dimensions. For the sake of this article, we focus on simple parametrizations of the seed nodes on the domain boundary.

When modeling a closed surface $\mathbb{M} \subset \mathbb{R}^2$, a natural parametric node set for interpolation is $\Lambda = \{\lambda_k\}_{k=1}^{N_d} \in [-\pi, \pi)$. For the purposes of this article, we restrict ourselves to equispaced samples. If adaptive sampling is required, these samples can be clustered appropriately.

For $\mathbb{M} \subset \mathbb{R}^3$, we use the parametric node set $\Lambda = \{\boldsymbol{\lambda}_k\}_{k=1}^{N_d} = \{\lambda_k, \theta_k\}_{k=1}^{N_d}$, where $-\pi \leq \lambda < \pi$ and $-\pi/2 \leq \theta < \pi/2$. However, obtaining the $(\lambda_k, \theta_k)$ pairs is not quite as simple as in the case of $\mathbb{M} \subset \mathbb{R}^2$. While equispaced points in $[-\pi, \pi)$ correspond to equispaced Cartesian samples for $\mathbb{S}^1$, this is not true for $\mathbb{S}^2$. As was done previously [35], we use quasi-uniform Cartesian nodes on the sphere transformed into spherical coordinates in the rectangle $[-\pi, \pi) \times [-\pi/2, \pi/2)$ to obtain our parametric node sets. There are many quasi-uniform node sets used for this purpose in the RBF literature, such as minimal energy or maximal determinant nodes [35]. However, these node sets are typically obtained by solving expensive optimization problems. Other node sets, such as icosahedral nodes, are only available for certain values of $N_d$. We select a node set that is easily computed on-the-fly in $O(N_d)$ operations for any value of $N_d$: the so-called generalized spiral points, developed by Rakhmanov, Saff, and Zhu [28]. Interestingly, the algorithm presented in [28] was improved by Thomsen in an online discussion group [30, 40]. We opt to use Thomsen's generalized spiral algorithm, presented in simple form on [30]. Once the generalized spiral points are generated, we then transform them into spherical coordinates to obtain the parametric node set $\Lambda$.

**2.1.2. Evaluation nodes.** While a good choice of interpolation nodes controls geometric modeling error, the choice of evaluation nodes affects the errors in RBF-FD discretizations of PDEs on surfaces and domain boundaries. The goal is to find a set of parametric evaluation nodes in the rectangle $[-\pi, \pi) \times [-\pi/2, \pi/2)$ (or in $[-\pi, \pi]$ for curves) that results in quasi-uniformly-spaced Cartesian nodes on the surface. While this can be accomplished by solving optimization problems or rejection sampling in parametric space, we will adopt an approach that is a combination of parametric supersampling and *Cartesian* thinning. This is explained in the context of our node generator in the next section.

**2.2. Convergence of geometric models.** In this section, we test the convergence of our new geometric model for closed surfaces with the goal of understanding its behavior on reconstructing both infinitely smooth and finitely smooth domain boundaries. We use the boundaries presented in [35], repeated here for reference. The functions corresponding to the one dimensional (1D) curves are given by (where two dimensional is shortened to 2D)

$$2D\ C^{\infty} : \left[1 + A\exp\left(\frac{-(1-\cos\lambda)^2}{\sigma_1}\right)\right] \boldsymbol{x}_{ideal}, \tag{4}$$

$$2D\ C^{2} : \left[1 + B\exp\left(\frac{-(1-\cos^2\lambda)^{1.5}}{\sigma_2}\right)\right] \boldsymbol{x}_{ideal}, \tag{5}$$

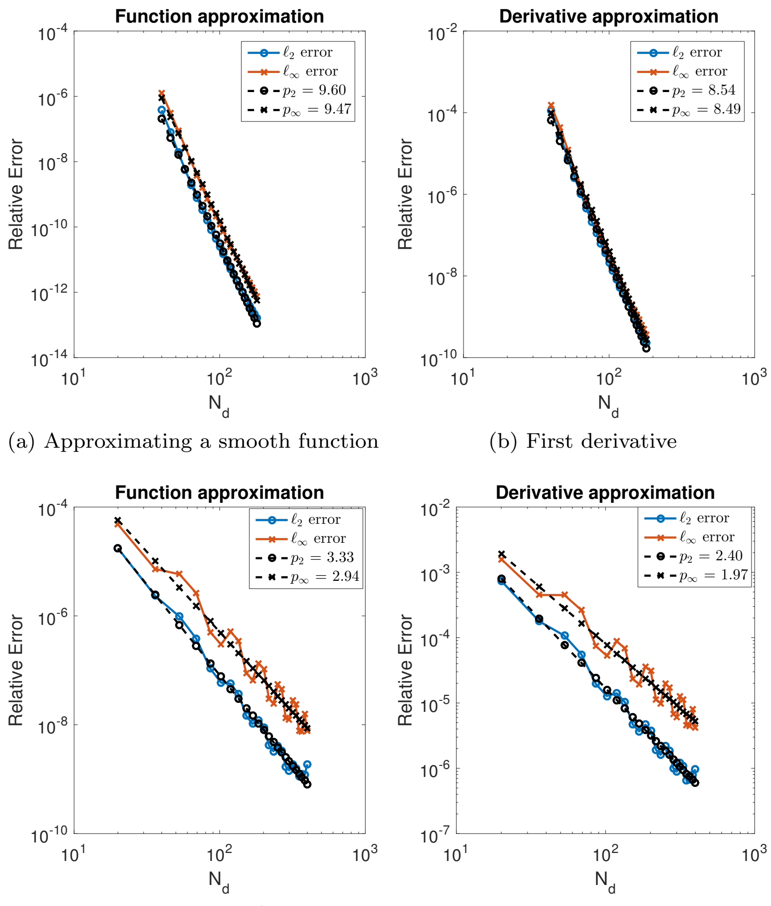
$$\boldsymbol{x}_{ideal}(\lambda) = \left(x_c + a\cos\lambda, y_c + b\sin\lambda\right), \tag{6}$$

where $-\pi \leq \lambda \leq \pi$. For the boundary represented by the $C^{\infty}$ function, we use $x_c = y_c = 0.9$, $a = 0.04$, $b = 0.05$, $A = 0.09$, and $\sigma_1 = 0.1$. For the boundary represented by the $C^2$ function, we use $x_c = y_c = 0.2$, $a = b = 0.1$, $B = 0.04$, and $\sigma_2 = 0.9$. The functions corresponding to the 2D surfaces are given by (where three dimensional is shortened to 3D)

$$3D\ C^{\infty} : \left[1 + A\exp\left(\frac{\mathrm{r_c^2}}{\sigma_1}\right)\right] \boldsymbol{x}_{ideal}, \tag{7}$$

$$3D\ C^{3} : \left[1 + B\exp\left(\frac{\mathrm{r_c^{1.5}}}{\sigma_2}\right)\right] \boldsymbol{x}_{ideal}, \tag{8}$$

$$\boldsymbol{x}_{ideal}(\lambda) = \left(x_c + a\cos\lambda\cos\theta, y_c + b\sin\lambda\cos\theta, z_c + c\sin\theta\right), \tag{9}$$



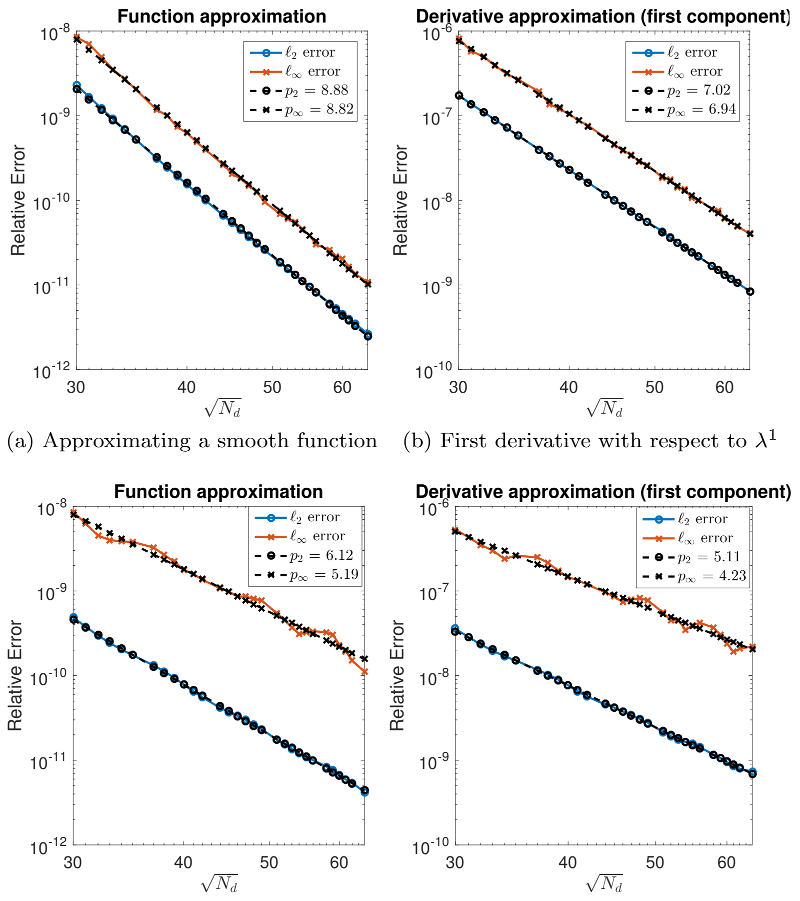
(a) Approximating a smooth function (b) First derivative

(c) Approximating a $C^2$ function (d) First derivative

Fig. 1. *Errors in function and derivative approximation for closed* $1D$ *curves using* $\phi(r) = r^7$. *The dashed lines in the figures on the bottom row are lines of best fit with slopes shown in the legends.*

where $-\pi \leq \lambda \leq \pi$, $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$, and $r_c = 1 - \cos\theta\cos\theta_c\cos(\lambda - \lambda_c) - \sin\theta\sin\theta_c$. For the boundary represented by the $C^\infty$ function, we use $x_c = y_c = z_c = 0.9$, $a = 0.1$, $b = 0.2$, $c = 0.09$, $A = 0.09$, and $\sigma_1 = 0.2$. For the boundary represented by the $C^3$ function, we use $x_c = y_c = 0.1$, $z_c = 0.2$, $a = b = c = 0.1$, $B = 0.04$, and $\sigma_2 = \frac{16}{25}$. We set $\lambda_c = 0$ and $\theta_c = \frac{\pi}{2}$. Note that since the functions represent curves/surfaces, their first parametric derivatives are the tangent vectors; for a detailed description on generating these and other geometric quantities from the interpolant, see [35].

We first present results using PHS SBFs for interpolating the above closed 1D curves, then present analogous results for interpolating the above closed 2D surfaces. In the 1D case, we use $\phi(r) = r^7$, while in the 2D case we use $\phi(r) = r^6 \log r$. Figure 1 shows the results of modeling 1D curves of differing smoothness with PHS SBFs. Figures 1(a) and 1(b) show the errors in approximating an infinitely smooth function and its first derivative using PHS SBFs. Figure 1(a) shows that we get a convergence rate of approximately $N_d^{-9}$ (ninth order) when interpolating the $C^\infty$ function with $\phi(r) = r^7$, and lose an order when approximating its derivative. This is an order higher than the theoretical $O(N_d^{-8})$ convergence rate for PHS RBFs in



(a) Approximating a smooth function (b) First derivative with respect to $\lambda^1$

(c) Approximating a $C^3$ function (d) First derivative with respect to $\lambda^1$

FIG. 2. *Errors in function and derivative approximation for closed 2D surfaces using $\phi(r) = r^6 \log r$. The results for the other first derivative are similar and therefore omitted. The dashed lines in the figures on the bottom row are lines of best fit with slopes shown in the legends.*

1 dimension [10]. In contrast, Figures 1(c) and 1(d) show that when approximating a boundary represented by a $C^2$ function, the convergence rates are limited by the smoothness of the function, with each subsequent derivative converging at one order lower, as indicated by the slopes of the dashed lines.

Figure 2 shows similar results for modeling boundaries that are 2D surfaces. Let $h_d \propto \frac{1}{\sqrt{N_d}}$. The predicted convergence rate for the kernel $\phi(r) = r^6 \log r$ is $O(h_d^8)$ in 2 dimensions [10], and the observed convergence rate in Figure 2(a) is slightly higher. Figure 2(b) shows that we attain the theoretically predicted $O(h_d^7)$ convergence rate when approximating the first derivative. Figure 2(c) shows that the approximation order is limited in the case of approximating a $C^3$ function. In the infinity norm, we have lost about three orders of convergence when compared to the $C^\infty$ case, and one further order when approximating the derivative.

It is important to note that while spectral convergence rates can be obtained for infinitely smooth functions using infinitely smooth SBFs [35], the shape parameters in that work had to be carefully selected by running extensive experiments. In contrast, the PHS SBFs in this study required no tuning to achieve high order convergence rates. Our experiments also indicated that using spherical harmonics in conjunction with SBFs resulted in spectral convergence rates for modeling 2D surfaces. However, we believe that our current formulation strikes an excellent balance between computational cost and accuracy for both $C^\infty$ and finitely smooth boundaries. When selecting $N_d$ in practical applications, it is reasonable to select it by ensuring that the error of the geometric model (from theory) is equal to or lower than the error from an RBF-FD discretization.

## 3. Robust node generation on irregular domains and surfaces.

Having tested our geometric modeling technique, we present our algorithm (Algorithm 1) for generating scattered nodes on irregular domains $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, using a combination of Poisson disk sampling and the new geometric modeling technique. Algorithm 1 has the important feature that it ensures that interior nodes maintain a user-specified separation distance of $h$ from the boundary and between each other. In the remainder of this section, we will discuss the salient features of Algorithm 1, present some modifications to allow for time-varying embedded boundaries, explore the quality of the node sets it produces (using histograms), and test its suitability for RBF-FD discretizations.

**Algorithm 1** Node generation for domains with smooth boundaries.

```
   Given: $\mathcal{X}_d = \{(\boldsymbol{X}_d)_\ell\}_{\ell=1}^{N_d}$, a set of seed nodes on domain boundary.
   Given: $h$, the average separation distance between nodes.
   Generate: $\mathcal{X}_b = \{(\boldsymbol{X}_b)_j\}_{j=1}^{N_b}$, a set of boundary nodes with spacing $h$.
   Generate: $\eta_b$, the set of outward unit normals on the boundary.
   Generate: $\mathcal{X}_i = \{\boldsymbol{x}_k\}_{k=1}^{N_i}$, a set of interior nodes with spacing $h$.
   Generate: $\mathcal{X} = \mathcal{X}_i \cup \mathcal{X}_b$.
1: Obtain $\mathcal{X}_b$ using Algorithm 2.
2: Evaluate derivatives of (1) at the parametric evaluation points obtained from
   Algorithm 2 to obtain $\eta_b$.
3: Use the normals $\eta_b$ to project $\mathcal{X}_b$ inwards a distance $h$, giving a set $\hat{\mathcal{X}_b}$; this set
   defines an inner boundary and an inner domain.
4: Build a kd-tree on the set $\hat{\mathcal{X}_b}$.
5: Generate the oriented bounding volume (OBV) corresponding to $\hat{\mathcal{X}_b}$ using Algorithm 3.
6: Fill the OBV with points of (approximate) spacing $h$ using Poisson disk sampling
   in Algorithm 4 [3].
7: Test each sample against the outward normal at its closest inner boundary point
   (found using the kd-tree). If it is outside the inner domain, discard.
8: All remaining Poisson disk samples (not including $\hat{\mathcal{X}_b}$) form the set $\mathcal{X}_i$.
9: Set $\mathcal{X} = \mathcal{X}_i \cup \mathcal{X}_b$.
```

### 3.1. Sampling on surfaces.

A node generation algorithm for RBF-FD methods must generate boundary nodes that are approximately a user-specified distance $h$ apart. Since all domain boundaries are submanifolds of $\mathbb{R}^d$, this problem involves generating a set of parametric evaluation nodes that result in approximately quasi-uniform Cartesian node sets on surfaces. We present an algorithm based on two simple ideas: supersampling of our geometric model, and decimation. As implemented in Algorithm 2, we first supersample the parametric interpolant (1) to the surface at a large set of parametric evaluation points in $[-\pi, \pi)$ (curves) and $[-\pi, \pi) \times [-\pi/2, \pi/2)$ (surfaces). The resulting Cartesian nodes are not quasi-uniformly spaced, but tend to "bunch" according to the parametric map. Thus, the Cartesian node set is now thinned (or *decimated*) to approximately enforce that all nodes be no closer than some separation distance $h$. This requires only the implementation of a ball query (range search) algorithm—in our case provided by our kd-tree implementation—in conjunction with a simple depth-first traversal of the node set (in any order). This approach is often referred to as *sample elimination*. A strength of our approach is that the resulting set of samples comes from the geometric model, and retains the high-order accuracy conferred by the model. Algorithm 2 clearly only approximately enforces that points be a distance $h$ apart in the *Euclidean* norm, much like Bridson's algorithm for Poisson disk sampling [3]. We will explore the spatial distributions of the node sets produced by Algorithm 2 in section 3.6.1.

**Algorithm 2** Sampling on surfaces.

```
    Given: $\mathcal{X}_d = \{(\boldsymbol{X}_d)_\ell\}_{\ell=1}^{N_d}$, a set of seed nodes on domain boundary.
    Given: $h$, the average separation distance between nodes.
    Given: $\tau$, the supersampling parameter.
    Generate: $\mathcal{X}_b = \{(\boldsymbol{X}_b)_j\}_{j=1}^{N_b}$, a set of boundary nodes with spacing $h$.
    Generate: $\Lambda^e$, a set of parametric evaluation nodes for (1)
    Generate: the SBF geometric model in (1).
 1: Using (1), fit a geometric model to the nodes in $\mathcal{X}_d$.
 2: To find $N_b$ corresponding to a value of $h$, estimate the surface area (or perimeter) $a_d$ of the OBV corresponding to $\mathcal{X}_d$ (generated by Algorithm 3). Then, $N_b = a_d h^{-(d-1)}$.
 3: Evaluate (1) at $\hat{N}_b = \tau N_b$ parametric evaluation points either in $[-\pi, \pi)$ or $[-\pi, \pi) \times [-\pi/2, \pi/2)$. Let $\hat{\mathcal{X}}_b$ be the resulting candidate set of Cartesian evaluation points.
 4: Build a kd-tree on $\hat{\mathcal{X}}_b$.
 5: Initialize $g$, an $\hat{N}_b$ array of flags, to 1.
 6: Store $\hat{\mathcal{X}}_b$ in $\hat{N}_b \times d$ matrix $\hat{X}_b$.
 7: for $k = 1, \hat{N}_b$ do
 8:     if $g(k) \neq 0$ then
 9:         Set idxs = indices of points within distance $h$ of $\hat{X}_b$ (using, for instance, a kd-tree ball query).
10:         Set g(idxs $\neq k$) = 0.
11:     end if
12: end for
13: Set $\hat{g}$ = find ($g = 1$), the vector of indices corresponding to flags of 1.
14: Collect all parametric evaluation nodes corresponding to indices in $\hat{g}$ into array $\Lambda^e$.
15: Evaluate t(1) at $\Lambda^e$ to obtain $\mathcal{X}_b$, the $N_b$ Cartesian points on the boundary.
```

## 3.2. Oriented bounding boxes (OBBs) from principal component analysis (PCA).

Algorithm 1 requires an OBV for node generation. Almost any simple shape will serve as a bounding volume, but we restrict ourselves to OBBs, since intersection tests of vectors against boxes simply involve testing a set of linear inequalities. Further, the Poisson disk sampling algorithm is trivial to implement for a rectangular/cuboidal domain. Our goal is to use a technique that has low computational complexity, is easy to implement, and is meshfree. Further, a technique that produces approximately minimal volume OBBs is desirable, since this

**Algorithm 3** OBB generation using PCA.

```
**Given:** $X_b$, the $N_b \times d$ matrix containing a set of points for which we want an OBB.
**Generate:** $B$, the matrix of OBB vertices.
1: Compute $\boldsymbol{x}_c = \frac{1}{N_b}\left[\sum_{i=1}^{N_b} X_b(i,1)\ \sum_{i=1}^{N_b} X_b(i,2)\ \sum_{i=1}^{N_b} X_b(i,3)\right]$, the centroid of $X_b$.
2: Compute $M = \mathbf{1} \otimes \boldsymbol{x}_c$, the $N_b \times d$ matrix with $\boldsymbol{x}_c$ as each of its rows.
3: Compute $C = \frac{1}{N_b}(X_b - M)^T(X_b - M)$, the $d \times d$ normalized covariance matrix corresponding to $X_b$.
4: Decompose $C$ as $C = VDV^T$, where $V$ contains eigenvectors (rotations about origin), $D$ contains eigenvalues (scaling).
5: Compute unrotated boundary points $\hat{X}_b = X_b V$.
6: Find column minimum and maximum values for $\hat{X}_b$, i.e., two vertices of the unrotated bounding box.
7: Using these two vertices, find the side lengths of the box along each coordinate direction.
8: Using two vertices and side lengths, find the other $2^d - 2$ vertices of the bounding box.
9: Store all unrotated bounding box vertices in $\hat{B}$, a $2^d \times d$ matrix.
10: Compute the OBB vertices $B = \hat{B}V^T$.
```

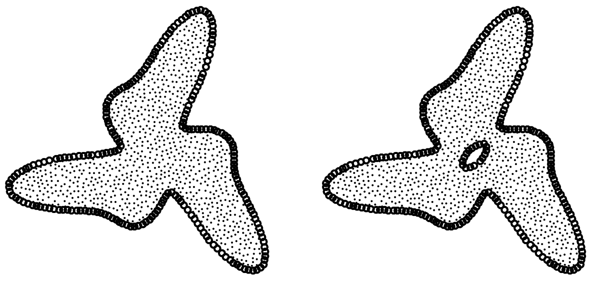
will result in fewer inside/outside tests. PCA fits all our requirements [9], and can easily be used to generate an OBB for our domain in $O(N_b)$ operations. This procedure is summarized in Algorithm 3.

**3.3. Poisson disk sampling.** Algorithm 1 is primarily structured around Poisson disk sampling inside the OBB generated by Algorithm 3. Our approach for Poisson disk sampling in the OBB is a straightforward implementation of the algorithm described in [3]. For completeness, we present this procedure in Algorithm 4, adapted to our notation. The complexity of this algorithm is $O(\hat{k}N_{obb})$. Since $\hat{k}$ is a constant, the cost scales as $O(N_{obb})$. In section 3.6, we will explore the impact of $\hat{k}$ on the uniformity of node distributions and on the accuracy for RBF-FD discretizations of PDEs. We explore the node distributions obtained from the use of Algorithm 4 within Algorithm 1 in section 3.6.2.

**3.4. Domains with time-varying embedded boundaries.** Biological problems often involve domains with not only irregular outer boundaries, but irregular embedded inner boundaries, e.g., platelets and red blood cells in a blood vessel [34]. While the authors have begun developing numerical methods for simulations in such complex domains [36, 31, 32], our methods currently lack a robust node generation algorithm that can modify its node sets locally when the embedded inner boundaries deform, and are added or removed. Fortunately, Algorithm 1 is easily modified to tackle this problem. Consider the scenario where Algorithm 1 has generated node sets $\mathcal{X}_b$ and $\mathcal{X}_i$ on the boundary and interior of an irregular domain $\Omega_0$ without any embedded boundaries. Now consider the inclusion of irregular boundaries $\Gamma = \{\Gamma_j\}_{j=1}^{N_p}$ enclosing domains $\{\Omega_j\}_{j=1}^{N_p}$ into $\Omega_0$ in such a way that these boundaries define a new domain $\Omega = \Omega_0 \backslash \bigcup_{j=1}^{N_p} \Omega_j$. Our node generator must automatically and efficiently generate a node set in $\Omega$ that *deletes* the nodes contained in $\tilde{\Omega} = \bigcup_{j=1}^{N_p} \Omega_j$. In fact, our algorithm also deletes any nodes that are within a distance $h$ of the embedded boundaries, again ensuring that the global separation distance is approximately $h$. This is detailed in Algorithm 5. This procedure involves only locally changing the

**Algorithm 4** Fast Poisson disk sampling.

```
    Given: $h$, the minimum distance between nodes.
    Given: $d$, the spatial dimension.
    Given: $\hat{k}$, the number of samples to choose before rejection (the Poisson neighborhood size).
    Given: $B$, the matrix of vertices of the domain OBB.
    Generate: The set of $N_{obb}$ Poisson disk samples in the domain OBB.
 1: Initialize a $d$-dimensional Cartesian background grid $\mathcal{G}$ with cell size $\frac{h}{\sqrt{d}}$. A value of $-1$ in $\mathcal{G}$ indicates no sample, any nonnegative integer is the linear index of the sample in a cell.
 2: Generate the first uniform random node $\boldsymbol{x}_0$ within the domain OBB. Insert this node into $\mathcal{G}$.
 3: Initialize the array of node indices (active list) $\mathcal{I}$ with the index of 0 (corresponding to $\boldsymbol{x}_0$).
 4: while $\mathcal{I}$ is not empty do
 5:     Choose a random index $i$ from $\mathcal{I}$.
 6:     Generate $\hat{k}$ uniform random nodes in the spherical annulus centered at $\boldsymbol{x}_i$ with inner radius $h$ and outer radius $2h$. Place these nodes in set $\mathcal{P}$.
 7:     Compare all of $\mathcal{P}$ against existing nodes within distance $h$ (using $\mathcal{G}$ to facilitate comparisons).
 8:     If none of the nodes in $\mathcal{P}$ are sufficiently far from existing samples, remove $i$ from the active list.
 9:     If any of the nodes in $\mathcal{P}$ are sufficiently far from existing samples, add their indices to the active list, store these valid samples in the set $\mathcal{V}$.
10:     If any of the nodes in $\mathcal{P}$ are outside the domain OBB, reject them.
11: end while
12: The set of $N_{obb}$ Poisson disk samples is $\mathcal{V}$.
```



(a) Star domain (b) Star with embedded ellipse

Fig. 3. *Local node modification for embedded boundaries using Algorithm* 5. *The embedded ellipse is shown using circles.*

original node set as new objects are introduced, and also accounts for cases where an embedded boundary may touch the domain outer boundary. It is important to note that this algorithm features potential early termination: a point is only tested against bounding boxes and embedded boundaries until it is found to be contained within an embedded boundary. While other efficiency improvements are possible, we forgo them for simplicity. Figure 3 shows an example of the node sets obtained using Algorithm 5. In Figure 3(a), we show the interior nodes on the star domain. Figure 3(b)

**Algorithm 5** Node set modification for (closed) embedded boundaries.

```
    Given: $h$, the average separation distance between nodes in domain $\Omega_0$.
    Given: $\alpha$, the ratio of the numbers of inner boundary nodes to outer boundary nodes (see section 4.2).
    Given: $\chi_b = \{(\boldsymbol{X}_b)_j\}_{j=1}^{N_b}$, a set of boundary nodes on $\partial\Omega_0$.
    Given: $\eta_b$, the set of outward unit normals on the boundary $\partial\Omega_0$.
    Given: $\chi = \{\boldsymbol{x}_k\}_{k=1}^{N}$, the set of nodes on $\Omega_0$ (including the boundary).
    Given: $N_d^\Gamma$ seed nodes on each of the embedded boundaries $\Gamma_j, j = 1, \ldots, N_p$.
    Generate: $Z$, the $N \times 2$ array whose first column indicates whether a point $\boldsymbol{x}_k \in \Omega$, and second column is $j$ if $\boldsymbol{x}_k \in \Omega_j$.
    Generate: $\tilde{\chi}$, the (modified) set of $\tilde{N}$ nodes on the irregular domain $\Omega = \Omega_0 \backslash \tilde{\Omega}$.
 1: Using (1), fit a geometric model to the seed nodes on each embedded boundary $\Gamma_j, j = 1, \ldots, p$.
 2: Set $N_b^\Gamma = \alpha N_b$.
 3: For $j = 1, \ldots, N_p$, obtain $\chi_b^{\Gamma_j}$, the $N_b^\Gamma$ boundary points on the $j$th embedded boundary using Algorithm 2.
 4: Evaluate derivatives of the $N_p$ boundary interpolants at $N_b^\Gamma$ parametric points to obtain $\eta_b^{\Gamma_j}$, the set of unit normals on each boundary (pointing into $\Omega_0$).
 5: Extend each embedded boundary $\Gamma_j$ by distance $h$ in their normal directions to obtain $\tilde{\Gamma}_j$ and the corresponding points $\chi_b^{\tilde{\Gamma}_j}$.
 6: Let $\chi_b^{\tilde{\Gamma}} = \bigcup_{j=1}^{N_p} \chi_b^{\tilde{\Gamma}_j}$.
 7: Add $\chi_b^{\tilde{\Gamma}}$ to the domain kd-tree currently containing $\chi_b$.
 8: Generate the OBB for each $\tilde{\Gamma}_j$ using Algorithm 3 on the $N_b^\Gamma$ boundary points.
 9: Initialize the 2D array (map) Z with dimensions $N \times 2$ to zeros.
10: for $k = 1, N$ do
11:     for $j = 1, N_p$ do
12:         if $\boldsymbol{x}_k$ is within the $j$th OBB then
13:             Find the closest point $\zeta$ on $\tilde{\Gamma}_j$ using the domain kd-tree.
14:             Use $\zeta$ to test $\boldsymbol{x}_k$ against $\tilde{\Gamma}_j$.
15:             if $\boldsymbol{x}_k$ is inside $\tilde{\Gamma}_j$ then
16:                 Set $Z(k,1) = 1$, $Z(k,2) = j$.
17:                 Break and go to next $k$ value.
18:             end if
19:         end if
20:     end for
21: end for
22: Let $g = \text{find } (Z(:,1) = 0)$ be the list of indices of points with $\Omega$.
23: Set $\tilde{\chi} = \chi(g = 0)$, $\tilde{N} = \text{length } (\tilde{\chi})$.
```

shows the node set obtained when an ellipse is embedded into the star domain. It is easy to see that Algorithm 5 only locally modifies node sets, as desired.

Algorithm 5 does not take into account the case where an embedded boundary is removed. There are two options within the above framework for modifying the node sets in such a situation. The first option is to fully generate a new node set on $\Omega$ after an embedded boundary is removed; this would involve using Algorithm 5 to modify the original node set on $\Omega_0$ to account for the embedded boundaries that were *not removed.* This option is rather wasteful. Instead, Algorithm 5 associates each grid point $\boldsymbol{x}_k$ with an embedded boundary $\Omega_j$. Thus, to handle a *removed* embedded boundary $\Omega_j$, we need only find all nodes in the original node set that are contained within $\Omega_j$ by searching the second dimension of the array $Z$ for the value $j$. Of course, this second option relies on knowing which of the embedded boundaries were removed. If this is not known, the first option discussed above is preferable.

**3.5. Boundary refinement and ghost nodes.** The numerical solution of PDEs with RBF-FD sometimes requires denser node sets near domain boundaries [11, 12, 31], and/or ghost nodes outside the domain boundary to enforce boundary conditions [11, 1, 32]. Our node generator must, therefore, possess these capabilities. Generating ghost nodes is straightforward: given a set of boundary nodes and unit outward normals, simply copy the nodes a distance $h$ outside the domain along the outward normals. Further, Algorithm 1 can be easily modified to provide a node set refined near a boundary: simply copy the boundary nodes at some user-specified distance inward from the boundary. This places a layer of nodes between $\mathcal{X}_b$ and the inner boundary $\hat{\mathcal{X}}_b$. This approach generalizes straightforwardly to multiple layers of boundary refinement. However, a drawback of this approach is that it does not generate *graded* boundary-refined node sets, i.e., node sets that smoothly vary in space. We leave this generalization for future work.

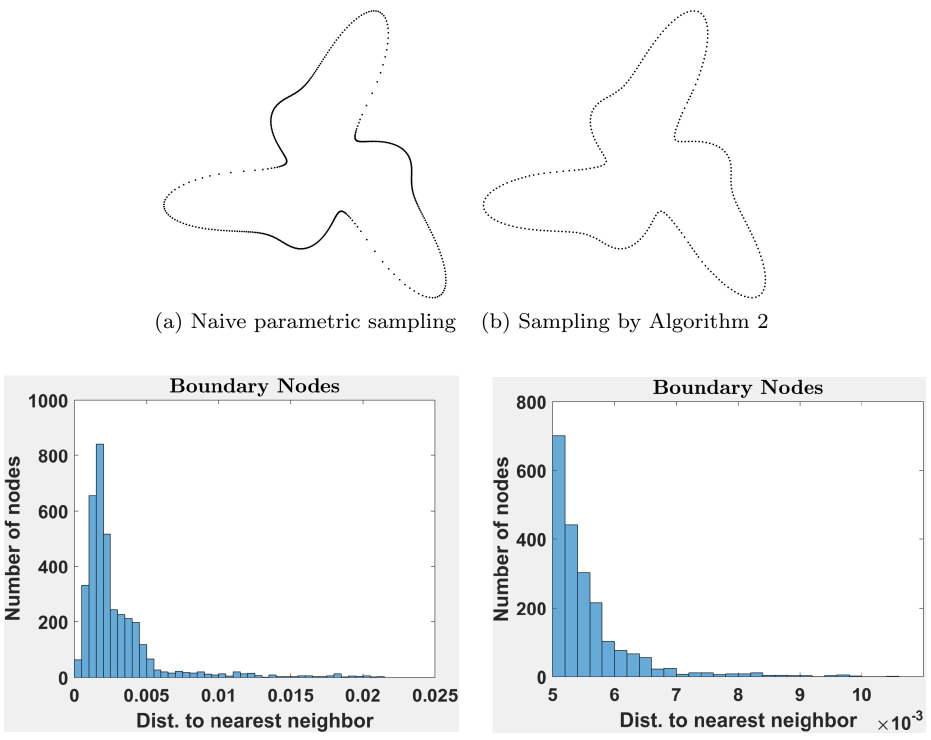
**3.6. Some results.** We now study three aspects of our algorithm. First, we study the spatial distributions of domain boundary nodes obtained from Algorithm 2. Next, we explore the influence of the Poisson neighborhood size $\hat{k}$ on the spatial distributions of interior nodes obtained from Algorithm 4. Finally, we explore the effect of the Poisson neighborhood size $\hat{k}$ on the stability and errors in an RBF-FD discretization.

**3.6.1. Spatial distribution of boundary nodes.** We first explore the effect of Algorithm 2 on the node distributions produced on surfaces, and compare it to a naive sampling of the parametric map. To briefly illustrate the efficacy of this procedure, we sample two functions (corresponding to domain boundary s) whose maps from parametric space to Cartesian space are likely to produce distorted point sets. The 2D shape (the star domain) is given by the following $C^0$ function:

$$r(\lambda) = |\cos(1.5\lambda)|^{\sin(3\lambda)}, \tag{10}$$

$$x(\lambda) = r(\lambda)\cos(\lambda), y(\lambda) = r(\lambda)\sin(\lambda), \tag{11}$$

where $\lambda \in [0, 2\pi)$. The three dimensional (3D) shape is a classic red blood cell shape, obtained by smoothly distorting the sphere, and is therefore $C^\infty$ [21, 24]. For this test, we interpolate the star shape with the SBF interpolant at $N_d = 128$ points (due to its low smoothness) and attempt to generate Cartesian samples with an average node spacing of $h = 0.005$. We interpolate the 3D shape with the SBF interpolant at $N_d = 700$ points, and attempt to generate Cartesian samples with an average node spacing of $h = 0.05$. The resulting node sets and their histograms are shown in Figures 4 and 5. Figure 4(a) clearly shows the clustering in the naive sampling technique for the star domain, corroborated by the histogram in Figure 4(c) which shows a large number of nodes with a distance of $h = 0.0025$ to the nearest neighbor (rather than the desired $h = 0.005$). Examining the histogram, we also see that the nodes are also sometimes very far apart (as far apart as $h = 0.02$). In contrast, Algorithm 2 produces much more uniform node sets, which can be verified visually (Figure 4(b)) and from the corresponding histogram (Figure 4(d)). In this case, the histogram shows the greatest number of nodes in the bin corresponding to $h = 0.005$, with exponentially fewer nodes in bins corresponding to larger $h$ values. The 3D



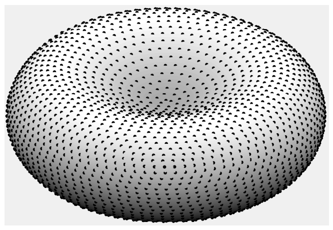
(a) Naive parametric sampling (b) Sampling by Algorithm 2

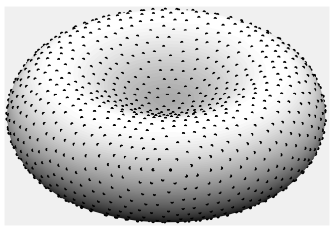
(c) Histogram for $h = 0.005$ (naive sampling) (d) Histogram for $h = 0.005$ (Algorithm 2)

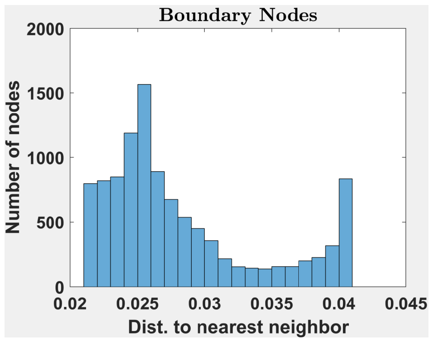
FIG. 4. *Comparison of surface sampling techniques for the star domain using supersampling parameter* $\tau = 2$.

results are shown in Figures 5(a) and 5(b). Figure 5(c) (naive sampling) shows that most nodes are in a bin close to $h = 0.03$, but are distributed over a relatively wide range. With naive sampling, there appear to be no nodes whatsoever in the desired $h = 0.05$ bin! In marked contrast, the histogram in Figure 5(d) (corresponding to sampling by Algorithm 2) shows the most number of nodes in the $h = 0.05$ bin (as desired), with exponentially fewer nodes in bins corresponding to larger $h$ values. In all cases, Algorithm 2 still appears to produce quasi-uniform node sets.

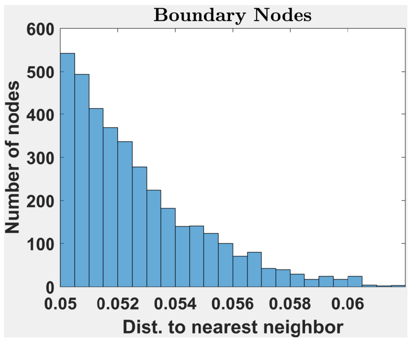
**3.6.2. Spatial distribution of interior nodes.** We turn our attention to the node sets produced by Algorithm 1 in the interiors of domains. Recall that this is accomplished by applying Algorithm 4 in the domain bounding box to obtain $N_{obb}$ nodes, then eliminating those nodes that lie outside the parametric SBF interpolant to the boundary. It is important to note that Algorithm 4 utilizes the Poisson neighborhood size $\hat{k}$ to control the cost of sampling. However, $\hat{k}$ also determines the number of nodes against which samples are compared to specify the desired separation distance $h$. Indeed, one could imagine setting $\hat{k} = N_{obb}$ to obtain a perfectly spaced set of nodes at a cost of $O(N_{obb}^2)$ for node generation. Since $\hat{k}$ controls both computational cost and spatial distribution, it is important to explore the effect of this parameter on spatial distributions of nodes. To do so, we compute histograms of distances to the nearest neighbor for node sets in the interior of the star domain with $h = 0.005$ (2 dimensions), and the red blood cell with $h = 0.05$ (3 dimensions). The results are shown in Figure 6. From Figure 6, it is clear that Algorithm 1 produces node sets that are quasi-uniform regardless of the $\hat{k}$ value used. Most nodes lie in the desired


(a) Naive parametric sampling


(b) Sampling by Algorithm 2



(c) Histogram for $h = 0.05$ (naive sampling)



(d) Histogram for $h = 0.05$ (Algorithm 2)

FIG. 5. *Comparison of surface sampling techniques for the red blood cell domain using supersampling parameter $\tau = 2$.*

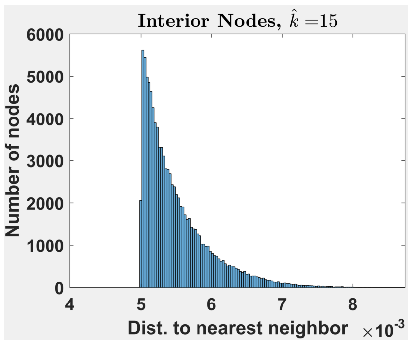
bins of $h = 0.005$ (in 2 dimensions) and $h = 0.05$ (in 3 dimensions), and the number of nodes in bins corresponding to larger values of $h$ drops off exponentially in both 2 and 3 dimensions. Very similar node distributions were obtained when objects were embedded within (results not shown).

**3.6.3. Stability for RBF-FD discretizations.** Our motivation for designing Algorithm 1 was at least partly because nodes obtained from popular node generators such as Gmsh were not always suitable for RBF-FD discretizations. Figure 7 shows an example of this. A sixth-order RBF-FD discretization of the Laplacian in the unit ball resulted in eigenvalues with positive real parts when using a Gmsh-generated node set with $N = 4561$ nodes (Figure 7(a)). In contrast, the same high-order discretization on $N = 5157$ nodes generated by Algorithm 1 results in a well-behaved spectrum (Figure 7(b)) with a relatively small spread even along the imaginary axis. We note that it may indeed be possible to obtain appropriate nodes for RBF-FD discretizations from Gmsh if the right parameters and algorithms are used. However, Algorithm 1 required no fine tuning for this example.

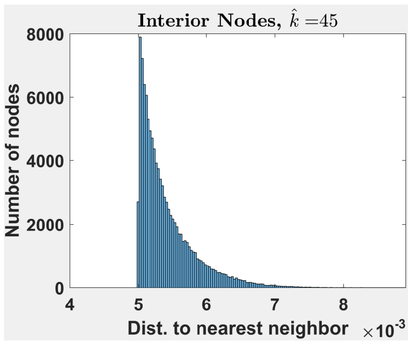
**3.6.4. Errors in RBF-FD discretizations.** The goal of this article was to design a node generator that produces node sets suitable for RBF-FD discretizations. In this section, we provide some evidence as to that suitability by exploring the effects of the parameter $\hat{k}$ on the errors and convergence rates from RBF-FD discretizations of the heat equation:

$$\frac{\partial c(\boldsymbol{x},t)}{\partial t} = \Delta c + f(\boldsymbol{x},t), \boldsymbol{x} \in \Omega, \tag{12}$$

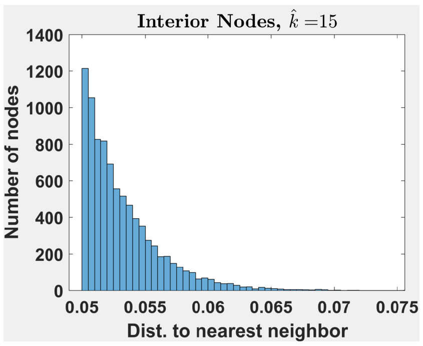
$$\frac{\partial c(\boldsymbol{x},t)}{\partial \boldsymbol{n}} = g(\boldsymbol{x},t), \boldsymbol{x} \in \gamma, \tag{13}$$



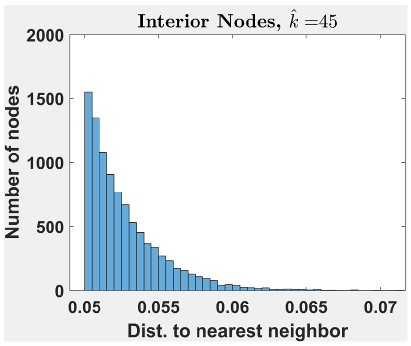
(a) Star, $\hat{k} = 15$, $h = 0.005$



(b) Star, $\hat{k} = 45$, $h = 0.005$

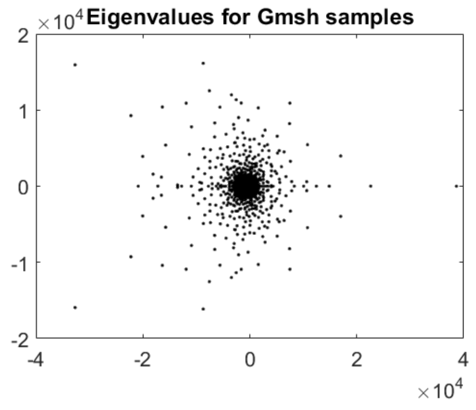


(c) Red blood cell, $\hat{k} = 15$, $h = 0.05$

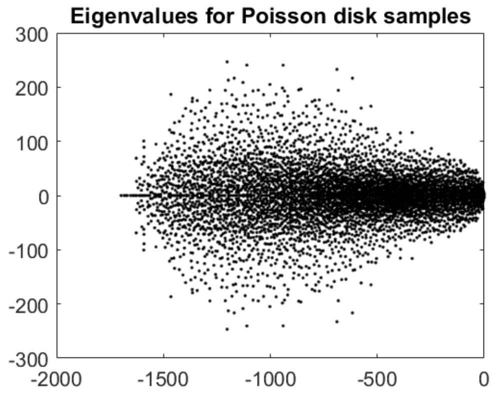


(d) Red blood cell, $\hat{k} = 45$, $h = 0.05$

FIG. 6. *Histograms of node sets generated by Algorithm* 1 *in the interiors of the star domain (top row) and the red blood cell domain (bottom row) as a function of the Poisson neighborhood size* $\hat{k}$.



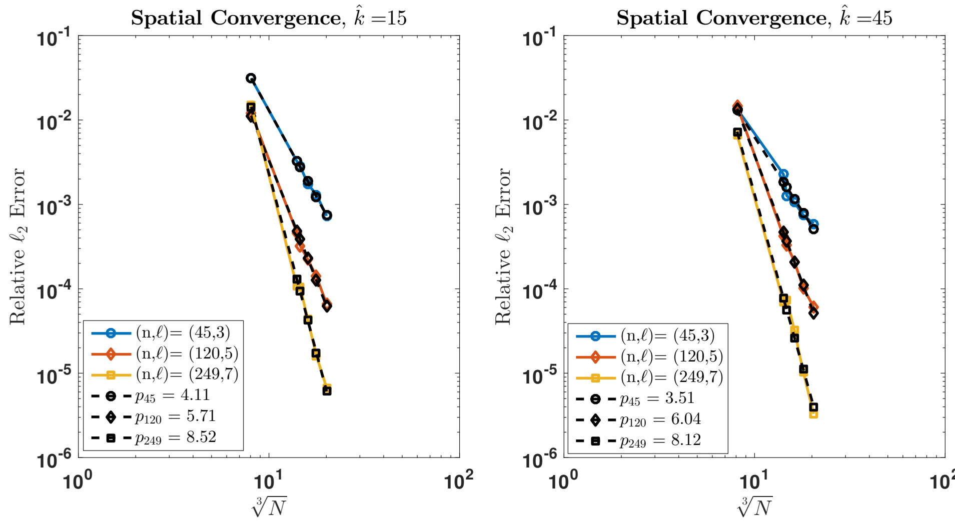
(a) Eigenvalues on Gmsh nodes



(b) Eigenvalues on Algorithm 1 nodes

FIG. 7. *Eigenvalues of the discrete Laplacian formed by a sixth-order RBF-FD method in the unit ball for nodes obtained from Gmsh (left) and Algorithm* 1 *with* $\hat{k} = 15$ *and* $\tau = 2$ *(right).*

where $\Omega$ is the red blood cell domain seen earlier, and $\gamma$ is its curved boundary. We have restricted ourselves to a 3D test for brevity. We use $N_d = 800$ seed nodes as the starting point for the node generator; this is sufficient to ensure that the errors



(a) Convergence rates for $\hat{k} = 15$ (b) Convergence rates for $\hat{k} = 45$

FIG. 8. *Errors in the solution to* (12)–(13) *in the red blood cell domain* ($N_d = 800$) *as a function of the stencil size n, number of nodes N, polynomial degree $\ell$, and Poisson neighborhood size $\hat{k}$.*

in geometric modeling match the high orders of accuracy in RBF-FD. To measure convergence rates, we manufacture a solution to the heat equation by prescribing $c(\boldsymbol{x}, t)$, then computing a corresponding $f(\boldsymbol{x}, t)$ and $g(\boldsymbol{x}, t)$ that make the prescribed solution hold. We use the infinitely smooth ansatz

$$c(\boldsymbol{x}, t) = c(x, y, z, t) = 1 + \sin(\pi x)\cos(\pi y)\sin(\pi z)e^{-\pi t}. \tag{14}$$

For the spatial discretization, we use the overlapped RBF-FD method [31], and the ghost node technique outlined in [32]. The RBF-FD discretization is achieved by using PHS RBFs in conjunction with high-degree polynomials, where the convergence rates are dictated purely by the polynomial degree [12, 11, 1]. Since these discretization techniques are explained in detail in the references, we restrict ourselves to a description of our results. Primarily, when increasing the RBF-FD stencil size $n$ and consequently the polynomial degree $\ell$, we expect to see higher orders of convergence. We run these convergence studies and calculate $\ell_2$ errors in the numerical solution for different values of $n$, $\ell$, $\hat{k}$, and $h \propto \frac{1}{\sqrt[3]{N}}$. The results are shown in Figure 8. Studying Figures 8(a) and 8(b) closely, we see slightly different convergence rates between the two figures for the same stencil size and polynomial degree. From the data (not shown), the errors for $\hat{k} = 45$ appear to be slightly lower than the errors for $\hat{k} = 15$, but not significantly so at higher values of $N$. This is not surprising, given the similarity in the histograms for these two $\hat{k}$ values, but it is reassuring since it indicates that we can use $\hat{k} = 15$ and still achieve low errors and high orders of convergence on fairly irregular domains.

## 4. Complexity analysis.

We now derive the computational complexity of Algorithms 1 and 5.

### 4.1. Complexity of one-time node generation.

The total computational cost of our node generator is a sum of the individual costs of the following steps:

- Forming a kd-tree on the domain boundary: The kd-tree is formed on $N_b$ points for a cost of $C_1 = O(dN_b \log N_b)$. Each subsequent nearest neighbor search costs $O(\log N_b)$ operations.
- SBF boundary representation: Since this involves $N_d$ seed nodes, it requires the solution of the $N_d \times N_d$ dense linear system (3), which has an asymptotic cost of $O(N_d^3)$. This interpolant is evaluated to obtain $\tau N_b$ boundary points for a cost of $O(N_b N_d)$, then thinned for a worst-case cost of $O(N_b \log N_b)$ using the kd-tree. If we assume $N_d = \gamma N_b$, where $\gamma \in (0, 1]$ is some small number, the evaluation cost can be written as $O(\gamma N_b^2)$. Cost: $C_2 = O(\gamma^3 N_b^3) + O(\gamma N_b^2) + O(N_b \log N_b)$.
- OBB: The primary cost in the PCA computation of the OBB is the $O(d^2 N_b)$ cost of forming the $d \times d$ covariance matrix. The subsequent eigendecomposition of the $d \times d$ matrix is computed in $d^3$ flops. Then, the rotations and max/min computations to find the unrotated bounding box cost $O(N_b)$. The costs of rotations of the bounding boxes can be neglected when $d = 2, 3$. Cost: $C_3 = O(N_b)$.
- Poisson disk sampling in the OBB: Using Algorithm 4, Poisson disk samples for the OBB can be computed for a cost of $C_4 = O(N_{obb})$.
- Eliminate OBB nodes outside the domain boundary: This requires finding the closest boundary point to each of the $N_{obb}$ Poisson disk samples in the OBB using the kd-tree. The total cost is therefore given by $C_5 = O(N_{obb} \log N_b)$.

The total cost is given by $C_{tot} = \sum_{i=1}^{5} C_i$. Dropping the big-$O$ notation for convenience, we have

$$C_{tot} = dN_b \log N_b + \gamma^3 N_b^3 + \gamma N_b^2 + N_b \log N_b + N_b + N_{obb} + N_{obb} \log N_b. \tag{15}$$

All our costs are currently in terms of $N_b$ and $N_{obb}$. However, it is more appropriate to express $C_{tot}$ in terms of the total number of points in the domain $N = N_i + N_b$. This requires dimension-specific simplifications.

### 4.1.1. 2D complexity estimates.

We now present the complete 2D complexity estimates in terms of $N$. Assuming the nodes are approximately uniformly spaced on both the boundary and the interior, we have

$$h_i = \left(\frac{a}{N_i}\right)^{\frac{1}{2}}, h_b = \frac{q}{N_b}. \tag{16}$$

If we wish the nodes to be spaced comparably on the boundary and interior, it is reasonable to assume that $h_i = h_b = h$. Solving for $N_b$ then gives

$$N_b = qa^{-\frac{1}{2}} N_i^{\frac{1}{2}}. \tag{17}$$

Substituting (17) into (15) and neglecting sublinear terms in $N_i$ gives us

$$C_{tot} = \gamma^3 q^3 a^{-\frac{3}{2}} N_i^{\frac{3}{2}} + \gamma \frac{q^2}{a} N_i + 2qa^{-\frac{1}{2}} N_i^{\frac{1}{2}} + 2N_{obb}. \tag{18}$$

We must now relate $N_{obb}$ to $N_i$ to get the total cost in terms of $N_i$. We know that

$$h_i = \left(\frac{a}{N_i}\right)^{\frac{1}{2}} = \left(\frac{a_{obb}}{N_{obb}}\right)^{\frac{1}{2}}, \tag{19}$$

$$\implies N_{obb} = \frac{a_{obb}}{a} N_i. \tag{20}$$

Substituting this new expression into (18) and retaining only linear and higher-order terms in $N_i$, we obtain

$$C_{tot} = \gamma^3 q^3 a^{-1.5} N_i^{1.5} + \gamma \frac{q^2}{a} N_i + 2\frac{a_{obb}}{a} N_i. \tag{21}$$

Since $N = N_i + N_b$ and $N_b << N_i$, it is reasonable to replace $N_i$ by $N$, giving us

$$C_{tot} = \gamma^3 q^3 a^{-1.5} N^{1.5} + \gamma \frac{q^2}{a} N + 2\frac{a_{obb}}{a} N. \tag{22}$$

The constant in front of the leading-order $N^{1.5}$ term is typically small in practice for large $N$ and smooth domain boundaries. In such a case, the $O(N)$ terms dominate. Thus, for small $\gamma$ and reasonably compact domains, we have

$$C_{tot} \approx \left(\gamma \frac{q^2}{a} + 2\frac{a_{obb}}{a}\right) N. \tag{23}$$

Our node generator thus has a theoretical computational complexity of $O(N)$ in 2 dimensions, despite the use of a global RBF interpolant on the boundary. As we show in section 4.3, we recover the $O(N)$ complexity in practice.

**4.1.2. 3D complexity estimates.** We proceed much in the same way as in the 2D case, but obtain a different estimate. First, assuming the nodes are approximately uniformly spaced on both the boundary and the interior, we now have

$$h_i = \left(\frac{v}{N_i}\right)^{\frac{1}{3}}, h_b = \left(\frac{s}{N_b}\right)^{\frac{1}{2}}. \tag{24}$$

Now, letting $h_i = h_b = h$, solving for $N_b$ gives

$$N_b = s v^{-\frac{2}{3}} N_i^{\frac{2}{3}}. \tag{25}$$

Using (25) in (15) and neglecting sublinear terms in $N_i$, we have

$$C_{tot} = \gamma^3 s^3 v^{-2} N_i^2 + \gamma s^2 v^{-\frac{4}{3}} N_i^{\frac{4}{3}} + 2N_{obb}. \tag{26}$$

Once again relating $N_{obb}$ to $N_i$ using the volume of the OBB $v_{obb}$, we have

$$C_{tot} = \gamma^3 s^3 v^{-2} N_i^2 + \gamma s^2 v^{-\frac{4}{3}} N_i^{\frac{4}{3}} + 2\frac{v_{obb}}{v} N_i. \tag{27}$$

For a sufficiently large $N_b$ (small $h$) and fixed $N_d$, $\gamma$ can be as low as $O(10^{-3})$. Thus, both the quadratic and superlinear terms in the above expansion may be small for a compact domain, leaving us with

$$C_{tot} \approx 2\frac{v_{obb}}{v} N_i \approx 2\frac{v_{obb}}{v} N. \tag{28}$$

Even if the superlinear term is retained, this represents a complexity of $O(N^{1.\bar{33}})$. However, in practice, we observe close to $O(N)$ complexity, as we show in section 4.3.

**4.2. Complexity of node set modification.** We now derive the complexity associated with Algorithm 5. The total *worst-case* computational cost for removing nodes from the interior of $N_p$ embedded boundaries can be written as

$$C_{mod} = O(N_p(N_d^\Gamma)^3) + O(N_p N_b^\Gamma (N_d^\Gamma)^2) + O(N_p N_b^\Gamma) + O(N N_p), \tag{29}$$

where the first term corresponds to fitting the SBF geometric model to each embedded boundary, the second term to evaluations of the models to obtain normals, the third term to the computation of the OBBs of the embedded boundaries, and the last term to testing each of the $N$ domain nodes against the $N_p$ OBBs. Now, let

$$N_b^\Gamma = \alpha N_b, N_d^\Gamma = \alpha N_d. \tag{30}$$

If approximately the same spacing between nodes is maintained on all boundaries, we have $h_\Gamma = h_b$, which in 2 dimensions gives us $\alpha = \frac{q^\Gamma}{q}$ (rounded to the nearest integer). Similarly, in 3 dimensions, we have $\alpha = \frac{s^\Gamma}{s}$. Dropping the big-$O$ notation for convenience, $C_{mod}$ can be written as

$$C_{mod} = \alpha\gamma^2(\alpha^2\gamma + 1)N_p N_b^3 + \alpha N_p N_b + N N_p. \tag{31}$$

Using the fact that $N = N_i + N_b$, (31) can be written as

$$C_{mod}^{2\mathrm{D}} = \alpha\gamma^2(\alpha^2\gamma + 1)N_p q^3 a^{-1.5} N_i^{1.5} + \alpha N_p q a^{-\frac{1}{2}} N^{\frac{1}{2}} + N N_p. \tag{32}$$

By a similar argument as in section 4.1, the $N_i^{1.5}$ term is small, leaving an estimate of

$$C_{mod}^{2\mathrm{D}} \approx N N_p, \tag{33}$$

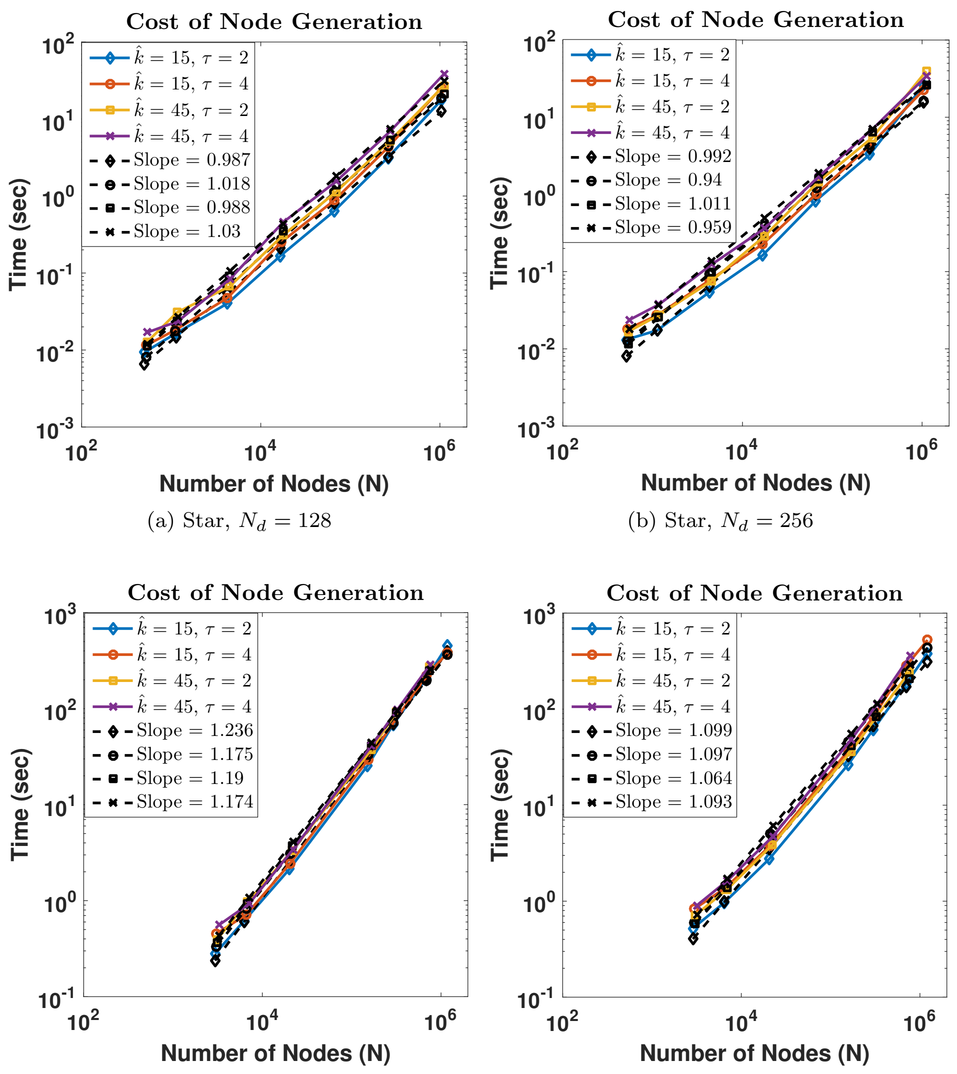
where sublinear terms in $N$ and $N_i$ have been neglected. Since the number of embedded boundaries is always much smaller than the number of nodes, i.e., $N_p << N$, we have $C_{mod}^{2\mathrm{D}} = O(N)$. The 3D derivation is similar. Using the definition of $N_b$ in terms of $N_i$ in 3 dimensions, (31) can be rewritten as

$$C_{mod}^{3\mathrm{D}} = \alpha\gamma^2(\alpha^2\gamma + 1)N_p s^3 v^{-2} N_i^2 + \alpha N_p s v^{-\frac{2}{3}} N^{\frac{2}{3}} + N N_p. \tag{34}$$

Again, in practice, the linear term in $N$ dominates. Thus, we have

$$C_{mod}^{3\mathrm{D}} \approx N N_p, \tag{35}$$

which is in practice $O(N)$ since $N_p << N$. If a bounding box hierarchy is used to keep track of the embedded boundaries, the $O(NN_p)$ term can be further shrunk to $O(N \log N_p)$, but we leave this approach for future work. It is important to note that our estimates for node generation only constitute the worst case. In general, it is unlikely that a Poisson disk sample needs to be tested against *all* OBBs of the embedded inner boundaries. Indeed, the disk sample is deleted as soon as it is found to lie within any embedded boundary, requiring no additional tests. The true complexity estimate in this scenario is nondeterministic. We leave this analysis for future work also.

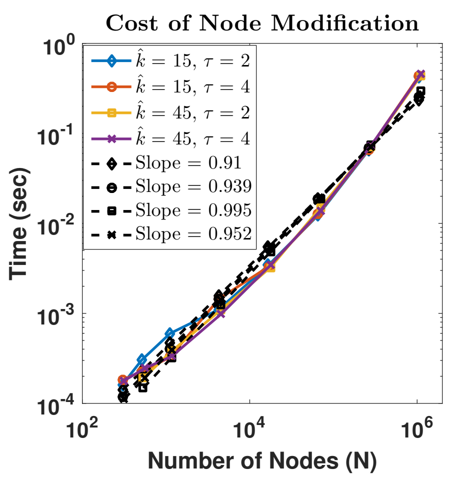


(a) Star, $N_d = 128$ (b) Star, $N_d = 256$

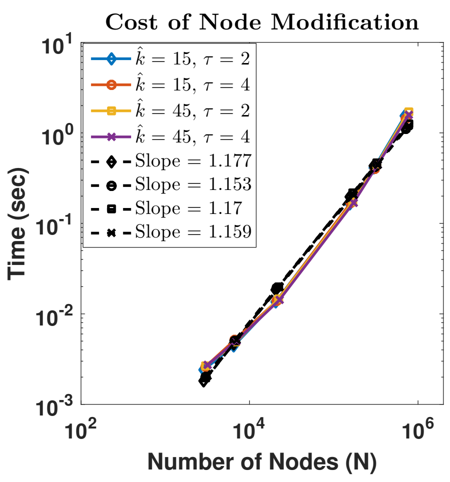
(c) Bumpy sphere, $N_d = 400$ (d) Bumpy sphere, $N_d = 800$

FIG. 9. *Cost of node generation using Algorithm* 1 *on* 2*D (top row) and* 3*D (bottom row) irregular domains (on a loglog plot). The timings are shown as a function of the Poisson neighborhood size $\hat{k}$, the supersampling parameter $\tau$, the number of data sites $N_d$, and the total number of nodes $N$. The dashed lines are lines of best fit used to measure slopes.*

**4.3. Scaling of the node generator.** We now present scaling results (in the form of timings) for our node generator. All timings were done using a C/C++ code on the six-core Intel Coffee Lake 8700-K (12 logical cores with hyperthreading) clocked at 4.56 GHz, with 16 GB of 2600 MHz DDR4 RAM. We focus on the star domain from section 3 and the bumpy sphere domain from [21, 37]. We perform two experiments: the first measures the cost of node generation in these two irregular domains, while the second measures the cost of modifying these node sets with embedded inner boundaries. The results of the first experiment are shown in Figure 9, and the results from the second experiment are shown in Figure 10.



(a) Star with embedded ellipse



(b) Bumpy sphere with embedded red blood cell

FIG. 10. *Cost of node set* modification *using Algorithm* 5 *on* 2D *(left) and* 3D *(right) irregular domains (on a loglog plot). The number of inner boundary data nodes is fixed to* $N_d^\Gamma = 24$ *in* 2 *dimensions and* $N_d^\Gamma = 200$ *in* 3 *dimensions. The timings are shown as a function of the Poisson neighborhood size* $\hat{k}$*, the supersampling parameter* $\tau$*, and the total number of nodes* $N$ *(measured prior to modification). The dashed lines are lines of best fit used to measure slopes.*

Figure 9 verifies that the cost of node generation is indeed approximately $O(N)$ in both 2 dimensions (Figure 9(a)) and 3 dimensions (Figure 9(b). The cost in each case goes up slightly as $N_d$, $\hat{k}$, and $\tau$ are increased (though this is hard to see from the graphs), but the slopes remain approximately linear. Code profiling shows that the bulk of the time is taken up by kd-tree operations for large $N$. We also verified that the geometric modeling costs scaled as $O(N_b N_d)$ (not shown). We return to this in the discussion section. For the next experiment, we place a single ellipse tilted at an angle of $\pi/4$ with respect to the $x$-axis at the center of the star domain; for the 3D analogue of this experiment, we place a single red blood cell in the interior of the bumpy sphere domain. We then measure the costs of modifying the node sets as the node spacing $h$ is decreased and $N$ is increased. It is important to note that this also has the effect of increasing the number of nodes on the boundaries of these inner embedded objects. The results are shown in Figure 10 for different values of $\hat{k}$ and $\tau$. Both the 2D and 3D experiments (Figures 10(a) and 10(b), respectively) show once again that our asymptotic estimates hold true: we see that the cost of node set modification is approximately $O(N)$ (see figure captions). Obviously, the precise cost depends on the shape, volume, and number of the inner embedded boundaries. Nevertheless, the experiment verifies that one can indeed modify these node sets much more quickly than one can generate them, reflecting the local nature of our node set modification algorithm. From these results, the results from sections 3.6.1–3.6.4, we conclude that it is reasonable to set $\tau = 2$ and $\hat{k} = 15$ for our applications. It may be reasonable to adapt $\tau$ according to the local curvature of the domain boundary. We leave this for future work.

**5. Discussion.** In this article, we presented new algorithms for one-shot node generation on irregular domains. The algorithm utilizes a parameter-free high-order boundary representation based on SBF interpolation, with a simple strategy for obtaining quasi-uniform node sets on domain boundaries. When used in conjunction with a bounding box based on PCA, this allowed for the use of Poisson disk sampling without hand tuning. The simplicity of the algorithm allowed for a straightforward complexity analysis of our algorithms in 2 and 3 dimensions. We demonstrated that our algorithms achieve a scaling of $O(N)$ both in theory and in practice. Further, we demonstrated that our algorithm for local node set modification also exhibited a complexity of $O(N)$, albeit with a much smaller constant than in the node generation case. We demonstrated that the node sets were quasi-uniform, and that they were suitable for RBF-FD discretizations even for the value of $\hat{k} = 15$ and $\tau = 2$.

Profiling revealed that our kd-tree implementation was the primary bottleneck in our algorithm, with the nearest neighbor search and ball query operation costs dominating the cost of node generation. A faster data structure for these operations would likely significantly improve the wall-clock times for node generation and modification. Similarly, a bounding box hierarchy would enable rapid node set modification when hundreds of time-varying embedded boundaries are involved. These are areas of future research. The second significant bottleneck is the high-order boundary representation, with asymptotic costs of $O(N^{1.5})$ in 2 dimensions and $O(N^2)$ in 3 dimensions. Though the constants in front of these terms are small, it may still be desirable to use a high-order boundary representation that costs $O(N_d)$ operations to obtain; this should also speed up evaluation of the interpolant significantly. The first author plans to pursue this area of research. Finally, the authors plan to leverage this node generation framework to investigate biological problems involving coupled bulk-surface chemical transport.

## REFERENCES

[1] G. A. Barnett, *A Robust RBF-FD Formulation Based on Polyharmonic Splines and Polynomials*, Ph.D. thesis, University of Colorado Boulder, Boulder, CO, 2015.

[2] V. Bayona, M. Moscoso, M. Carretero, and M. Kindelan, *RBF-FD formulas and convergence properties*, J. Comput. Phys., 229 (2010), pp. 8281–8295.

[3] R. Bridson, *Fast Poisson disk sampling in arbitrary s*, in ACM SIGGRAPH 2007 Sketches, SIGGRAPH '07, ACM, New York, 2007, 22, https://doi.acm.org/10.1145/1278780.1278807.

[4] T. Cecil, J. Qian, and S. Osher, *Numerical methods for high dimensional Hamilton-Jacobi equations using radial basis functions*, J. Comput. Phys., 196 (2004), pp. 327–347.

[5] G. Chandhini and Y. Sanyasiraju, *Local RBF-FD solutions for steady convection–diffusion problems*, Internat. J. Numer. Methods Engrg., 72 (2007), pp. 352–378.

[6] G. P.-T. Choi, K. T. Ho, and L. M. Lui, *Spherical conformal parameterization of genus-*0 *point clouds for meshing*, SIAM J. Imaging Sci., 9 (2016), pp. 1582–1618, https://doi.org/10.1137/15M1037561.

[7] O. Davydov and D. T. Oanh, *Adaptive meshless centres and RBF stencils for Poisson equation*, J. Comput. Phys., 230 (2011), pp. 287–304.

[8] O. Davydov and R. Schaback, *Optimal stencils in Sobolev spaces*, preprint, arXiv:1611.04750, 2016.

[9] D. Dimitrov, C. Knauer, K. Kriegel, and G. Rote, *On the bounding boxes obtained by principal component analysis*, in Twenty-second European Workshop on Computational Geometry Delphi, Greece, 2006, p. 193.

[10] G. E. Fasshauer, *Meshfree Approximation Methods with MATLAB*, Interdiscip. Math. Sci. 6, World Scientific, Singapore, 2007.

[11] N. Flyer, G. A. Barnett, and L. J. Wicker, *Enhancing finite differences with radial basis functions: Experiments on the Navier-Stokes equations*, J. Comput. Phys., 316 (2016), pp. 39–62, https://doi.org/10.1016/j.jcp.2016.02.078.

[12] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett, *On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy*, J. Comput. Phys., 321 (2016), pp. 21–38, https://doi.org/10.1016/j.jcp.2016.05.026.

[13] N. Flyer, E. Lehto, S. Blaise, G. B. Wright, and A. St-Cyr, *A guide to RBF-generated finite differences for nonlinear transport: shallow water simulations on a sphere*, J. Comput. Phys., 231 (2012), pp. 4078–4095.

[14] N. Flyer and G. B. Wright, *Transport schemes on a sphere using radial basis functions*, J. Comput. Phys., 226 (2007), pp. 1059–1084.

[15] N. Flyer and G. B. Wright, *A radial basis function method for the shallow water equations on a sphere*, R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci., 465 (2009), pp. 1949–1976.

[16] A. L. Fogelson and R. D. Guy, *Immersed-boundary-type models of intravascular platelet aggregation*, Comput. Methods Appl. Mech. Engrg., 197 (2008), pp. 2087–2104.

[17] B. Fornberg, T. A. Driscoll, G. Wright, and R. Charles, *Observations on the behavior of radial basis function approximations near boundaries*, Comput. Math. Appl., 43 (2002), pp. 473–490, https://doi.org/10.1016/S0898-1221(01)00299-1.

[18] B. Fornberg and N. Flyer, *Fast generation of 2-D node distributions for mesh-free PDE discretizations*, Comput. Math. Appl., 69 (2015), pp. 531–544.

[19] B. Fornberg and E. Lehto, *Stabilization of RBF-generated finite difference methods for convective PDEs*, J. Comput. Phys., 230 (2011), pp. 2270–2285.

[20] E. Fuselier, T. Hangelbroek, F. Narcowich, J. Ward, and G. B. Wright, *Localized bases for kernel spaces on the unit sphere*, SIAM J. Numer. Anal., 51 (2013), pp. 2538–2562, https://doi.org/10.1137/120876940.

[21] E. J. Fuselier and G. B. Wright, *A high-order kernel method for diffusion and reaction-diffusion equations on surfaces*, J. Sci. Comput., 56 (2013), pp. 535–565.

[22] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Internat. J. Numer. Methods Engrg., 79 (2009), pp. 1309–1331, https://doi.org/10.1002/nme.2579.

[23] J. Hake, P. M. Kekenes-Huskey, and A. D. McCulloc, *Computational modeling of subcellular transport and signalings*, Curr. Opin. Struct. Biol., 25 (2014), pp. 92–97.

[24] E. Lehto, V. Shankar, and G. B. Wright, *A radial basis function (RBF) compact finite difference (FD) scheme for reaction-diffusion equations on surfaces*, SIAM J. Sci. Comput., 39 (2017), pp. A2129–A2151, https://doi.org/10.1137/16M1095457.

[25] P.-O. Persson and G. Strang, *A simple mesh generator in MATLAB*, SIAM Rev., 46 (2004), pp. 329–345, https://.doi.org/10.1137/S0036144503429121.

[26] C. Piret, *The orthogonal gradients method: A radial basis functions method for solving partial differential equations on arbitrary surfaces*, J. Comput. Phys., 231 (2012), pp. 4662–4675.

[27] C. Piret and J. Dunn, *Fast RBFC OGR for solving PDEs on arbitrary surfaces*, in Numerical Computations: Theory and Algorithms, AIP Conf. Proc., 1776, 2016, 070005, https://doi.org/10.1063/1.4965351.

[28] E. A. Rakhmanov, E. Saff, and Y. Zhou, *Minimal discrete energy on the sphere*, Math. Res. Lett, 1 (1994), pp. 647–662.

[29] M. L. Rodriguez, P. J. McGarry, and N. J. Sniadecki, *Review on cell mechanics: Experimental and modeling approaches*, Appl. Mech. Rev., 65 (2013), 060801.

[30] E. B. Saff, *Spiral Points on the Sphere*, https://my.vanderbilt.edu/edsaff/spheres-manifolds/.

[31] V. Shankar, *The overlapped radial basis function-finite difference (RBF-FD) method: A generalization of RBF-FD*, J. Comput. Phys., 342 (2017), pp. 211–228, https://doi.org/10.1016/j.jcp.2017.04.037.

[32] V. Shankar and A. L. Fogelson, *Hyperviscosity-based stabilization for radial basis function-finite difference (RBF-FD) discretizations of advection-diffusion equations*, J. Comput. Phys., 372 (2018), pp. 616–639, https://doi.org/10.1016/j.jcp.2018.06.036.

[33] V. Shankar and S. D. Olson, *Radial basis function (RBF)-based parametric models for closed and open curves within the method of regularized stokeslets*, Internat. J. Numer. Methods Fluids, 79 (2015), pp. 269–289, https://doi.org/10.1002/fld.4048.

[34] V. Shankar, G. Wright, A. Fogelson, and R. Kirby, *Augmenting the Immersed Boundary Method with Radial Basis Functions (RBFs) for the Modeling of Platelets in Hemodynamic Flows*, Internat. J. Numer. Methods Fluids, 79 (2015), pp. 536–557.

[35] V. Shankar, G. B. Wright, A. L. Fogelson, and R. M. Kirby, *A study of different modeling choices for simulating platelets within the immersed boundary method*, Appl. Numer. Math., 63 (2013), pp. 58–77, https://doi.org/10.1016/j.apnum.2012.09.006.

[36] V. Shankar, G. B. Wright, A. L. Fogelson, and R. M. Kirby, *A radial basis function (RBF) finite difference method for the simulation of reaction-diffusion equations on stationary platelets within the augmented forcing method*, Internat. J. Numer. Methods Fluids, 75 (2014), pp. 1–22, https://doi.org/10.1002/fld.3880.

[37] V. Shankar, G. B. Wright, R. M. Kirby, and A. L. Fogelson, *A radial basis function (RBF)-finite difference (FD) method for diffusion and reaction–diffusion equations on surfaces*, J. Sci. Comput., 63 (2014), pp. 745–768.

[38] D. A. Steinman, *Image-based computational fluid dynamics modeling in realistic arterial geometries*, Ann. Biomed. Eng., 30 (2002), pp. 483–497.

[39] C. A. Taylor and C. Figueroa, *Patient-specific modeling of cardiovascular mechanics*, Annu. Rev. Biomed. Eng., 11 (2000), pp. 109–134.

[40] K. Thomsen, *Generalized Spiral Points: Further Improvement*, https://groups.google.com/forum/#!topic/sci.math/CYMQX7HO1Cw (2007).

[41] G. B. Wright and B. Fornberg, *Scattered node compact finite difference-type formulas generated from radial basis functions*, J. Comput. Phys., 212 (2006), pp. 99–123, https://doi.org/10.1016/j.jcp.2005.05.030.

[42] C. Yuksel, *Sample elimination for generating poisson disk sample sets*, Comput. Graph. Forum, 34 (2015), pp. 25–32, https://doi.org/10.1111/cgf.12538.