MIPRO 2021, September 27 - October 01, 2021, Opatija, Croatia

Discretized Boundary Surface Reconstruction

Mitja Jančič[1,2], Viktor Cvrtila[3], Gregor Kosec[2]
[1] “Jožef Stefan” International Postgraduate School, Ljubljana, Slovenia
[2] “Jožef Stefan” Institute, Parallel and Distributed Systems Laboratory, Ljubljana, Slovenia
[3] Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
mitja.jancic@ijs.si, cvrtilaviktor@gmail.com, gregor.kosec@ijs.si

Abstract **– Domain discretization is an essential part of the solution procedure in numerical simulations. Meshless methods simplify the domain discretization to positioning of nodes in the interior and on the boundary of the domain. However, generally speaking, the shape of the boundary is often undefined and thus needs to be constructed before it can be discretized with a desired internodal spacing. Domain shape construction is far from trivial and is the main challenge of this paper. We tackle the simulation of moving boundary problems where the lack of domain shape information can introduce difficulties. We present a solution for 2D surface reconstruction from discretization points using cubic splines and thus providing a surface description anywhere in the domain. We also demonstrate the presented algorithm in a simulation of phase-change-like problem.**

Keywords – meshless; moving boundary; surface reconstruction; simulation; cubic splines

I. Introduction

Tractable solutions to partial differential equations (PDEs) are not easily obtained. Often advanced mathematical procedures or a series of simplifications are needed to obtain a closed form solution to a real-life problem [1]. Therefore, in practice, we often rely on numerical treatment that provides us with a numerical approximation. For that, different numerical methods for solving PDEs have been proposed. Most commonly used, e.g. Finite Difference Method [2], Finite Element Method [3], Finite Volume Method [4], Boundary Element Method [5], require a mesh to operate, while meshless methods approximate the differential operators only using scattered nodes [6] as shown in Fig. 1. This is an important advantage as the node positioning is considered to be easier then mesh generation, however, far from trivial. For that reason, several dedicated node positioning algorithms emerged [7, 8, 9, 10].

Historically speaking, meshless methods were introduced in the 1990s. Since then, different numerical procedures have been proposed, e.g. meshless Element Free Galerkin [11], the Local Petrov-Galerkin [12], h-p Cloud Method [13] and others. In this paper, we will use the meshless generalization of the traditional finite difference method (FDM) – the Radial Basis Function-generated Finite Differences (RBF-FD) originally proposed by Tolstykh [14]. The RBF-FD has already been used in a vast variety of applications ranging from linear elasticity [15], 4-dimensional problems [16], geosciences [17], fluid mechanics [10], dynamic thermal rating of power lines [18], etc.

The fact that the domain discretization in the context of meshless methods is heavily simplified makes the meshless methods very attractive in the context of moving boundary problems, e.g. phase change problems [19]. Providing a good discretization of a moving phase front is no easy task, as satisfying the quasi-uniform internodal spacing h is crucial to assure the stability of the numerical method [20]. The only way to satisfy the quasi-uniform spacing h condition on moving boundary problems is by repositioning the nodes from the domain. However, the node repositioning needs to bo performed with a minimum cost to domain shape distortion. To reduce the distortion, the domain boundary shape must be known even between the discretization points, meaning, a proper surface reconstruction algorithm from a set of boundary nodes is needed.

Surface reconstruction has already been addressed in the context of numerical simulations – Non-uniform rational basis splines (NURBS), often used in computer graphics for representing curves and surfaces [21], are used in numerical simulations using the finite element analysis, e.g. Isogeometric analysis (IGA) [22]. In this paper, we tackle surface reconstruction in two-dimensional domain space where the surface is represented as a two-dimensional curve. We provide an algorithm that uses cubic splines to reconstruct the domain shape from the discretization points and thus provides us with the domain shape information anywhere on the boundary.

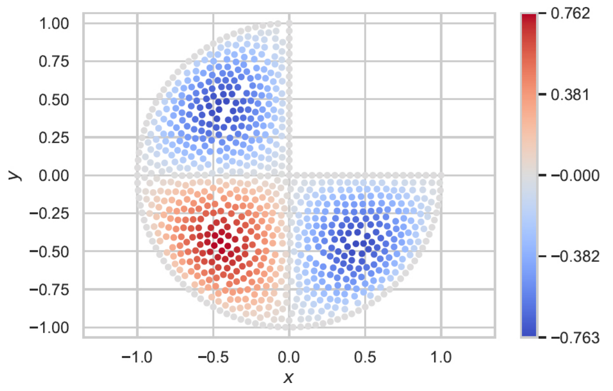

Figure 1. Example solution of Poisson’s problem with Dirichlet boundary conditions on $N = 881$ scattered nodes in the domain.

The paper is organized as follows: In section II a short presentation to local strong form meshless meth-

Authorized licensed use limited to: Institute Jozef Stefan. Downloaded on December 07,2022 at 07:22:38 UTC from IEEE Xplore. Restrictions apply.

ods, namely the RBF-FD is given. In section III and section IV our proposed surface reconstruction algorithm is thoroughly explained. The algorithm is then used to solve a problem from section V, while results are presented in section VI. Final conclusions are gathered in the final section VII.

II. Local strong form meshless methods

A general idea of the meshless methods is to use the local discretization points and construct an approximation of the considered field. This approximation is later used for manipulation with differential operators using the ansatz

$$(\mathcal{L}u)(\boldsymbol{x}) \approx \sum_i w_i^{\mathcal{L}} u(\boldsymbol{p}_i). \tag{1}$$

Here $\mathcal{L}$ is a differential operator and index i runs over the set of closest neighbors $\boldsymbol{p}_i$ of $\boldsymbol{x}$. Equality of (1) is enforced and the weights $\boldsymbol{w}$ are computed. Different methods can be used to compute the weights, we will use the RBF-FD. Often used RBFs, e.g. Gaussians, include a shape parameter that can play a crucial role in the overall method stability [23]. However, using Polyharmonic splines (PHS) and additionally augmenting them with polynomials helps overcome the stability issues [24].

Some solution procedures, including the RBF-FD based, have been implemented using the object-oriented approach and C++'s strong template system. Node positioning, support selection, differential operator approximation, PDE discretization and other modules are all available as part of the Medusa library [25], also used in this paper.

III. Surface reconstruction

Consider a moving boundary problem. The altering distance between the neighboring nodes can, firstly, increase to a point where the numerical methods become unstable due to the violation of quasi-uniform internodal spacing h requirement, or secondly, become too large to achieve a desired accuracy of numerical solution in a specific domain area. To avoid such difficulties, repositioning the nodes in the domain is required, however, during the repositioning process the domain shape must be preserved as much as possible. Since the domain shape is, generally speaking, undefined, it first has to be constructed from a set of points before the boundary can be discretized. In this section we present a solution for 2D surface reconstruction from a set of points using cubic splines and provide complete information about the surface shape at hand.

Suppose we are given a set of k points

$$X = \{\boldsymbol{x}_i \in \mathbb{R}^2,\ i = 0, \dots, k-1\}$$

representing the boundary $\partial\Omega$ of the domain Ω, which is parametrized by a *Jordan curve* $\boldsymbol{\gamma}\colon [a,b] \to \mathbb{R}^2$. We do not have access to neither the domain nor the curve. Suppose there exist knots $\{t_j \in [a,b], j = 0, \dots, k-1\}$ such that $\boldsymbol{\gamma}(t_j) = \boldsymbol{x}_i$ for all j. The points are given in no particular order, i.e. $0 \le i < j \le k-1$ does not necessarily imply that $t_i < t_j$. Our task is to find a curve $\tilde{\boldsymbol{\gamma}}\colon [\tilde{a}, \tilde{b}] \to \mathbb{R}^2$ that approximates the original curve $\boldsymbol{\gamma}$. As there are many possible curves that interpolate X, obtaining the original curve is impossible without providing additional information or constraints. We, therefore, assume that the given points are *dense enough* to adequately describe the curve in the following way.

Let $Y = \boldsymbol{\gamma}(\mathbb{R}) \subset \mathbb{R}^2$ be the image of the curve $\boldsymbol{\gamma}$. Suppose $\boldsymbol{x}_p$ and $\boldsymbol{x}_q$ are neighboring points to $\boldsymbol{x}_i$ in the sense that t_i is the only knot between t_p and t_q. The indices p and q are both dependant on the choice of i, i.e. $p = p(i)$ and $q = q(i)$, however, the explicit dependency is omitted in our writing. Let us then define an open neighborhood $U_i = \{\boldsymbol{\gamma}(t), t_p < t < t_q\}$ of $\boldsymbol{x}_i$ in Y. To help clarify the notation used, an illustration is provided in Fig. 2.

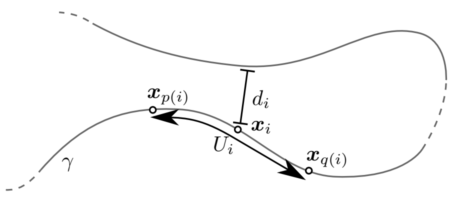

Figure 2. The notation introduced in section III.

Additionally, define

$$d_i = \mathrm{d}(\boldsymbol{x}_i, Y - U_i)$$

where d is the standard metric on $\mathbb{R}^2$. As long as

$$d_i \ge \max\{\mathrm{d}(\boldsymbol{x}_i, \boldsymbol{x}_p), \mathrm{d}(\boldsymbol{x}_i, \boldsymbol{x}_q)\} \tag{2}$$

and

$$\operatorname*{arg\,min}_{j=0,\dots,k-1} \mathrm{d}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \{p, q\} \tag{3}$$

for all $i = 0, \dots, k$, we can find one of the neighboring points $\boldsymbol{x}_p$ or $\boldsymbol{x}_q$ of an arbitrary point $\boldsymbol{x}_i$ by finding the nearest point among $\boldsymbol{x}_j$ for $j = 0 \dots, i-1, i+1, \dots, k-1$.

When inequality (2) is not satisfied, the discretization alone does not provide sufficient information to determine which points of X are in which part of the curve $\boldsymbol{\gamma}$. A similar problem can occur if condition (3) does not hold for some index, i.e. if the nearest neighbor to $\boldsymbol{x}_i$ is not $\boldsymbol{x}_p$ or $\boldsymbol{x}_q$.

IV. Algorithm and implementation

The surface reconstruction algorithm works in three steps. First we determine the correct order of given points, i.e. we find a permutation σ on $\{0, \dots, k-1\}$, such that $\sigma(i) < \sigma(j)$ implies that $t_i < t_j$ for all indices $0 \le i, j \le k-1$. In the second step, an approximation $\tilde{\boldsymbol{\gamma}}$ of the starting curve $\boldsymbol{\gamma}$ is obtained by fitting a cubic spline on ordered starting points. Finally, in the third step, we use the node positioning algorithm [9] to discretize the curve for use in further calculations.

Ordering the starting points

To find the appropriate permutation σ, firstly, the list of ordered points $\{\boldsymbol{x}'_i\}_{i=0}^{k-1}$ is initialized and an arbitrary starting point is chosen and assigned to the first position $\boldsymbol{x}'_0$. Now find the nearest neighbor $\boldsymbol{x}_p$ of $\boldsymbol{x}'_0$, and assign it to $\boldsymbol{x}'_1$. The list is then build up inductively: Once $\boldsymbol{x}'_j$ is defined, find the nearest neighbor $\boldsymbol{x}_p$ to $\boldsymbol{x}'_j$. If $\boldsymbol{x}_p$ is

Authorized licensed use limited to: Institute Jozef Stefan. Downloaded on December 07,2022 at 07:22:38 UTC from IEEE Xplore. Restrictions apply.

not $\boldsymbol{x}'_{j-1}$, set $\boldsymbol{x}'_{j+1}$ to $\boldsymbol{x}_p$. Otherwise let $\boldsymbol{x}_p$ be the second nearest neighboring point. Now compare $\boldsymbol{x}_p$ to $\boldsymbol{x}'_{j-2}$. If these are not equal, set $\boldsymbol{x}'_{j+1}$ to $\boldsymbol{x}_p$. The process is repeated until we cannot find a point $\boldsymbol{x}_p$ that is not already in the ordered list. The process is also presented as pseudocode in Alg. 1.

In this paper, a k-d tree constructed from X is used to query for nearest neighbors. Since it is more economical to store two arrays of indices rather than two arrays of vectors, the permutation is stored instead of a full list of ordered vectors when implementing the ordering procedure.

Algorithm 1: Point enumeration

```
   Data: An array of points on the plane $x[k]$.
   Result: An array representing a desired
           permutation $\sigma[k]$.
   Result: An array representing the inverse
           permutation $\sigma^{-1}[k]$.
1  begin
2      create integer arrays $\sigma[k]$, $\sigma^{-1}[k]$;
3      initialize k-d tree $T$ based on points $x$;
4      set $\sigma[0] \leftarrow 0$, $\sigma^{-1}[0] \leftarrow 0$;
5      for $j \leftarrow 0$ to $k-2$ do
6          set $n_{\text{knn}} \leftarrow 2$;
7          while $\sigma[j+1]$ is not set do
8              find the $n_{\text{knn}}$-th nearest point $x[p]$ to
                 $x[\sigma[j]]$ in $T$;
9              if $p$ does not equal $\sigma[j - n_{knn} + 1]$
                 then
10                 set $\sigma[j+1] \leftarrow p$, $\sigma^{-1}[p] \leftarrow j+1$;
11             else
12                 increment $n_{\text{knn}}$ by one;
13             end
14         end
15     end
16 end
```

Domain shape reconstruction

The second step of the surface reconstruction fits a cubic spline $\tilde{\boldsymbol{\gamma}} \colon \mathbb{R} \to \mathbb{R}^2$ to the points from the ordered list $\{\boldsymbol{x}'_i\}_i$. When $\tilde{\boldsymbol{\gamma}}$ is obtained, all the surface shape information required is at our disposal - even between the discretization nodes. However to be able to reconstruct the entire domain Ω in the third and final step of the surface reconstruction algorithm, we must be able to distinguish between the interior and exterior of the curve $\tilde{\boldsymbol{\gamma}}$.

Suppose we are given a set of points $\{\boldsymbol{z}_i\}_i \subset \partial\Omega$ and a set of accompanying normals $\{\boldsymbol{n}_i\}_i \subset S^1$. To determine if an arbitrary point $\boldsymbol{z}$ lies in Ω, find the nearest point $\boldsymbol{z}_i$ to $\boldsymbol{z}$, and check if the vectors $\boldsymbol{z} - \boldsymbol{z}_i$ and $\boldsymbol{n}_i$ point in opposite directions. This can be done by computing the scalar product

$$\langle \boldsymbol{z} - \boldsymbol{z}_i, \boldsymbol{n}_i \rangle \,. \tag{4}$$

If the above scalar product (4) is negative, we conclude that $\boldsymbol{z} \in \Omega$, otherwise not.

However, making any conclusions based on the sign of the equation (4) is not reliable. This basic idea typically fails in the proximity of sharp corners of $\boldsymbol{\gamma}$ or more generally speaking, where the curve is not differentiable. We thus modify the algorithm to use the information provided by the interpolating spline $\tilde{\boldsymbol{\gamma}}$. Let

$$s_0 < \cdots < s_{k-1} < s_k$$

be the knots for the interpolated points $\tilde{\boldsymbol{\gamma}}(s_i) = \boldsymbol{x}'_i$ for all $i = 0, \ldots, k-1$ and $\tilde{\boldsymbol{\gamma}}(s_k) = \boldsymbol{x}'_0$. For an arbitrary query point $\boldsymbol{x}$, we find the closest point on the curve by minimizing the function

$$f(t) = \mathrm{d}(\boldsymbol{x}, \tilde{\boldsymbol{\gamma}}(t)). \tag{5}$$

Note that f typically has many local minima. To obtain the correct one, we find the nearest point $\boldsymbol{x}_q$ to $\boldsymbol{x}$ among X and use the inverse permutation $p = \sigma^{-1}(q)$. The desired value t can now be approximated using bisection on the interval $[s_{p-1}, s_{p+1}]$. Suppose t_{min} is the correct global minimum of f. Generally speaking t_{min} is dependant on $\boldsymbol{x}$, but for the sake of brevity we use $t_{\text{min}} = t_{\text{min}}(\boldsymbol{x})$ unless otherwise specified.

Now the scalar product (4) is rewritten to take form

$$\left\langle \boldsymbol{x} - \tilde{\boldsymbol{\gamma}}(t_{\text{min}}), \tilde{\boldsymbol{\gamma}}''(t_{\text{min}}) \right\rangle . \tag{6}$$

Note that this procedure requires that the normals 'point outwards'. Our construction for $\tilde{\boldsymbol{\gamma}}$ does not guarantee this, which we compensate for by introducing a constant c

$$c = -\operatorname{sgn}\left(\left\langle \boldsymbol{x}_{\text{int}} - \tilde{\boldsymbol{\gamma}}(t_{\text{min}}(\boldsymbol{x}_{\text{int}})), \tilde{\boldsymbol{\gamma}}''(t_{\text{min}}(\boldsymbol{x}_{\text{int}}))\right\rangle\right), \tag{7}$$

as the sign of the value of equation (6) when applied to a point $\boldsymbol{x}_{\text{int}}$ from the interior, i.e. $\boldsymbol{x}_{\text{int}} \in \Omega$. Note the constant c essentially flips the normals.

The equation (6) is then finally modified to

$$\left\langle \boldsymbol{x} - \tilde{\boldsymbol{\gamma}}(t_{\text{min}}), c\tilde{\boldsymbol{\gamma}}''(t_{\text{min}}) \right\rangle \tag{8}$$

and enables us to determine if a point $\boldsymbol{x}$ is in the interior of Ω or not.

Discretization

The only remaining step for a complete surface reconstruction is to discretize the curve $\tilde{\boldsymbol{\gamma}}$. The discretization is done by employing the node positioning algorithm proposed in [9]. A detailed description of the node positioning algorithm used is out of the scope of this paper.

V. Problem setup

We demonstrate the proposed surface reconstruction algorithm from chapter IV on a simplified moving-boundary problem – a simulation of dendrite-like growth also known as *solidification procedure*. A dendrite in metallurgy is a typical tree-like crystal structure that develops as molten metal solidifies [26]. The dynamics of a real-life problem is thus mainly governed by the phase-transition physics from molten metal to a solid crystal structure.

A. Moving boundary

Let the initial domain $\Omega = B_m / B_d$ be an annulus between a larger static circle B_m with radius R_m representing the boundary of the molten metal and smaller non-

Authorized licensed use limited to: Institute Jozef Stefan. Downloaded on December 07,2022 at 07:22:38 UTC from IEEE Xplore. Restrictions apply.

static circle B_d initially with radius $R_d < R_m$ representing the dendrite's initial state

$$B_m = \left\{ \boldsymbol{x} \in \mathbb{R}^2,\ \|\boldsymbol{x}\| \leq R_m \right\} \text{ and}$$
$$B_d = \left\{ \boldsymbol{x} \in \mathbb{R}^2,\ \|\boldsymbol{x}\| \leq R_d \right\}.$$

Generally speaking the velocity of the phase-transition front during the solidification is a function of the temperature field in the proximity. However, in this work, we simplify the problem to a degree, where this dependency is discarded – the velocity is instead synthetically defined to result in a dendrite-like shape. To achieve that, all nodes $\boldsymbol{x}_i$ from the ∂B_d boundary are assigned a velocity $\boldsymbol{v}_i = \boldsymbol{v}_i(\boldsymbol{x}_i)$ that depends on the position of the node $\boldsymbol{x}_i$ and on the boundary normal $\boldsymbol{n}_i = \boldsymbol{n}_i(\boldsymbol{x}_i)$

$$\boldsymbol{v}_i = v_d \Big(\frac{1}{20} + \cos^2(2\phi_i) \Big) \boldsymbol{n}_i. \tag{9}$$

Here, v_d is a constant model parameter and ϕ_i is the polar angle of $\boldsymbol{x}_i$. This essentially means that at any given time step all the nodes from the dendrite boundary are moved $\boldsymbol{x}_i^{t+\mathrm{d}t} = \boldsymbol{x}_i^t + \mathrm{d}t\boldsymbol{v}_i(\boldsymbol{x}_i)$ which results in a dendrite-like growth.

B. *Temperature field*

Although the dynamics of the problem at hand is simplified by uncoupling the phase-transition front velocity and temperature field in the proximity, the latter is still computed at every time step, as shown in our implementation scheme in Fig. 3.

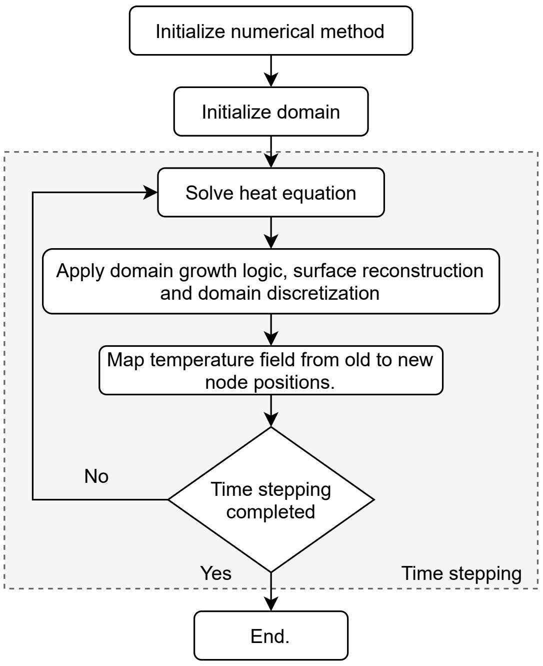
Initialize numerical method
Initialize domain
Solve heat equation
Apply domain growth logic, surface reconstruction and domain discretization
Map temperature field from old to new node positions.
Time stepping completed
No
Yes
Time stepping
End.

Figure 3. Implementation scheme.

The temperature field is governed by the dimensionless heat equation

$$\frac{\partial}{\partial t} T = \nabla^2 T \tag{10}$$

The equation is discretized and finally takes the form

$$T_{i+1}(\boldsymbol{x}) = T_i(\boldsymbol{x}) + \mathrm{d}t\nabla^2 T_i(\boldsymbol{x}) \tag{11}$$

before it is numerically solved for all nodes $\boldsymbol{x} \in \Omega$.

VI. Results

The problem from section V is now simulated using our in-house Medusa library for meshless simulations. Since this is a theoretical problem, the simulation is done in a dimensionless sense.

The outer radius R_m, representing the molten metal boundary, is constant and set to 1, while the initial radius of B_d is set to $R_d = 0.1$. On every time step all the dendrite boundary nodes are assigned a velocity as defined in equation (9), where $v_d = 0.04$. The temperature field is obtained on every time step before the domain growth logic is applied, as noted in implementation scheme in Fig. 3. After the surface had been reconstructed and nodes repositioned, it is important to map the temperature field from the old node positions to the new. This is achieved using the Inverse distance weighting (IDW), a procedure also known as Sheppard's interpolation [1].

In this work, we used the RBF-FD with polyharmonic splines augmented with monomials of second order to compute the weights from equation (1) and consequently compute the temperature field in the interior of Ω. The temperatures at both boundaries, i.e. molten metal and dendrite boundary, are kept constant at 1 and 0 respectively.

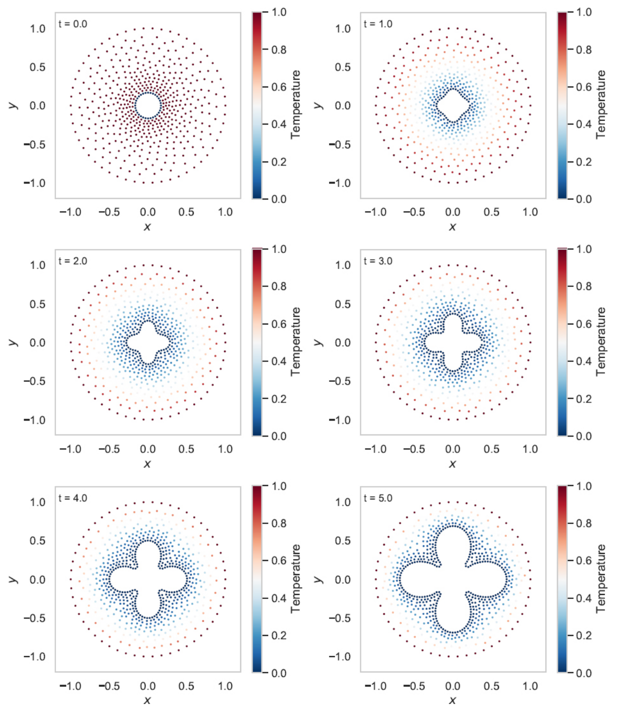

Figure 4. Timelapse of dendrite-like growth using the proposed surface reconstruction algorithm.

The simulation is done by explicit time-marching for $N_t = 500$ time steps where a single time step is $\mathrm{d}t = 0.01$ units long, with a total simulation time $t_{tot} = 5$ units.

Authorized licensed use limited to: Institute Jozef Stefan. Downloaded on December 07,2022 at 07:22:38 UTC from IEEE Xplore. Restrictions apply.

The timelapse of the simulation is demonstrated in Fig. 4 for some selected simulation times. Also notice that the internodal distance $h(\boldsymbol{x})$ is smaller closer to the dendrite boundary, which is of great importance in a more realistic case where the dendrite tip velocity is a function of the temperature field in the proximity and thus needs to be accurately computed. In our case, the decreasing internodal distance h provides us a with well defined and distinguishable dendrite shape.

The number of nodes at the initialization time is $N = 506$, while there are $N = 637$ nodes at the simulation end. The number of nodes increases as the dendrite boundary is moving but also has a finer discretization compared to its surroundings. The total execution time is approximately 21 seconds on a `Windows Linux Subsystem (Ubuntu 20.04 LTS)` with `Intel(R) Core(TM) i7-9750H CPU @ 2.6GHz` and `16 GB` of RAM. The C++ code was compiled using `g++ (GCC) 9.3.0` for Linux with `-fopenmp -O3 -DNDEBUG` flags.

It is important to note that between two time steps all the domain shape information, apart from the discretization nodes, is discarded. Therefore, the domain shape is reconstructed at every single time step. In Fig. 4 we can observe how even after the 500 simulation time steps, the shape of the boundary remains smooth and symmetric. There are no visible irregularities or sights of unexpected discontinuities in the direction of normal vectors. This observation allows us to conclude that the proposed algorithm for surface reconstruction is stable.

VII. Conclusions

A detailed presentation of the proposed surface reconstruction algorithm is provided in this paper. In the first step ordering the discretized points in order is required, in the second step, cubic splines are used to reconstruct the surface shape and finally, in the third step, node positioning algorithm is used to obtain a new set of nodes on the boundary. We also explain how to distinguish between the interior and exterior of the reconstructed domain shape.

Additionally, we demonstrate the proposed algorithm on a moving boundary problem, approximating the dendrite-like shape. At every time step, the surface is reconstructed and temperature field is computed using the RBF-FD approximation. However, in this paper, we discard the coupling between the moving dendrite tip velocity and the temperature field in the proximity. Instead, we assign a synthetically defined velocity to the boundary nodes simulating the dendrite-like growth. The next step could, therefore, be the removal of this simplification and implementation of the actual phase-transition physics making the simulation more realistic.

In this paper, the surface reconstruction is executed at every time step, because we choose to discard all the available information about the domain shape between the discretization nodes after every time step. This is generally speaking unnecessary – some domain information between two time steps doesn't change and could be simply propagated to the next time step.

Acknowledgments

The authors would like to acknowledge the financial support of the ARRS research core funding No. P2-0095.

References

[1] Bor Plestenjak. Numericne metode, 2010.

[2] Gordon D Smith, Gordon D Smith, and Gordon Dennis Smith Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

[3] Olgierd Cecil Zienkiewicz and Robert Leroy Taylor. *The finite element method, vol. 2*. Butterworth-Heinemann, 2000.

[4] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[5] Mohammad H Aliabadi. *The boundary element method, volume 2: applications in solids and structures*, volume 2. John Wiley & Sons, 2002.

[6] Ted Belytschko, Yury Krongauz, Daniel Organ, Mark Fleming, and Petr Krysl. Meshless methods: an overview and recent developments. *Computer methods in applied mechanics and engineering*, 139(1-4):3–47, 1996.

[7] Xiang-Yang Li, Shang-Hua Teng, and Alper Ungor. Point placement for meshless methods using sphere packing and advancing front methods. In *ICCES'00, Los Angeles, CA*. Citeseer, 2000.

[8] Rainald Löhner and Eugenio Oñate. A general advancing front technique for filling space with arbitrary objects. *Int. J. Numer. Methods Eng.*, 61(12):1977–1991, 2004.

[9] Jure Slak and Gregor Kosec. On generation of node distributions for meshless PDE discretizations. *SIAM Journal on Scientific Computing*, 41(5):A3202–A3229, jan 2019.

[10] Gregor Kosec. A local numerical solution of a fluid-flow problem on an irregular domain. *Adv. Eng. Software*, 120:36–44, 2018.

[11] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free galerkin methods. *Int. J. Numer. Methods Eng.*, 37(2):229–256, 1994.

[12] Satya N. Atluri and Tulong Zhu. A new meshless local petrov-galerkin (mlpg) approach in computational mechanics. *Comput. Mech.*, 22(2):117–127, 1998.

[13] C. Armando Duarte and J. Tinsley Oden. H-p clouds—an h-p meshless method. *Numerical Methods for Partial Differential Equations: An International Journal*, 12(6):673–705, 1996.

[14] A. I. Tolstykh and D. A. Shirobokov. On using radial basis functions in a "finite difference mode" with applications to elasticity problems. *Comput. Mech.*, 33(1):68–79, 2003.

[15] Jure Slak and Gregor Kosec. Refined meshless local strong form solution of Cauchy–Navier equation on an irregular domain. *Engineering analysis with boundary elements*, 100:3–13, 2019.

[16] Mitja Jančič, Jure Slak, and Gregor Kosec. Monomial augmentation guidelines for rbf-fd from accuracy versus computational time perspective. *Journal of Scientific Computing*, 87(1):1–18, 2021.

Authorized licensed use limited to: Institute Jozef Stefan. Downloaded on December 07,2022 at 07:22:38 UTC from IEEE Xplore. Restrictions apply.

[17] Bengt Fornberg and Natasha Flyer. *A primer on radial basis functions with applications to the geosciences*, volume 87 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, sep 2015.

[18] M. Maksić, V. Djurica, A. Souvent, J. Slak, M. Depolli, and G. Kosec. Cooling of overhead power lines due to the natural convection. *International Journal of Electrical Power & Energy Systems*, 113:333–343, December 2019.

[19] Juan C. Álvarez Hostos, Erick A. Gutierrez-Zambrano, Joselynne C. Salazar-Bove, Eli S. Puchi-Cabrera, and Alfonso D. Bencomo. Solving heat conduction problems with phase-change under the heat source term approach and the element-free galerkin formulation. *International Communications in Heat and Mass Transfer*, 108:104321, 2019.

[20] Bengt Fornberg and Natasha Flyer. Solving pdes with radial basis functions. *Acta Numerica*, 24:215–258, 2015.

[21] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996.

[22] Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39-41):4135–4195, 2005.

[23] Natasha Flyer, Bengt Fornberg, Victor Bayona, and Gregory A. Barnett. On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy. *J. Comput. Phys.*, 321:21–38, 2016.

[24] Victor Bayona, Natasha Flyer, Bengt Fornberg, and Gregory A. Barnett. On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs. *J. Comput. Phys.*, 332:257–273, 2017.

[25] Jure Slak and Gregor Kosec. Medusa: A C++ library for solving pdes using strong form mesh-free methods, 2019.

[26] Ryo Kobayashi. Modeling and numerical simulations of dendritic crystal growth. *Physica D: Nonlinear Phenomena*, 63(3-4):410–423, 1993.

Authorized licensed use limited to: Institute Jozef Stefan. Downloaded on December 07,2022 at 07:22:38 UTC from IEEE Xplore. Restrictions apply.