

A Fully Lagrangian Meshfree Framework for PDEs on Evolving Surfaces

Pratik Suchde^{a,*}, Jörg Kuhnert^a

^a*Fraunhofer ITWM, 67663 Kaiserslautern, Germany*

Abstract

We propose a novel framework to solve PDEs on moving manifolds, where the evolving surface is represented by a moving point cloud. This has the advantage of avoiding the need to discretize the bulk volume around the surface, while also avoiding the need to have a global mesh. Distortions in the point cloud as a result of the movement are fixed by local adaptation. We first establish a comprehensive Lagrangian framework for arbitrary movement of curves and surfaces given by point clouds. Collision detection algorithms between point cloud surfaces are introduced, which also allow the handling of evolving manifolds with topological changes. We then couple this Lagrangian framework with a meshfree Generalized Finite Difference Method (GFDM) to approximate surface differential operators, which together give a method to solve PDEs on evolving manifolds. The applicability of this method is illustrated with a range of numerical examples, which include advection-diffusion equations with large deformations of the surface, curvature dependent geometric motion, and wave equations on evolving surfaces.

Keywords: Lagrangian, Moving Manifold, Evolving Surface, Meshfree, GFDM, Meshfree contact detection

1. Introduction

The need to numerically solve PDEs on evolving surfaces arises in various fields. From the modelling of surfactants [55] and other fluid dynamics [19], to the modelling of airbags [22] and parachutes [1]. PDEs on deforming curves and surfaces also appear in the modelling of biomembranes [12] and cell motility [13], in visualization [27], and image processing [25].

Most methods for solving PDEs on moving surfaces are mesh-based. Both finite elements [10, 11, 37] and finite volumes [46] have been used in this context. The movement of the surface is achieved by moving the discretizing mesh. This introduces the trouble of mesh distortion which needs an expensive remeshing [32]. Note that this is the same issue with moving mesh methods for volumetric flow with free surfaces. To avoid the need of meshing and remeshing, surfaces are often discretized with a cloud of numerical points. As a result, a significant amount of work has been done towards the development of meshfree methods for solving PDEs on static surfaces (for example, [16, 33, 39]). Some of these meshfree methods have also been extended to handle evolving surfaces.

To the end of solving PDEs on stationary surfaces, a lot of work has been done to derive meshfree methods that scale with the dimension of the surface itself, rather than the dimension of the embedding space [17, 33]. This significantly decreases the computational cost as compared to methods which discretize the embedding space (or a subset of the same dimension). However, when it comes to PDEs on evolving surfaces, most meshfree methods require a discretization of the bulk volume surrounding the surface [3, 30, 49, 50, 54]. The movement of the discrete manifold is captured by some notion of tracking on this surrounding grid or point cloud. This is similar in essence to interface tracking for free surface volumetric flows done in static mesh-based methods [21], and all inaccuracies in that context carry over to the present context. Thus, by requiring tracking of the surface, these methods lose one of the fundamental advantages of meshfree methods.

*Corresponding author

Email address: pratik.suchde@itwm.fraunhofer.de (Pratik Suchde)

Moreover, they require a discretization of a higher dimensional space, which makes them computationally expensive.

In the context of volumetric flows, it is well known that Lagrangian frameworks can accurately capture advection as well as the shape of interfaces and free surfaces [28, 53]. In this paper, we introduce such a fully Lagrangian framework to accurately capture the movement of an evolving surface. We present a new meshfree method for solving PDEs on evolving surfaces that scales with the manifold dimension. The surface is discretized with a cloud of numerical points, without any need to discretize the embedding space around the surface. Movement of the surface is captured using a fully Lagrangian framework. Spatial derivatives are computed by virtual projections to the tangent space at each point. A major advantage of a moving point cloud over a moving mesh is that point cloud distortion is much easier to fix. For point clouds defining volumes, this has been shown in, for example, [9]. Here, we will show that the same also holds good for surfaces. To this end, we present methods to fix distortions in surface point clouds by adding and removing points where necessary by purely local considerations.

We note that moving Lagrangian particles have already been used to solve PDEs on evolving surfaces by several authors. The novelty in the present work lies in the fact that only the surface is being discretized, and not the bulk around it. In contrast, [4] uses Lagrangian particles in a band around the manifold, and a regular grid in the background to interpolate particle locations and to fix distortion. While [29] and [50] use Lagrangian particles for the moving surface with a regular fixed grid of the dimension of the embedding space in the background for reference and neighbourhood information.

The paper is organized as follows. Section 2 contains some information about the setup of surface point clouds. Section 3 describes how a Lagrangian framework can be established for surface PDEs, and includes the details about the Lagrangian movement, the required adaptation of nodes, and contact handling. Section 4 then goes on to show how this can be used to solve PDEs on moving manifolds in a Lagrangian way, and presents numerical examples. The paper is then concluded with a short discussion in Section 5.

2. Preliminaries

To distinguish between the cases of PDEs on surfaces and those on volume domains, we use the term ‘volumetric’ to denote the volume domain case.

Throughout this work, we establish initial point clouds as per [9]. Notation and definitions for the point clouds are in accordance with [56, 58]. Conventions typically used for volumetric meshfree GFDM point clouds [9, 24] are carried over to the surface case here. Throughout this paper, we consider only smooth orientable 2 manifolds in \mathbb{R}^3 . However, the ideas presented in this paper can easily be generalized to higher dimensions and co-dimensions.

The smooth oriented surface or manifold M is discretized with N non-uniformly spaced numerical points also referred to as nodes or particles. These points are simply locations where approximations are carried out, and they do not carry mass. The N points include points both in the interior and at the boundary (if any) of the manifold. Approximations at a numerical point $i = 1, 2, \dots, N$ are done based on a support or neighbourhood S_i of n_{S_i} nearby points, within a distance of h . $h = h(\vec{x}, t)$ is referred to as the interaction radius or smoothing length. The spatial distribution of points is described by three parameters: h , r_{max} , and r_{min} . It is ensured that there is no hole of size $r_{max}h$ within the point cloud, and that no two points are closer than $r_{min}h$. Both r_{max} and r_{min} are global constants, taken to be the same for all simulations, and are not dependent on the PDE being solved. As a result, the smoothing length h serves not just as the size of the support at each point, but also as the spatial discretization size. r_{max} and r_{min} determine the number of neighbouring points in each support domain. We use $r_{max} = 0.45$ and $r_{min} = 0.2$, which result in about 15 – 20 points in each neighbourhood. These values are carried over from conventions on point cloud spacing used in meshfree GFDM volumetric flow solvers [9, 24, 58]. All distance computations are done in the embedding space (here, in \mathbb{R}^3), and *not* along the manifold. To satisfy these maximum and minimum distance criteria on manifolds, which could change with time, addition and deletion of points on the surface is carried out, which is explained in Sections 3.4 and 3.5.

3. The Lagrangian Framework

In this section we present details about how a meshfree Lagrangian framework can be established for surfaces. To the best of the authors' knowledge, such a fully Lagrangian meshfree setting without any background grid has never been done for manifolds.

We consider an advection velocity \vec{v} , which can have components both normal and tangential to the surface. In contrast to a lot of existing methods for PDEs on evolving surfaces, we do not split the velocity into its normal and tangential components, and use the entire velocity as the Lagrangian transport velocity. Furthermore, unlike many mesh-based Lagrangian methods for surface PDEs, we do not assume any correspondence between the point clouds at different time levels. At a given time level, no mapping is preserved to the initial configuration of points, and all approximations are done at the present point cloud directly.

3.1. The Actual Movement

The Lagrangian motion step involves solving the ODE system

$$\frac{d\vec{x}}{dt} = \vec{v}, \quad (1)$$

where \vec{v} is the advection velocity. In most volumetric Lagrangian frameworks for meshfree methods, Eq. (1) is solved by a first order method which assumes the velocity is constant between two time levels. The same is also done in Lagrangian and semi-Lagrangian methods for surfaces. Both in the moving mesh-context [32], and meshless manifold tracking context [50]. This involves each point of the point cloud, or equivalently each node of the mesh, being moved with the given velocity field as

$$\vec{x}^{(n+1)} = \vec{x}^{(n)} + \vec{v}^{(n)} \Delta t, \quad (2)$$

where the bracketed superscripts indicate the time level. Here, time-integration is done between the time levels t^n and t^{n+1} , and it is assumed that $\vec{v}^{(n+1)}$ is unknown. Alternatively, similar methods are also done by moving with the velocity $\vec{v}^{(n+1)}$, if known, or an average of $\vec{v}^{(n+1)}$ and $\vec{v}^{(n)}$. In each case, the velocity is taken to be constant within each time step.

In our earlier work for meshfree volumetric Lagrangian flows [57], we have shown the inaccuracies surrounding the first order movement similar to Eq.(2), which lead to large defects in volume conservation. To get accurate Lagrangian movement with Eq. (2), most authors tend to very small time steps. The same arguments of inaccuracy presented in [57] for volumetric flows carry over to the surface case here. Thus, we use the more accurate second order method for point cloud movement

$$\vec{x}^{(n+1)} = \vec{x}^{(n)} + \vec{v}^{(n)} \Delta t + \frac{1}{2} \frac{\vec{v}^{(n)} - \vec{v}^{(n-1)}}{\Delta t} \Delta t^2. \quad (3)$$

Alternatively, if $\vec{v}^{(n+1)}$ is known at the time of updating point locations, $\vec{v}^{(n)}$ and $\vec{v}^{(n-1)}$ can be replaced by $\vec{v}^{(n+1)}$ and $\vec{v}^{(n)}$ respectively in Eq. (3). This would be the case if the new velocity has been determined before the movement step. Note that since Eq. (3) retains the explicit nature of Eq. (2), the second order method comes at virtually no extra cost over the first order one.

To emphasize the need of such a second order method according to Eq. (3), a simple case of a rotating quarter-sphere is considered in Section 3.7.1.

We note that most practical applications for PDEs on evolving surfaces will rely on time-dependent velocity fields. While even higher order integration methods for the movement step can be developed for time-independent velocity fields, they can not be easily generalized to the time-dependent case for moving point clouds [57], so they are not very interesting in the present context.

In both Eq. (2) and Eq. (3), each point of the point cloud is moved explicitly and independently of all other points. Here, it is assumed that any implicit or global constraint on the motion of the surface is already represented in \vec{v} .

3.2. Normal Computation

Each interior point i of the manifold must be equipped with a unit normal \vec{n}_i and two unit tangents $\vec{t}_{1,i}$ and $\vec{t}_{2,i}$. Furthermore, each boundary point i (if any) of the manifold must be equipped with a unit manifold normal \vec{n}_i , a unit tangent \vec{t}_i , and a unit boundary normal \vec{v}_i . Note that \vec{v}_i is normal to the boundary of the manifold, but lies in the tangent space of the manifold. At each point, these normals and tangents form an orthonormal system of vectors.

Multiple possibilities exist for the computations of these normals and tangents on point clouds. A popular way to determine these is by using principal component analysis (PCA) [33, 44] which computes eigenvalues of local covariance matrices.

PCA is known to be sensitive to outliers and noise in the point cloud. Weighted PCA approaches have been used to overcome the same [51]. To use a more robust method to compute normals on point clouds, we use a minimization approach that maximizes the angles between the normal and the neighbouring points. This is based on the procedure used for computing normals at the free surface for volumetric flow by [52]. For an interior point i , the manifold normal \vec{n}_i is set to be the unit normal vector minimizing the functional

$$\sum_{j \in S_i} \cos^2(\theta_{ij}) W_{ij} = \vec{n}_i^T \sum_{j \in S_i} \left(W_{ij} \frac{(\vec{x}_j - \vec{x}_i)(\vec{x}_j - \vec{x}_i)^T}{\|\vec{x}_j - \vec{x}_i\|^2} \right) \vec{n}_i, \quad (4)$$

where θ_{ij} is the angle between the normal \vec{n}_i and the distance vector $\vec{x}_j - \vec{x}_i$, and W_{ij} is a Gaussian weighting function to emphasize the effect of the closest neighbours. Once \vec{n}_i is known, the tangents can be computed so as to obtain an orthonormal system of vectors.

For boundary points, the boundary normal \vec{v}_i is first computed considering only the boundary neighbours. Based on this, the boundary tangent is computed. After which, the manifold normal can be established at the boundary point.

This process of normal and tangent computation needs to be done at each time step, since the positions of each point are time-dependent.

We emphasize that we consider surfaces that are orientable, and the computed normal field should also result in an oriented surface. For example, for the case of the surface of a sphere, all the normals must either be pointing inwards, or all outwards. The need of an oriented normal field becomes especially important, for example, when computing the curvature of the surface by the surface divergence of the normal field $\kappa = -\frac{1}{2} \nabla_M \cdot \vec{n}$.

The same is explicitly checked by a sweep through all points when the normals are computed for the first time during a simulation. At later time steps, to maintain the orientation of the normal field, newly computed normals are compared with the old values. The orientation of \vec{n}_i^{new} is chosen such that

$$\vec{n}_i^{\text{new}} \cdot \vec{n}_i^{\text{old}} > 0, \quad (5)$$

where the superscripts *new* and *old* indicate the normals at the current and previous time levels respectively. Similar considerations are also done for normal computation at newly added points (to be explained in a coming section), but based on the normals of neighbouring points.

3.3. Neighbour Searching

Efficient neighbour searching is crucial for the efficiency of meshfree methods for surface PDEs, just as it is in the volumetric case. Neighbour search algorithms used in volumetric meshfree methods [8, 9, 48] directly carry over to the surface case being considered here. For each node, its neighbouring nodes are determined as the nodes within a certain distance h from it. The naive approach to neighbour searching would involve computing distances between every pair of nodes in the computational domain. However, this procedure is exorbitantly expensive. To avoid excessive distance computations, the domain is usually split into multiple regions referred to as boxes or cells. Using such a decomposition, for each node, distances only need to be computed with other nodes within the same box (or possibly also adjacent boxes). Several different data structures have been used to this end. One of the ways to do the same is to use quadtree or octree type searching algorithms. For the Lagrangian case, it needs to be ensured that when a new point is added, it is added in the correct box. To save time, it is often desirable to preserve the tree structure for a few time steps, and reconstruct it only every p time steps.

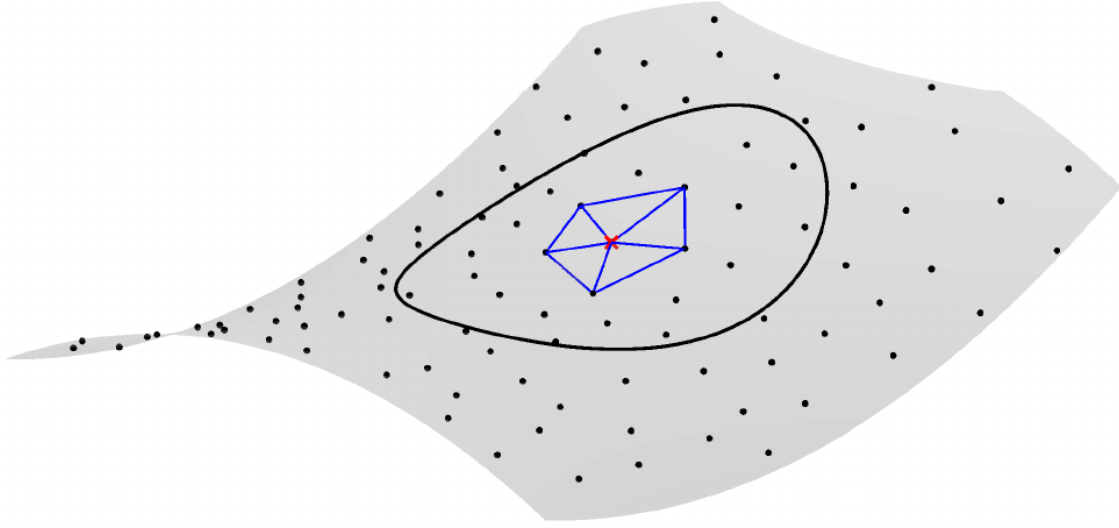


Figure 1: Local triangulation of a point on a surface. The central point is marked with a red cross. The black circle around it marks the neighbourhood of the central point. The blue triangles indicate the triangulation computed to determine locations where points need to be added to maintain regularity of the point cloud. The grey region indicates the surface specified by the black points.

3.4. Addition of Points

Just like the mesh-based case, Lagrangian movement of a surface point cloud can cause distortions which manifest as ‘holes’ in the point cloud where no points are present, or ‘clusters’ where too many points are present. To fix this, points are added in holes, and removed or merged when they come too close. We now propose a method for hole identification and filling in the absence of a background discretization of the embedding space.

Several volumetric meshfree methods use locally defined meshes [34] for a variety of purposes, from integration in weak form methods [2], to support domain selection [35], to post-processing [24]. We propose to use locally defined triangulations to identify holes in the surface point cloud. For this, at each point, we compute a ‘one-ring’ of triangles based solely on the locations of neighbouring points. An example of such a triangulation within a support domain is shown in Figure 1.

It is important to note the difference between the above mentioned locally computed triangulation, and the generation of an actual mesh on the entire surface. The local triangulations are much easier to generate. They scale very well when computed in parallel as the procedure for each point is independent of that for all other points. There is no imposed restriction for the triangles to be ‘good’. All the local 1-ring of triangles need not stitch together to form a global mesh of the surface. In fact, in most cases, they do not, unless the support domains are considered very large, which is rarely done in practice. Furthermore, these triangles are not preserved between time steps. In fact, they need not even be stored. Unlike the computation of a triangulation on an entire surface [5], the computation of these local triangles does not require any mapping to a parameter space.

These locally defined triangles, as shown in Figure 1, are used to determine locations of holes where points need to be filled. For this, we recall the volumetric point cloud spacing conventions introduced in Section 2, which are carried over to the surface case here. We intend to ensure that there is no hole of size $r_{max}h$ in the point cloud, where the smoothing length $h = h(\vec{x}, t)$ can be a function of both space and time. We set $r_{max} = 0.45$, following volumetric point cloud spacing conventions [9]. Thus, for any triangle with a circumradius greater than $r_{max}h$, a point is added at its circumcenter, but only as long as the circumcenter is within the triangle. Once this process is carried out at all points in the domain, it is then repeated for the newly created points. This process of filling points is illustrated in Figure 2.

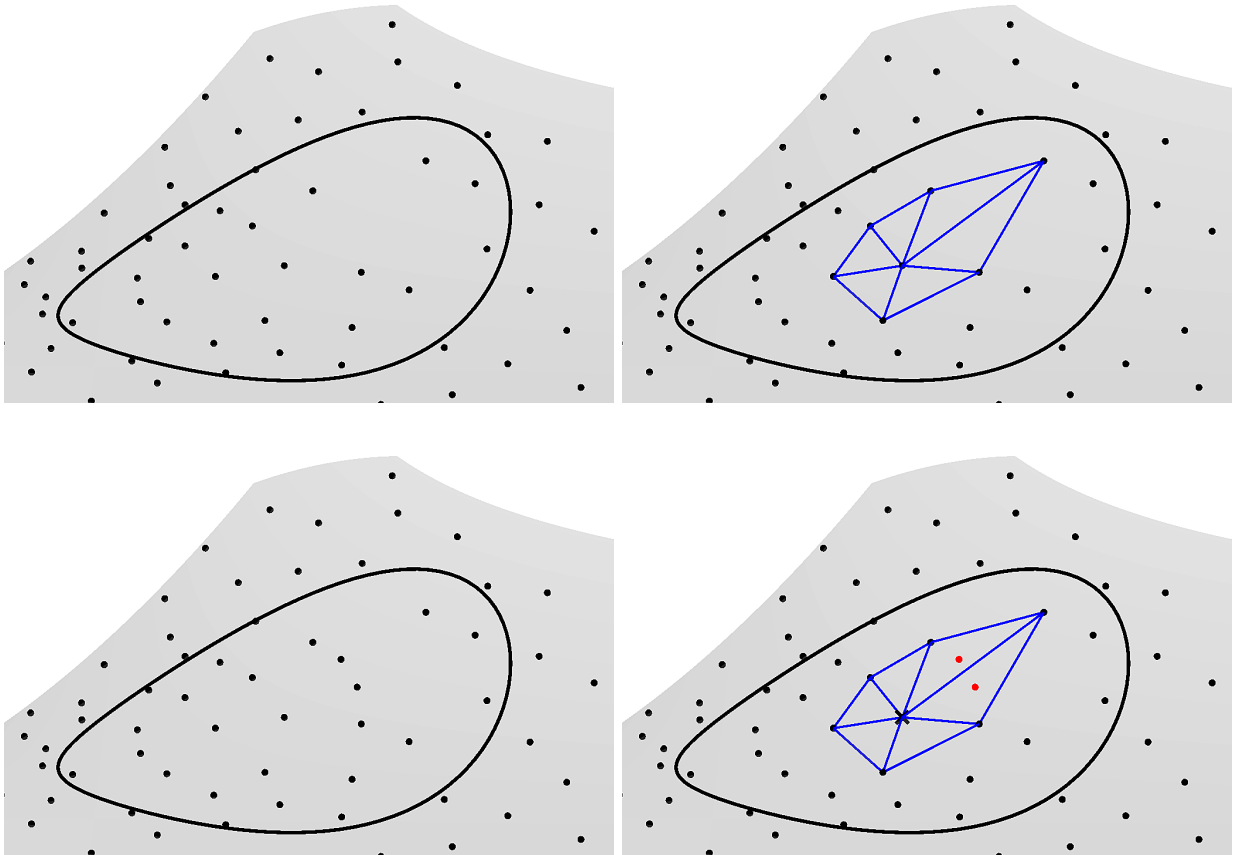


Figure 2: Addition of points in regions with insufficient points. Clockwise from top left: Initial configuration (top left), triangulation of a point within the support radius (top right), addition of red points in ‘large’ triangles (bottom right), and final configuration (bottom left).

For each newly created point, we need to approximate all physical quantities there. A considerable amount of work has been done in determining optimal ways to perform approximations at these new points for the volumetric Lagrangian framework [9, 15, 23] and those can easily be adapted for the needs of the surface-based case.

We note that for boundary points, we need to ensure to not form a triangle between three boundary points, as that could lead to addition of points outside the domain. To ensure a sufficient number of points on the boundaries, a separate addition process is also done only for boundary points, with the distance between two adjacent boundary points checked.

3.4.1. Curvature Corrected Addition

The above process of addition at the circumcenter of large triangles assumes the surface to be piecewise linear. This can be improved by correcting the new location to take the curvature into account. For a point being added, we start by approximating a normal at that point by

$$\vec{n}_{approx} = \frac{\vec{n}_1 + \vec{n}_2 + \vec{n}_3}{3}, \quad (6)$$

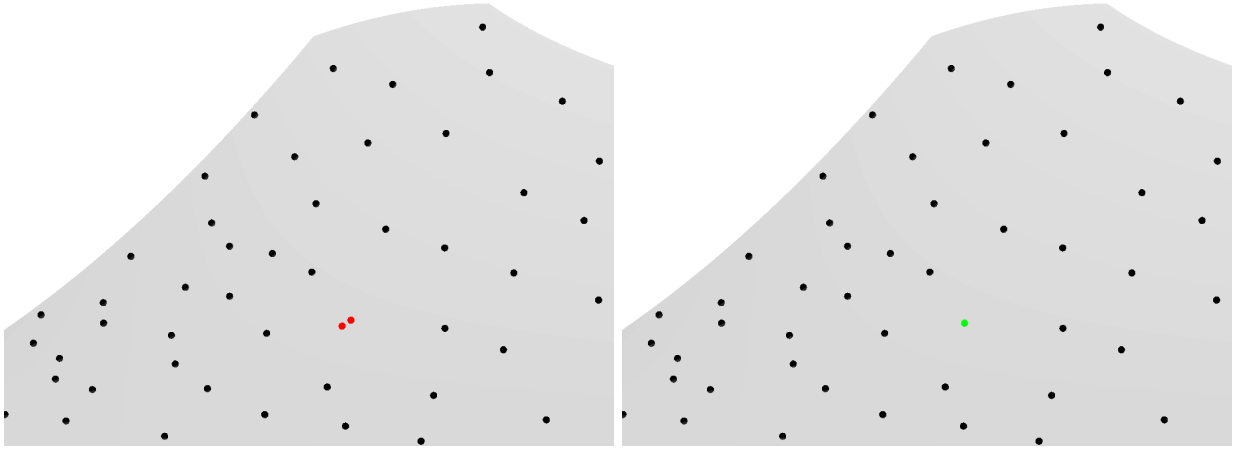


Figure 3: Merging points that are too close. The two red points in the left figure are merged to the green point in the right figure.

where the summation is over the three points in the triangle being considered. The newly created point is then moved in the direction of this approximated normal

$$\vec{x}_{new} = \vec{x}_c + d_\kappa \vec{n}_{approx}, \quad (7)$$

where \vec{x}_c is the original location at the circumcenter, and \vec{x}_{new} is the corrected location. The distance d_κ of moving the newly created point is computed based on the angles between the distance vectors and the normal field, in a manner similar to the procedure of normal computation explained earlier. The following constraint is enforced:

$$\sum_j \delta \vec{x}_j \cdot \vec{n}_j = 0, \quad (8)$$

where the sum is over the three points of the triangle, and $\delta \vec{x}_j = \vec{x}_j - \vec{x}_{new}$. This leads to

$$d_\kappa = \frac{\sum_j (\vec{x}_j - \vec{x}_c) \cdot \vec{n}_j}{\sum_j \vec{n}_{approx} \cdot \vec{n}_j}. \quad (9)$$

3.5. Deletion or Merging of Points

To prevent clustering of points as a result of Lagrangian movement, numerical points within $r_{min}h$ of each other are merged into a single point, with all physical properties interpolated at the new location. This process is illustrated in Figure 3.

In this process, priority is given to boundary points. If one boundary point and one interior point are within a distance of $r_{min}h$ from each other, instead of merging the two points, the interior point is deleted, and values are re-interpolated at the boundary point. This is done so as to not artificially deform the boundary.

These procedures of addition and deletion of points can be used to obtain adaptive refinement, when needed. Changing the smoothing length h will trigger the required point addition or merging algorithm. For example, refinement can be done based on some error bounds, as has been done in the context of volumetric meshfree GFDMs by [47, 59]. Alternatively, refinement can be carried out based on the gradient of a numerical solution $\nabla\phi$. The smoothing length h can be reduced with increasing $\|\nabla\phi\|$, which would result in a higher number of points in those locations. This process is illustrated in a numerical example in Section 3.7.3.

3.6. Contact Detection and Topological Change

Effectively modelling topological change of manifolds, and contact between different manifolds is one of the most challenging parts of setting up a comprehensive Lagrangian framework. Collision detection is usually done with the help of a mesh [61]. Even particle methods rely on a background mesh for the same [29]. A lot of work has been done for collision detection between two volume phases, by both mesh-based methods [61] and meshfree methods [31]. Several adaptations of these are needed to apply such methods for collision detection between surfaces given by point clouds.

As a preliminary, each meshfree node is assigned a ‘chamber’ attribute to indicate the difference between different manifolds. Note that for each point i , the n_{S_i} points in its neighbourhood S_i all belong to the same chamber. We start with collision between different manifolds, and later extend these algorithms for self-intersection which involves contact between different parts of the same manifold.

In this paper, we do contact detection in two steps. For a particular point i in chamber c_i , the first step involves a spatial search to determine which points of other chambers could possibly come into contact with the current chamber. To do this efficiently, and to avoid the need to compute distances with every point of other chambers, we use the neighbour searching algorithms mentioned above. After this stage, we assume that each point i has a list of own-chamber neighbours S_i , and a list of other-chamber ‘neighbours’ S_i^{ext} which are within a distance h of point i . Note that, unlike S_i , there is no need to store S_i^{ext} after contact detection is done. Furthermore, $i \in S_i$ but $i \notin S_i^{ext}$.

The next step involves the actual contact determination. We propose doing this in two parts. One involves determining penetrated points, i.e. determining parts of a surface that have ‘crossed’ another surface. For this, we first compute a signed distance DC_i to the other chamber, where DC_i indicates the distance to contact. DC_i is computed as the distance of the point i from an approximated surface given by the points in S_i^{ext} . The sign depends on the normals of S_i^{ext} . First, the distance to the surface given by S_i^{ext} is approximated as a quadratic polynomial

$$d^{S_i^{ext}} = d_0 + d_1x + d_2y + d_3x^2 + d_4y^2 + d_5xy, \quad (10)$$

where the coefficients d_k are determined based on the positions and normals of the points in S_i^{ext} . For this, we use the fact that the distance should be 0 at every point in S_i^{ext} . Thus, $d^{S_i^{ext}}(\vec{x}_m) = 0, \forall m \in S_i^{ext}$. Furthermore, the gradient of the distance function should be along the normal to the surface. At a dummy location obtained by moving a point in S_i^{ext} by a fixed distance ξ in the normal direction, the distance function should return ξ . Thus, $d^{S_i^{ext}}(\vec{x}_m \pm \xi \vec{n}_m) = \pm \xi, \forall m \in S_i^{ext}$ and sufficiently small $\xi > 0$. We take ξ to be a third of the minimum distance between points. These two conditions together form, in general, an over-determined system, which is solved in a least squares sense to determine the coefficients d_k . Then, DS_i is computed by evaluating Eq. (10) at \vec{x}_i .

It is important to note that, unlike the volume-based case, the sign of the distance computed above does not directly determine whether or not the point has penetrated another chamber. In the volume case, each boundary surface has normals pointing either inwards or outwards, which is known a-priori. However, in the case of manifolds, the concepts of interior and exterior are not always well defined. And thus, the direction of the normals are not always known a-priori. This is illustrated in Figure 4. In Figure 4, the red and blue manifolds are moving towards each other. It shows the two possibilities of the normal direction of the red manifold. As shown in the figure on the right, it is quite possible that the normals of the chambers are pointing in the same direction. Thus, checking for conflicting Lagrangian information, as is often done [50], is not always enough.

Thus, the sign of DC_i at one time step alone does not give us enough information. To solve this problem, we store the information about signed distance for each point. If the sign of DC_i has flipped compared to the previous time step, then we can say that the point has penetrated another chamber. Using a soft penetration model, $|DC_i|$ is then used for force computation, which depends on the considered physical model, as would be done in mesh-based methods [61].

The above method checks for opposing chambers that crossed each other in the previous movement step. The second part of contact detection involves determining opposite chamber points that are almost in contact. This is done by keeping track of points that are within the deletion distance of $r_{min}h$, but have not

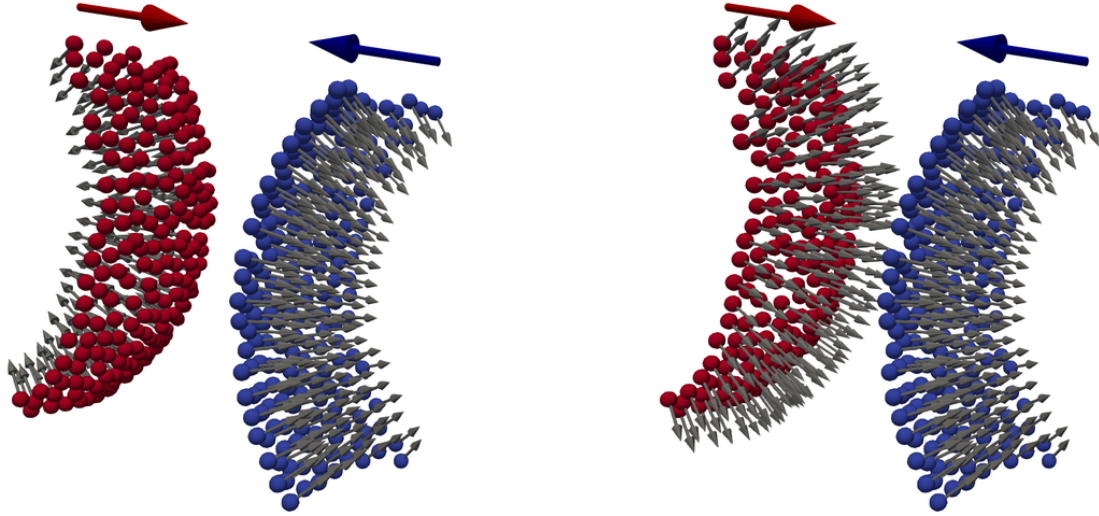


Figure 4: Different possible orientations of manifolds coming into contact. The colours of the points indicate the different manifolds. The grey arrows are the manifold normals. The red and blue arrows indicate the velocities of the manifold. The blue and red manifolds are moving towards each other in both cases. In the figure on the left, the normals of the two manifolds point away from each other. While in the figure on the right, they point towards each other.

yet crossed into the opposite chamber. Once again, force computation can be done based on the distance, which will depend on the considered physical model.

The processing of detected contacts between manifolds is differentiated into 3 types:

- *Non-penetration contact*: This is the most standard form of contact, indicating that two surfaces can not cross each other. Handling this numerically takes the form of adding the correct force once particles penetrate or come too close to another chamber, as described above.
- *Delete contact*: This is the case, for example, when two fluid droplets modelled only by their surfaces come into contact with one another. Thus, the points of each chamber coming into contact are deleted. For the geometric handling of this case, a numerical point is deleted if it has just penetrated into another chamber, or if it is within deletion distance of a point of another chamber.
- *Merge contact*: This third situation happens when two manifolds ‘stick’ together, such that the resultant can be treated as a single manifold, with modified properties. For this, points of different chambers are merged into a single chamber.

Numerical examples of the first two types of contact, based purely on the geometry of the problem, without any physical model, are presented in Section 3.7.3.

The above methods can also be extended quite easily for contact detection between different parts of the same manifold. For this, the only difference is the identification of the equivalents for S_i and S_i^{ext} defined above, which is done based on the direction of the normals.

3.7. Numerical Results

Before solving PDEs on evolving surfaces, we first test the Lagrangian framework by only performing advection of the manifold. In each of the simulations presented below, the point clouds are irregularly spaced. The initial point clouds are obtained starting from a CAD file by placing points using an advancing front technique for point clouds, similar to that done by [9].

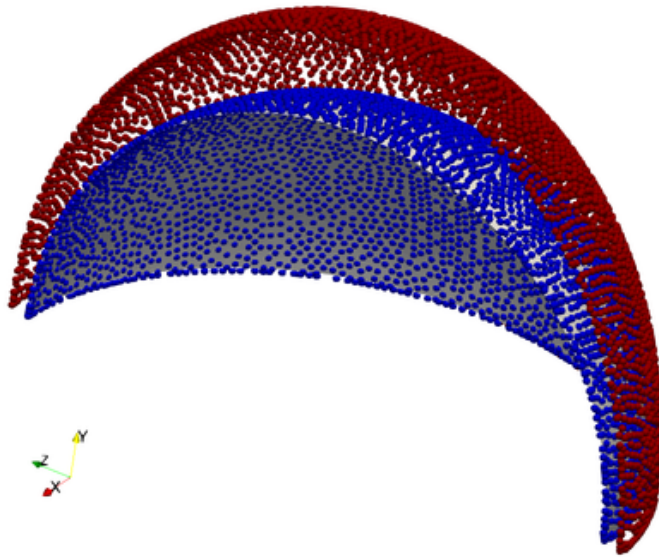


Figure 5: Quarter sphere point clouds after one full rotation. The blue point cloud is a result of the second order movement, Eq. (3), while the red point cloud is a result of the first order movement, Eq. (2). The grey background shell represents the original configuration at initial time.

3.7.1. Rotating Quarter Sphere

To show the need of movement by the second order method according to Eq. (3), we consider the case of a quarter of a unit sphere rotating about its center. To illustrate the effect of the movement only, no addition and deletion of points is done in this example. The sphere is centered at the origin, and the velocity of the surface is given by

$$\vec{v} = \begin{pmatrix} y \\ -x \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ z \\ -y \end{pmatrix}, \quad (11)$$

where the terms correspond to a rotation about the z and x axis, respectively. Clearly, the velocity is tangential to the surface at every point. Thus, updating point locations with the conventionally used first order movement, Eq. (2), results in points going off the sphere. As a result, numerically, the radius of the quarter sphere increases with every time step. This issue is no longer present for the second order movement, Eq. (3). Figure 5 shows the point clouds after one full rotation based on the two different movement methods in comparison with the original quarter sphere. The quarter sphere surface is discretized with $N = 3792$ irregularly spaced points. The time step is fixed as $\Delta t = 0.05$. Figure 5 illustrates the inaccuracy of the first order movement whenever the advection velocity has a rotational component. The blue point cloud from the second order movement is virtually at the initial location, while the red one from the first order movement is off it. To quantify this error, we measure the distance from the unit sphere. After one full rotation, the mean distance of the resultant point clouds from the unit sphere is 1.37×10^{-1} for the first order movement, and 5.57×10^{-4} for the second order movement. It is important to note here that for the first time step, the second order method reverts to the first order one due to unavailability of a previous velocity, and that the mean error after the first time step is 1.12×10^{-4} for both methods. The error as a result of the second order movement barely increases after the first time step, whereas that of the first order movement increases by several orders of magnitude. Thus, we now only use the movement according to Eq. (3) for the remainder of the paper.

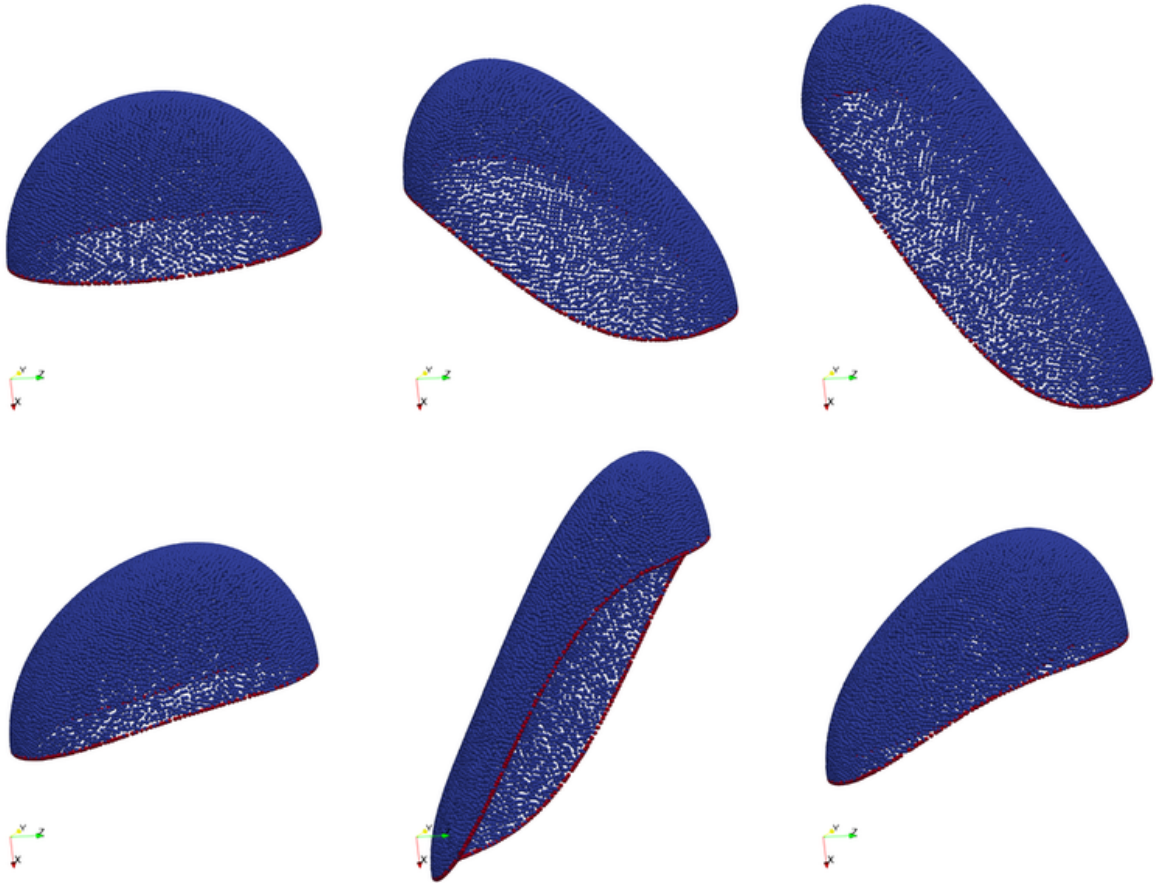


Figure 6: Elongation and contraction of a unit hemisphere, with addition and deletion of points to maintain regularity. Clockwise from top left: At times $t = 0$ (top left), $t = 0.09$ (top center), $t = 0.24$ (top right), $t = 0.55$ (bottom right), $t = 0.745$ (bottom center) and $t = 0.975$ (bottom left). The red points indicate manifold boundaries.

3.7.2. Deforming Hemisphere

To illustrate the addition and deletion of points in the Lagrangian framework, we consider stretching and contracting of a unit hemisphere as shown in Figure 6. The velocity field is given by

$$\vec{v} = \begin{pmatrix} 2\pi \cos(2\pi t) \sin(\frac{\pi}{2}z) \\ 0 \\ 0 \end{pmatrix}, \quad (12)$$

which is a modification of the example considered in [6]. Figure 6 shows the evolution of the surface for $\Delta t = 0.005$. The number of points N on this evolving surface as a function of time is shown in Figure 7. As the sphere is stretched, to maintain regularity of the point cloud, points are added on the manifold. For the same reason, points are deleted from the manifold as the sphere contracts.

This serves as a good example of surface deformation because the surface returns to its original shape. We know that at every multiple of $t = 0.5$, the manifold should once again take the initial shape. The resulting error in the Lagrangian movement can thus be checked by measuring the mean distance from the

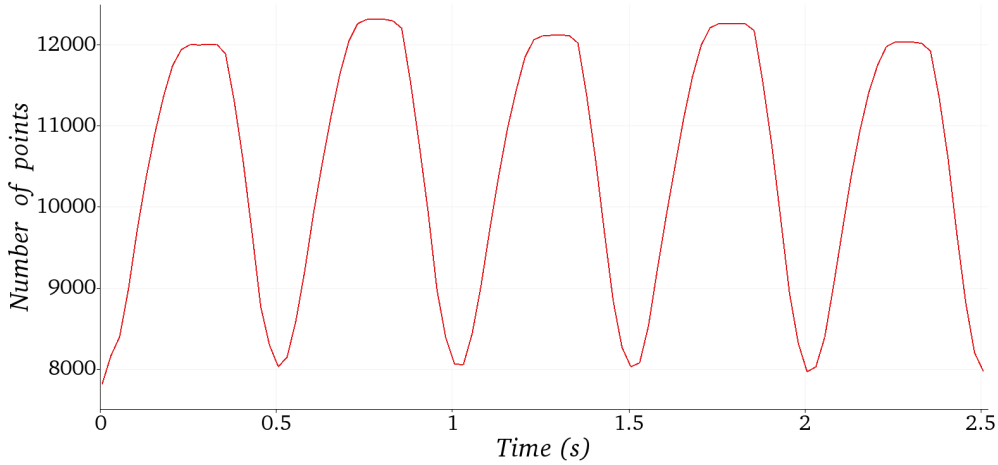


Figure 7: Number of points as a function of time for the distorting hemisphere test case.

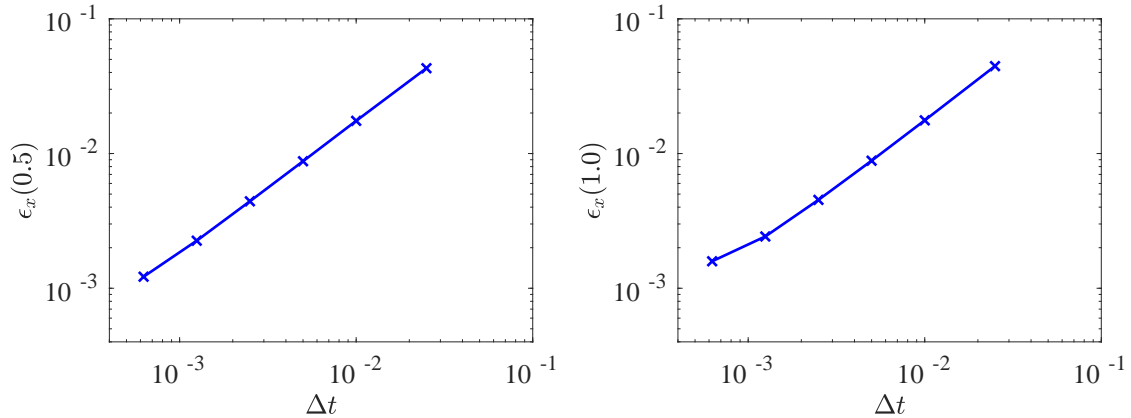


Figure 8: Convergence of error in location as the deforming hemisphere returns to its original state at $t = 0.5$ (left) and $t = 1.0$ (right).

unit sphere, as done in the previous example.

$$\epsilon_x(t) = \frac{1}{N(t)} \sum_{i=1}^{N(t)} \left| \|\vec{x}_i\|^2 - 1 \right|. \quad (13)$$

We reiterate that ϵ_x should be 0 only for integer multiples of $t = 0.5$. The convergence of the error ϵ_x at $t = 0.5$ and $t = 1.0$ is plotted in Figure 8 for a varying time step. The two plots also show that the accumulation of the Lagrangian movement error with time is not significant. The figure illustrates that a numerical convergence order of $\mathcal{O}(\Delta t)$ is observed. The main reason for the small convergence order is that the rate determining step is the very first time step, where a first order movement is necessary as no previous velocity is available. We note that the same was quantified in the previous example. This could be avoided by prescribing an initial velocity derivative, which is not done here.

3.7.3. Joining Spheres

To illustrate the contact handling algorithms, we consider two unit spheres moving towards each other.

Simulations are done for the same with both delete contact, and non-penetration contact enforced, as described in Section 3.6.

The two examples considered in this section are only used to illustrate the applicability of the geometric collision detection and penetration avoidance algorithms presented earlier. Thus, no physical model has been applied. The delete contact case is handled as described in Section 3.6. For the non-penetration contact case, since no force is added, a slight modification is required to avoid penetration. Once penetration has been detected geometrically, as described earlier, penetrated points are projected back so as to ensure that the two discrete manifolds do not cross each other. Velocities are recomputed as averaged values after the projection.

In both simulations, we use a time step of $\Delta t = 0.03$, and a smoothing length of $h = 0.1$ which corresponds to $N = 15018$ total points in each of the two spheres at the initial time level. The spheres move towards each other with a velocity of $\vec{v} = (-\text{sign}(x_1)0.5, 0, 0)$, where ‘sign’ is the signum function, and the origin is between the two spheres at initial state. The center of the spheres are at an initial distance of 2.2 units apart.

For the non-penetration contact case, Figure 9 shows the evolution of the spheres as they move towards one another. Note that Figure 9 only shows a slice of the two spheres, such that only the half away from the viewing angle is seen.

The delete contact simulation uses the same setup with one difference. To better process the contacts, we apply an adaptive refinement. For regions closer to the interface, the smoothing length h is reduced to $h = 0.05$. As a result, the hole filling algorithm explained in Section 3.4 causes the point cloud to become finer near the interface. Figure 10 shows the result when enforcing the delete contact. It also illustrates the adaptive refinement. Once again, only a slice of the result is shown. We note that the jagged interface between the two surfaces after deletion on contact is a result of unevenly spaced points. This becomes more regular (more straight, in this case) as the number of points is increased, or as refinement is done near the interface.

3.7.4. Addition of Points

The examples considered so far use addition of new points at the circumcenter of ‘large’ triangles. We now consider curvature corrected addition as per Section 3.4.1. We consider the case of addition of points on the stationary surfaces of a sphere, and of a torus given by

$$\left(c - \sqrt{x^2 + y^2}\right)^2 + z^2 = a^2, \quad (14)$$

with $c = 3$ and $a = 1$. In each case, the smoothing length is decreased which results in the point cloud getting finer. The error in the location of addition of new points is measured as the mean distance off the surface. This error is give by Eq. (13) for the sphere. For the torus, it is given by

$$\epsilon_x = \frac{1}{N(t)} \sum_{i=1}^{N(t)} \left| \left(c - \sqrt{x^2 + y^2}\right)^2 + z^2 - a^2 \right|. \quad (15)$$

For both considered geometries, points are added to 4 point clouds of different discretization sizes to observe how the error in location converges with increasing number of points. In each case, the initial smoothing length h_0 is halved, which results in approximately quadrupling the number of points. The resultant error after the entire addition procedure is plotted in Figure 11 for different starting discretizations. The corresponding number of points in each domain before the refining is done is shown in Table 1. Figure 11 illustrates that using the curvature corrected addition results in about halving the error, and that the errors converge faster than $\mathcal{O}(h^2)$. Further improvements to curvature corrected addition remains an open problem, and will not be discussed in the present work.

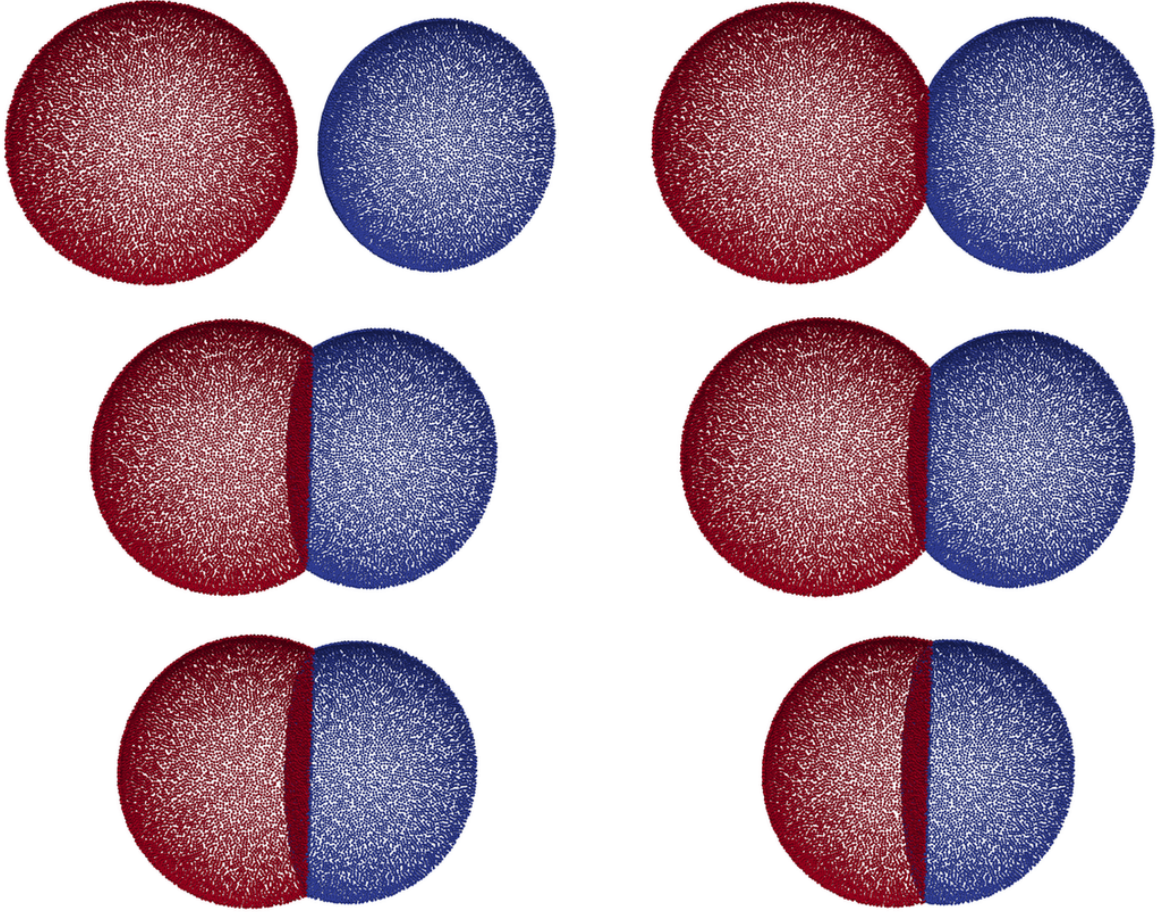


Figure 9: Two sphere surfaces moving towards each other, with non-penetration contact enforced. Clockwise from top left: At times $t = 0$ (top left), $t = 0.36$ (top right), $t = 0.72$ (middle right), $t = 1.08$ (middle left), $t = 1.44$ (bottom left) and $t = 1.8$ (bottom right). The colour represents the different manifolds. To make the result easier to visualize, only the half of the domain away from the viewing angle is shown in the figure.

Table 1: Number of points (before refining) corresponding to the discretization sizes mentioned in Figure 11. In each case, the final number of points after refining, when the errors are measured, are approximately 4 times that of the initial number mentioned below.

h_o	Sphere	Torus
0.4	770	7801
0.2	3520	34836
0.1	14994	147112
0.05	63627	612828

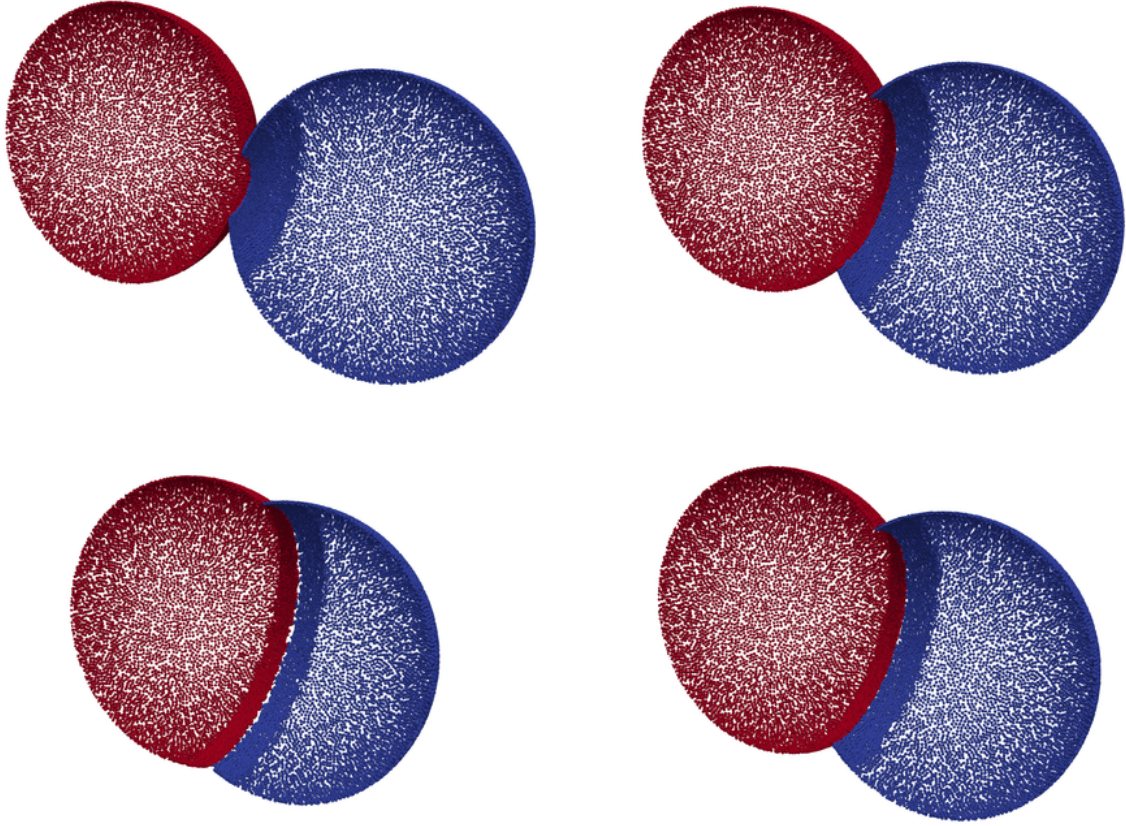


Figure 10: Two sphere surfaces moving towards each other, with delete contact enforced. Clockwise from top left: At times $t = 0.03$ (top left), $t = 0.63$ (top right), $t = 0.93$ (bottom right), and $t = 1.5$ (bottom left). The colour represents the different manifolds. To make the result easier to visualize, only the half of the domain away from the viewing angle is shown in the figure.

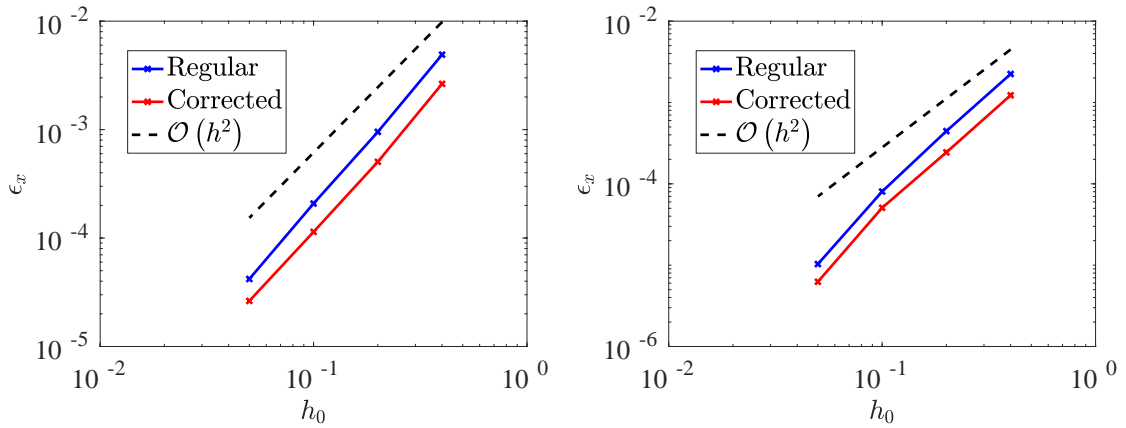


Figure 11: Error in location of newly added points against the discretization size. The blue lines indicate the errors after regular addition at the circumcenter. The red lines indicate the errors when the curvature corrected addition according to Section 3.4.1 is used. Errors are shown for the surface of a sphere (left) and the surface of a torus (right).

4. PDEs on moving manifolds

We now use the meshfree Lagrangian framework for surfaces developed above to solve PDEs on moving manifolds. In contrast to many existing methods for PDEs on evolving surfaces [6], in this method, integration is done on the surface at the existing time level, and no map is maintained between the initial and current state. At each time step, differential operators are computed for the point positions at that level directly.

The moving Lagrangian framework for evolving PDEs developed in Section 3 of this paper can be used to solve PDEs on moving manifolds with a wide variety of methods to define the numerical derivatives. In this paper, we use meshfree Generalized Finite Difference Methods (GFDMs) for surface PDEs for the same. This is based on our earlier work for PDEs on stationary surfaces [56]. For the sake of completeness, we provide a short description about how they are computed.

4.1. Differential Operators

Meshfree GFDMs [14, 18, 26, 38] are strong form methods. For volume-domains, they have been shown to be robust methods, with a wide variety of applications [9, 24, 42, 45]. They have also been referred to under the name of Generalized Moving Least Squares (GMLS) [43]. Differential operators are computed using a least squares approach while ensuring that monomials up to a certain order are differentiated exactly.

Here, we use the GFDM formulation for surface PDEs from [56]. This relies on discretizing differential operators entirely in the tangent space. This method scales with the true dimension of the manifold. It has the advantage of avoiding differential geometry complexities. One of the biggest advantages of this method is that most existing developments in volume-based numerical methods can directly be carried over to surfaces. GFDMs also have the advantage that it is straightforward to handle a wide variety of boundary conditions.

Differential operators are computed by virtual projections to the tangent space at each point which recovers a Euclidean problem in the dimension of the tangent space. Below, we explain how the differential operators are computed. For details about the same, we refer to [56].

4.1.1. Surface Gradients

For a function u defined on the surface, its surface gradient is given by

$$\nabla_M u = \nabla \hat{u}, \quad (16)$$

where ∇_M is the surface gradient, ∇ is the volumetric gradient, and \hat{u} is a normal extension of u to a band around the surface such that $\vec{n} \cdot \nabla \hat{u} = 0$. Thus, computing a numerical approximation to $\nabla_M u$ can be done by computing a numerical approximation to $\nabla \hat{u}$. Since $\vec{n} \cdot \nabla \hat{u} = 0$, we only have to approximate the tangential components $\vec{t}_k \cdot \nabla \hat{u}$ for $k = 1, 2$. For each point i , these tangential derivatives $\vec{t}_k \cdot \nabla \hat{u}$ are approximated in the tangent plane T_i spanned by $\vec{t}_{1,i}$ and $\vec{t}_{2,i}$.

For this, given a point i on the surface, its neighbouring points $j \in S_i$ are projected to the tangent space T_i . The projection of the point $j \in S_i$ to T_i will be referred to by j_{T_i} . This projection is done along the manifold normal \vec{n}_i . This process is illustrated for a 1-dimensional manifold in \mathbb{R}^2 in Figure 12. The projection to the tangent space is not actually done at the numerical level. Only the distances between the central point i and the projected locations j_{T_i} are required. These are computed by rotating the distance vector $\delta \vec{x}_{ij} = \vec{x}_j - \vec{x}_i$. Using these distances, volumetric differential operators are computed in the tangent space.

For a central point i , and projected locations j_{T_i} , usual 2-dimensional gradients are computed along the $\vec{t}_{1,i}$ and $\vec{t}_{2,i}$ directions, given by

$$\nabla_T \hat{u} \approx \tilde{\nabla}_T \hat{u} = \begin{pmatrix} \sum_{j \in S_i} c_{ijT}^{t_1} \hat{u}_{jT} \\ \sum_{j \in S_i} c_{ijT}^{t_2} \hat{u}_{jT} \end{pmatrix}, \quad (17)$$

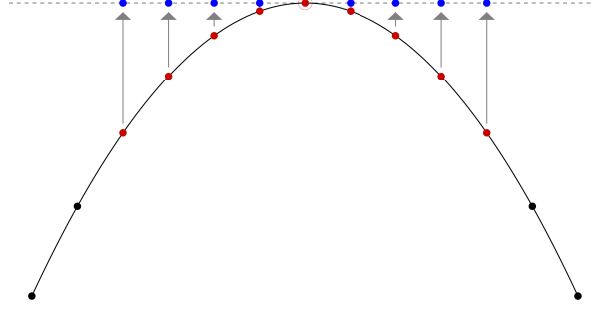


Figure 12: Projection to the tangent plane along the central normal. The central point is marked with an extra circle around it. Its neighbouring red points are projected to the tangent plane to the blue locations.

where ∇_T denotes the gradient in the tangent plane, the $\tilde{\cdot}$ overhead denotes the discrete derivative, and $\hat{u}_{j_T} = u_j$ denotes the function value at the projected location j_T . The stencil coefficients c_{ij_T} are computed using a least squares approach while ensuring that monomials up to a certain order are differentiated exactly:

$$\sum_{j \in S_i} c_{ij_T}^{t_k} m_{j_T} = \frac{\partial}{\partial t_k} m(\vec{x}_i) \quad \forall m \in \mathcal{P}_T, \quad (18)$$

$$\min J_i = \sum_{j \in S_i} \left(\frac{c_{ij_T}^{t_k}}{W_{ij_T}} \right)^2, \quad (19)$$

for $k = 1, 2$, where \mathcal{P}_T is the set of monomials, usually up to order 2, in $\vec{t}_{1,i}$ and $\vec{t}_{2,i}$ on the tangent plane. These are then rotated to obtain the numerical surface gradients. The surface derivatives are given by

$$\tilde{\nabla}_{M,i} u = \begin{pmatrix} \sum_{j \in S_i} c_{ij}^{M,x} u_j \\ \sum_{j \in S_i} c_{ij}^{M,y} u_j \\ \sum_{j \in S_i} c_{ij}^{M,z} u_j \end{pmatrix}, \quad (20)$$

where $c_{ij}^{M,x}$ are the stencil coefficients for the surface gradient in the x direction, and similarly for the other directions. We have

$$\begin{pmatrix} c_{ij}^{M,x} \\ c_{ij}^{M,y} \\ c_{ij}^{M,z} \end{pmatrix} = R^T \begin{pmatrix} c_{ij}^{t_1} \\ c_{ij}^{t_2} \\ c_{ij}^n \end{pmatrix}, \quad (21)$$

for $c_{ij}^n = 0$, and a rotation matrix R given by

$$R^T = (\vec{t}_1 \quad \vec{t}_2 \quad \vec{n}), \quad (22)$$

for column vectors \vec{t}_1 , \vec{t}_2 , and \vec{n} .

4.1.2. Surface Divergence

We first rewrite the numerical surface gradients computed above as

$$\tilde{\nabla}_{M,i}u = (G_{1i}u, G_{2i}u, G_{3i}u)^T. \quad (23)$$

Thus, $G_{1i}u = \sum_{j \in S_i} c_{ij}^{M,x} u_j$, and similarly for G_{2i} and G_{3i} . Using this notation, the divergence of a vector valued function $\vec{v} = (v^1, v^2, v^3)$ defined on the manifold is given by

$$\tilde{\nabla}_M \cdot \vec{v} = \sum_{k=1,2,3} G_{ki} v^k. \quad (24)$$

4.1.3. Surface Laplacian

The numerical surface Laplacian or Laplace Beltrami is given by

$$\tilde{\Delta}_M u = \sum_{j \in S_i} c_{ij}^{\Delta_M} u_j. \quad (25)$$

To determine the stencil coefficients $c_{ij}^{\Delta_M}$, we proceed in a manner similar to what was done above for the gradients. We first compute a 2 dimensional volumetric Laplacian operator on the tangent plane:

$$\sum_{j \in S_i} c_{ijT}^{\Delta_T} m_{jT} = \Delta_T m(\vec{x}_i) \quad \forall m \in \mathcal{P}_T, \quad (26)$$

$$\min J_i = \sum_{j \in S_i} \left(\frac{c_{ijT}^{\Delta_T}}{W_{ijT}} \right)^2. \quad (27)$$

Since the Laplacian is rotationally invariant, the tangent plane Laplacian directly gives us the surface Laplacian $c_{ij}^{\Delta_M} = c_{ijT}^{\Delta_T}$.

The verification of the concept and implementation of these differential operators for PDEs on stationary surfaces has been done in [56].

4.2. Numerical Results

All sparse linear systems arising from the discretization of the PDEs are solved using a BICGSTAB2 iterative solver [60], without the use of any preconditioner. Monomials up to the order of 2 are used in the computation of all the differential operators.

4.2.1. Convection Diffusion Reaction Equation on an Expanding Sphere

We consider the case of an advection diffusion reaction equation on an evolving manifold

$$\frac{D\phi}{Dt} + \phi \nabla_M \cdot \vec{v} = \alpha \Delta_M \phi + f(\phi), \quad (28)$$

where \vec{v} is the advection velocity of the surface, $\frac{D\phi}{Dt}$ is the advective surface material derivative, and α is the diffusion coefficient. Note that, in general, \vec{v} can have a component both normal and tangential to the manifold.

The time integration proceeds as follows. In each time step, first the Lagrangian movement step of Eq. (3) is performed. After that, integration of the remaining terms of Eq. (28) is done with a first order implicit method

$$\frac{\phi^{(n+1)} - \phi^{(n)}}{\Delta t} + \phi^{(n+1)} \nabla \cdot \vec{v}^{(n+1)} = \alpha \Delta_M \phi^{(n+1)} + f(\phi^{(n+1)}), \quad (29)$$

where the bracketed superscripts indicate the time level. Discretization of the spatial operators is done as mentioned in Section 4.1.

To check experimental orders of convergence of the proposed method, we consider the simple example of an expanding sphere. The initial unit sphere, discretized with irregularly spaced points, expands at a constant rate. The surface at time t is given by

$$x^2 + y^2 + z^2 = (r(t))^2, \quad (30)$$

with $r(t) = 1 + 0.5t$. The considered velocity field is $\vec{v} = 0.5\vec{n}$. The numerical solution is compared to a manufactured solution

$$\phi_{\text{exact}}(\vec{x}, t) = \exp(-6t)xy. \quad (31)$$

Since the exact solution and the location of the surface is known analytically, this serves as a good test case for the Lagrangian movement, as well as for the discretization of the spatial derivatives. The reaction term f is determined analytically so that the manufactured solution satisfies Eq. (28) with $\alpha = 1$. This makes use of the following:

$$\nabla_M \cdot \vec{v} = \frac{1}{r(t)}, \quad (32)$$

$$\Delta_M \phi_{\text{exact}} = -\frac{6\phi_{\text{exact}}}{(r(t))^2}. \quad (33)$$

Relative errors in the numerical solution are measured at $t = 1$ by

$$\epsilon_2 = \left(\frac{\sum_{i=1}^N \|\phi_i - \phi_{\text{exact}}(\vec{x}_i)\|^2}{\sum_{i=1}^N \|\phi_{\text{exact}}(\vec{x}_i)\|^2} \right)^{\frac{1}{2}}. \quad (34)$$

Note that the numerical errors found are not just a function of the numerical differential operators, but also of the Lagrangian movement and the addition/deletion of points to maintain point cloud regularity.

The errors and convergence orders are tabulated in Table 2 with a changing spatial discretization and a small time step of $\Delta t = 0.4h^2$. Convergence orders are measured with respect to the number of points at the initial time. A second order convergence is observed for the same. The numerical convergence with a changing time step is shown in Table 3 for $h = 0.1$. In this case, first order convergence is observed.

Table 2: Errors at $t = 1$ and convergence orders with h for the expanding sphere test case. h is the smoothing length, N is the number of points in the entire domain at the initial time, ϵ_2 is the relative error, and r is the order of convergence of ϵ_2 .

h	N	ϵ_2	r
0.4	480	6.57×10^{-2}	—
0.2	1 806	1.93×10^{-2}	1.85
0.1	7 446	4.76×10^{-3}	1.98
0.05	30 054	1.19×10^{-3}	1.98

Table 3: Errors at $t = 1$ and convergence orders with respect to the time step Δt for the expanding sphere test case. ϵ_2 is the relative error, and r is the order of convergence of ϵ_2 .

$\Delta t \times 10^2$	ϵ_2	r
1	5.45×10^{-2}	—
1/2	2.74×10^{-2}	0.99
1/4	1.38×10^{-2}	0.99
1/8	6.99×10^{-3}	0.98
1/16	3.56×10^{-3}	0.97

4.2.2. Diffusion on a Deforming Half Pipe

Next, we consider another convection diffusion reaction case according to Eq. (28). Unlike the previous case, here, the manifold has a boundary, the surface velocity \vec{v} is dependent on the solution of the PDE, and also has a tangential component. Moreover, the deformation of the surface is more pronounced. Such cases of high surface deformation are important to consider here since these are the examples where the advantage of a moving meshfree method over a moving mesh method become evident. Under such high deformation, a moving mesh method would require a lot of remeshing, which would significantly slow down the simulation. The initial domain is taken to be one half of a pipe with radius 0.25 bent in an arc of radius 0.75, as shown in the first image in Figure 13. The velocity field is given by

$$\vec{v} = 0.5 \phi \vec{n} + \begin{pmatrix} -yz \\ 0 \\ xz \end{pmatrix}, \quad (35)$$

where \vec{n} is the unit normal vector, and ϕ is governed by Eq. (28). The first term in Eq. (35) causes the surface to expand with the ‘temperature’ ϕ , while the second term causes an overall twisting motion with the two halves twisting in opposite directions. An initial condition with discontinuities is considered. At $t = 0$, the domain has two ‘hot’ regions with $\phi = 1$, with the rest of the domain at $\phi = 0$. We set $f = 0$ and $\alpha = 0.2$ in Eq. (28). The domain is discretized with a smoothing length of $h = 0.05$ that corresponds to an initial number of points of $N = 11\,802$. Time integration is done as mentioned in the previous test case, with the velocity \vec{v} taken as a function of ϕ at the previous time level. For a time step $\Delta t = 0.01$, Figure 13 shows the evolution of the surface and the solution ϕ . The initial half pipe twists in two different directions, with a slight expansion in the regions of high ϕ , as ϕ diffuses.

4.2.3. Geometric Motion

We consider motion of a surface dependent on its curvature. First, we consider the mean curvature flow [7], in which the velocity of the surface is given by

$$\vec{v} = \kappa \vec{n}, \quad (36)$$

where \vec{n} is the unit surface normal. The mean curvature κ is given by

$$\kappa = -\frac{1}{2} \nabla_M \cdot \vec{n}. \quad (37)$$

An initial dumbbell shape is used as the domain as shown in Figure 14. The dumbbell is composed of two unit spheres with their centers 4 units apart, connected by a cylindrical shape of radius 0.2. There is a sharp change between the dumbbell ends and the handle. The simulation is done with $h = 0.12$, which corresponds to an initial number of points of $N = 20\,015$, and a time step of $\Delta t = 0.005$. The results show the dumbbell shrinking, causing a neck pinch singularity, leading to the separation of the two ends, which evolve towards spheres as they contract. This matches the known behaviour of the dumbbell shape under mean curvature flow [20, 40]. As a second geometric motion test case, we consider the averaged mean curvature flow [41]. The surface velocity is slightly modified from Eq. (36) to

$$\vec{v} = (\kappa - \bar{\kappa}) \vec{n}, \quad (38)$$

where $\bar{\kappa}$ denotes the global average of the curvature. The considered computational domain is the torus defined by

$$\left(c - \sqrt{x^2 + y^2} \right)^2 + z^2 = a^2, \quad (39)$$

for $c = 3$ and $a = 1$. A coarsely discretized domain is used with $h = 0.9$, which corresponds to $N = 1\,490$ points at the initial state. The results for a time step of $\Delta t = 0.05$ are shown in Figure 15. The torus

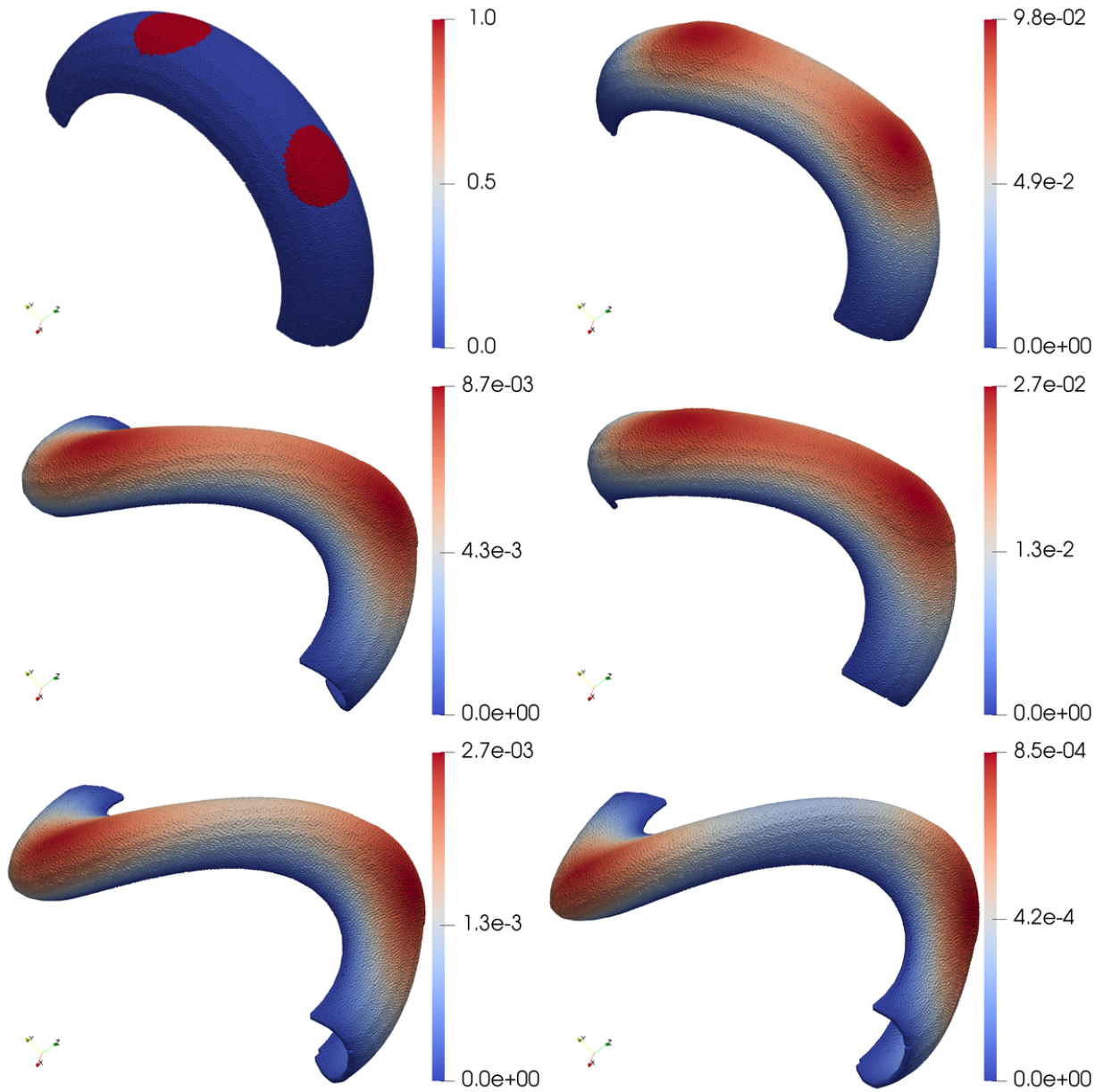


Figure 13: Diffusion on a deforming half pipe: Surface domain and solution shown at selected times. Clockwise from top left: At times $t = 0$ (top left), $t = 0.4$ (top right), $t = 0.8$ (middle right), $t = 1.2$ (middle left), $t = 1.6$ (bottom left) and $t = 2.0$ (bottom right). The colour represents the solution ϕ .

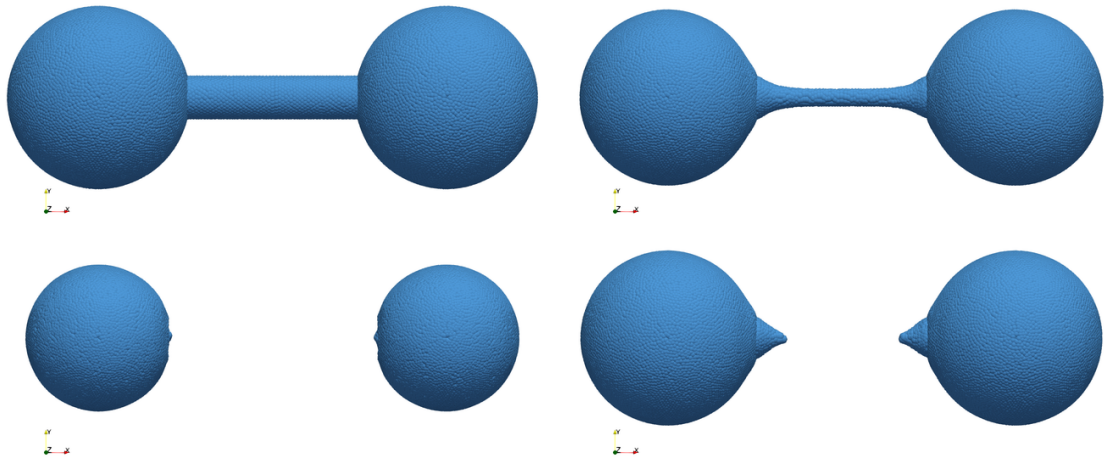


Figure 14: Mean curvature flow for a dumbbell shape: Clockwise from top left: At times $t = 0$ (top left), $t = 0.05$ (top right), $t = 0.06$ (bottom right), and $t = 0.2$ (bottom left).

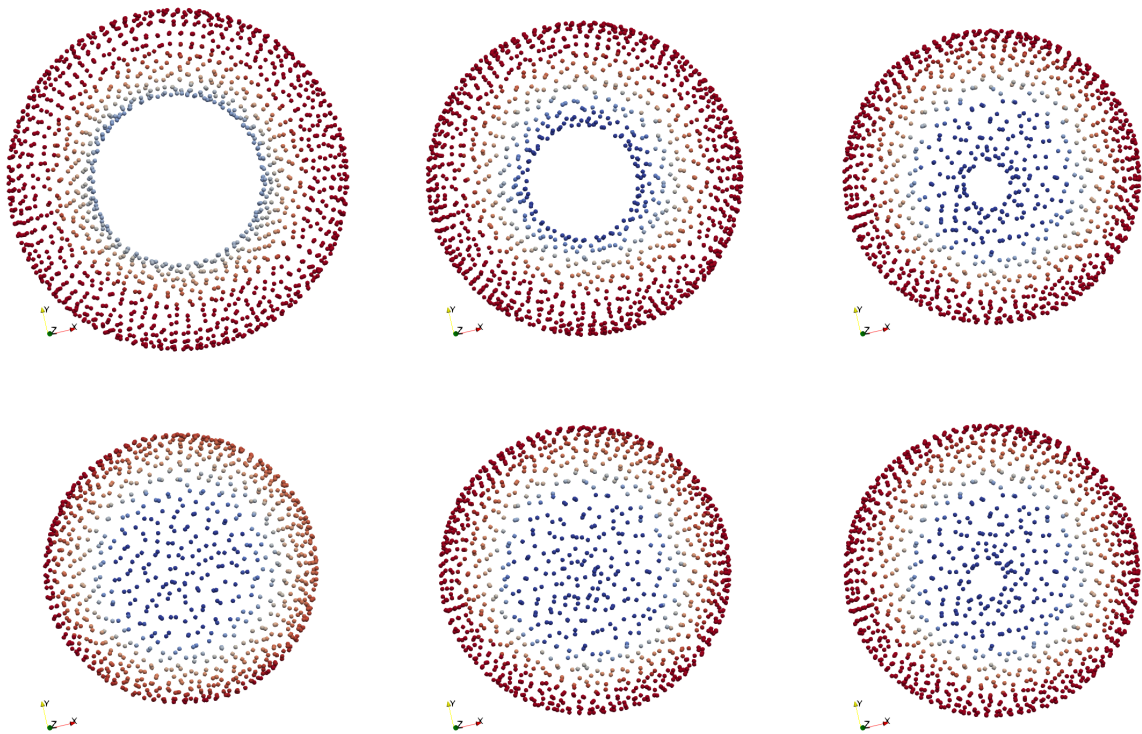


Figure 15: Averaged mean curvature flow for a torus: Clockwise from top left: At times $t = 0$ (top left), $t = 4$ (top center), $t = 6.8$ (top right), $t = 7.05$ (bottom right), $t = 7.55$ (bottom center) and $t = 10$ (bottom left). The points are coloured according to the curvature κ to make the results easier to visualize. The initial torus contracts and evolves to an ellipsoid.

contracts towards the center, resulting in a topological change from genus 1 to 0, after which the domain slowly evolves towards a sphere.

Both results show that the capability of the present method to capture topological change well. For both the dumbbell and the torus simulations, the contact handling algorithms introduced in Section 3.6 detect the topological change, with the delete contact enforced, and the simulations do not need to be restarted after the singularities arise, which is often the case.

An important point to note here is that to maintain stability, the values of κ have to be smoothed before the velocity is computed. This requirement of smoothing has also been reported by other authors [29]. The smoothing is done with a wide Gaussian smoothing kernel on the support domain

$$\tilde{\kappa}_i = \frac{\sum_{j \in S_i} \widetilde{W}_{ij} \kappa_j}{\sum_{j \in S_i} \widetilde{W}_{ij}}, \quad (40)$$

with weights

$$\widetilde{W}_{ij} = \exp\left(-\frac{\|\vec{x}_j - \vec{x}_i\|^2}{h_i^2}\right). \quad (41)$$

4.2.4. Wave Equation on an Evolving Surface

As the last experiment, we solve a wave equation on an evolving surface [36]:

$$\frac{D}{Dt} \frac{D\phi}{Dt} + \frac{D\phi}{Dt} \nabla_M \cdot \vec{v} = c^2 \Delta_M \phi + f, \quad (42)$$

where \vec{v} is the surface velocity, and $f = f(\phi, \vec{x}, t)$ is a forcing term. Eq. (42) is considered with initial conditions

$$\phi(\vec{x}, 0) = g_1(\vec{x}), \quad (43)$$

$$\frac{D\phi}{Dt}(\vec{x}, 0) = g_2(\vec{x}). \quad (44)$$

Numerical integration in each time step begins with a movement step according to Eq. (3), and is followed by

$$\frac{\phi^{(n+1)} - 2\phi^{(n)} + \phi^{(n-1)}}{(\Delta t)^2} + \frac{\phi^{(n+1)} - \phi^{(n)}}{\Delta t} \nabla_M \cdot \vec{v}^{(n+1)} = c^2 \Delta_M \phi^{(n+1)} + f^{(n)}. \quad (45)$$

The numerical differential operators are computed as explained in Section 4.1, and $\phi^{(-1)}$ is computed based on Eq. (44). We start by validating the scheme with a manufactured solution

$$\phi_{\text{exact}}(\vec{x}, t) = \sin(\omega t) xy. \quad (46)$$

The considered surface at time t is given by

$$x^2 + y^2 + z^2 = (r(t))^2, \quad (47)$$

with $r(t) = 1 + 0.5t$. The considered velocity field is $\vec{v} = 0.5\vec{n}$. The forcing term and initial conditions are determined such that ϕ_{exact} in Eq. (46) satisfies Eq. (42) with $c = 1$. Similar to earlier, we note that the following holds:

$$\nabla_M \cdot \vec{v} = \frac{1}{r(t)}, \quad (48)$$

$$\Delta_M \phi_{\text{exact}} = -\frac{6\phi_{\text{exact}}}{(r(t))^2}. \quad (49)$$

For a varying spatial discretization, convergence of the relative errors of the numerical solution as compared to the analytical solution are given in Table 4. The simulations use $\omega = \sqrt{6}$ in Eq.(46), and a time step of $\Delta t = h/20$. Convergence orders are measured with the number of points in the domain at initial time. The table shows that the numerical results match the manufactured solution. An experimental order of convergence of 2 is observed which matches the expectation given the use of second order monomials in the computation of numerical derivatives.

Table 4: Errors at $t = 1$ and convergence orders with h for the wave equation test case with a manufactured solution. h is the smoothing length, N is the number of points in the entire domain at the initial time, ϵ_2 is the relative error, and r is the order of convergence of ϵ_2 .

h	N	ϵ_2	r
0.4	480	7.88×10^{-3}	–
0.2	1 806	1.96×10^{-3}	2.1
0.1	7 446	4.90×10^{-4}	1.95
0.05	30 054	1.23×10^{-4}	1.98

To further emphasize the capability of the present method to handle complex surfaces with large deformations, we consider the wave equation on a deforming Armadillo, with no forcing term. The considered initial conditions are

$$\phi^{(0)} = -\cos(x) \sin(2y) \cos(z), \quad (50)$$

$$\phi^{(-1)} = -\cos(x - c\Delta t) \sin(2y - c\Delta t) \cos(z - c\Delta t). \quad (51)$$

Furthermore, we set $f = 0$, $c = 7.0$, $\Delta t = 0.01$, and $h = 0.2$ which corresponds to $N = 13\,828$ points at the initial time. The velocity is given by

$$\vec{v} = -0.3\phi\vec{n} + \begin{pmatrix} -0.5x \\ 0.25y \end{pmatrix} + \begin{pmatrix} -0.5\text{sign}(x) \\ 0 \\ 0 \end{pmatrix}. \quad (52)$$

The simulation results are shown in Figure 16. The Armadillo stretches along its length as the wave travels in all directions. The collapsing of the Armadillo along its width triggers the contact detection algorithms, and results in a topological change in the surface.

5. Conclusion

We presented a novel fully Lagrangian framework to solve PDEs on moving manifolds. A surface is discretized with a point cloud, with no mesh or surrounding particles to discretize the volume around it. The method can handle arbitrary movements in the surface including those not prescribed a-priori and those that cause large deformations in the manifold. Distortions in the point cloud as a result of the evolution of the surface are fixed with purely local considerations. Similar to the volumetric case, the ease of fixing distortion is one of the biggest advantages of the moving surface point cloud over the moving surface mesh. We further presented a robust algorithm for contact handling for surface point clouds, that also enables simulations with topological changes in the surface.

To solve PDEs on evolving-in-time surfaces, this fully Lagrangian framework was used with a meshfree Generalized Finite Difference Method to approximate numerical derivatives. The Lagrangian framework was validated with several numerical experiments with different types of motion. And the applicability of the overall scheme was illustrated with applications to advection diffusion reaction equations on evolving surfaces, wave equations on evolving surfaces, and curvature-based geometric flow. This work forms a foundation for a purely meshfree Lagrangian method for PDEs on moving interfaces, with lots of potential applications ranging from shell mechanics to flow on curved surfaces.

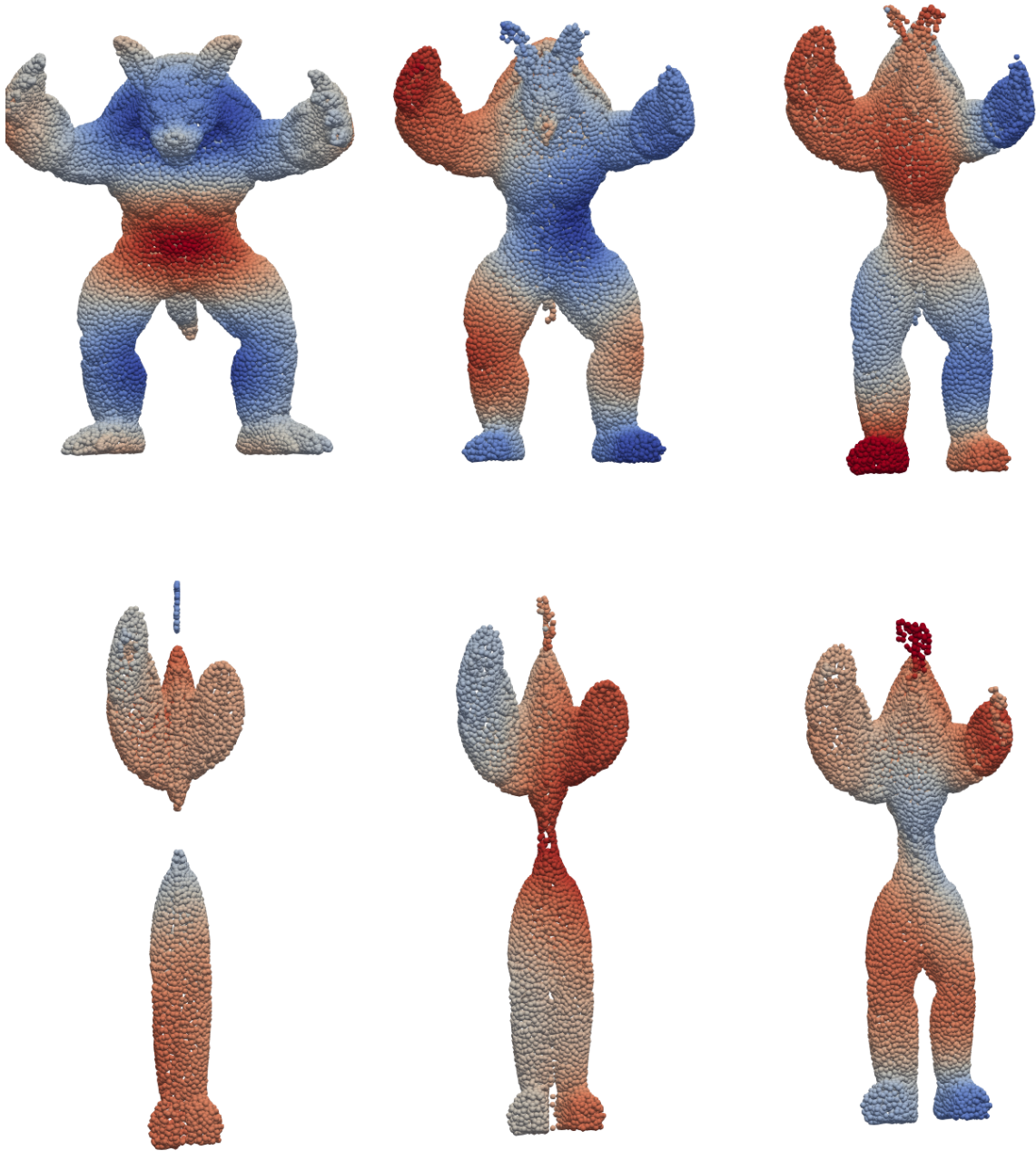


Figure 16: Wave equation on a deforming armadillo: Clockwise from top left: At times $t = 0$ (top left), $t = 0.24$ (top center), $t = 0.48$ (top right), $t = 0.72$ (bottom right), $t = 0.96$ (bottom center) and $t = 1.2$ (bottom left). The points are coloured according to the solution ϕ of the wave equation.

A limitation of the present work is that differential operators are computed on a stationary point cloud at the beginning of each time step. As a result, achieving higher order accuracy for the overall scheme could be quite problematic. A possible solution to this could be to use more accurate operator splitting, as done, for example, in [29]. Another improvement that needs to be explored is the correction of locations of points added to fill holes in the point cloud. In this paper, we considered the evolution of surfaces in both normal and tangential directions. However, evolution of particles within a (possibly stationary) surface was not considered. The extension to the present work to tackle the possibility of evolution within the manifold, for example for Lagrangian flow on a surface, will be considered in our future work.

Acknowledgements

The armadillo model is courtesy of the Stanford Computer Graphics Laboratory of Stanford University.

References

- [1] M. Accorsi, J. Leonard, R. Benney, and K. Stein. Structural modeling of parachute dynamics. *AIAA journal*, 38(1):139–146, 2000.
- [2] N. S. Atluri and T. Zhu. A new meshless local petrov-galerkin (mlpg) approach in computational mechanics. *Computational Mechanics*, 22(2):117–127, 1998.
- [3] S. Auer and R. Westermann. A semilagrangian closest point method for deforming surfaces. *Computer Graphics Forum*, 32(7):207–214, 2013.
- [4] M. Bergdorf, I. F. Sbalzarini, and P. Koumoutsakos. A lagrangian particle method for reaction–diffusion systems on deforming surfaces. *Journal of Mathematical Biology*, 61(5):649–663, Nov 2010.
- [5] H. Chen and J. Bishop. Delaunay triangulation for curved surfaces. *Meshing Roundtable*, pages 115–127, 1997.
- [6] S. Chen and J. Wu. Discrete conservation laws on evolving surfaces. *SIAM Journal on Scientific Computing*, 38(3):A1725–A1742, 2016.
- [7] T. H. Colding, W. P. Minicozzi, II, and E. K. Pedersen. Mean curvature flow. *Bull. Amer. Math. Soc. (N.S.)*, 52(2):297–333, 2015.
- [8] J. M. Domínguez, A. J. C. Crespo, M. Gómez-Gesteira, and J. C. Marongiu. Neighbour lists in smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 67(12):2026–2042, 2011.
- [9] C. Drumm, S. Tiwari, J. Kuhnert, and H.-J. Bart. Finite pointset method for simulation of the liquid - liquid flow field in an extractor. *Computers & Chemical Engineering*, 32(12):2946 – 2957, 2008.
- [10] G. Dziuk and C. M. Elliott. Finite elements on evolving surfaces. *IMA Journal of Numerical Analysis*, 27(2):262–292, 2007.
- [11] G. Dziuk and C. M. Elliott. Finite element methods for surface pdes. *Acta Numerica*, 22:289–396, 2013.
- [12] C. M. Elliott and B. Stinner. Modeling and computation of two phase geometric biomembranes using surface finite elements. *J. Comput. Phys.*, 229(18):6585–6612, Sept. 2010.
- [13] C. M. Elliott, B. Stinner, and C. Venkataraman. Modelling cell motility and chemotaxis with evolving surface finite elements. *Journal of The Royal Society Interface*, 9(76):3027–3044, 2012.
- [14] C.-M. Fan, C.-N. Chu, B. arler, and T.-H. Li. Numerical solutions of waves-current interactions by generalized finite difference method. *Engineering Analysis with Boundary Elements*, 2018.
- [15] Y. Farjoun and B. Seibold. Solving one dimensional scalar conservation laws by particle management. In M. Griebel and M. A. Schweitzer, editors, *Meshfree Methods for Partial Differential Equations IV*, pages 95–109, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [16] N. Flyer, G. B. Wright, and B. Fornberg. Radial basis function-generated finite differences: A mesh-free method for computational geosciences. In W. Freeden, M. Z. Nashed, and T. Sonar, editors, *Handbook of Geomathematics*, pages 1–30, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [17] E. J. Fuselier and G. B. Wright. A high-order kernel method for diffusion and reaction-diffusion equations on surfaces. *Journal of Scientific Computing*, 56(3):535–565, Sep 2013.
- [18] L. Gavete, F. Ureña, J. Benito, A. García, M. Ureña, and E. Saete. Solving second order non-linear elliptic partial differential equations using generalized finite difference method. *Journal of Computational and Applied Mathematics*, 318:378 – 387, 2017. Computational and Mathematical Methods in Science and Engineering CMMSE-2015.
- [19] P. A. Gilman. Magnetohydrodynamic shallow water equations for the solar tachocline. *The Astrophysical Journal Letters*, 544(1):L79, 2000.
- [20] M. A. Grayson. A short note on the evolution of a surface by its mean curvature. *Duke Math. J.*, 58(3):555–558, 06 1989.
- [21] D. Gueyffier, J. Li, A. Nadim, R. Scardovelli, and S. Zaleski. Volume-of-fluid interface tracking with smoothed surface stress methods for three-dimensional flows. *Journal of Computational Physics*, 152(2):423 – 456, 1999.
- [22] A. Hirth, A. Haufe, and L. Olovsson. Airbag simulation with ls-dyna past-present-future. In *Proceedings of the 6th European LS-DYNA Users Conference*, Gothenburg, Sweden, 2007.
- [23] A. Iske. On the construction of mass conservative and meshless adaptive particle advection methods. In V. M. A. Leitao, C. J. S. Alves, and C. Armando Duarte, editors, *Advances in Meshfree Techniques*, pages 169–186, Dordrecht, 2007. Springer Netherlands.

- [24] A. Jefferies, J. Kuhnert, L. Aschenbrenner, and U. Giffhorn. Finite pointset method for the simulation of a vehicle travelling through a body of water. In M. Griebel and A. M. Schweitzer, editors, *Meshfree Methods for Partial Differential Equations VII*, pages 205–221, Cham, 2015. Springer International Publishing.
- [25] H. Jin, A. J. Yezzi, and S. Soatto. Region-based segmentation on evolving surfaces with application to 3d reconstruction of shape and piecewise constant radiance. In T. Pajdla and J. Matas, editors, *Computer Vision - ECCV 2004*, pages 114–125, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [26] A. Katz and A. Jameson. Meshless scheme based on alignment constraints. *AIAA journal*, 48(11):2501–2511, 2010.
- [27] T. Kim, J. Tessendorf, and N. Thuerey. Closest point turbulence for liquid surfaces. *ACM Transactions on Graphics (TOG)*, 32(2):15, 2013.
- [28] H. Lamb. *Hydrodynamics*. Cambridge university press, 1993.
- [29] S. Leung, J. Lowengrub, and H. Zhao. A grid based particle method for solving partial differential equations on evolving surfaces and modeling high order geometrical motion. *Journal of Computational Physics*, 230(7):2540 – 2561, 2011.
- [30] S. Leung and H. Zhao. A grid based particle method for moving interface problems. *J. Comput. Phys.*, 228(8):2993–3024, May 2009.
- [31] S. Li, D. Qian, W. K. Liu, and T. Belytschko. A meshfree contact-detection algorithm. *Computer Methods in Applied Mechanics and Engineering*, 190(24):3271 – 3292, 2001.
- [32] Y. Li, X. Qi, and J. Kim. Direct discretization method for the cahn–hilliard equation on an evolving surface. *Journal of Scientific Computing*, May 2018.
- [33] J. Liang and H. Zhao. Solving partial differential equations on point clouds. *SIAM Journal on Scientific Computing*, 35(3):A1461–A1486, 2013.
- [34] G.-R. Liu. *Meshfree methods: moving beyond the finite element method*. Taylor & Francis, Boca Raton, FL, USA, 2009.
- [35] G.-R. Liu and G.-Y. Zhang. *Smoothed point interpolation methods: G space theory and weakened weak forms*. World Scientific, Singapore, 2013.
- [36] C. Lubich and D. Mansour. Variational discretization of wave equations on evolving surfaces. *Math. Comp.*, 84(292):513–542, 2015.
- [37] C. Lubich, D. Mansour, and C. Venkataraman. Backward difference time discretization of parabolic differential equations on evolving surfaces. *IMA Journal of Numerical Analysis*, 33(4):1365–1385, 2013.
- [38] M. Luo, C. G. Koh, W. Bai, and M. Gao. A particle method for two-phase flows with compressible air pocket. *International Journal for Numerical Methods in Engineering*, 108:695–721, Nov. 2016.
- [39] T. März and C. B. Macdonald. Calculus on surfaces with general closest point functions. *SIAM Journal on Numerical Analysis*, 50(6):3303–3328, 2012.
- [40] U. F. Mayer. A numerical scheme for moving boundary problems that are gradient flows for the area functional. *European Journal of Applied Mathematics*, 11(1):6180, 2000.
- [41] U. F. Mayer. A singular example for the averaged mean curvature flow. *Experiment. Math.*, 10(1):103–107, 2001.
- [42] I. Michel, S. M. I. Bathaeian, J. Kuhnert, D. Kolymbas, C.-H. Chen, I. Polymerou, C. Vrettos, and A. Becker. Meshfree generalized finite difference methods in soil mechanics—part ii: numerical results. *GEM - International Journal on Geomathematics*, 8(2):191–217, Nov 2017.
- [43] D. Mirzaei, R. Schaback, and M. Dehghan. On generalized moving least squares with diffuse derivatives. *IMA J. Numer. Anal.*, 32(3):983–1000, 2012.
- [44] N. J. Mitra and A. Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 322–328, New York, NY, USA, 2003. ACM.
- [45] A. Möller and J. Kuhnert. Simulation of the glass flow inside a floating process / Simulation de l'écoulement du verre dans le procédé float. *Revue Verre*, 13(5):28–30, 2007.
- [46] S. Nemaadjieu. *A stable and convergent O-method for general moving hypersurfaces*. CASA-report. Technische Universiteit Eindhoven, 2014.
- [47] D. T. Oanh, O. Davydov, and H. X. Phu. Adaptive rbf-fd method for elliptic problems with point singularities in 2d. *Applied Mathematics and Computation*, 313:474 – 497, 2017.
- [48] J. Onderik and R. Đurikovič. Efficient neighbor search for particle-based fluids. *Journal of the Applied Mathematics, Statistics and Informatics (JAMSI)*, 4(1):29–43, 2008.
- [49] A. Petras, L. Ling, C. Piret, and S. Ruuth. A least-squares implicit rbf-fd closest point method and applications to pdes on moving surfaces. *Journal of Computational Physics*, 381:146 – 161, 2019.
- [50] A. Petras and S. Ruuth. PDEs on moving surfaces via the closest point method and a modified grid based particle method. *Journal of Computational Physics*, 312:139 – 156, 2016.
- [51] F. Petronetto, A. Paiva, E. S. Helou, D. E. Stewart, and L. G. Nonato. Mesh-free discrete laplace-beltrami operator. *Comput. Graph. Forum*, 32(6):214–226, Sept. 2013.
- [52] E. O. Reséndiz-Flores, J. Kuhnert, and F. R. Saucedo-Zendejo. Application of a generalized finite difference method to mould filling process. *European Journal of Applied Mathematics*, page 120, 2017.
- [53] R. Samulyak, X. Wang, and H.-C. Chen. Lagrangian particle method for compressible fluid dynamics. *Journal of Computational Physics*, 362:1 – 19, 2018.
- [54] A. Sokolov, O. Davydov, and S. Turek. Numerical study of the rbf-fd level set based method for partial differential equations on evolving-in-time surfaces. Technical report, Fakultät für Mathematik, TU Dortmund, Nov. 2017. Ergebnisberichte des Instituts für Angewandte Mathematik, Nummer 579.
- [55] H. A. Stone. A simple derivation of the timedependent convectediffusion equation for surfactant transport along a deforming interface. *Physics of Fluids A: Fluid Dynamics*, 2(1):111–112, 1990.

- [56] P. Suchde and J. Kuhnert. A meshfree generalized finite difference method for surface PDEs. *Computers & Mathematics with Applications*, 2019.
- [57] P. Suchde and J. Kuhnert. Point cloud movement for fully lagrangian meshfree methods. *Journal of Computational and Applied Mathematics*, 340:89 – 100, 2018.
- [58] P. Suchde, J. Kuhnert, S. Schröder, and A. Klar. A flux conserving meshfree method for conservation laws. *International Journal for Numerical Methods in Engineering*, 112(3):238–256, 2017.
- [59] M. Ureña, J. J. Benito, F. Ureña, A. García, L. Gavete, and L. Benito. Adaptive strategies to improve the application of the generalized finite differences method in 2d and 3d. *Mathematical Methods in the Applied Sciences*, 2017.
- [60] H. A. van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.
- [61] P. Wriggers. *Computational Contact Mechanics*. Springer, 2006.