SIAM J. SCI. COMPUT.
Vol. 41, No. 5, pp. A3202–A3229

ON GENERATION OF NODE DISTRIBUTIONS FOR MESHLESS PDE DISCRETIZATIONS*

JURE SLAK† AND GREGOR KOSEC‡

Abstract. In this paper we present an algorithm that is able to generate locally regular node layouts with spatially variable nodal density for interiors of arbitrary domains in two, three, and higher dimensions. It is demonstrated that the generated node distributions are suitable to use in the RBF-FD method, which is demonstrated by solving thermo-fluid problem in two and three dimensions. Additionally, local minimal spacing guarantees are proven for both uniform and variable nodal densities. The presented algorithm has time complexity $O(N)$ to generate N nodes with constant nodal spacing and $O(N \log N)$ to generate variably spaced nodes. Comparison with existing algorithms is performed in terms of node quality, time complexity, execution time, and PDE solution accuracy.

Key words. node generation algorithms, variable density discretizations, meshless methods for PDEs, RBF-FD

AMS subject classifications. 65D99, 65N99, 65Y20, 68Q25

DOI. 10.1137/18M1231456

1. Introduction.

In recent years, a number of meshless approaches have been developed to numerically solve partial differential equations (PDEs) with the desire to circumvent the problem of polygonization encountered in the classical mesh-based numerical methods. The major advantage of meshless methods is the ability to solve PDEs on a set of scattered nodes, i.e., without a mesh. This advantage was advertised even to the point that arbitrary nodes could be used (see [23, p. 14] and [31]), making node generation seemingly trivial. Nevertheless, it soon turned out that such simplification leads to unstable results.

Although node placing is considered much easier than mesh generation, certain care still needs to be taken when generating node sets for meshless methods. Many methods require sufficiently regular nodes for adequate precision and stability. Among others, this also holds for the popular radial basis function-generated finite differences method (RBF-FD) [12]. Despite the need for quality node distributions, solving PDEs with strong form meshless methods utilizing radial basis functions (RBFs) has become increasingly popular [13], with recent uses in linear elasticity [35], contact problems [36], geosciences [12], fluid mechanics [19], dynamic thermal rating of power lines [21], and even the financial sector [15].

Since one of the key advantages of mesh-free methods is the ability to use highly spatially variable node distributions, which can adapt to irregular geometries and allow for refinement in critical areas, many specialized algorithms for genera-

*Submitted to the journal's Methods and Algorithms for Scientific Computing section December 10, 2018; accepted for publication (in revised form) August 19, 2019; published electronically October 15, 2019.

https://doi.org/10.1137/18M1231456

Funding: This work was supported by FWO grant G018916N, the ARRS research core funding P2-0095, and Young Researcher program PR-08346.

†Jožef Stefan Institute, Department E6, Parallel and Distributed Systems Laboratory, and Faculty of Mathematics and Physics, University of Ljubljana, 1000 Ljubljana, Slovenia (jure.slak@ijs.si, http://e6.ijs.si/~jslak/).

‡Jožef Stefan Institute, Department E6, Parallel and Distributed Systems Laboratory, 1000 Ljubljana, Slovenia (gregor.kosec@ijs.si, http://e6.ijs.si/~gkosec/).

tions of such node layouts have been developed. Most of them can generally be categorized into either mesh-based, iterative, advancing front, or sphere-packing algorithms.

The most basic way to generate such node sets is to employ existing tools and algorithms for mesh generation, use the generated nodes, and simply discard the connectivity relations. Such an approach was reasoned by Liu [23, p. 14] as follows: "There are very few dedicated node generators available commercially; thus, we have to use preprocessors that have been developed for FEM." This is problematic for two reasons: it is computationally wasteful, and some authors have reported that such node layouts yielded unstable operator approximations [32], making them unable to obtain a solution. Besides the above shortcomings, such an approach is also conceptually flawed, since the purpose of mesh-free methods is to remove meshing from the solution procedure altogether. To this end, other approaches were researched, often inspired by the algorithms for mesh generation.

A common iterative approach is to position nodes by simulating free charged particles, obtaining so-called minimal energy nodes [17]. Other iterative methods include bubble simulation [24], Voronoi relaxation [1], or a combination of both [6]. Iterative methods are computationally expensive and require an initial distribution. Additionally, the user is often required to consider trade-offs between the number of iterations and node quality. Despite their expensive nature, the produced distributions are often of high quality, which makes iterative methods useful for improving node distributions generated by other algorithms [11].

The next category consists of advancing front methods, which usually begin at the boundary and advance toward the domain interior, filling it in the process. Löhner and Oñate [25] present a general advancing front technique that can be used for filling space with arbitrary objects. These methods, especially if generating a mesh, are often restricted to two dimensions [30]. Another example of a two-dimensional advancing front approach is inspired by dropping variable-sized grains into a bucket [11], which yields quality variable density node distributions and is computationally efficient in practice [34].

The last category is the circle or sphere packing methods [22], which generate high quality node distributions but are often computationally expensive. With inspiration from the graphics community, Poisson disk sampling [7] has become of interest. It can be used to efficiently generate nodes in arbitrary dimensions [5] and has just recently been used as a node generation algorithm [32] providing nodal distributions of sufficient quality for the RBF-FD method.

To the best of our knowledge, algorithms presented in [11, 32] are currently among the best available. However, they have some shortcomings, namely, [11] only works in two dimensions and [32] does not support variable nodal spacing. In this paper, we present an algorithm that overcomes these shortcomings. The presented algorithm works in two, three, and higher dimensions and supports variable density distributions. It also has minimal spacing guarantees and is provably computationally efficient. The main shortcoming of the presented algorithm is that it requires a discretized boundary as an input, which will be addressed in future work. For algorithms that can fill domains with varying density, conformal mappings can be used to generate nodes on curved surfaces by appropriately modifying the node density [11]. The paper by Shankar, Kirby, and Fogelson [32] also includes an algorithm for generation of an appropriate boundary discretization, based on RBF geometric model and supersampling. This paper does not deal with the task of generating a boundary discretization and focuses on discretizations of domain interiors, assuming that the boundary dis-

cretization already exists when required. The extension of the algorithm to curved surfaces will be addressed in future work.

The rest of the paper is organized as follows: in section 2 the requirements for node generation algorithms are discussed, in section 3 recently introduced algorithms for generating nodal distributions, suitable for strong form meshless methods, are presented, in section 4 a new algorithm is presented, in section 5 the algorithms are compared, and some numerical examples are presented in section 6.

2. Node placing algorithm requirements. In this section we examine a list of properties that an ideal node-positioning algorithm should possess and discuss the rationale behind each property. The properties are loosely ordered by decreasing importance.

1. *Local regularity.* Nodal distributions produced by the algorithm should be locally regular throughout the domain, i.e., the distances between nodes should be approximately equal. This definition of local regularity is somewhat soft and imprecise. The requirement stems from the fact that local strong form meshless methods are often sensitive to nodal positions and large discrepancies in distances to the nearest neighbors or other irregularities can cause ill-conditioned approximations, making the distribution inappropriate for solving PDEs. Thus, this point should be read in practice as follows: "The distributions produced by the algorithm should yield quality PDE solutions when using local strong form methods, if reasonable spacing function h was given."
2. *Minimal spacing guarantees.* Computational nodes that are positioned too closely can severely impact the stability of some meshless methods [23]. Thus, provable minimal spacing guarantees are desirable. For constant spacing h, the condition

$$\|p - q\| \geq h \tag{2.1}$$

   is required for any two different points p and q. For variable nodal spacing, the algorithm should guarantee a local lower bound for internodal spacing.
3. *Spatially variable densities.* Many node distribution algorithms rely on a constant discretization step h, as do some efficient implementations [5]. Spatially variable nodal distributions are often required when dealing with irregular domains or adaptivity [36]. The algorithm should be able to generate distributions with spatially variable nodal spacing, which can be assumed to be given as a function $h\colon \mathbb{R}^d \to (0, \infty)$. The changes in variability should be gradual and smooth in order to satisfy the requirement of local regularity. The algorithm should work without any continuity assumptions for reasonable h (see the remarks in subsection 4.3) and should see a constant step h as a special case of variable step $h(p)$, not the other way around.
4. *Computational efficiency and scalability.* Time complexity of the algorithm should ideally be linear in the number of generated nodes. Quasilinear time complexity (e.g., $O(N \log N)$) is acceptable, while time complexity that is $\Omega(N^\alpha)$ for $\alpha > 1$ is undesirable. The algorithm should also be computationally efficient in practice, making it feasible to use as a node generation algorithm in an adaptive setting.
5. *Compatibility with boundary discretization.* Assume that a boundary discretization $\mathcal{X}_b$ of $\partial\Omega$ conforming to the spacing function h already exists. More precisely, we are given a set $\mathcal{X}_b$ of points such that for any two neighboring points p and q, the norm $\|p - q\|$ is approximately equal to $h(p)$ or

$h(q)$. The generated discretization of the whole domain Ω should seamlessly join with the boundary discretization. This helps to prevent problems often encountered when enforcing boundary conditions (see [32, sec. 3.5] and references therein).

6. *Compatibility with irregular domains.* The algorithm should inherently work with any irregular domain Ω, given its characteristic (i.e., "is element of") function

$$\chi_\Omega \colon \mathbb{R}^d \to \{0, 1\},$$
$$\chi_\Omega(p) = \begin{cases} 1, & p \in \Omega, \\ 0, & p \notin \Omega. \end{cases} \tag{2.2}$$

   Any algorithms that fill axis- or otherwise oriented bounding boxes or produce nodes in a certain nonconstant space outside Ω are seen as impaired in this aspect. Desirably, as the volume of Ω decreases, no matter what the shape of Ω is, so should the number of operations required by the algorithm.
7. *Dimension independence.* The algorithm should ideally be formulated for a general (low) dimension d without any special cases. One-, two-, and three-dimensional versions of the algorithm should also allow a single general implementation.
8. *Direction independence.* The produced distributions and running time of the algorithm should be independent of the orientation of the domain Ω or the coordinate system used.
9. *No free parameters.* The algorithm should aim to minimize the number of free or tuning parameters and work well for all domains and density functions, without any user intervention. The aim is to require algorithms to be robust and work "out of the box." Any free parameters should be explored and well understood, default values should be recommended, and varying the parameters should not drastically change the algorithm's behavior.
10. *Simplicity.* Algorithms with simpler formulations and implementations are preferred.

3. State of the art algorithms. To the best of our knowledge, recently published algorithms by Fornberg and Flyer [11] and Shankar, Kirby, and Fogelson [32] best satisfy the requirements described in section 2 and are hence used as a base for further development. Both algorithms are first briefly described in the following sections.

3.1. Algorithm by Fornberg and Flyer. The node positioning algorithm by Fornberg and Flyer [11] was published in 2015 in a paper titled "Fast Generation of 2-D Node Distributions for Mesh-Free PDE Discretizations." The algorithm in its base form constructs discretizations for two-dimensional axis-aligned rectangles and is presented as Algorithm 3.1. In the following discussion, the first letters of the authors' surnames (FF) will be used to refer to the algorithm.

Initially, the lower side of the rectangle is filled with nodes, spaced according to the given spacing function h. The algorithm works as an advancing front algorithm, starting from $y = y_{\min}$ and advancing toward $y = y_{\max}$. In each iteration the lowest ($\min(y)$) candidate p from the current list of potential node locations is found, removed from potential candidates, and added to the final list. New candidates are spaced accordingly away from p and are inserted into the list of potential node locations. The iteration continues until $y = y_{\max}$ limit is reached.

Algorithm 3.1 Node positioning algorithm by Fornberg and Flyer.

Input: Box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, a function $h\colon [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \to (0, \infty)$, and $n \in \mathbb{N}$.
Output: An array of points in $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ distributed according to h.

1: **function** FF($x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, h, n)
2: $pts \leftarrow$ an empty array of points ▷ This is the final array of points.
3: $candidates \leftarrow$ points spaced according to h from $x_{\min}$ to $x_{\max}$ at y coordinate $y_{\min}$ ▷ This variable represents potential point locations, candidates for actual points that will be in the final result.
4: $(y_{\min}, i_{\min}) \leftarrow$ findMinimum($candidates$) ▷ Find minimal point with respect to y
5: **while** $y_{\min} \le y_{\max}$ **do** coordinate and return its value and index.
6: $p \leftarrow candidates[i_{\min}]$
7: append p to pts
8: remove points closer than $h(p)$ from $candidates$
9: find nearest remaining points in $candidates$ to the left and to the right of p
10: add n new points to $candidates$, equispaced on the circular sector with center p, spanning from the nearest left to the nearest right point
11: $(y_{\min}, i_{\min}) \leftarrow$ findMinimum($candidates$)
12: **end while**
13: **return** pts
14: **end function**

For irregular domains Ω the authors recommend running the above algorithm for the bounding box of Ω, denoted $\mathrm{bb}(\Omega)$, and later discard the nodes outside Ω. If present, the boundary discretization is superimposed onto the discretization generated by Algorithm 3.1. Additionally, internal nodes whose closest boundary node p is less than $h(p)/2$ away are discarded as well.

A few local "repel algorithm" iterations are recommended in the vicinity of the boundary to improve the quality. This part will be omitted from consideration, as it is an iterative improvement scheme that can be performed equivalently on any node distribution generated by any other algorithm [19]. The behavior of FF near the boundaries is thus excluded from analysis, as it is designed to work with the "repel algorithm."

3.1.1. Time complexity analysis. The complexity of the Algorithm 3.1 is not given by the authors and is hence derived in this section. Denote the number of generated nodes with N and the size of array *candidates* at the start of iteration i with s_i. Everything up to the while loop on line 5 is negligible compared to the main loop and takes $O(1)$ time for creation of lists and $O(s_0)$ for candidate generation and minimum extraction. In the main loop, lines 6–7 consume (amortized) constant time and lines 8–11 take time proportional to the size of *candidates* array, i.e., $O(s_i)$ time. Total time complexity is therefore

$$T_{\mathrm{FFbox}} = O(1) + O(s_0) + \sum_{i=1}^{N} (O(1) + O(s_i)) = O(N \max_{1 \le i \le N} s_i) := O(NS), \tag{3.1}$$

where S is defined as $S = \max_{1 \le i \le N} s_i$.

Precisely analyzing s_i and S is difficult for general function h. However, for a fixed square box and constant spacing h it holds that $N = \Theta(\frac{1}{h^2})$ and $s_i = \Theta(n\frac{1}{h}) = \Theta(n\sqrt{N})$. The time complexity in this case is hence $O(nN\sqrt{N})$.

For irregular domains Ω additional work is required. If N denotes the final number of nodes, the algorithm will generate approximately $\frac{|\operatorname{bb}\Omega|}{|\Omega|}N$ nodes in the case of constant h. Superimposing the boundary discretization with N_b nodes and testing all generated nodes for proximity takes $O(N_b \log N_b + \frac{|\operatorname{bb}\Omega|}{|\Omega|}N \log N_b)$ time for building and querying the k-d tree of boundary nodes. These terms are dominated by the node generation in the interior and the time complexity of the algorithm by Fornberg and Flyer for generating node distributions for irregular domains for constant spacing h is

$$T_{\mathrm{FF}} = O\left(n\left(\frac{|\operatorname{bb}\Omega|}{|\Omega|}N\right)^{1.5}\right). \tag{3.2}$$

For variable spacing, the overhead of generated nodes due to the irregularity of Ω and the advancing front size have to be evaluated using integrals, making the time complexity expression somewhat more complicated and less illustrative.

3.1.2. Implementation notes. Authors provided a MATLAB implementation of Algorithm 4.1 in the appendix ot their article [11]. This implementation has been translated to C++ using the Eigen matrix library [16] and the `nanoflann` library for k-d trees, provided by Blanco and Rai [4]. The translation is mostly faithful to the original with a few inefficiencies removed. The C++ implementation is approximately 6 times faster than the original MATLAB implementation (both tested on the same computer).

3.2. Algorithm by Shankar, Kirby, and Fogelson. In 2018, Shankar, Kirby and Fogelson published a node generation algorithm in a paper titled "Robust Node Generation for Meshfree Discretizations on Irregular Domains and Surfaces" [32]. Their node generation algorithm is designed to work on surfaces and in three dimensions, but it does not support variable nodal spacing. We will focus our attention on the part that generates discretizations of the domain interior, when boundary discretization has already been constructed. The main part of the node generation for the interior is based on Poisson disk sampling of the oriented bounding box $\operatorname{obb}(\Omega)$ of domain Ω, described in a paper by Bridson [5]. The relevant part of the node generation algorithm is presented as Algorithm 3.2. In the following discussion the first letters of the authors' surnames (SKF) will be used to refer to the algorithm.

The algorithm starts by taking points on the boundary and their corresponding outward unit normals and shifting them toward the domain's interior by h. An oriented bounding box (OBB) of the shifted boundary points is then constructed using principal component analysis (PCA) [18] as described by Dimitrov et al. [10]. The main part of the algorithm, spanning lines 3 to 24, is the Poisson disk sampling algorithm, which generates the internal discretization of the oriented bounding box using a background grid G as a spatial search structure. Finally, points outside the domain, bounded by $\mathcal{X}$, are discarded. Here, a k-d tree is used to test all candidates for inclusion by testing against the outward normal of their closest boundary point. The remaining points along with the original boundary discretization constitute the final discretization. As an inward-shifted array of points was used to construct the internal discretization, spacing of at least h is guaranteed.

3.2.1. Time complexity analysis. The authors themselves provide the time complexity analysis of the algorithm. Translated to our notation, the running time of

Algorithm 3.2 Node positioning algorithm by Shankar, Kirby, and Fogelson.

Input: Domain Ω and its dimension d.
Input: A nodal spacing step $h > 0$.
Input: A list of boundary points $\mathcal{X}$ of size N_b, moved h toward domain interior.
Output: A list of points in Ω distributed according to spacing function h.

```
1: function SKF(Ω, h, 𝒳, n)
2:     obb ← OBB(𝒳)                          ▷ Generate an oriented bounding box of 𝒳 using PCA.
3:     G ← d-dimensional grid of size h/√d of −1.          ▷ Maps points to their indices.
4:     p ← uniform random node inside obb
5:     G[INDEX(p)] ← 0          ▷ INDEX returns d-d index of p, and its sequential index is 0.
6:     S ← [p]                                  ▷ Resulting list of accepted samples.
7:     A ← {0}                                              ▷ Set of active indices.
8:     while A ≠ ∅ do
9:         i ← RANDOMELEMENT(A)                  ▷ Get a uniform random element of A.
10:        b ← false                  ▷ Indicates if any valid points were generated.
11:        for j ← 1 to n do
12:            p ← uniform random point in annulus with center S[i] and radii h and 2h
13:            if not OUTSIDE(p, obb) and not TOOCLOSE(p, h, G, S) then
14:                ADD(A, SIZE(S))              ▷ Add sequential index of p to active set A.
15:                G[INDEX(p)] ← SIZE(S)        ▷ Mark grid cell taken by p as occupied.
16:                APPEND(S, p)                 ▷ Append p to the list of accepted samples.
17:                b ← true                     ▷ Flag that an accepted sample was generated.
18:                break for
19:            end if
20:        end for
21:        if b = false then          ▷ Point S[i] failed to generate any accepted samples.
22:            REMOVE(A, i)           ▷ Point with index i is removed from the active set.
23:        end if
24:    end while
25:    T ← KDTREEINIT(𝒳)                  ▷ Initialize spatial search structure on points 𝒳.
26:    S ← FILTER(T, S)                   ▷ Discard points outside the region, bounded by 𝒳.
27:    return S
28: end function
```

the interior fill algorithm is

$$T_{\mathrm{SKF}} = O\left(n\frac{|\operatorname{obb}(\Omega)|}{|\Omega|}N\right). \tag{3.3}$$

This represents the running time of the Poisson disk sampling. The PCA analysis and tree construction are linear or log-linear in N_b and are thus dominated by the Poisson disk sampling.

3.2.2. Implementation notes. There is a small difference between the algorithm as described by Shankar, Kirby, and Fogelson [32] and Bridson [5]. The Bridson version generates up to n candidates for each point and stops as soon as one candidate is accepted. The version in the SKF algorithm generates all n points and adds all accepted candidates. Algorithm 3.2 uses the original, Bridson version and to obtain the SKF version, one needs to remove the break statement on line 18. Since the authors of the SKF algorithm claim to use a faithful implementation of the algorithm as presented by Bridson and only list the algorithm for completeness, we decided to use the Bridson version in our tests. All matrix and tensor operations were again imple-

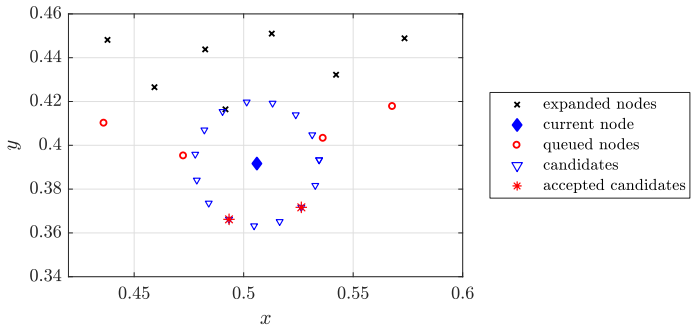

FIG. 1. *Generation and selection of new candidates in the proposed algorithm.*

mented using the Eigen matrix library and the k-d tree operations were implemented using `nanoflann`.

4. New node placing algorithm. From the discussion presented in section 3 it is clear that although state of the art placing algorithms provide a solid spatial discretization methodology for strong form meshless methods, there is still room for improvements, especially in the generalization to higher dimensions, flexibility regarding variable nodal density, and treatment of irregular domains. Improving upon the work of Fornberg and Flyer [11] and Shankar, Kirby, and Fogelson [32], we propose a new algorithm that overcomes some of the limitations of FF and SKF algorithms. We will refer to the proposed node placing algorithm as PNP in the rest of the paper.

The PNP algorithm is, similarly to SKF, based on Poisson disk sampling. Poisson disk sampling has certain stochastic properties, such as the fact that it produces a "blue noise distribution" that is an excellent fit for graphical applications like dithering [5, 7]. In the context of the PDE solution procedure a slightly different distribution is required that primarily follows appropriate spacing and regularity criteria. Therefore, the PNP algorithm deviates from the original Poisson disk sampling in order to effectively produce tightly packed regular distributions needed in the solution of PDEs.

The PNP algorithm begins either with a given nonempty set of "seed nodes" or with an empty domain. In the context of PDE discretizations, some nodes on the boundary are usually already known and can be used as seed nodes, possibly along with additional nodes in the interior. If the algorithm starts with no nodes, it adds a seed node randomly within the domain. Before the main iteration loop, seed nodes are put in a queue, waiting to be processed. In each iteration i, a node p_i is dequeued and *expanded*, by generating a set of candidates C_i, which are positioned on a sphere with center p_i and radius r_i, where r_i is obtained from the function h, $r_i = h(p_i)$. Candidates that lie outside of the domain or are too close to already existing nodes are rejected and remaining candidates are enqueued for expansion. Node p_i is accepted as a domain node and will not be touched any more. The iteration continues until the queue is empty. Figure 1 demonstrates a core operation of the algorithm, i.e., the expansion, with possible selection of new candidates and the rejection process.

Figure 2 illustrates the execution of the algorithm. The first panel shows an initial setup on a unit square. For demonstration purposes, the nodes in the initial boundary discretization along with a single node in the interior were chosen as the seed nodes. The subsequent panels in Figure 2 illustrate the progression of the algorithm. The discretization grows from the initial nodes inward toward the empty interior, until no more acceptable candidates can be found due to already existing nodes. The

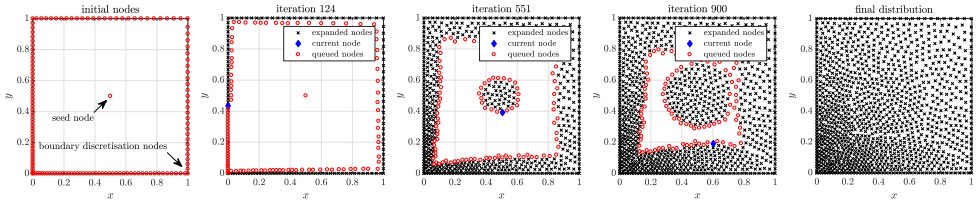

FIG. 2. *Run-time progress of the proposed algorithm (left to right). Unit square* $[0,1]^2$ *was sampled with nodal spacing function* $h(x,y) = 0.015\,(1+x+y)$.

advancing front nature of the algorithm can be seen, but the front itself is not a straight line as in FF.

An efficient implementation with an implicit queue contained in the array of final points and the k-d tree spatial structure [29] is presented as Algorithm 4.1. In practice, the comparison on line 10 should be done with some tolerance, due to the inexactness of the floating point arithmetic, i.e., the line should in practice read $\|c_{i,j} - n_{i,j}\| \geq (1-\varepsilon) r_i$ for e.g., $\varepsilon = 10^{-10}$.

Algorithm 4.1 Proposed node positioning algorithm.

Input: Domain Ω and its dimension d.
Input: A nodal spacing function $h\colon \Omega \subset \mathbb{R}^d \to (0,\infty)$.
Input: A list of starting points $\mathcal{X}$; this includes the possible boundary discretization and seed nodes.
Output: A list of points in Ω distributed according to spacing function h.

```
1:  function PNP(Ω, h, X)
2:      T ← KDTREEINIT(X)                    ▷ Initialize spatial search structure on points X.
3:      i ← 0                                ▷ Current node index.
4:      while i < |X| do                     ▷ Until the queue is not empty.
5:          p_i ← X[i]                       ▷ Dequeue current point.
6:          r_i ← h(p_i)                     ▷ Compute its nodal spacing.
7:          for each c_{i,j} in CANDIDATES(p_i, r_i) do   ▷ Loop through candidates.
8:              if c_{i,j} ∈ Ω then          ▷ Discard candidates outside the domain.
9:                  n_{i,j} ← KDTREECLOSEST(T, c_{i,j})   ▷ Find nearest node for proximity test.
10:                 if ‖c_{i,j} − n_{i,j}‖ ≥ r_i then ▷ Test that c_{i,j} is not too close to other nodes.
11:                     APPEND(X, c_{i,j})   ▷ Enqueue c_{i,j} as the last element of X.
12:                     KDTREEINSERT(T, c_{i,j})   ▷ Insert c_{i,j} into the spatial search structure.
13:                 end if
14:             end if
15:         end for
16:         i ← i + 1                        ▷ Move to the next nonexpanded node.
17:     end while
18:     return X
19: end function
```

The algorithm includes the generation of new candidates in line 7 that needs to be further defined. Three options are proposed and evaluated below:

1. *Random candidates.* The candidate set C_i in each iteration consists of n random points chosen on a d-dimensional sphere with center p_i and radius r_i, recalling the original Poisson disk sampling.
2. *Fixed pattern candidates.* The candidate set C_i in each iteration consists of a fixed discretization of a unit ball, translated to p_i and scaled by r_i. The

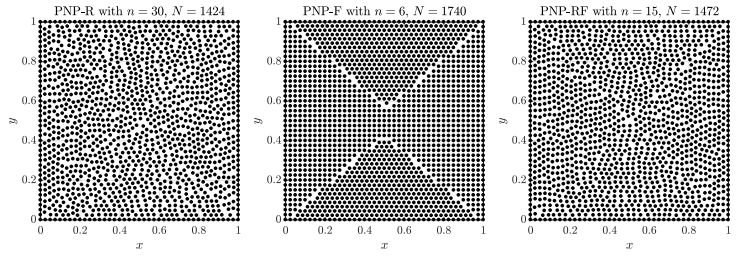

FIG. 3. *Comparison of different types of candidate generation when filling the unit square* $[0, 1]^2$ *with* $h = 0.025$.

discretization of a unit ball in two dimensions is obtained simply by $C_{\text{unit}}(k) = \{(\cos\varphi, \sin\varphi);\ \varphi \in \{0, \varphi_0, 2\varphi_0, \ldots, (k-1)\varphi_0\}, \varphi_0 = \frac{2\pi}{k}\}$. In d-dimensions, the discretization of a ball with radius r is obtained using d-dimensional spherical coordinates and recursively discretizing a $d-1$ dimensional ball. Using, e.g., $k = 6$ results in 14 candidates in three dimensions, and using $k = 12$ results in 48. In three dimensions, the parameter k represents the number of points lying on the great circle.

3. *Randomized pattern candidates.* The candidate set C_i is obtained from the fixed set in point 2, by applying a random rotation to all the points.

The three ways of candidate generation were used to produce node distributions on a unit square, shown in Figure 3. Different types of candidate generation are abbreviated as PNP-R, PNP-F, and PNP-RF for random, fixed pattern, and randomized fixed pattern variants, respectively.

The fixed pattern candidate generation algorithm stands out, as the gaps where the advancing fronts of nodes joined are clearly visible. Due to reproduction of space-efficient hex-packing it also has the most nodes. Other two algorithms generate visually similar distributions, but a higher value of n needed to be used for the randomized version to produce similar results. We decided to use the randomized fixed pattern for candidate distribution, as it produces results similar to the random version with lower time complexity.

The presented algorithm has a few differences compared to the original Poisson disk sampling. First and foremost, the algorithm works with variable nodal spacing and is able to generate distributions with spatially variable densities. Each node is used only once to generate the new candidates. Third, the candidates are generated uniformly on the sphere with random offsets, as opposed to being generated at random on an annulus. This packs the candidates more tightly and reduces the gaps. It also improves running time, as candidates better cover the unoccupied space at the cost of removing the stochastic properties of the sampling, which are not relevant to the PDE solutions. Fourth, the candidates that are outside Ω are immediately discarded, only continuing the generation of candidates actually inside Ω, once again improving execution time. More details about the impact of the above differences are investigated in section 5 and can be observed in Figures 4 and 11.

The PNP algorithm exhibits gaps between nodes where the advancing fronts meet in Figure 3, but the gaps are never large enough that another node could be placed inside and are even emphasized visually if the marker size is comparable to nodal spacing (see Figure 4). The exact place where the advancing fronts meet is dependent on the position of the seed nodes. If the algorithm is run from a seed node in the domain interior instead of from the boundary nodes, these types of fronts do

not appear throughout the domain, but gaps form at the boundary instead. With PDE discretization in mind it is less problematic to have them appear in the domain interior, and this is how the algorithm is run for the rest of the paper.

Additionally, the algorithm can be easily modified to return the indices of the nodes where the fronts meet. For each node i, we can check if any candidates generated from it are accepted and added on line 11. If that is not the case, index i can be added to the list of *terminal nodes*, which is returned after the algorithm finishes. Regularization can then be performed on those or neighboring nodes, if necessary.

4.1. Time complexity analysis. Output sensitive time complexity is straightforward to analyze. Let us denote the number of given starting points in $\mathcal{X}$ with $N_b = |\mathcal{X}|$. It is assumed that N_b is significantly less than N, e.g., $N_b = O(N^{\frac{d-1}{d}})$ as is the case when $\mathcal{X}$ represents the boundary discretization. The initial construction of the spatial index costs $O(N_b \log N_b)$ and initialization of other variables costs $O(1)$. The number of iterations of the main loop is equal to the number of generated points, denoted by N. A total of n candidates are generated in the ith iteration and, in the worst case, two k-d tree operation on the tree with at most $i + N_b$ nodes are performed, taking $O(\log(i + N_b))$ time for each candidate. All other operations are (amortized) constant. Thus the total time complexity of the algorithm is equal to

$$T_{\text{PNP}} = O(N_b \log N_b) + O(1) + \sum_{i=1}^{N} [nO(\log(i + N_b)) + O(1)] = O(nN \log N). \tag{4.1}$$

The above analysis shows that the time complexity of the algorithm is dominated by the spatial search structure used, which adds an undesired factor $\log N$. If h is assumed to be constant, the algorithm could be sped up by using a uniform-grid based spatial search structure, similar to one used in Algorithm 3.2. Using such a search structure requires a rectangular grid, usually with spacing $h/\sqrt{d}$, such that there is at most one point per grid cell. When constructed on the rectangle $\operatorname{obb}(\Omega)$, the time complexity of its allocation and initialization is proportional to the number of cells, which leads to time complexity

$$O\left(\frac{|\operatorname{obb}\Omega|}{(h/\sqrt{d})^d}\right) = O\left(\frac{|\operatorname{obb}\Omega|}{|\Omega|}N\right), \tag{4.2}$$

using the fact that for constant h the number of nodes is $N = \Theta(|\Omega|/h^d)$.

The subsequent insertions and queries in the grid are all $O(1)$, thus improving the time complexity of the algorithm for constant h to

$$T_{\text{PNP-grid}} = O\left(\frac{|\operatorname{obb}\Omega|}{|\Omega|}N + nN\right). \tag{4.3}$$

Furthermore, the factor $\frac{|\operatorname{obb}\Omega|}{|\Omega|}$ can be eliminated by using a hash map of cells instead of a grid; however, the practical benefit of that approach shows only with very irregular domains.

Using the background grid for a spatial structure is feasible even with moderately spatially variable h, by allowing more than one point per cell. For even higher variability, hierarchical grids could be used, but a k-d tree-like search structure covers all cases. For a specific use case, k-d tree can be replaced with any spatial search structure, as desired by the user, obtaining time complexity

$$T_{\text{PNP-general}} = O(P(N) + Nn(Q(N) + I(N))), \tag{4.4}$$

where P is the precomputation/initialization time used by the data structure on N nodes, $Q(N)$ is the time spent on a radius query, and $I(N)$ is the time spent for new element insertion.

4.2. Implementation notes. As in the previous two algorithms, all matrix and tensor operations were implemented using the Eigen matrix library and the k-d tree operations were implemented using `nanoflann`.

4.3. Remarks. Algorithms 3.1 and 4.1 do not necessarily terminate, depending on the nodal spacing function used. The integral

$$N(h) := \int_\Omega \frac{\mathrm{d}\Omega}{h(p)^d} \tag{4.5}$$

approximately measures the number of points required and can be infinite even if function h is smooth and positive on Ω. Simply taking a one-dimensional example $\Omega = (0, 1)$ and $h(x) = \frac{0.1}{x}$ is enough to trick the algorithm into sampling indefinitely. As a precaution to that and more practically, as a memory limit, the maximal number of points $N_{\max}$ can be specified by the user and the algorithm can be terminated prematurely.

5. Satisfaction of the requirements. This section compares all three node placing algorithms, namely, FF (Algorithm 3.1), SKF (Algorithm 3.2), and PNP (Algorithm 4.1). The results of the comparison presented in this section are summarized at the end in Table 3. The following subsections roughly follow the requirements postulated in section 2.

5.1. Local regularity. The most important feature that an algorithm should possess is regularity of the distributions. This property is initially tested visually, by observing plots of nodal distributions, which is feasible only in two dimensions. Among other things, local regularity states also that large discrepancies in distances to nearest neighbors are not desired. This can be tested in arbitrary dimensions, by observing distances to nearest neighbors, using various statistics and histogram plots to determine their properties. Finally, the accuracy and stability of solutions of PDEs on generated node distributions can be compared to fully determine the quality of distributions generated by the three algorithms.

We begin our analyses by comparing the three algorithms on the unit square $[0, 1] \times [0, 1]$. Node distributions were generated using constant density $h = 0.025$ and the expected number of nodes is $N(h) = 1600$. Node distribution for all three algorithms is shown in Figure 4. Parameters n for various algorithms were chosen as recommended in their respective papers ($n = 5$ for FF and $n = 15$ for SKF), with $n = 15$ also being used for the algorithm presented in this paper.

The SKF algorithm generated substantially fewer nodes than the other two. It also has significant gaps between the boundary and internal nodes as well as visually more irregular distributions. The FF algorithm generates a smooth distribution without any significant defects in the interior. The PNP algorithm exhibits gaps on diagonals, where advancing fronts from the sides have merged, but behaves better near the boundaries. The part of the distribution where the advancing fronts meet is shown in the rightmost panel in Figure 4 to give a better perspective on the size of the gaps.

In terms of the number of nodes, FF gives the best result, since it produced only 45 nodes fewer than expected, followed by PNP that produces 128 less nodes. The worst performance is demonstrated by SKF with deficiency of 573 nodes.

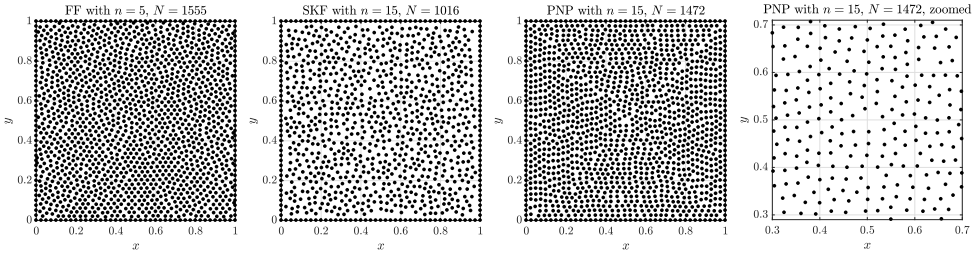

FIG. 4. *Node distributions on the unit square* $[0,1]^2$ *with* $h = 0.025$ *generated with different algorithms. Rightmost figure shows the enlarged PNP distribution in the center where the advancing fronts meet.*

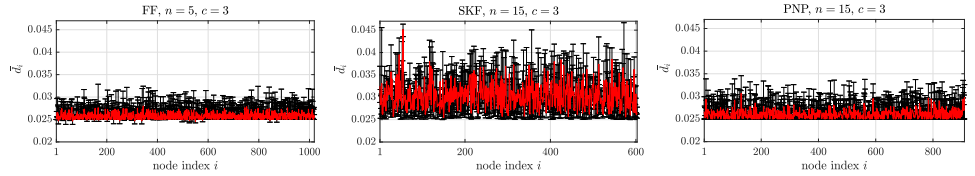

FIG. 5. *Average distances to the nearest neighbors for internal nodes. Error bars show minimal and maximal distances to three nearest neighbors.*

TABLE 1
Numerical quantities related to internodal distance and hole regularity.

alg.	mean $\bar{d}_i$	std $\bar{d}_i$	mean$(d_i^{\max} - d_i^{\min})$	min s_j	mean s_j	max s_j
FF	0.02575	0.00065	0.00208	0.028071	0.03438	0.04352
SKF	0.03042	0.00275	0.02894	0.029737	0.04470	0.07008
PNP	0.02604	0.00086	0.00276	0.028949	0.03568	0.05164

To analyze local regularity, distances to nearest neighbors are observed in the interior of the domain. For each node p_i at least $2h$ away from the boundary, its c closest neighbors (excluding i itself) are found and denoted by $p_{i,j}$ for $j = 1, \ldots, c$ with distances to these neighbors computed as $d_{i,j} = \|p_i - p_{i,j}\|$. Figure 5 shows average distances of each node to its three closest neighbors, i.e., the plot of $\bar{d}_i = \frac{1}{3}\sum_{j=1}^{3} d_{i,j}$ for each considered node p_i. Along with the average distance, the interval $[d_i^{\min}, d_i^{\max}]$ is shown, where

$$d_i^{\min} = \min_{j=1,2,3} d_{i,j}, \qquad d_i^{\max} = \max_{j=1,2,3} d_{i,j}.$$

FF and PNP algorithms show similar behavior, with the average distance being close to h with little variability between distances to closest neighbors. The FF algorithm has a few nodes a bit closer than h together, but keeps the internodal distance closer to h and with less spread than PNP. The SKF algorithm performs worse with most of its distances to c nearest neighbors being on average closer to 0.03 and with a significantly larger spread. The numerical results representing these quantities are shown in Table 1. The first two columns of the table demonstrate that the prescribed nodal spacing h is much better obeyed in PNP and FF algorithms, and the last column shows that the average spread of the internodal distances in the SKF algorithm is more than two times greater than in FF and PNP.

Besides distances to the nearest neighbors, we can also take a look at the empty space between the generated nodes. This can be done by computing the Voronoi

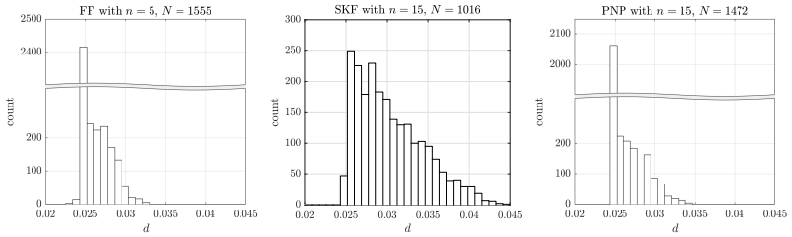

FIG. 6. *Histogram of distances to three nearest neighbors for node distributions on unit square* $[0,1]^2$ *with* $h = 0.025$.

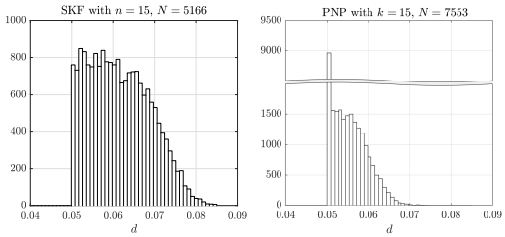

FIG. 7. *Histogram of distances to six nearest neighbors for internal nodes of distributions on unit cube* $[0,1]^3$ *with* $h = 0.05$.

diagram vertices v_j that lie inside the domain and observing the diameters s_j of the largest circles centered at v_j not containing any nodes. Formally, s_j are given as

$$s_j = 2 \min_i \|v_j - p_i\|. \tag{5.1}$$

Note that the largest value of s_j is the diameter of the largest empty circle. The basic statistics of s_j for the three considered algorithms are presented in Table 1.

Additional insight is offered with histograms of distances to three nearest neighbors (Figure 6). As expected, the largest count is in the bin around h. PNP and SKF algorithms have no distances in bins below h, but the FF algorithm does put a small number of nodes at a distance less than h (see subsection 5.2). The irregularities visible in the SKF algorithm distribution in Figure 4 are reflected in the histogram. The histogram has a much heavier tail than the PNP and FF histograms, with far fewer nodes exactly at distance h. PNP and FF histograms show more tightly packed distributions with slimmer tails, but the tail of the PNP histogram is a bit longer and more spread out.

Next, the PNP and SKF algorithms are compared in three dimensions. The unit cube $[0,1] \times [0,1] \times [0,1]$ is filled with a constant density $h = 0.05$, starting from the boundary in the PNP case. The expected number of nodes is $N(h) = 8000$. Histograms of distances to the closest $c = 6$ nodes for internal nodes are shown in Figure 7 for the PNP and SKF algorithms.

The histograms behave similarly to their two-dimensional counterparts. SKF algorithm again generated significantly fewer nodes than the PNP algorithm. PNP has a large number of neighbors at distance h and a lighter tail, while the distances in the SKF case are more spread out.

Further visual confirmation of regularity for variable density cases is demonstrated in subsection 5.3 (see Figures 8, 9, and 10), and more importantly, by the solutions of PDEs on generated node sets [13, 32, 35], thus confirming sufficient local regularity.

Additionally, section 6 considers sample solutions to PDE examples and discusses accuracy, eigenvalue stability, and convergence properties. Our experiments have shown that SKF distributions cause stability problems when using small stencils, such as, e.g., the closest 7 nodes. The likely cause of this instability is higher node irregularity in SKF node distributions. PNP and FF distributions had no problems with small stencils.

5.2. Minimal spacing requirements. Point 2 discusses minimal spacing guarantees. Provable minimal spacing guarantees are very desirable, since nodes that are positioned too closely can effect the stability of strong form methods. The FF algorithm does not strictly respect the spacing h. When running the algorithm with $h = 0.005$ on a unit square $[0, 1]^2$, some pairs of points in the domain interior were closer than $0.95h$. Although the violations do not appear to be significant and do not affect the quality in practice, no bound of form $\|p_j - p_i\| \geq \alpha h$ for $\alpha > 0$ and $i \neq j$ is known.

The SKF algorithm enforces the spacing between nodes to be greater than or equal to h in the interior and on the boundary, both times leveraging specialized spatial search structures. The algorithm thus has the usual minimal spacing guarantee for constant nodal spacing:

$$\|p - q\| \geq h \tag{5.2}$$

for $p \neq q$.

A similar argument can be made for the PNP algorithm: each new candidate is checked using a k-d tree against all previous ones, proving the minimal spacing guarantee for constant h. For variable h, the above argument yields the bound

$$\|p_i - p_j\| \geq \min_{p\in\Omega} h(p) \tag{5.3}$$

for $i \neq j$. This bound is dependent on a global property of h and can be very coarse. More precisely, local bounds when considering spatially variable distributions are defined by Mitchell et al. [28]. If an ordered list of points, numbered 1 to N, is considered, then the minimal spacing guarantee, called the *empty disk property*, is satisfied if

$$\|p_i - p_j\| \geq f(p_i, p_j) \tag{5.4}$$

for $1 \leq i < j \leq N$, where f is a function evaluated at previously accepted node p_i and new candidate p_j. Four basic variations were proposed, based on which point's spacing is taken into account when positioning new candidates:

- *Prior-disks:* $f(p_i, p_j) = h(p_i)$,
- *Current-disks:* $f(p_i, p_j) = h(p_j)$,
- *Bigger-disks:* $f(p_i, p_j) = \max\{h(p_i), h(p_j)\}$,
- *Smaller-disks:* $f(p_i, p_j) = \min\{h(p_i), h(p_j)\}$.

The PNP procedure satisfies none of these variations. The following proposition establishes a version of the empty disk property (5.4) of PNP.

PROPOSITION 5.1. *Let the points p_i, $i = 1, \ldots, N$, be a list of nodes generated by Algorithm* 4.1*, where first N_b nodes were given as initial nodes. The minimal spacing inequality*

$$\|p_k - p_j\| \geq h(p_{\beta(j)}) \tag{5.5}$$

holds for all $N_b \leq k < j < N$. The function β represents the predecessor function.

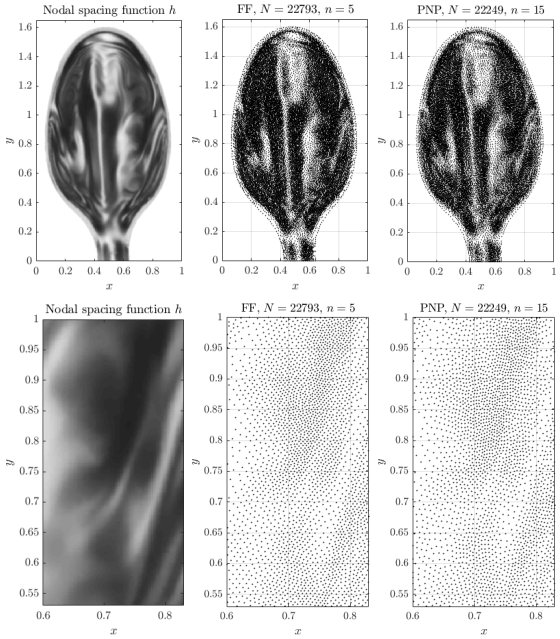

FIG. 8. *Illustration of variable density node sampling, with the nodal spacing function h obtained from the image on the left using* (5.8). *Enlarged variants are present to better assess the node quality.*

Proof. Algorithm 4.1 begins with N_b initial nodes. Each candidate is generated from a unique existing node, thus giving rise to a predecessor-successor relation. Predecessor function $\beta\colon \{N_b+1,\dots,N\} \to \{1,\dots,N\}$ for an accepted candidate p_j that was generated from p_i is defined as $\beta(j) = i$. Note that predecessors for the first N_b initially given points are not defined.

Consider an accepted candidate p_j, generated from a node p_i. The candidate was generated at a distance $h(p_i)$ from p_i, thus satisfying the equality

$$\|p_i - p_j\| = h(p_i) = h(p_{\beta(j)}). \tag{5.6}$$

In particular, this means that Algorithm 4.1 satisfies the *prior-disks* property for predecessor-successor pairs. The distance d to the nearest neighbor of p_j among already accepted nodes is then found and if $d \geq h(p_i)$, the candidate is accepted. This means that the following inequality holds for all $k < j$,

$$\|p_k - p_j\| \geq d \geq h(p_i) = h(p_{\beta(j)}), \tag{5.7}$$

establishing the desired property. ∎

5.3. Spatial variability. An important feature of the FF and PNP algorithms is the ability to generate node sets with variable nodal spacing on irregular domains. The SKF algorithm does not support variable nodal spacing and is excluded from this analysis. As an example, the image shown in the top left corner of Figure 8 is chosen as a source for the nodal spacing function h. The image is a modified version of an image showing stress distribution in a plastic spoon under a photoelasticity experiment [2]. It features an irregular domain and rapidly varying dark and light regions, which presents a more challenging case usually found in PDE discretizations.

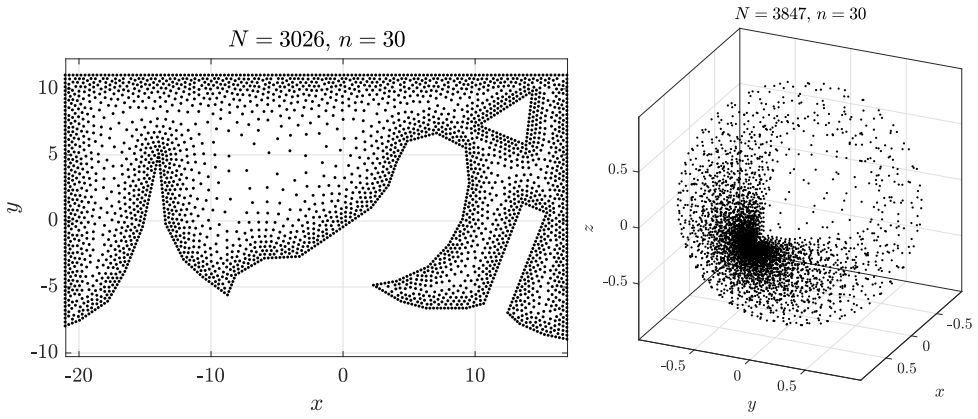

FIG. 9. *Example of generated variable density distributions for a nonconvex domain with nontrivial boundaries.*

The conversion from gray levels to the nodal spacing function is the same as used by Fornberg and Flyer [11]. Normalization factor $h_0 = 1.5$ was used to adjust the number of nodes for maximal visibility. The nodal spacing function h is thus constructed from the image as

$$h(x,y) = h_0\, s\left(\frac{I_{\lfloor wx \rfloor, \lfloor wy \rfloor}}{255}\right),\ s(g) = 0.002 + 0.006\, g + 0.012\, g^8, \tag{5.8}$$

where I_{ij} represents the gray level, ranging from 0 to 255, of the pixel in the ith row and the jth column of the image and w is the width of the image. The node distributions obtained by filling the spoon shape with the aforementioned density using the PNP and FF algorithms are shown in the first row of Figure 8. The bottom row shows an enlarged portion of the image and the corresponding distributions, so that individual nodes are visible for easier visual assessment.

Both generated node sets conform to the supplied nodal spacing function. The total number of nodes is similar in both cases, with PNP having fewer nodes than FF. Enlarged portions show that PNP and FF distributions are locally regular, visually similar, and respect the variable nodal spacing function h.

Further examples of two- and three-dimensional spatially variable distributions are shown in Figure 9. The two-dimensional domain is a nonconvex polygon with a hole and the three-dimensional domain is a spherical shell with one of the octants cut out. Figure 10 displays successive enlargement of a nodal distribution used to solve a contact problem [36], which illustrates the graded nature of the refinement and its local regularity in the most zoomed panel.

5.4. Computational efficiency and scalability. Point 4 concerns computational efficiency in two aspects: theoretical time complexity and execution time.

The time complexity of the FF algorithm is proven in subsection 3.1.1 and given by (3.2)

$$T_{\mathrm{FF}} = O\left(n\left(\frac{|\,\mathrm{bb}\,\Omega|}{|\Omega|}N\right)^{1.5}\right) \tag{5.9}$$

for constant spacing h and is similar for variable spacing. There are no immediate benefits if h is assumed to be constant. The SKF algorithm benefits from the assumption

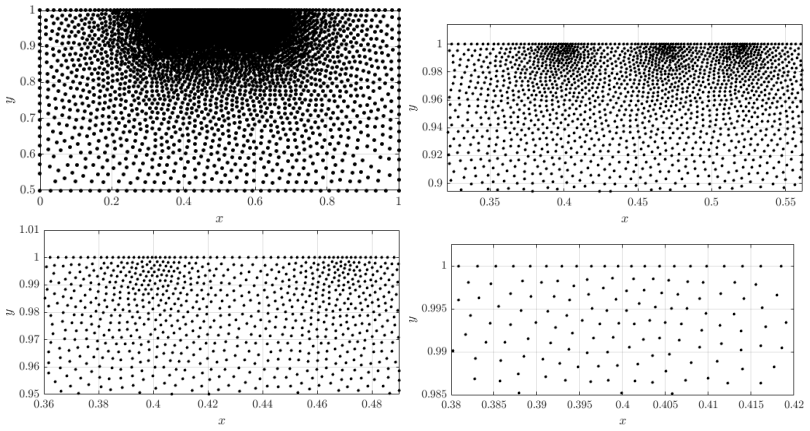

FIG. 10. *Discretization of a contact region and successive enlargements.*

of constant h and has time complexity given by (3.3) in subsection 3.2.1:

$$T_{\mathrm{SKF}} = O\left(\frac{\mathrm{obb}(\Omega)|}{|\Omega|} nN\right). \tag{5.10}$$

The PNP algorithm has time complexity

$$T_{\mathrm{PNP}} = O(nN \log N), \tag{5.11}$$

as analyzed in subsection 4.1. If h is assumed constant, the time complexity is further reduced to $O(\frac{|\mathrm{obb}(\Omega)|}{|\Omega|} N + nN)$ using grid spatial search structure and even to $O(Nn)$ using hashing for irregular domains.

The PNP algorithm is better for a domain irregularity factor compared to both the SKF and FF algorithms. In the case of constant h it shares the same remaining factor Nn with SKF and for variable densities it is strictly better than FF.

Next, we compare the running time and scalability of proposed algorithms. All time measurements were done on a laptop computer with an Intel Core i7-7700HQ CPU @ 2.80GHz processor and 16 GB DDR4 RAM. Code was compiled using g++ (GCC) 8.1.1 for Linux with -std=c++11 -O3 -DNDEBUG flags.

Note that we implemented all three algorithms in the same manner with great emphasis on optimization of the code in order to provide a fair comparison.

All algorithms were run on a unit square $[0,1]^2$ with the same parameters as in subsection 5.1. The nodal spacing function h was varied as $h = \frac{1}{n}$ for such n that the total number of nodes N reached approximately $N = 10^6$. Each run was executed 10 times and the median time was taken. The results are shown in Figure 11.

In two dimensions the FF algorithm performs better than the others for small N. This is also expected, as the algorithm generates nodes in a much simpler (and deterministic) way than the other two approaches. The SKF algorithm is next in terms of performance, with its grid-based search structure. The PNP algorithm is the slowest, due to the k-d tree search structure. Nonetheless, 10^6 nodes are generated in 5 to 10 seconds, which is significantly less than the time that would be spent on solving the PDE on these nodes.

The trends for large N coincide with the theoretical time complexities with SKF being an $O(N)$ algorithm, PNP being an $O(N \log N)$, and FF being $O(N\sqrt{N})$.

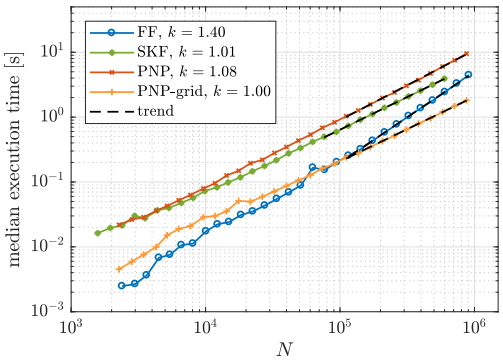

FIG. 11. *Execution times for the considered algorithms when filling* $[0,1]^2$ *with successively smaller densities. Each data point represents a median of* 10 *runs. Standard deviation of run times from the median was below* 3% *in all cases. Value* k *in the legend indicates the slope of the line.*

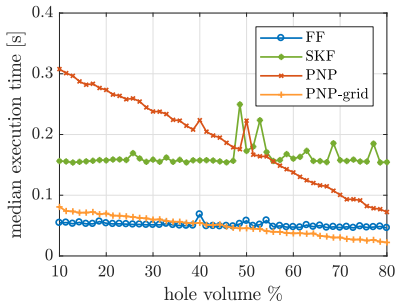

FIG. 12. *Execution time when filling domains* $\Omega(\alpha)$ *which have decreasing area.*

PNP was also run using the same grid search structure as SKF, denoted in Figure 11 by "PNP-grid." It shows a significant improvement over the use of k-d tree spatial search structure and agrees with the predicted linear time complexity. This also shows that the PNP algorithm itself is about three times faster than SKF, when compared using the same search structure and the same number of candidates. PNP with a gird-based structure also comes close to FF for smaller N and constant h. The execution time of the PNP and SKF algorithms was tested also in three dimensions and the results are equivalent.

Additionally, we analyze the execution time of the three algorithms when dealing with irregular domains. Both the FF and SKF algorithms do no have time complexity proportionate to $|\Omega|$, but rather to the volume of its (oriented) bounding box, which can be arbitrarily larger. In practice this means that the PNP algorithm inherently benefits in execution time by a factor of $\frac{|\operatorname{bb}(\Omega)|}{|\Omega|}$.

This is illustrated in Figure 12, which shows the execution time of the considered algorithms when filling increasingly "emptier" domains

$$\Omega(\alpha) = [0,1]^2 \setminus \left(\frac{1}{2} - \alpha, \frac{1}{2} + \alpha\right)^2. \tag{5.12}$$

Domains $\Omega(\alpha)$ are chosen in such a way that the bounding box is equal to $[0,1]^2$ for all α and that the limit of the ratio between the bounding box and domain volume approaches zero as α approaches $1/2$.

The difference in the behavior of execution time is substantial and shows that both versions of PNP really scale with volume of Ω, while the execution time of FF and SKF remains almost constant, as predicted by time complexity analysis. This means that around 30,000 nodes can be generated, for less than 8,000 to remain in the final set.

5.5. Compatibility with boundary discretizations. The next point discusses the compatibility between interior and boundary discretizations. All three algorithms treat boundary discretizations separately from discretizing the interior of Ω. Due to the box-fill nature of FF, the generated discretization of the interior is chopped off at the boundary of Ω when the boundary discretization is superimposed. Nodes that are closer to the boundary nodes than a given threshold are discarded. If the threshold is strictly h, gaps between the boundary and the interior discretization can occur. The authors recommend setting the threshold to $h/2$ and performing a few iterations of a repel-type algorithm that is executed locally on the nodes near the boundary to smooth the transition between both discretizations. The SKF algorithm possesses a similar problem, but deals with it differently. It generates internal nodes in a slightly reduced oriented bounding box, which is computed from boundary nodes that were shifted by h to the interior. This prevents the generation of internal nodes too close to the boundary of the box; however, the nodes still need to be tested for inclusion, which is done using the shifted nodes and their normals. This causes gaps near the boundary, which can be observed in the sample distribution in Figure 4 (second panel).

The PNP algorithm bypasses the aforementioned problems altogether, by offering the option to use the boundary discretization as a starting point of the interior discretization and thus allowing for a smooth transition near the boundary. Similar irregularities to those present near the boundary in the SKF algorithm are formed when the advancing fronts from the opposite sides meet, but they appear in the interior of Ω (see Figure 4, rightmost panel), where they have less impact on the stability of the solution. Consequently no need to smooth the irregularities with expensive iterative repel techniques arose.

5.6. Compatibility with irregular domains. Another requirement deals with irregular domains. The FF algorithm has a disadvantage of being only able to fill axis-aligned boxes, which results in potentially a lot of unnecessarily generated nodes. This approach is somewhat improved in the SKF algorithm, where oriented bounding boxes are used, in general reducing the number of generated nodes compared to FF. The number of unnecessarily generated nodes could be reduced even further by decomposing an unfavorably shaped domain into smaller domains, which can be better bounded by cuboids. The smaller domains can then be filled separately and combined together, provided that the node generation algorithm behaves well near boundaries. An appropriate domain decomposition would also enable immediate parallel execution of the algorithm.

Of the three discussed algorithms only PNP never generates any unnecessary nodes in the exterior of the given domain Ω, never evaluates nodal spacing function h outside of Ω, and has the property that the total number of generated nodes and the time complexity scale directly with $|\Omega|$. The impact of unnecessary node generation outside Ω on the execution time is illustrated in subsection 5.4; however, the slowdown introduced by bounding boxes is in practice often acceptable.

The strength of the PNP algorithm which allows it to generate nodes only inside Ω can also become its disadvantage. If seed nodes are supplied only in one part of

Ω and the domain has a bottleneck in the middle (such as an hourglass shape) of girth approximately equal to nodal spacing h in that area, the algorithm might fail to advance through such a bottleneck and would not generate any nodes in the other part. The FF and SKF algorithms do not suffer from this problem, and it can also be circumvented in PNP by supplying at least one seed node in each problematic part of the domain.

5.7. Direction and dimension independence. Points 7 and 8 deal with direction and dimension independence. The FF algorithm is only two-dimensional and directionally dependent, because the advancing front progresses with respect to the increasing y coordinate. For inconveniently rotated or badly shaped domains, filling via increasing last coordinate might perform badly. Choosing a filling direction is the first step of the algorithm, and it can have a significant effect on the running time and the generated node distribution. The algorithm is also not easily generalizable to higher dimensions, as it is not immediately obvious how to extend the concept of the "closest left" point to higher-dimensional spaces.

The SKF algorithm is better in this aspect. Using PCA it computes oriented bounded boxes, which provides independence from rotations. The main parts of the SKF algorithm, i.e., PCA and Poisson disk sampling, all work in arbitrary dimensions. Similarly, all parts of the PNP algorithm are formulated for a general dimension d and the formulation of the fill procedure is independent of the coordinate system. The same is true for the implementation: there is a single implementation for all values of d and the space dimension can truly be a run-time parameter. The coordinates of points are only accessed in the internals of the k-d tree operations; all other expressions are coordinate-free.

5.8. Free parameters. Point 9 states that the developed algorithm should aim to minimize the number of free or tuning parameters. All three algorithms have a parameter influencing the number of candidates, which represents a time-quality trade-off. Authors of FF set $n = 5$ and anything above has similar distributions with a higher execution time. Authors of SKF analyze the effect of the number of candidates more precisely and recommend $n = 15$ in two and three dimensions, with a higher number of candidates corresponding to lower errors. For the PNP algorithm we similarly recommend $n = 15$ in two dimensions, and $n = 30$ in three dimensions with increasing n for higher dimensions. Anything above $n = 30$ in two dimensions gives very similar results and is computationally wasteful.

6. Solution of PDEs on generated nodes.

6.1. Poisson's equation. The decisive factor of node distribution quality for strong form methods is its ability to support construction of a good approximation of differential operators. A basic test of this ability is to solve Poisson's equation on nodes generated by all three algorithms and compare the accuracy of the solutions.

A d-dimensional boundary value problem

$$\begin{aligned} \nabla^2 u &= f \quad \text{in } \Omega = [0,1]^d, \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{6.1}$$

with $u(x_1, \dots, x_d) = \prod_{i=1}^{d} \sin(\pi x_i)$ and $f(x_1, \dots, x_d) = -d\pi^2 \prod_{i=1}^{d} \sin(\pi x_i)$ is considered in $d = 2$ and $d = 3$ dimensions.

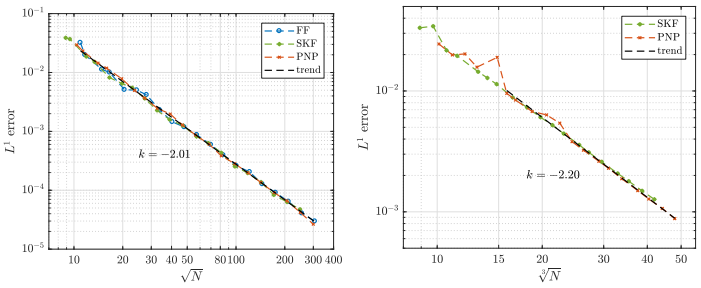

FIG. 13. *Accuracy of the numerical solution of* (6.1) *for considered algorithms when filling* $[0,1]^d$ *with successively smaller densities in two dimensions (left) and three dimensions (right).*

The solution is obtained using the popular strong form RBF-FD method [13, 26, 35]. Polyharmonic radial basis functions (PHS)

$$\varphi(r) = \begin{cases} r^k, & k \text{ odd}, \\ r^k \log r, & k \text{ even} \end{cases} \tag{6.2}$$

with $k = 3$ augmented with monomials up to order 2 are used to construct the approximations on a stencil of 15 closest nodes in two dimensions and 42 closest in three dimensions. The final system is solved using the BiCGSTAB iterative algorithm with tolerance 10^{-15} and 100 iterations with ILUT preconditioner with fill factor 20 and drop tolerance 10^{-5}.

The L^1 error between the correct solution u and obtained solution u_h is evaluated on an independent uniform grid of points G, three times denser than the densest discretization used in the solution of the problem, and computed as

$$L^1 = \|u_h - u\|_1 \approx \frac{1}{|G|} \sum_{p \in G} |u(p) - u_h(p)|. \tag{6.3}$$

Node distributions generated by the three considered algorithms are tested using the same parameters as in subsections 5.1 and 5.4. The nodal spacing function h varies as $h = \frac{1}{n}$ for such n that the total number of nodes N reached approximately $N = 10^5$. The results are shown in Figure 13.

In both the two- and three-dimensional cases we observe convergence with expected rates for large N. All node sets give well-behaved solutions with very similar accuracy. Similar results are obtained in three dimensions.

6.2. Eigenvalue stability. An often observed property of numerical discretization methods is the spectrum of discretized partial differential operator. For example, the spectrum of discretized Laplace operator should have only eigenvalues with negative real part, and a relatively small spread along the imaginary axis [32]. Figure 14 shows the spectrum of Laplace operator discretized with second order RBF-FD PHS on PNP nodes shown in Figure 9. There are no eigenvalues with positive real part and also imaginary spread is relatively small, which additionally confirms the stability of RBF-FD PHS differentiation on scattered nodes.

Additionally, we tested several different setups with different stencil sizes and approximation orders on nodes distributed with all three positioning algorithms with minimal differences observed in the spectrum.

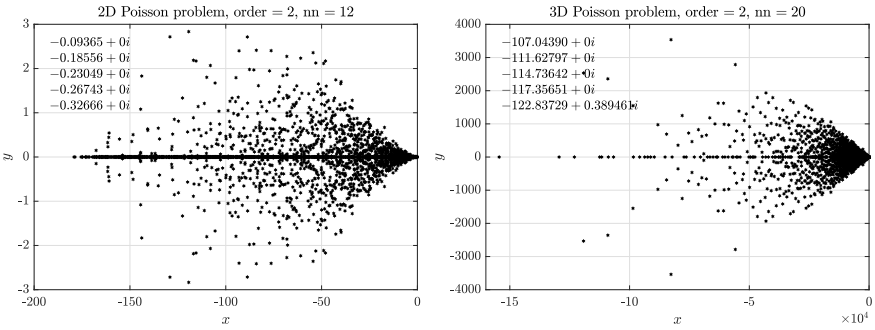

FIG. 14. *Spectra of the Laplacian operator discretized with RBF-FD PHS r^3, augmented with monomials or order* 2 *on PNP nodes. Note the different scales on the axes of both plots. Variable* nn *denotes the number of nearest neighbors used to construct the stencil. The* 5 *eigenvalues with the largest real parts are given in the top left corner of each plot.*

6.3. Thermo-fluid problem. Finally, the PNP algorithm is tested on a more complex problem. The goal is to demonstrate the capability of the meshless solution procedure on PNP nodes solving a transient nonlinear convection dominated problem in two- and three-dimensional irregular domains with mixed Dirichlet and Neumann boundary conditions. The natural convection problem governed by coupled Navier–Stokes, mass continuity, and heat transfer equations is chosen for a test case. First, a well-known de Vahl Davis benchmark test [9] is solved to demonstrate correctness of the solution procedure, both in two and three dimensions. Once we attain confidence in the solution procedure we extend the demonstration to irregular domains.

The natural convection benchmark problem is governed by the following equations:

$$\frac{\partial \boldsymbol{v}}{\partial t} + (\boldsymbol{v}\cdot\nabla)\boldsymbol{v} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2\boldsymbol{v} + \frac{1}{\rho}\boldsymbol{b}, \tag{6.4}$$

$$\nabla\cdot\boldsymbol{v} = 0, \tag{6.5}$$

$$\boldsymbol{b} = \rho(1-\beta(T-T_{\mathrm{ref}}))\boldsymbol{g}, \tag{6.6}$$

$$\frac{\partial T}{\partial t} + \boldsymbol{v}\cdot\nabla T = \frac{\lambda}{\rho c_p}\nabla^2 T, \tag{6.7}$$

where $\boldsymbol{v}(u,v,w)$, p, T, μ, λ, c_p, ρ, $\boldsymbol{g}$, β, T_{ref}, and $\boldsymbol{b}$ stand for velocity, pressure, temperature, viscosity, thermal conductivity, specific heat, density, gravitational acceleration, coefficient of thermal expansion, reference temperature for Boussinesq approximation, and body force, respectively. The de Vahl Davis test is defined on a unit square domain Ω, where vertical walls are kept at constant temperatures with ΔT difference between the cold and hot side, while horizontal walls are adiabatic. In generalization to three dimensions we assume also front and back walls to be adiabatic [37]. No-slip velocity boundary conditions are assumed on all walls. The problem is characterized by Rayleigh (Ra) and Prandtl (Pr) numbers, defined as

$$\mathrm{Pr} = \frac{\mu c_p}{\lambda}, \mathrm{Ra} = \frac{g\beta\rho c_p \Delta T h^3}{\lambda\mu}, \tag{6.8}$$

with h standing for characteristic length, in our case set to 1. All cases considered in this paper are computed at $\mathrm{Pr} = 0.71$.

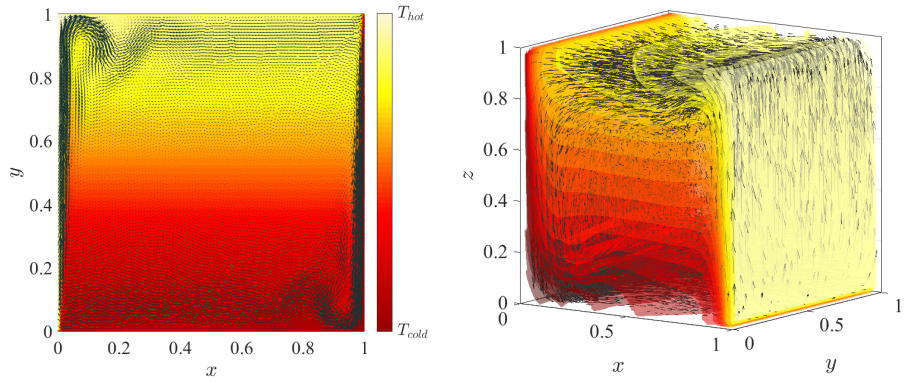

FIG. 15. *Temperature contour and velocity quiver plots for the* $\mathrm{Ra} = 10^8$ *case in two dimensions (left) and the* $\mathrm{Ra} = 10^6$ *case in three dimensions (right).*

The problem is solved with implicit time stepping, where each time step begins with a computation of intermediate velocity $(\tilde{\boldsymbol{v}}_2)$

$$\tilde{\boldsymbol{v}}_2 = \boldsymbol{v}_1 + \Delta t \left[-(\boldsymbol{v}_1 \cdot \nabla)\tilde{\boldsymbol{v}}_2 + \frac{\mu}{\rho}\nabla^2 \tilde{\boldsymbol{v}}_2 + \frac{1}{\rho}\boldsymbol{b}(T_1) \right]. \tag{6.9}$$

The computed velocity is coupled with mass continuity by an iterative velocity-correction scheme, where it is assumed that the correction depends only on the pressure term

$$\boldsymbol{v}_2 = \tilde{\boldsymbol{v}}_2 - \frac{\Delta t}{\rho}\nabla p. \tag{6.10}$$

Applying divergence on (6.10) yields a pressure Poisson equation

$$\nabla^2 p = \frac{\rho}{\Delta t}\nabla \cdot \tilde{\boldsymbol{v}}_2 \text{ in } \Omega, \quad \frac{\partial p}{\partial n} = \frac{\rho}{\Delta t}\tilde{\boldsymbol{v}}_2 \cdot \boldsymbol{n} \text{ on } \partial\Omega, \quad \text{subjected to } \int_\Omega p = 0, \tag{6.11}$$

which is solved first to get the pressure field. With computed pressure the velocity is corrected following (6.10). Steps (6.11) and (6.10) are repeated until the convergence criterion is not met. Once the velocity is satisfactorily divergence free, the temperature field, coupled with momentum equation through Boussinesq approximation, is updated as

$$T_2 = T_1 + \Delta t \left[-\boldsymbol{v}_2 \cdot \nabla T_2 + \frac{\lambda}{\rho c_p}\nabla^2 T_2 \right]. \tag{6.12}$$

All spatial operators are discretized using RBF-FD with r^3 PHS radial basis functions, augmented with monomials up to order 2, with the closest 25 nodes used as a stencil. For the time discretization time step $\Delta t = 10^{-3}$ was used for all cases. Nodal distance $h = 0.01$ is used for simulations in two dimensions and $h = 0.25$ for simulations in three dimensions. Boundaries with Neumann boundary conditions are additionally treated with ghost nodes [3].

In Figure 15 steady state temperature contour and velocity quiver plots for the $\mathrm{Ra} = 10^8$ case in two dimensions and the $\mathrm{Ra} = 10^6$ case in three dimensions are presented. A more quantitative analysis is done by comparing characteristic values, i.e., peak positions and values of cross section velocities, with data available in the literature [8, 20, 37, 14]. We analyze six different cases, namely, $\mathrm{Ra} = 10^6, 10^7, 10^8$

TABLE 2
Comparison of results computed with RBF-FD on FF nodes and reference data.

	Ra	$v_{max}(x, 0.5)$			x			$u_{max}(0.5, y)$			y		
		present	[8]	[20]	present	[8]	[20]	present	[8]	[20]	present	[8]	[20]
2D	10^6	0.2628	0.2604	0.2627	0.037	0.038	0.039	0.0781	0.0765	0.0782	0.847	0.851	0.861
	10^7	0.2633	0.2580	0.2579	0.022	0.023	0.021	0.0588	0.0547	0.0561	0.870	0.888	0.900
	10^8	0.2557	0.2587	0.2487	0.010	0.011	0.009	0.0314	0.0379	0.0331	0.918	0.943	0.930
	Ra	$w_{max}(x, 0.5, 0.5)$			x			$u_{max}(0.5, 0.5, z)$			z		
		present	[37]	[14]	present	[37]	[14]	present	[37]	[14]	present	[37]	[14]
3D	10^4	0.2295	0.2218	0.2252	0.850	0.887	0.883	0.2135	0.1968	0.2013	0.168	0.179	0.183
	10^5	0.2545	0.2442	0.2471	0.940	0.931	0.935	0.1564	0.1426	0.1468	0.144	0.149	0.145
	10^6	0.2564	0.2556	0.2588	0.961	0.965	0.966	0.0841	0.0816	0.0841	0.143	0.140	0.144

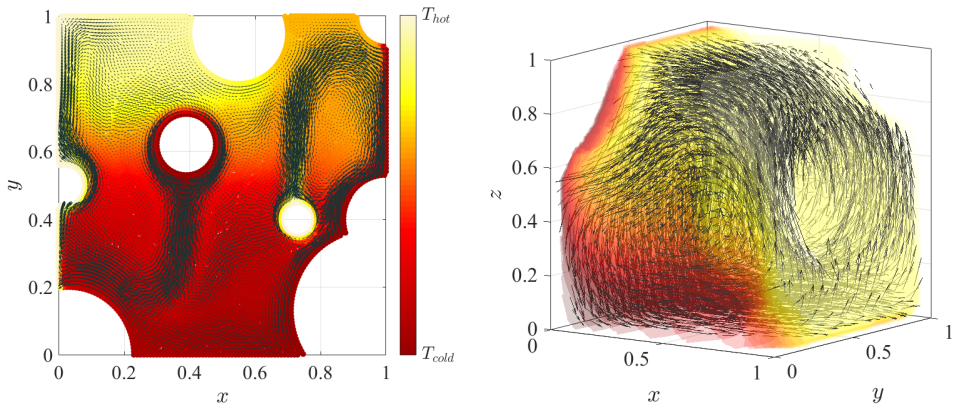

FIG. 16. *Temperature contour and velocity quiver plots of solutions in irregular two-dimensional domain (left) and irregular three-dimensional domain (right).*

in two dimensions, and Ra $= 10^4, 10^5, 10^6$ in three dimensions. The comparison in presented in Table 2.

Finally, in Figure 16 we demonstrate the solution of the transient convection dominated problem in an irregular two- and three-dimensional domain with mixed Dirichlet–Neumann boundary conditions on nodes positioned with the proposed algorithm. Note that this case, a solution of natural convection in an irregular domain, includes several potential complications, such as Neumann boundary conditions on curved boundaries, concavities, convection dominated transport, and nonlinearities.

7. Conclusions. A new algorithm for generating variable density node distributions in interiors of arbitrary dimensions is proposed. The algorithm has many desirable properties, such as direction independence, support for irregular domains by only discretizing the area actually contained in the domain interior, good compatibility with boundary discretizations, and good scaling behavior. We prove that the time complexity of the proposed algorithm scales as $O(N)$ for constant spacing and $O(N \log N)$ for variable spacing. A minimal nodal spacing guarantee for constant and variable nodal spacing functions is also proven. With examples it is shown that the proposed algorithm produces locally smooth distributions that are suitable for the RBF-FD method for solving partial differential equations. The algorithm is compared against two other state-of-the-art algorithms, and the summary of the findings is presented in Table 3.

TABLE 3
Comparison of FF, SKF, and PNP algorithms.

Property/algorithm	FF	SKF	PNP
Supports variable density	yes	no	yes
Supports three-dimensional distributions	no	yes	yes
Supports irregular domains	yes, using BB	yes, using OBB	yes, natively
Compatibility with boundary nodes	n/a	no	yes
Dimension independence	no	yes	yes
Direction independence	no	yes	yes
Randomized	minimal (only starting line)	yes (fully)	yes (controlled)
Minimal spacing guarantees	no	yes (constant h)	yes (constant and variable h)
Time complexity	$O\left(n\left(\frac{\lvert \mathrm{bb}(\Omega)\rvert}{\lvert\Omega\rvert}N\right)^{1.5}\right)$	$O\left(n\frac{\lvert \mathrm{obb}(\Omega)\rvert}{\lvert\Omega\rvert}N\right)$	$O(nN\log N)$ ($O(nN)$ if h constant)
Computational time	best for smaller N, 5 s for 10^6 nodes	6 s for 10^6 nodes	10 s for 10^6 nodes, 2 s if h constant
PDE accuracy	satisfactory	satisfactory with larger support sizes	satisfactory
Number of free parameters	1 (no. of cand. n)	1 (no. of cand. n)	1 (no. of cand. n)

The algorithm is also included in the Medusa library [27] for solving PDEs with strong form meshless methods, but a standalone implementation of the algorithm is available from the library's website as well [33].

At least three directions are open for future research. The first one deals with effective adaptive modification of parts of generated distributions with target complexity $O(N_{\text{old}} + N_{\text{new}})$, where N_{old} and N_{new} stand for the number of removed old nodes and the number of added new nodes. The second direction is to generalize the algorithm to (parametric) surfaces, again with desired $O(N)$ time complexity irrespective of the surface. The third direction is to investigate parallelization opportunities on different parallel architectures ranging from shared memory multicore central processing units and general purpose graphics processing units to distributed computing. A potential approach, suitable for shared memory, is to independently build the discretization from several seed nodes. The bottleneck in such an approach is the manipulation of global kd-tree search structure, especially on GPGPUS. Alternative simpler search structures, such as spatial grids, could be used instead, as is common practice in computer graphics community. A second option, also suitable for distributed computing, is via domain decomposition, where main problems arise in load balancing and appropriate partitioning of the complex higher-dimensional domains.

REFERENCES

[1] M. BALZER, T. SCHLÖMER, AND O. DEUSSEN, *Capacity-constrained point distributions: A variant of Lloyd's method*, ACM Trans. Graphics, 28 (2009), https://doi.org/10.1145/1531326.1531392.

[2] S. BAUER, *Image number K7245 − 1*, United States Department of Agriculture, https://www.ars.usda.gov/oc/images/photos/k7245-1/.

[3] V. BAYONA, N. FLYER, B. FORNBERG, AND G. A. BARNETT, *On the role of polynomials in RBF-FD approximations:* II. *Numerical solution of elliptic PDEs*, J. Comput. Phys., 332 (2017), pp. 257–273, https://doi.org/10.1016/j.jcp.2016.12.008.

[4] J. L. Blanco and 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.
[5] R. Bridson, *Fast Poisson Disk Sampling in Arbitrary Dimensions*, in Proceedings of SIGGRAPH sketches, 2007, p. 22, https://doi.org/10.1145/1278780.1278807.
[6] Y. Choi and S. Kim, *Node generation scheme for meshfree method by Voronoi diagram and weighted bubble packing*, in Proceedings of the Fifth U.S. National Congress on Computational Mechanics, Boulder, CO, 1999.
[7] R. L. Cook, *Stochastic sampling in computer graphics*, ACM Trans. Graphics, 5 (1986), pp. 51–72, https://doi.org/10.1145/7529.8927.
[8] H. Couturier and S. Sadat, *Performance and accuracy of a meshless method for laminar natural convection*, Numer. Heat Trans. B, 37 (2000), pp. 455–467, https://doi.org/10.1080/10407790050051146.
[9] G. de Vahl Davis, *Natural convection of air in a square cavity: A bench mark numerical solution*, Internat. J. Numer. Methods Fluids, 3 (1983), pp. 249–264, https://doi.org/10.1002/fld.1650030305.
[10] D. Dimitrov, C. Knauer, K. Kriegel, and G. Rote, *On the bounding boxes obtained by principal component analysis*, in Proceedings of the 22nd European Workshop on Computational Geometry, 2006, pp. 193–196.
[11] 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, https://doi.org/10.1016/j.camwa.2015.01.009.
[12] B. Fornberg and N. Flyer, *A Primer on Radial Basis Functions with Applications to the Geosciences*, SIAM, Philadelphia, 2015, https://doi.org/10.1137/1.9781611974041.
[13] B. Fornberg and N. Flyer, *Solving PDEs with radial basis functions*, Acta Numer., 24 (2015), pp. 215–258, https://doi.org/10.1017/S0962492914000130.
[14] T. Fusegi, J. M. Hyun, K. Kuwahara, and B. Farouk, *A numerical study of three-dimensional natural convection in a differentially heated cubical enclosure*, Int. J. Heat Mass Transf., 34 (1991), pp. 1543–1557, https://doi.org/10.1016/0017-9310(91)90295-p.
[15] A. Golbabai and E. Mohebianfar, *A new method for evaluating options based on multi-quadric RBF-FD method*, Appl. Math. Comput., 308 (2017), pp. 130–141, https://doi.org/10.1016/j.amc.2017.03.019.
[16] G. Guennebaud, B. Jacob, et al., *Eigen v*3, 2010, http://eigen.tuxfamily.org.
[17] D. P. Hardin and E. B. Saff, *Discretizing manifolds via minimum energy points*, Not. AMS, 51 (2004), pp. 1186–1194.
[18] I. Jolliffe, *Principal Component Analysis*, 2nd ed., Springer Ser. Statist., Springer, New York, 2011, https://doi.org/10.1007/978-3-642-04898-2_455.
[19] G. Kosec, *A local numerical solution of a fluid-flow problem on an irregular domain*, Adv. Eng. Softw., 120 (2018), pp. 36–44, https://doi.org/10.1016/j.advengsoft.2016.05.010.
[20] G. Kosec and B. Šarler, *Solution of thermo-fluid problems by collocation with local pressure correction*, Internat. J. Numer. Methods Heat Fluid Flow, 18 (2008), pp. 868–882, https://doi.org/10.1108/09615530810898999.
[21] G. Kosec and J. Slak, *RBF-FD based dynamic thermal rating of overhead power lines*, in Advances in Fluid Mechanics XII, WIT Trans. Eng. Sci. 120, WIT Press, Southampton, 2018, pp. 255–262, https://doi.org/10.2495/afm180261.
[22] X.-Y. Li, S.-H. Teng, and A. Ungor, *Point placement for meshless methods using sphere packing and advancing front methods*, in Proceedings of ICCES'00, Los Angeles, CA, 2000.
[23] G.-R. Liu, *Mesh Free Methods: Moving Beyond the Finite Element Method*, CRC Press, Boca Raton, FL, 2002, https://doi.org/10.1201/9781420040586.
[24] Y. Liu, Y. Nie, W. Zhang, and L. Wang, *Node placement method by bubble simulation and its application*, CMES Comput. Model. Eng. Sci. , 55 (2010), pp. 89–110, https://doi.org/10.3970/cmes.2010.055.089.
[25] R. Löhner and E. Oñate, *A general advancing front technique for filling space with arbitrary objects*, Internat. J. Numer. Methods Engrg., 61 (2004), pp. 1977–1991, https://doi.org/10.1002/nme.1068.
[26] B. Mavrič and B. Šarler, *Local radial basis function collocation method for linear thermoelasticity in two dimensions*, Internat. J. Numer. Methods Heat Fluid Flow, 25 (2015), pp. 1488–1510, https://doi.org/10.1108/hff-11-2014-0359.
[27] *Medusa Library*, http://e6.ijs.si/medusa/.
[28] S. A. Mitchell, A. Rand, M. S. Ebeida, and C. Bajaj, *Variable radii Poisson-disk sampling, extended version*, in Proceedings of the 24th Canadian Conference on Computational Geometry, Vol. 5, 2012.

[29] A. W. Moore, *An Introductory Tutorial on KD-trees*, Ph.D. thesis, University of Cambridge, Cambridge, 1991, https://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1991_1/moore_andrew_1991_1.pdf.

[30] 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.

[31] K. Reuther, B. Sarler, and M. Rettenmayr, *Solving diffusion problems on an unstructured, amorphous grid by a meshless method*, Int. J. Therm. Sci., 51 (2012), pp. 16–22, https://doi.org/10.1016/j.ijthermalsci.2011.08.017.

[32] V. Shankar, R. M. Kirby, and A. L. Fogelson, *Robust node generation for meshfree discretizations on irregular domains and surfaces*, SIAM J. Sci. Comput., 40 (2018), pp. 2584–2608, https://doi.org/10.1137/17m114090x.

[33] J. Slak and G. Kosec, *Standalone Implementation of the Proposed Node Placing Algorithm*, http://e6.ijs.si/medusa/static/PNP.zip.

[34] J. Slak and G. Kosec, *Fast generation of variable density node distributions for mesh-free methods*, in WIT Trans. Eng. Sci., 122 (2018), pp. 163–173, https://doi.org/10.2495/be410151.

[35] J. Slak and G. Kosec, *Refined meshless local strong form solution of Cauchy–Navier equation on an irregular domain*, Eng. Anal. Bound. Elem., 100 (2018) pp.3–13, https://doi.org/10.1016/j.enganabound.2018.01.001.

[36] J. Slak and G. Kosec, *Adaptive radial basis function-generated finite differences method for contact problems*, Internat. J. Numer. Methods Engrg., 2019, https://doi.org/10.1002/nme.6067.

[37] P. Wang, Y. Zhang, and Z. Guo, *Numerical study of three-dimensional natural convection in a cubical cavity at high Rayleigh numbers*, Int. J. Heat Mass Transf., 113 (2017), pp. 217–228, https://doi.org/10.1016/j.ijheatmasstransfer.2017.05.057.