Reconstruction of surfaces given by point clouds

Viktor Cvrtila*, Miha Rot*†,
* Institut "Jožef Stefan", Department of Communication Systems, Jamova cesta 39, 1000 Ljubljana, Slovenia
†Jozef Stefan International Postgraduate School, Jamova cesta 39, 1000 Ljubljana, Slovenia
Viktor.Cvrtila@ijs.si

***Abstract*—One of the most general ways to represent a three-dimensional domain is with a dense point cloud that describes the boundary. This representation is convenient as it is both the output of a 3D scan and relatively simple to obtain from alternative surface description methods. However, a point cloud on its own is often insufficient for further calculations. We would like to use said point cloud to create a more convenient description of the true shape, which would allow us to use specialised discretization algorithms, and a way to determine its interior. In this paper we propose an algorithm that fits parametrized surfaces to discrete neighbourhoods that cover the point cloud and uses a partition of unity to ensure that the surfaces match along their edges. We then use this local parametrization to construct the characteristic function of the domain, i.e. a function which can determine if a given point is inside the domain or not.**

Keywords*—surface reconstruction, point clouds, meshless methods

I. INTRODUCTION

Many real-world problems can be expressed mathematically as a system of partial differential equations (PDEs). Often, there are no known analytical solutions to such problems. Knowing how to numerically solve PDEs is crucial in many areas, from meteorology [1] to modelling of gas turbines [2]. In recent years, much work has been done in the field of meshless numerical solvers [3]–[5]. In contrast to other methods - like the finite element method that requires a triangularization of the computational domain - these solve PDEs on a set of scattered nodes. In practice, there are many different ways to present the domains of PDEs. Triangle meshes or non-uniform rational basis spline (NURBS) patches [6] are common choices but both come with unique caveats. The first is computationally expensive to produce and is not suited for meshless methods while the latter cannot be used directly and requires further processing, e.g. by converting it to a mesh [7] or by discretization [8].

This paper focuses on another representation – *point clouds*. A point cloud is a finite set of points $X \subset \mathbb{R}^d$. Point clouds can be measured (e.g. when using 3D scanners) or produced during computation when discretizing a known domain. This typically produces point clouds $X \subset \partial D$ that describe the boundary of some domain $D \subset \mathbb{R}^d$. Working with point clouds directly would allow for more flexibility when working with complex domains. Unfortunately, point clouds carry less data than other representations and introduce new problems. Namely, for a point cloud $X \subset \partial D$

- there is no natural way to *rediscretize* the surface ∂D. In other words there is no way to make the point cloud more or less dense.
- There is no natural way to determine if a given point is inside the domain D.

The goal of this paper is to present an algorithm that alleviates these problems. This algorithm differs from others by creating a locally parametrized surface, where the local parametrization maps are created using radial basis functions. Similar work has been done in [9], with a focus on the visual aspects of the reconstructed surface. They define a procedure, that takes patches of a surface and ensures that they match near their boundaries. It focuses primarily on how the resulting patches look, e.g. by making sure that the resulting patches have no holes between them. We, on the other hand, can pick as many patches as we need, so we sidestep this problem. Because of this we can also use a much simpler procedure to make sure the patches match. Analogous problem has also been tackled implicitly [10], with the surface given by a point cloud reconstructed as an isosurface of a scalar field. We will discuss why this is not the best option for our use case in section II. We, however, use similar interpolants. For an overview of other surface reconstruction methods, see [11].

In section II we present the algorithm, split into three parts; we discuss the preliminary point cloud decomposition and estimations in II-A. The creation of local parametrizations is described in II-B. We finish by constructing the characteristic function in II-C. The algorithm is then demonstrated on a simple two-dimensional point cloud in section III.

II. THE ALGORITHM

Before we present the algorithm, we explain the reasoning behind our decisions in its design. If we ignore approximations of surfaces - such as triangular meshes - the most common ways of defining surfaces using equations are as a parametrization or as an isosurface of a scalar field. The latter is often simpler and more general. It is easier to produce a scalar field such that the given points lie on a desired isosurface; one can construct such a field using interpolation. However, it is less convenient – even finding a point on an isosurface is non-trivial and typically involves solving a non-linear equation. On the other hand, parametrization is harder to produce, but easier to work with. With a parametrized surface, we can easily

use existing discretization algorithms [8] to produce nodes of desired density. We choose parametrization knowing that once we invest effort into its construction, further work is made easier. As described, the algorithm is not dependant on the dimension d of the domain $D \subseteq \mathbb{R}^d$. We choose the dimension to equal 2, as it is easier to illustrate the procedure.

We first give a brief overview of the algorithm. It accepts a point cloud $X \subset \partial D$. This point cloud should be dense enough to sufficiently describe the features of the boundary ∂D. We intend to produce a series of local parametrizations $\boldsymbol{p}_i$ and the characteristic function c.

1) First, select a *discrete neighbourhood* X_i for each point $\boldsymbol{x}_i \in X$, i.e. a subset $\boldsymbol{x}_i \in X_i \subset X$. Neighbouring points $\boldsymbol{x}_{i,j} \in X_i$ are typically close to $\boldsymbol{x}_i$.
2) Second, calculate a rough approximation for normals $\boldsymbol{v}_i$ on ∂D at each point $\boldsymbol{x}_i$ in X.
3) Next, create a parametrization domain $U_i \subset \mathbb{R}^2$ for each discrete neighbourhood X_i. At the same time, create a series of knot vectors $\boldsymbol{u}_{i,j}$, corresponding to each point $\boldsymbol{x}_{i,j} \in X_i$.
4) Fit a function $\boldsymbol{s}_i \colon U_i \to \mathbb{R}^2$ to each discrete neighbourhood, such that $\boldsymbol{s}_i(\boldsymbol{u}_{i,j}) = \boldsymbol{x}_{i,j}$.
5) In parallel with the previous step, construct a sufficiently smooth partition of unity $\phi_i \colon R^2 \to [0,1]$ around each point $\boldsymbol{x}_i \in X$.
6) Using the functions from the previous steps, construct the local parametrizations $\boldsymbol{p}_i \colon U_i \to \mathbb{R}^2$. These differ from the functions $\boldsymbol{s}_i$, since the union $\bigcup_i \boldsymbol{p}_i(U_i)$ is a manifold, whereas $\bigcup_i \boldsymbol{s}_i(U_i)$ is not.
7) Before we can construct the function that determines the interior, once again approximate normals on ∂D, this time based on the derivatives of $\boldsymbol{p}_i$.
8) Finally, construct a function $c \colon \mathbb{R}^2 \to \mathbb{R}$ that can determine if a point is inside D.

Section II-A will focus on steps 1 through 3, section II-B will focus on steps 4 through 6 and section II-C will cover the remaining steps.

In short, the algorithm accepts a point cloud $X \subset \partial D$ and produces a series of local parametrizations $\boldsymbol{p}_i \colon U_i \to \mathbb{R}^2$ as well as a function $c \colon \mathbb{R}^2 \to \mathbb{R}$, such that the sign of $c(\boldsymbol{x})$ determines whether $\boldsymbol{x}$ is in the interior, on the boundary or in the exterior of D.

At this step note two details. We have no intention of producing a global parametrization $\boldsymbol{p} \colon U \to \partial D$. By imitating the definition of a manifold, we achieve a great deal of generality without having to mesh the surface. We could, in principle, "glue" the parametrization domains U_i together. This would be impractical, as it would essentially create a covering space, the topology of which could be complex. Also, the definition of the function c is fairly vague. This is because we can choose how to construct the function. An elegant approach would be to fit a scalar field to the points $\boldsymbol{x}_i - \boldsymbol{n}_i, \boldsymbol{x}_i, \boldsymbol{x}_i + \boldsymbol{n}_i$, such that the sign of $c(\boldsymbol{x})$ determines whether $\boldsymbol{x}$ is in D, on the boundary, or in the exterior of D. The problem with this approach is that constructing c in such a way, that it agrees with the parametrizations $\boldsymbol{p}_i$, is not trivial. If we need to work with both the boundary and the interior, we can use another procedure. When given a point $\boldsymbol{x}$, we can find the nearest point on ∂D to $\boldsymbol{x}$ and decide based on the corresponding normal.

A. CONSTRUCTING THE PARAMETRIZATION DOMAINS

One of the main problems of fitting a parametrized surface to a point cloud is the fact that we do not have any obvious choice of domain. That is, the choice of knot vectors $\boldsymbol{u}_{i,j}$ is not obvious. This problem is solved in the first three steps of the algorithm. Since we decided to construct only local parametrizations, we can restrict ourselves to smaller domains, such that on each U_i, the image $\boldsymbol{p}_i(U_i)$ is a graph of a function, allowing for rotations.

To achieve this, we first select a small enough discrete neighbourhood for each point $\boldsymbol{x}_i$ from the point cloud X. Small enough, here, means that there should exist a connected neighbourhood of the discrete neighbourhood in ∂D, such that this neighbourhood is a graph of a function, up to rotation. We intend to fit a local parametrization to each such set at a later step. The simplest way to achieve this is by taking the neighbourhood of $\boldsymbol{x}_i$ to be its $(k-1)$-nearest neighbours for a small enough k. Let $\boldsymbol{x}_i$ be a point in X. Write $\{\boldsymbol{x}_i = \boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, \ldots, \boldsymbol{x}_{i,k}\} = X_i \subset X$. One way of constructing these would be using a k-d tree [12] to find the required points. After constructing a k-d tree, querying the k nearest points to a point $\boldsymbol{x}_i$ for some small fixed k is computationally inexpensive. This method also lends itself well to parallel computing, as demonstrated in [13].

Next, we wish to approximate the normals $\boldsymbol{n}_i$ on ∂D at $\boldsymbol{x}_i$. There are many ways to do this, e.g. using Voronoi covariance measures [14], or jets [15]. We use a simpler approach – we fit a plane to X_i using the least squares method, since this is a robust method and easy to implement and understand. This does not assume that the discrete neighbourhoods are almost flat. At this stage, we just need an approximation of the normal. We can chose the normal of the hyperplane $\boldsymbol{v}_i$ as a first approximation of the normal. The problem with this procedure is that we have no guarantee that the normals are picked consistently – some may be inward-pointing, while others may be outward-pointing. This does not pose a problem at this point, as we only need the direction represented by the normals.

With the discrete neighbourhoods X_i and the normals $\boldsymbol{v}_i$, we can construct the parametrization domains U_i. Write P_i for the plane passing through $\boldsymbol{x}_i$ perpendicular to $\boldsymbol{v}_i$. Let $\boldsymbol{A}_i \colon \mathbb{R}^2 \to \mathbb{R}^2$ be an affine map that maps P_i to $\mathbb{R}^2 \times \{0\}$ and $\boldsymbol{x}_i$ to $(0,0,0)$. We can then expect that if we apply $\boldsymbol{A}_i$ to a small connected open neighbourhood of X_i in ∂D, we get a graph of a function. With this in mind, define $\tilde{\boldsymbol{u}}_{i,j}$ to be the projection of $\boldsymbol{x}_{i,j}$ along $\boldsymbol{v}_i$

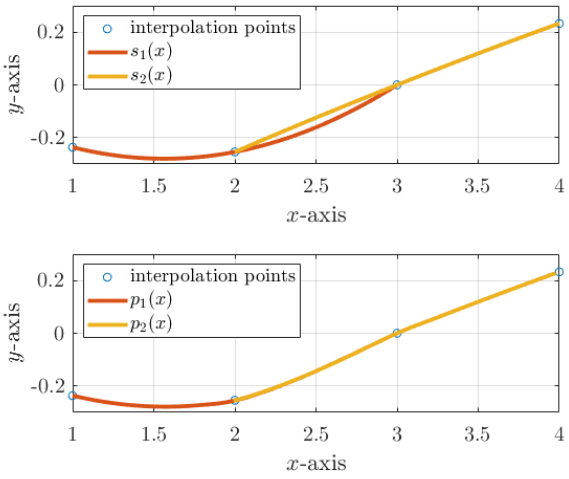

Fig. 1: The result of interpolating points on discrete neighbourhood $1, 2, 3$ and $2, 3, 4$ on the upper graph. The gluing operation is then applied to these maps on the lower graph.

onto P_i. Then define $\boldsymbol{u}_{i,j} \in \mathbb{R}^2$ as $\boldsymbol{A}_i \tilde{\boldsymbol{u}}_{i,j}$. Since it may be convenient later, we normalize the set of knot vectors so that the distance between $\boldsymbol{u}_{i,0}$ and the nearest vector among $\boldsymbol{u}_{i,1}, \ldots, \boldsymbol{u}_{i,k}$ equals one. For the moment, select the entire plane $\mathbb{R}^2$ as the domain U_i. When we construct the parametrization maps, we can shrink the domains U_i to equal a closed unit ball.

This procedure, which only has to be done once, is relitively fast for small k. Constructing a k-d tree with n points has $\mathcal{O}(n \log(n))$ time complexity, and queriying the k nearest neighbours has $\mathcal{O}(k \log(n))$. Using the least squares method to approximate the normals is $\mathcal{O}(k^2)$ per normal. The entire procedure is therefore $\mathcal{O}(kn \log(n) + k^2 n)$.

B. CONSTRUCTING THE LOCAL PARAMETRIZATIONS

We now discuss steps 4 through 6. Since we have the knot vectors $\boldsymbol{u}_{i,j}$ for each discrete neighbourhood X_i, fitting an interpolating surface to the points in X_i is simple. For example, let $\varphi\colon [0, \infty) \to \mathbb{R}$ be a radial basis function (RBF). We can then find an interpolant of the form

$$\boldsymbol{s}_i(\boldsymbol{x}) = \boldsymbol{Q}_i(\boldsymbol{x}) + \sum_{j=1}^{k} \alpha_{i,j} \varphi(||\boldsymbol{x} - \boldsymbol{x}_{i,j}||),$$

where Q_i are polynomials and $\alpha_{i,j} \in \mathbb{R}^2$, such that for each $j = 1, \ldots, k$

$$\boldsymbol{s}_i(\boldsymbol{u}_{i,j}) = \boldsymbol{x}_{i,j}.$$

For an overview of RBF interpolation, consult chapter 3 of [16]. The role of the polynomial $\boldsymbol{Q}_i$ is discussed in [17].

The problem here is that we do not have a proper local parametrization. The maps $\boldsymbol{s}_i$ only interpolate the points on the surface, so they do not agree inbetween the points; see the upper graph in Fig. 1. We construct two interpolating maps, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, which interpolate points with x-coordinates of 1, 2, 3 and 2, 3, 4 resp. Notice that the maps $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ do not agree between the middle two points; the union of the images of the maps do not form a manifold around these points.

This can be fixed by "gluing" the maps together using a partition of unity (PU). A PU is a series of functions $\phi_1, \ldots, \phi_n$ such that $\sum_{i=1}^{n} \phi_i$ equals the constant 1 function. If the support of these functions is small enough, they can be used to smoothly combine locally defined functions. A partition of unity around a set of points in $\mathbb{R}^2$ can be constructed as follows. First, choose a radius $r_i > 0$ for each point $\boldsymbol{x}_i$ from X. This should be chosen such that the ball $B(\boldsymbol{u}_{i,0}, r_i)$ contains the knot vectors corresponding to the nearest points to $\boldsymbol{x}_i$ in X_i. Then, define a function $\kappa\colon [0, \infty) \to [0, \infty)$ such that the support of κ equals $[0, 1]$ and that the derivatives $\kappa^{(j)}(0)$ and $\kappa^{(j)}(1)$ equal 0 for all $0 < j \leq d$. We can use, for example, the function

$$\kappa(x) = \begin{cases} \exp\left(-\frac{1}{1-x^2}\right) & \text{if } x \in [0, r) \\ 0 & \text{if } x \in [r, \infty) \end{cases}.$$

Then we simply define

$$\phi_i(\boldsymbol{u}) = \frac{\kappa\left(\frac{||\boldsymbol{u}||}{r_i}\right)}{\sum_{j=1}^{n} \kappa\left(\frac{||\boldsymbol{u}||}{r_j}\right)}.$$

The parametrizarion maps $\boldsymbol{p}_i$ are then evaluated as follows. Suppose we want to calculate $\boldsymbol{p}_i(u)$ for some $\boldsymbol{u} \in \mathbb{R}^2$. First, we calculate $\boldsymbol{s}_i(\boldsymbol{u})$, as the first approximation. Write $\boldsymbol{x}^{(0)} = \boldsymbol{s}_i(\boldsymbol{u})$. Then find all $\boldsymbol{x}_j$ from X, such that the distance between $\boldsymbol{x}^0$ and $\boldsymbol{x}_j$ is less than r_i. Write $\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_\ell}$ for these points. Now, for each index $j = 1, \ldots, \ell$ project the point $\boldsymbol{x}^{(0)}$ along $\boldsymbol{v}_{i_j}$ onto the plane P_{i_j} and applying the maps $\boldsymbol{A}_{i_j}$ to get $\boldsymbol{u}_1^{(0)}, \ldots, \boldsymbol{u}_\ell^{(0)}$; note that we repeated the procedure for calculating the knot vectors $\boldsymbol{u}_{i,j}$, but using the point $\boldsymbol{x}^{(0)}$. Now we define the value of $\boldsymbol{p}_i$ at $\boldsymbol{u}$ by

$$\boldsymbol{p}_i(\boldsymbol{u}) = \sum_{j=1}^{\ell} \boldsymbol{s}_{i_j}(\boldsymbol{u}_j^{(0)}) \phi_{i_j}(\boldsymbol{u}_j^{(0)}).$$

The lower graph in Fig. 1 shows the result of the gluing procedure, when applied to the functions displayed in the upper graph.

Since we have all the data from section II-A, each evaluation of a parametric map is mostly dependant on the size of the discrete neighbourhoods k. Calculating the coefficients $\alpha_{i,j}$ for each interpolant $\boldsymbol{s}_i$ is $\mathcal{O}(k^3 n)$. If we choose the radiuses r_i such that each ball with center x_i and radius r_i contains at most k other points of X, one evaluation is $\mathcal{O}(k^2 + k \log(n))$.

C. CONSTRUCTING THE CHARACTERISTIC FUNCTION

Our final task is to construct an approximation of the characteristic map. To do so, we must at this point compute better approximations for the normals. Since we have the parametrization maps $\boldsymbol{p}_i$, this is much simpler than in II-A. Let $\boldsymbol{n}_i^{(0)}$ equal the normalized cross product of the derivatives of $\boldsymbol{p}_i$ at $(0,0)$. Such a vector is indeed a normal on the image of $\boldsymbol{p}_i$. The problem here is that the normals $\boldsymbol{n}_i^{(0)}$ may not all face in the same direction. Some may be inward-facing, while others may be outward-facing. We must ensure that the choice of normal is consistent. There are many ways to achieve this.

One of the simpler ways is to propagate the direction of a given normal breadth-first to its nearest neighbours. Choose a normal $\boldsymbol{n}_i = \boldsymbol{n}_i^{(0)}$ and add its index i to a FIFO queue. Take the discrete neighbourhood X_i of $\boldsymbol{x}_i$. Let $\boldsymbol{n}_{i,j}^{(0)}$ be the normal at $\boldsymbol{x}_{i,j}$; that is, there exists some index m such that $\boldsymbol{x}^{i,j} = \boldsymbol{x}_m$, so $\boldsymbol{n}_{i,j}^{(0)} = \boldsymbol{n}_m^{(0)}$. If we have not visited it during the procedure, check if the dot product $\langle \boldsymbol{n}_m^{(0)}, \boldsymbol{n}_i^{(0)} \rangle$ is positive. If it is, set $\boldsymbol{n}_m = \boldsymbol{n}_m^{(0)}$, otherwise set $\boldsymbol{n}_m = -\boldsymbol{n}_m^{(0)}$. Then add the index m to the queue and repeat until the queue is empty. This procedure ensures that the normals are chosen consistently. However, we do not at this point know whether the normals are all inward- or outward-facing. We can solve this problem with the help of the characteristic function.

Suppose we are not interested in the local parametrizations, but only in the characteristic function. We can construct a function, whose sign determines the position of a point in relation to the domain. Using RBFs, we can create a map $c\colon \mathbb{R}^2 \to \mathbb{R}$ by interpolating

$$c(\boldsymbol{x}_i - \boldsymbol{n}_i) = -1,\ c(\boldsymbol{x}_i) = 0 \text{ and } c(\boldsymbol{x}_i + \boldsymbol{n}_i) = 1. \quad (1)$$

If we wish to interpolate only locally, we may construct a function c_i satisfying (1) for all points in a discrete neighbourhood X_i. Then we simply combine these using an appropriate partition of unity around the points of X. If $\phi_1, \ldots, \phi_n$ is such a partition of unity, define $c = \sum_{i=1}^{n} c_i \phi_i$. The advantage of using the normal vector approximations $\boldsymbol{n}_i$ over the vectors $\boldsymbol{v}_i$ from II-A is that the former are smoother in a way, since they were constructed using the smooth local parametrization maps.

If, on the other hand, we want both the reconstructed boundary and the characteristic function, we employ the following procedure. We again construct a function $c\colon \mathbb{R}^2 \to \mathbb{R}$. Say we want to calculate $c(\boldsymbol{x})$ for some $\boldsymbol{x} \in \mathbb{R}^2$. First find the nearest point to $\boldsymbol{x}$ in the point cloud X, say $\boldsymbol{x}_i$. Write $\boldsymbol{x}^{(0)}$ for an approximation of the point that minimizes the distance between $\boldsymbol{x}$ and the image of the unit ball under $\boldsymbol{p}_i$. We do not need a particularly good approximation, so we use a fixed small number of iteration steps of Newton's method. Let $\boldsymbol{n}^{(0)}$ be the normal at $\boldsymbol{x}^{(0)}$. Then define $\boldsymbol{c}(x) = \langle \boldsymbol{x}^{(0)} - \boldsymbol{x}, \boldsymbol{n}^{(0)} \rangle$.

If we do indeed take a small fixed number of iterations, evaluating the function c is still in $\mathcal{O}(k^2 + k\log(n))$

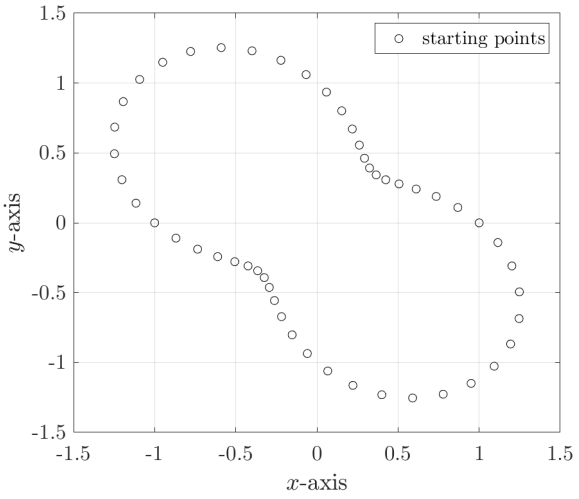

Fig. 2: The example point cloud produced from sampling curve (2).

The sign of both of these functions determines the position of a point $\boldsymbol{x}$. If $c(\boldsymbol{x})$ is negative, then $\boldsymbol{x} \in D$. If it equals zero, then $\boldsymbol{x}$ is on the boundary ∂D. Otherwise, $\boldsymbol{x}$ is outside of $\boldsymbol{D}$.

We can also use these maps to ensure that the normals all point in the right direction. Take a point $\boldsymbol{x}_0 \in D$, i.e. a point that we know to be inside the domain, and calculate $c(\boldsymbol{x}_0)$. If this value is positive, flip all the $\boldsymbol{n}_i$ vectors.

III. USAGE EXAMPLE

Consider the polar curve given by the equation

$$r(\theta) = 1 - \cos(\theta)\sin(\theta). \quad (2)$$

To test our algorithm, we first naively discretize the curve by taking the points corresponding to $\theta = 0, \frac{\pi}{50}, \frac{2\pi}{50}, \ldots, \frac{49\pi}{50}$. We get a set of fifty unevenly spaced points X; see Fig. 2. We choose neighbourhoods of size $k = 5$. First, we approximate the normals as stated in II-A. Note that at this stage, the normals are not consistent. After constructing the parametrization domains, we construct the interpolation maps. As before, they do not agree; they interpolate each series of consecutive five points, but the union of their images does not form a manifold. See Fig. 3 (or Fig. 1). Now, we construct the parametrization maps p_i. These are actual local parametrizations of a manifold. The reconstructed curve is displayed in Fig. 4. We finally construct the map c using both methods described above. We use a naive interior fill algorithm to check if the map c functions as intended. We take points on a grid and, using c, we check if they are in the interior of the domain. Fig. 5 displays the results of this fill algorithm.

IV. CONCLUSION

The focus of this paper was to introduce a parametric surface algorithm based on local RBF interpolation. When given a point cloud, the algorithm creates a series of local parametrizations around each point and joins them

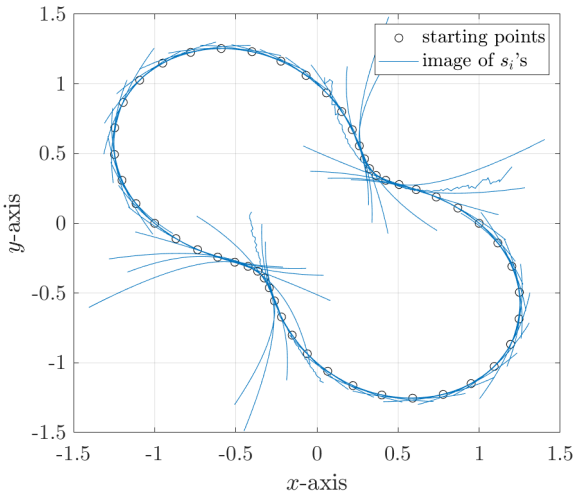

Fig. 3: Each curve represents the image of a map s_i for $i = 1, \ldots, 50$.

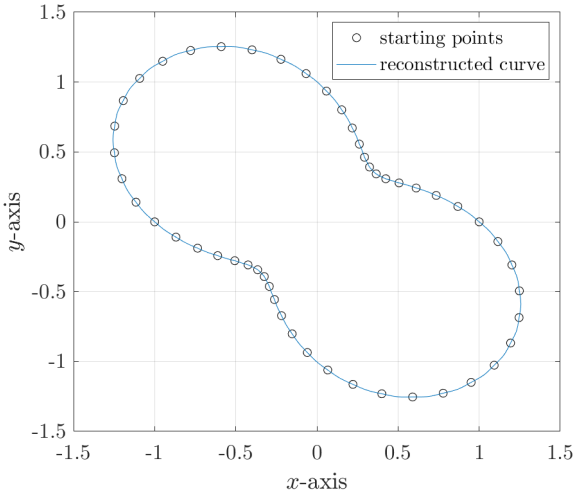

Fig. 4: The union of the images of p_i for $i = 1, \ldots, 50$.

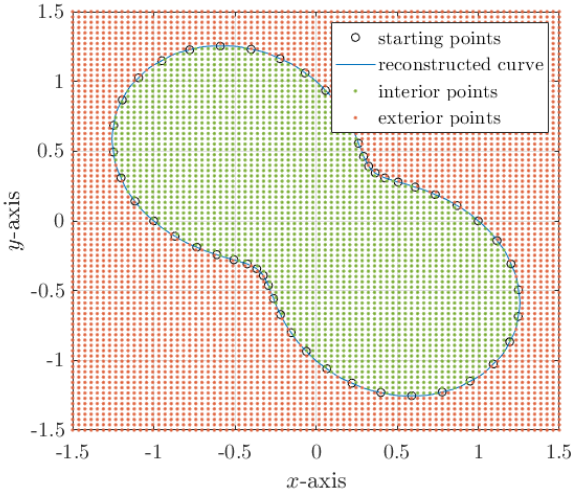

Fig. 5: The result of a naive fill algorithm using the c map.

into a smooth manifold by using the partition of unity. It also creates the characteristic function of the reconstructed surface based on the local parametrizations. Such an algorithm enables one to work with discretized surfaces – to rediscretize them or to discretize their interiors. While originally intended for surface reconstruction in two dimensions, it is not dependant on dimension. It can be easily modified to work on hypersurfaces of any dimension, due to the use of RBF interpolants. Further work would need to focus on testing the algorithm on more complex examples and direct comparison with alternative approaches.

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support of the Slovenian Research Agency (ARRS) research core funding No. P2-0095, Young Researcher program PR-10468, and project funding No. J2-3048.

REFERENCES

[1] G. Marchuk, *Numerical methods in weather prediction*. Elsevier, 2012.

[2] A. Benaarbia, Y. Rae, and W. Sun, "Unified viscoplasticity modelling and its application to fatigue-creep behaviour of gas turbine rotor," *International Journal of Mechanical Sciences*, vol. 136, pp. 36–49, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020740317324323

[3] V. Shankar, G. B. Wright, and A. Narayan, "A robust hyperviscosity formulation for stable rbf-fd discretizations of advection-diffusion-reaction equations on manifolds," *SIAM Journal on Scientific Computing*, vol. 42, no. 4, pp. A2371–A2401, 2020. [Online]. Available: https://doi.org/10.1137/19M1288747

[4] I. Tominec and E. Breznik, "An unfitted rbf-fd method in a least-squares setting for elliptic pdes on complex geometries," *Journal of Computational Physics*, vol. 436, p. 110283, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999121001789

[5] J. Slak and G. Kosec, "Medusa: A c++ library for solving pdes using strong form mesh-free methods," *ACM Trans. Math. Softw.*, vol. 47, no. 3, Jun. 2021.

[6] G. Legrain, "A nurbs enhanced extended finite element approach for unfitted cad analysis," *Computational Mechanics*, vol. 52, pp. 913–929, 2013.

[7] D. H. Adan and R. Cardoso, "Quasi-isotropic initial triangulation of nurbs surfaces," *European Journal of Computational Mechanics*, vol. 29, no. 1, pp. 27–82, 2020.

[8] U. Duh, G. Kosec, and J. Slak, "Fast variable density node generation on parametric surfaces with application to mesh-free methods," *SIAM Journal on Scientific Computing*, vol. 43, pp. A980–A1000, 2021. [Online]. Available: https://doi.org/10.1137/20M1325642

[9] Z. Deng, J. Bednařík, M. Salzmann, and P. Fua, "Better patch stitching for parametric surface reconstruction," in *2020 International Conference on 3D Vision (3DV)*, 2020, pp. 593–602.

[10] K. Drake, E. Fuselier, and G. Wright, "Implicit surface reconstruction with a curl-free radial basis function partition of unity method," 01 2021.

[11] A. Khatamian and A. R., *Journal of Information Processing Systems*, vol. 12, no. 3, pp. 338–357, 2016.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, sep 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[13] R. Trobec and M. Depolli, "A k-d tree based partitioning of computational domains for efficient parallel computing," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2021, pp. 284–290.

[14] L. Cuel, J.-O. Lachaud, Q. Mérigot, and B. Thibert, "Robust geometry estimation using the generalized voronoi covariance measure," *SIAM Journal on Imaging Sciences*, vol. 8, no. 2, pp. 1293–1314, 2015. [Online]. Available: https://doi.org/10.1137/140977552

[15] F. Cazals and M. Pouget, "Estimating differential quantities using polynomial fitting of osculating jets," vol. 22, 05 2003, pp. 177–187.

[16] B. Fornberg and N. Flyer, *A Primer on Radial Basis Functions with Applications to the Geosciences*. Society for Industrial and Applied Mathematics, 2015.

[17] N. Flyer, B. Fornberg, V. Bayona, and G. A. Barnett, "On the role of polynomials in rbf-fd approximations: I. interpolation and accuracy," *Journal of Computational Physics*, vol. 321, pp. 21–38, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999116301632