Multi-level Partition of Unity on Differentiable Moving Particles

JINJIN HE, Dartmouth College, USA
TAIYUAN ZHANG, Dartmouth College, USA
HIROKI KOBAYASHI, Toyota Central RD Labs., Inc., Japan
ATSUSHI KAWAMOTO, Toyota Central RD Labs., Inc., Japan
YUQING ZHOU, Toyota Research Institute of North America, United States of America
TSUYOSHI NOMURA, Toyota Central RD Labs., Inc., Japan
BO ZHU, Georgia Institute of Technology, United States of America

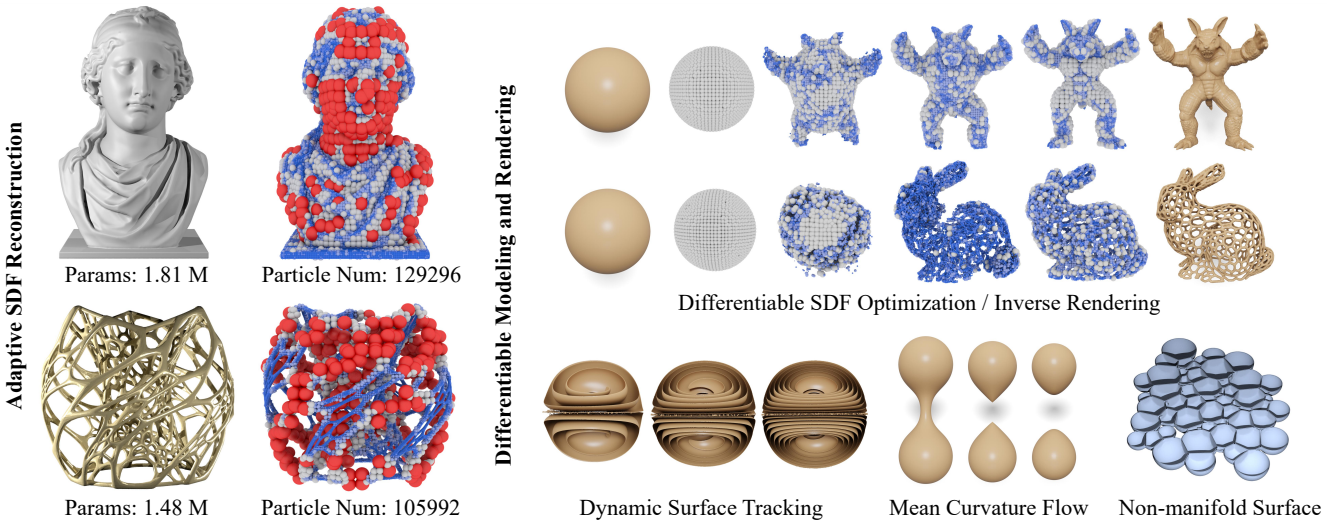

Fig. 1. We present a novel adaptive implicit representation based on differentiable particles with Multi-level Partition of Unity (MPU). Our method represents objects with complex geometries and topologies, with applications in adaptive SDF reconstruction, inverse rendering, and dynamic interface tracing.

We introduce a differentiable moving particle representation based on the multi-level partition of unity (MPU) to represent dynamic implicit geometries. At the core of our representation are two groups of particles, named feature particles and sample particles, which can move in space and produce dynamic surfaces according to external velocity fields or optimization gradients. These two particle groups iteratively guide and correct each other by alternating their roles as inputs and outputs. Each feature particle carries a set of coefficients for a local quadratic patch. These particle patches are assembled with partition-of-unity weights to derive a continuous implicit global shape. Each sampling particle carries its position and orientation, serving as dense surface samples for optimization tasks. Based on these moving particles, we develop a fully differentiable framework to infer and evolve highly detailed implicit geometries, enhanced by a multi-level background grid for particle adaptivity, across different inverse tasks. We demonstrated the efficacy of our representation through various benchmark comparisons with state-of-the-art neural representations, achieving lower memory consumption, fewer training iterations, and orders of magnitude higher accuracy in handling topologically complex objects and dynamic tracking tasks.

CCS Concepts: • **Computing methodologies** → **Point-based models**;

Additional Key Words and Phrases: Multi-Level Partition of Unity, Differentiable Particles, Inverse Rendering, Dynamic Interface Tracking

ACM Reference Format:
Jinjin He, Taiyuan Zhang, Hiroki Kobayashi, Atsushi Kawamoto, Yuqing Zhou, Tsuyoshi Nomura, and Bo Zhu. 2024. Multi-level Partition of Unity on Differentiable Moving Particles. *ACM Trans. Graph.* 43, 6 (December 2024), 21 pages. https://doi.org/10.1145/3687989

Authors' addresses: Jinjin He, jinjin.he.gr@dartmouth.edu, Dartmouth College, Hanover, New Hampshire, USA; Taiyuan Zhang, taiyuan.zhang.gr@dartmouth.edu, Dartmouth College, Hanover, New Hampshire, USA; Hiroki Kobayashi, hiroki.kobayashi@mosk.tytlabs.co.jp, Toyota Central RD Labs., Inc., Nagakute, Aichi, Japan; Atsushi Kawamoto, atskwmt@mosk.tytlabs.co.jp, Toyota Central RD Labs., Inc., Nagakute, Aichi, Japan; Yuqing Zhou, yuqing.zhou@toyota.com, Toyota Research Institute of North America, Ann Arbor, Michigan, United States of America; Tsuyoshi Nomura, nomu2@mosk.tytlabs.co.jp, Toyota Central RD Labs., Inc., Nagakute, Aichi, Japan; Bo Zhu, bo.zhu@gatech.edu, Georgia Institute of Technology, Atlanta, Georgia, United States of America.

0730-0301/2024/12-ART $15.00
https://doi.org/10.1145/3687989

1 INTRODUCTION

Developing efficient data structures and representations to characterize dynamic implicit geometries is one of the central research topics in computer graphics and computational physics. Dating back

to earlier years, the introduction of level sets [Osher and Sethian 1988] and adaptively sampled distance fields [Frisken et al. 2000] initiated the adoption of implicit signed distance functions (SDF) to feature complex object geometries and capture their dynamic motions. Traditionally, these SDF fields were discretized on a background grid to characterize the implicit geometry, later enhanced with adaptive techniques such as octrees [Frisken et al. 2000], and more recently, equipped with neural representations like instant neural graphics primitives (I-NGP) [Müller et al. 2022]. One of the main limitations of a grid-based representation (or a hybrid neural representation) is its inherent *isotropy*, which makes it difficult to economically depict anisotropic features such as very small furrows, thin beams and filaments, and non-manifold foam structures.

Particle-based approaches offer a fresh perspective for addressing the challenges related to complex and dynamic implicit geometry. Thanks to the Lagrangian nature of particles, particle-based representations exhibit inherent potential to accommodate spatial adaptivity and temporal coherence. This potential has been demonstrated by recent groundbreaking work in 3D Gaussian splatting [Kerbl et al. 2023] and their dynamic extensions [Luiten et al. 2024], which focus on reconstructing (particularly anisotropic) radiance fields based on spherical harmonics stored on a set of moving particles. In the realm of implicit geometry representation, particles have historically played an important role. On the one hand, extensive literature has been devoted to point-set surface reconstruction in the 3D geometry reconstruction, including notable works such as Moving Least Squares (MLS) [Levin 2004], Point Set Surface (PSS) [Alexa et al. 2001], and Algebraic Point Set Surface (APSS) [Guennebaud and Gross 2007]; on the other hand, the particle level set (PLS) method [Hieber and Koumoutsakos 2005] and its many variations (e.g., Adaptive PLS [Ianniello and Di Mascio 2010], Oriented PLS [Vartdal and Bøckmann 2013] and One-layer PLS [Zhao et al. 2018]) have been crucial for tracking the evolution of dynamic implicit geometry in various physics simulation applications. Particles play different roles in different cases, either as material points that carry physical quantities or as correction information carriers used to adjust the SDF on a background grid.

In this paper, we further explore the potential of using particles to facilitate implicit, dynamic geometry representation by introducing a moving particle representation based on the multi-level partition of unity (MPU) scheme [Ohtake et al. 2005]. The key idea of our representation is to devise two particle systems, including a sparse set of feature particles and a dense set of sample particles, which can iteratively guide each other during the dynamic evolution of an interface driven by external velocities or optimization gradients. These two groups of particles play different roles during optimization. On the one hand, each feature particle carries a quadratic surface patch, which is then stitched together using MPU coefficients to form a continuous and smooth implicit surface. On the other hand, each sample particle carries its position and orientation; many sample particles serve as data input to optimize the position and parameters of the feature particles for an implicit MPU surface.

For implicit surface construction, tracking, and optimization tasks, we combine moving particles and MPU to leverage the computational merits from both sides. First, using particles inherently enables inverse differentiation, thanks to the ease of calculating the derivative of each particle's position as well as its quadratic coefficients with respect to optimization objectives such as rendering pixel colors or target distance field values. This benefit is particularly attractive when compared to a neural approach (e.g., I-NGP [Müller et al. 2022]), for the ease of implementation and the efficiency of *derivative calculation*. Second, using MPU allows each particle to characterize high-order geometries that are more complex than a linear approximation (e.g., SDF value). This local expressiveness enables our representation to feature complex surfaces even with a very sparse usage of particles in space (in contrast to a dense narrow band of particles such as in particle level sets), opening up further designs of *surface adaptivity* that remain scarcely explored in the implicit geometry community.

We carried out a comprehensive set of benchmark tests to validate the performance of both our particle MPU representation and our differentiable pipeline. Compared to other state-of-the-art methods, our approach demonstrates efficacy with low computational cost (implicit surface reconstruction), robustness (differentiable SDF optimization), rapid training (inverse rendering), and heightened expressiveness (reconstructed/inversed surface quality). Figure 2 illustrates a differentiable optimization process using our pipeline.

Our main contributions can be summarized as:

(1) A novel dual particle representation consisting of feature particles and sample particles based on MPU for adaptive implicit geometry representation;
(2) A fully differentiable framework based on moving particles to accommodate inverse rendering, modeling, and dynamic tracking applications;
(3) A particle reseeding/deletion/projection mechanism to enable surface adaptivity;
(4) An extension to handle non-manifold geometries with unsigned distance fields.

2 RELATED WORK

Implicit Geometry Representations. Implicit geometry representations such as Moving Least Squares (MLS) Surfaces and Signed Distance Functions (SDF) have been widely studied in computer graphics [Bloomenthal and Bajaj 1997]. Due to their expensive visualization, especially for complex geometries, developing efficient adaptive implicit representations has gained much attention. Leveraging piecewise quadratic functions and the partition of unity, Ohtake et al. [2005] introduced a novel shape representation. Adaptivity in moving least squares surfaces was explored by Dey and Sun [2005] and Huang et al. [2010]. For adaptive SDF generation and isosurface meshing, notable contributions have been seen by Frisken et al. [2000], Akkouche and Galin [2001], Varadhan et al. [2004], Azernikov and Fischer [2005], Koschier et al. [2017], etc. In recent years, machine learning, particularly deep learning, has garnered attention for its potential to represent 3D shapes effectively. DeepSDF [Park et al. 2019a], DeepIMLS [Liu et al. 2021], and NGLoD [Takikawa et al. 2021] demonstrated the capacity to learn implicit representations from SDF and MLS surfaces. The work by Müller et al. [2022] innovatively reduced the computational cost of neural networks by augmenting a small neural network with a multi-resolution hash table of trainable feature vectors. Chen et al. [2023] departed from

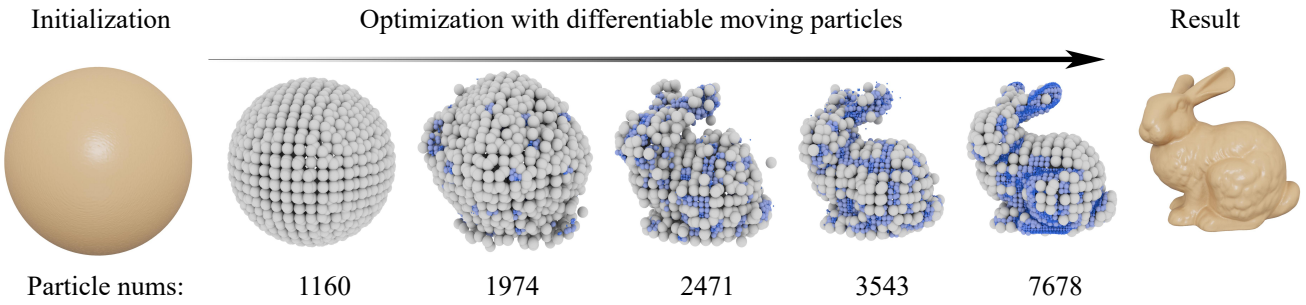

Fig. 2. This figure demonstrates the evolution of feature particles during the differentiable optimization process from the initial sphere to the target mesh with five intermediate states. Particles with deeper color have smaller radii that characterize smaller patches/regions.

fixed neural features on grid nodes and instead employed general radial bases with flexible kernel position and shape. Our method employs particles as the basic unit to represent implicit geometries to achieve compute and memory efficiency in dynamic tasks.

Multi-level Partition of Unity. The MPU method [Ohtake et al. 2005] can reconstruct an implicit surface with locally estimated quadratic functions given an oriented point set. With adaptive octree-based partition, points are approximated by quadric functions to capture local features and use the partition of unity to obtain global implicit functions. Attaching the estimated primitives to points instead of cells, Xiao [2011] introduced multi-level partition of unity algebraic point set surfaces with an efficient projection operator. Li et al. [2014] presented a new robust MPU method with a new error metric, controlling the approximation error between the surface and vertices of the triangle. Hu et al. [2010] presented a facial localization method using MPU implicit. Combining MPU and dynamic particle systems, Chen et al. [2018] can generate smooth and adaptive meshes from medical image datasets. Besides the geometry-related tasks, MPU can be applied to solve PDEs by developing multi-level particle partition of unity method as a mesh-free generalization of FEM [Schweitzer 2009, 2011]. Our proposed representation can be viewed as a moving-particle adaptation of the traditional MPU. Projecting and constraining feature particles onto the surface demonstrates advantages in adapting surface features and differentiating inverse modeling and rendering.

Particle-based Methods. Particle-based methods have always been highly valued due to flexibility compared to grid-based methods. Taking advantage of particles, various approaches achieved better results compared to the original grid-based structure like PLS methods [Enright et al. 2002; Ianniello and Di Mascio 2010], MPU-APSS [Xiao 2011], particle-based anisotropic surface meshing [Zhong et al. 2013], Particle-NeRF [Abou-Chakra et al. 2023], NeuralRBF [Chen et al. 2023], etc. Leveraging point set, Pauly et al. [2006] devised a point-based multi-scale surface representation, Duranleau et al. [2008] introduced a multi-resolution representation for point-set surfaces which is slightly larger than the original point set, Huang et al. [2019] can reconstruct an implicit surface from an un-oriented point set using global smoothness energy, Mercier et al. [2022] presented a simple, fast, and smooth scheme to approximate Algebraic Point Set Surfaces using non-compact kernels. With the great success of 3D Gaussian Splatting [Kerbl et al. 2023], particles carrying feature vectors have shown huge potential to obtain improvement on multiple tasks Jiang et al. [2024]; Xie et al. [2023]. Beyond geometry representations, particle-based methods have shown their advantages in simulation Becker and Teschner [2007]; Bell et al. [2005]; Deng et al. [2022] and rendering [Sakamoto et al. 2007; Tanaka et al. 2012]. Our method further squeezes the potential of the particle-based structure by introducing radius adaptivity and parameters to characterize complicated geometry.

Differentiable Geometry and Rendering. In recent times, there has been a growing interest in the research community in making traditional tasks differentiable, leveraging the success of machine learning. In the realm of differentiable isosurface extraction, MeshSDF [Remelli et al. 2020], Shape as Points [Peng et al. 2021], and FlexiCubes [Shen et al. 2023] improved isosurface representation for gradient-based optimizing unknown meshes concerning geometric, visual, or even physical attributes. Sellán et al. [2023] treated each SDF sample as a spherical region and employed an energy-based approach for under-sampling SDF reconstruction task. In the domain of differentiable rendering, Soft rasterizer [Liu et al. 2019], SDFDiff [Jiang et al. 2020], differentiable surface rendering [Cole et al. 2021], differentiable surface splatting [Yifan et al. 2019] have explored various differentiable rendering algorithms. Exploring other differentiable geometry processing tasks, Rakotosaona et al. [2021] presented an approach for optimizing triangle meshes both in 2D and on surfaces. Noteworthy work on surface evolution by [Mehta et al. 2022, 2023; Novello et al. 2023] delves into the use of smooth neural networks for modeling dynamic variations of implicit surfaces governed by the level set equation. Our method aligns with this trend and specifically focuses on surface evolution and dynamic surface tracking, capitalizing on the advantages offered by a particle-based representation in dynamic scenarios.

3 BACKGROUND

Implicit Surface. We define an implicit geometry S as the zero level set of an implicit function $f(\mathbf{x})$ that takes a point $\mathbf{x}$ as input and returns its signed distance value.

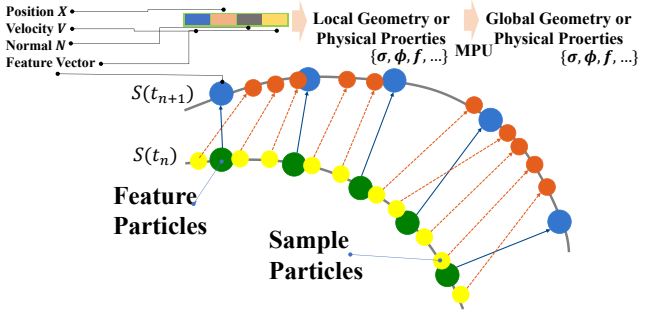

Fig. 3. Feature particles carry parameters encoding properties through MPU, while sample particles densely capture surface details. Parameters(feature vectors) carried by each feature particle encode local surface information of MPU. $S(t_n)$ and $S(t_{n+1})$ illustrate the evolution of particles driven by gradient flow from time step n to $n+1$.

Partition of Unity. We define an implicit geometry with Partition of Unity [Ohtake et al. 2005] by integrating a set of locally defined patches $\{\Omega_i\}_{i=1}^n$ into a global approximation Ω. Each patch Ω_i has a center $\mathbf{c}_i$ and a radius h_i. Specifically, we approximate $f(x)$ defined on Ω as a linear combination of implicit functions $f_i(x)$ defined on each local patch Ω_i with weights φ_i:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \varphi_i(\mathbf{x}) f_i(\mathbf{x}). \tag{1}$$

The weights satisfy the partition of unity property $\sum_{i=1}^n \varphi_i = 1$. Following the Shepard's method [Franke and Nielson 1980], given a set of non-negative compactly supported functions $\{w_i\}_{i=1}^n$, the partition of unity functions φ_i can be constructed as, $\varphi_i(\mathbf{x}) = \dfrac{w_i(\mathbf{x})}{\sum_{j=1}^n w_j(\mathbf{x})}$. We use the quadratic B-spline functions to calculate each $\{w_i\}$ as:

$$w_i(\mathbf{x}) = \begin{cases} 1 - 3r_i(\mathbf{x})^2 & , 0 \le r_i(\mathbf{x}) \le \frac{1}{3} \\ \frac{3}{2}(1 - r_i(\mathbf{x}))^2 & , \frac{1}{3} \le r_i(\mathbf{x}) \le 1 \\ 0 & , \text{otherwise} \end{cases} \tag{2}$$
$$r_i(\mathbf{x}) = \frac{|\mathbf{x} - \mathbf{c}_i|}{h_i},$$

where $\mathbf{c}_i$ and h_i are the center and radius of each patch Ω_i.

4 GEOMETRIC REPRESENTATION

4.1 Data Structure

Our geometric representation consists of two sets of particles — **feature particles** for patch representation and **sample particles** for patch sampling. Firstly, we employ a set of feature particles $\mathcal{P}_F$ with variable radii that are sparsely sampled on the target surface. Each feature particle characterizes a local quadratic patch with 14 parameters(10 quadratic parameters + 3 position + 1 radius). Secondly, we utilize another set of oriented sample particles $\mathcal{P}_S$ that are densely sampled on the target surface. Next, we will elaborate on their definition and construction.

Feature Particles. We employ a sparse set of feature particles $\mathcal{P}_F$ to represent moving patches around an implicit surface. Each feature particle carries a particle center $\mathbf{c} \in \mathbb{R}^3$, a support radius $\mathbf{h}$, and a set of parameters $\boldsymbol{\beta} = [\beta_0, \beta_1, \cdots, \beta_l]^\top$ defining the local quadratic patch. In particular, β is used to define a quadratic polynomial as:

$$\mathcal{F}(\mathbf{x}) = \mathbf{b}(\mathbf{x} - \mathbf{c})^\top \boldsymbol{\beta}, \tag{3}$$

where $\mathbf{b}(\mathbf{x})$ serves as the basis polynomial for the quadratic surface. In our problem, we define $\mathbf{b}$ as $\mathbf{b}(\mathbf{x}) = \left[x^2, y^2, z^2, xy, yz, zx, x, y, z, 1\right]^\top$ with $\boldsymbol{\beta}$ as a 10-dimensional vector. Here, the function $\mathcal{F}$ is used as f_i to define the local quadratic patch in Eq.(1). Eq.(3) can be further used to calculate the normal vector of the local patch as:

$$\mathbf{n} = \frac{\partial \mathcal{F}}{\partial \mathbf{x}} = \frac{\partial \mathbf{b}^\top}{\partial \mathbf{x}} \beta. \tag{4}$$

Similarly, we can also derive the surface curvature as:

$$\kappa = \nabla \cdot \frac{\nabla \mathbf{b}^\top \beta}{|\nabla \mathbf{b}^\top \beta|}. \tag{5}$$

The detailed derivation is provided in Appendix D.

Sample Particles. We maintain a dense set of sample particles on the implicit surface to facilitate surface approximation and surface evolution. Each sample particle carries a position and a normal vector, which can be taken as input (for reconstruction tasks) or calculated using Eq. (3) and Eq. (4) (for dynamic tracking or inverse modeling tasks).

Feature-Sample Particle Interaction. Feature and sample particles alternately guide and correct each other in various tasks. On the one hand, sample particles serve as inputs for implicit surface reconstruction, resulting in the construction of feature particles as outputs. On the other hand, the implicit MPU surface produced by feature particles acts as the reference surface, guiding the generation of new sample particles in space. This process is illustrated in Figure 3. This mutual-correction mechanism is particularly important in tasks involving dynamic surfaces, such as interface tracking or inverse modeling, which we will elaborate on in Sec. 6 and Sec. 7.

4.2 Geometric Fitting

With the feature particles and sample particles defined in Sec. 4.1, we describe the geometric fitting process to calculate the parameters carried on each feature particle by taking the sample particles as input. In particular, we will optimize both the *parameters* and the *locations* of each feature particle to fit an implicit geometry sampled by sample particles. We describe the processes as below.

Parameter Fitting. We calculate the values of β for each feature particle by minimizing an objective $\mathcal{L}_{\boldsymbol{\beta}}$ as:

$$\min_{\boldsymbol{\beta}} \mathcal{L}_{\boldsymbol{\beta}} = \frac{1}{\sum_{\mathbf{p} \in \mathcal{N}} w(\mathbf{p})} \sum_{\mathbf{p} \in \mathcal{N}} w(\mathbf{p}) f(\mathbf{p})^2 + \frac{1}{m} \sum_{i=1}^{m} \left(f(\mathbf{q}_i) - D(\mathbf{q}_i)\right)^2, \tag{6}$$

where $\mathbf{q}_i$ represents training data points with signed distance values interpolated from sample particles, $D(\mathbf{q}_i)$ denotes the distance function interpolated from the sample particles and their normals, and m is the total number of training data points. The $\mathbf{p}$ are neighboring sample particles around the feature particle and $\mathcal{N}$ denotes the set

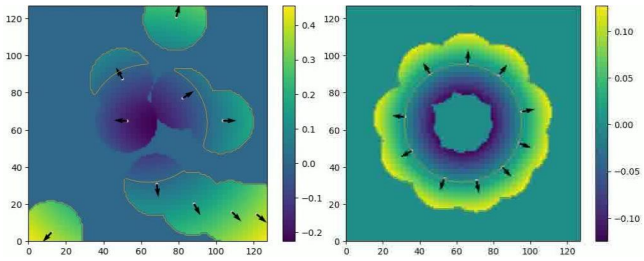

Fig. 4. We present the optimization results minimizing $\mathcal{L}$, with the target function being a circle in 2D. The contours represent the implicit surface and the boundary of feature particles. **Left**: randomly generated feature particles. **Right**: optimized feature particles by minimizing Eq. 8.

of neighboring sample particles for the feature particle. Details on obtaining the training data point $\mathbf{q}_i$ and the distance function are provided in Appendix B. The first term of Eq. (6) aims to optimize the function's value to 0 at each sample particle location, while the second term seeks to make the function's value at the training point locations converge to their respective distance values.

Position Fitting. In addition to polynomial fitting, we aim to keep the center of each $P_{\mathcal{F}}$ as close to the surface as possible to represent surface details and use fewer particles in the space. To achieve this, we optimize $\mathbf{c}$ for each feature particle by minimizing the objective:

$$\min_{\mathbf{c}} \mathcal{L}_{\mathbf{c}} = \sum_{i=1}^{N_{\mathcal{F}}} |f_i(\mathbf{c}_i)|^2, \tag{7}$$

this loss function enables each feature particle's position to converge near the surface where $f_i(\mathbf{c}_i) = 0$.

Total Loss. Thus, given input sample particles, we define the loss as the combination Eq. (6) and Eq. (7) to optimize the feature particles' parameters and positions:

$$\min_{\boldsymbol{\beta},\mathbf{c}} \mathcal{L} = \mathcal{L}_{\boldsymbol{\beta}} + \mathcal{L}_{\mathbf{c}}. \tag{8}$$

Optimization. We minimize Eq. (8) by separately optimizing Eq. (6) using the Weighted Least Squares (WLS) method [Nealen 2004] and Eq. (7) using the Newton projection method. We refer readers to Appendix C for more implementation details of WLS. For Newton projection, we take the following iteration for each feature particle until $f_i(\mathbf{c}_i) < 0.01\Delta\mathbf{c}_i$:

$$\mathbf{c}_i^{t+1} = \mathbf{c}_i^t - f_i(\mathbf{c}_i^t)\frac{\nabla f_i(\mathbf{c}_i^t)}{|\nabla f_i(\mathbf{c}_i^t)|}. \tag{9}$$

The local surface coefficients β of the feature particles are updated after each projection iteration.

In sum, we can minimize Eq. (8) by alternating between one-step weighted least squares and one-step Newton projection. In Figure 4, we randomly generate feature particles and obtain optimized $\mathbf{c}$ and $\boldsymbol{\beta}$ feature particles using the described method.

5 ADAPTIVITY

In our dynamic tracking and inverse modeling pipelines, we regularly reinitialize the position of feature particles. For other tasks, such as surface reconstruction, the reinitialization happens at the beginning of the algorithm. During reinitialization, we utilize a sparse adaptive grid [Setaluri et al. 2014] to facilitate the placement of feature particles with adaptive radii. Differing from a traditional MPU [Ohtake et al. 2005], where the octree serves as data storage, in our case, we use the octree as an adaptive spatial discretization to guide the generation of particles (similar ideas can be seen in Mercier et al. [2022]). In particular, we use the background grid as an auxiliary structure *during the reinitialization stage only* to place feature particles and maintain their distribution, ensuring that every sample particle is covered by at least one feature particle. We will elaborate the details of reinitialization as follows.

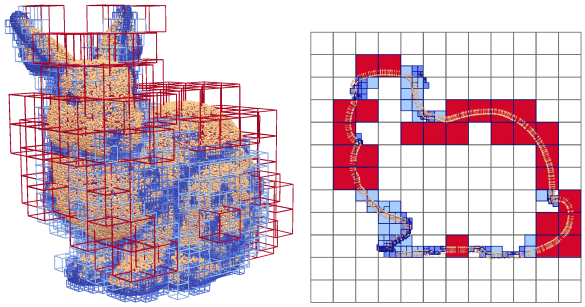

Fig. 5. Sparse grid activation is based on the presence of sample particles and surface curvature. **Left**: 3D visualization. **Right**: 2D slice. Activated cells are colored according to their layer index.

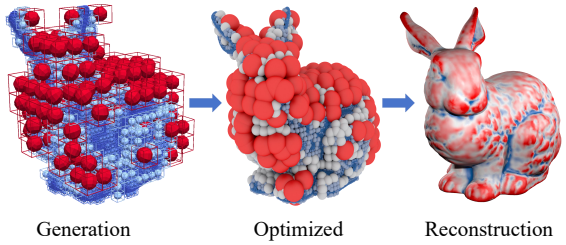

Fig. 6. Feature particles are generated according to the sparse grid and then projected to the surface. After an additional fixing step, the resulting surface can be reconstructed. **Left**: generated feature particles at the cell centers of the sparse grid. **Middle**: projected and fixed feature particles. The particles' radii are represented by a color gradient from red to blue, corresponding to large to small sizes. **Right**: reconstructed surface colored by curvature calculated from feature particles. The curvature increases as the color shifts toward red.

5.1 Feature Particle Reinitialization

Our feature particle reinitialization algorithm consists of two steps: *grid cell activation* and *particle generation*. First, we construct a hierarchy of sparse grids that adaptively cover the current implicit surface sampled by the sample particles. Our adaptive sparse grid has a structure similar to an Octree, and only the cells near the surface are activated. Like Octree, we activate cells with varying radii and layer them according to their radius. For example, cells with the largest radius form the coarsest layer, while those with

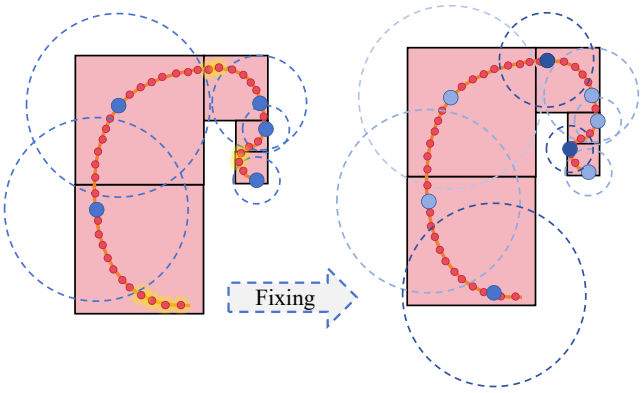

Fig. 7. By examining each activated cell, we add feature particles (colored by deep blue after fixing) with corresponding grid radius to fix holes if there are any isolated sample particles (colored by yellow).

the smallest radius form the finest layer. Then, we generate feature particles with adaptive radii based on the activated cells in the grid hierarchy.

ALGORITHM 1: Sparse gird activation

```
Input    : Sample Particles p, Curvature Carried κ_p, Threshold ε_0,
           Sparse Grid G, Finest Grid Number L_0
Variable : Current Layer Number L
L ← 1
while L ≤ L_0 do
    foreach p do
        Get grid G^L_{p_i} contains p_i
        if κ_{p_i} ≥ ε_0 then
            if L == 1 then
                Activate G^L_{p_i}
            end
            Get coarser gird G^{L-1}_{p_i}
            foreach G^L_c ∈ G^{L-1}_{p_i}'s children do
                Activate G^L_c
            end
        end
    end
    L ← L + 1
end
foreach G do
    if G_i not contain a p then
        Deactivate G_i
    end
    if L_{G_i} < L_0 then
        if Any G_c^{L_{G_i}+1} ∈ G_i's children is activated then
            Deactivate G_i
        end
    end
end
```

Grid Cell Activation. For the grid cell activation stage, assuming we are provided with a set of sample particles $\mathcal{P}_s$ sampled from an initial surface or the most recently evolved surface, the sparse grid hierarchy is constructed based on these sample particles. To address space efficiency concerns in a more adaptive manner, our auxiliary sparse grid is constructed considering both the positions of the sample particles and the surface's curvature on these points. The curvature of sample particles can be computed from the input mesh or our representation. We demonstrate how to calculate curvature from our feature particles in Appendix D.

Our activation algorithm iterates from coarse to fine grid layers. The second panel of Fig. 8 illustrates this process. Starting with the coarsest layer, we check each grid cell to determine if it contains sample particles and whether the maximum surface curvature carried on each sample particle in this cell exceeds a specified threshold ε_0, $max_{p_i \in node} \kappa_{p_i} > \varepsilon_0$. If both conditions are met, we activate the cell and the surrounding cells from the same layer that are located inside the coarser grid layer above. This process repeats iteratively until the final fine layer is reached. After checking and activating all grid cells in the current layer, we deactivate the parent grids from the coarser layer above that contains activated cells in the current layer. Also, during the iteration, some cells that do not contain a sample particle may become activated. We will deactivate those cells. We show the pseudo-code in Algorithm. 1.

A grid cell activation example is shown in Fig. 5, the sparse grid cell is activated by sample particles, and from the 2D Slice we see that our activation is both sparse in space and surface which is different from an octree in Gibou's survey [Gibou et al. 2018] and particle level set [Enright et al. 2002] which are sparse in space but dense on the surface.

Particle Generation. Upon activating the sparse grid cell, we generate a feature particle with a radius of $h_f = \alpha h_{cell}$ within each cell, where h_{cell} is the length of the grid, and α is a scaling variable that we set it to 1.15 in the experiments. During the initialization or re-initialization process, we assign a layer number to each feature particle, indicating the cell layer from which the feature particle is generated. Subsequently, we proceed feature particle fitting with a fixing step which will be discussed later.

5.2 Hole Fixing

We conduct a hole-fixing step after feature particle fitting to prevent holes on the surface due to the insufficient number of feature particles in local places. In particular, we will inspect each activated grid cell to identify any sample particles that are not contained within a feature particle. Subsequently, we will calculate the average position of the miss-contained sample particles within this cell and add a new feature particle at this position with a radius equal to half the cell's width. We iterate this process until every sample particle is contained by at least one feature particle. Fig. 6 illustrates the generation of feature particles and the resulting projection. The right of Fig. 6 shows the reconstruction result and colored curvature we compute using MPU.

So far, we have successfully obtained adaptive feature particles to reconstruct the implicit surface based on fixed sample particle inputs. We outline our implicit surface reconstruction in Algorithm 2. Figure 9 shows the results of our implicit surface reconstruction and the particle view (see Sec. 9.1 for more details). Next, we will discuss our optimization pipeline with particle differentiation.

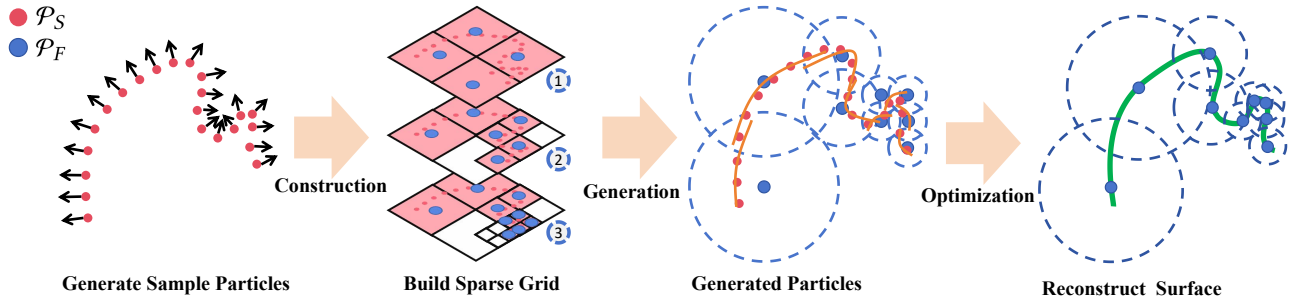

Fig. 8. The surface reconstruction pipeline of our adaptive implicit representation. After obtaining sample particles, we first construct a sparse grid using local curvature and position carried by particles. We then generate feature particles from the sparse grid and optimize their parameters.

Fig. 9. Our Implicit surface reconstruction results of different shapes , different colors of feature particles represent different radii. The figure demonstrates that our reconstructed structure is adaptive regarding both its spatial and surface details, capable of representing complex topologies and intricate surfaces. The numbers of feature particles used to reconstruct these shapes vary from 1k to 30k.

6 DIFFERENTIATION

In differentiable optimization tasks, our objective often revolves around minimizing a task-specific loss function, denoted as $\mathcal{L}_{\text{task}}$. As shown in Fig. 10, throughout optimization, we generate an output from the implicit surface based on our representation, corresponding to the specified task. Subsequently, we evaluate the loss function by comparing this output with the ground truth. Then, we back-propagate gradients from $\mathcal{L}_{\text{task}}$ to the underlying implicit surface represented by our feature particles.

6.1 Gradient Flow

Mathematically, we evaluate the equation $\frac{\partial \mathcal{L}_{task}}{\partial \theta} = \sum \frac{\partial \mathcal{L}_{task}}{\partial \mathbf{X}} \frac{\partial \mathbf{X}}{\partial \theta}$, where $\mathbf{X}$ is the output 3D points with specific information from our implicit surface according to the task. For example, it could be pixels transformed into 3D points for inverse rendering tasks or sample particles for Chamfer distance optimization. Here, θ is the parameter to optimize in our representation. In our experiment, θ includes the center $\mathbf{c}$ and quadratic parameter $\boldsymbol{\beta}$ of feature particles. We also need to obtain gradients on sample particles to optimize $\boldsymbol{\beta}$.

We use an automatic differentiation system to calculate gradients. Similar to NIE [Mehta et al. 2022], in the spirit of Lagrangian advection, the gradient $\frac{\partial \mathcal{L}_{task}}{\partial X}$ acts as instantaneous flow field $\mathbf{V}$ on these points. We calculate a global velocity $\mathbf{V}$ using partition of unity similar to Eq. (1) as:

$$\mathbf{V}(\mathbf{x}) = \sum_{i=1}^{k} \varphi(\mathbf{x}) \mathbf{V}_i, \tag{10}$$

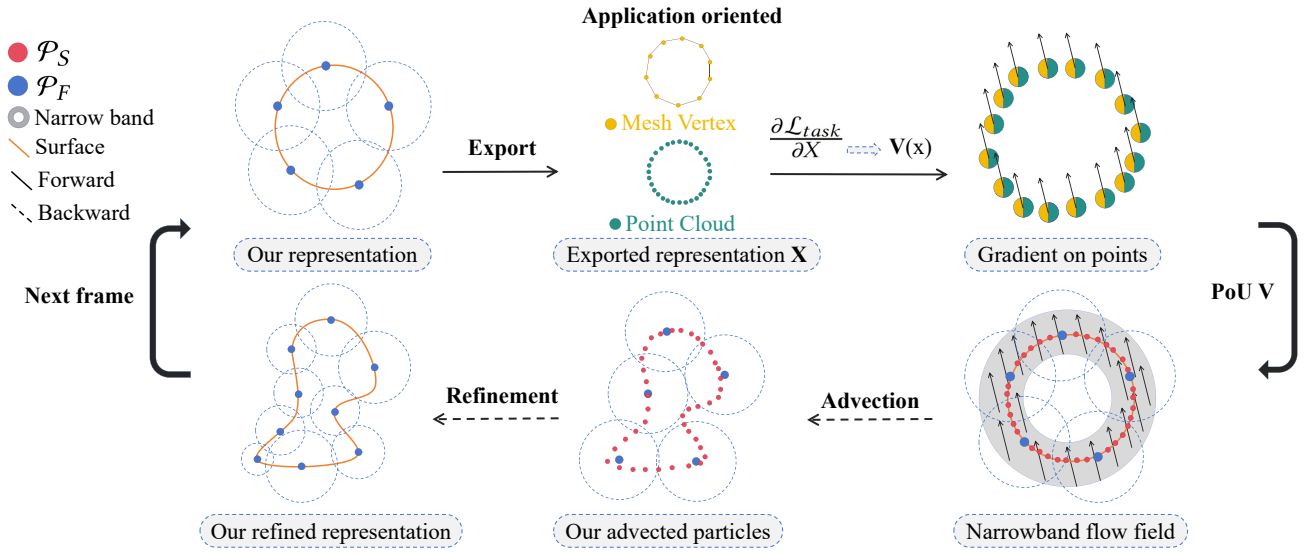

Fig. 10. Our differentiable evolution pipeline: this figure shows how we use gradients in differentiable tasks to advect our representation while preserving adaptivity and integrity. In each iteration, we export task-oriented points, calculate the task loss and gradients at each point, and then interpolate these gradients to create a narrow-band velocity. Finally, we advect the particles and refine them accordingly.

ALGORITHM 2: Implicit surface reconstruction

Input : Sample Particles $\mathbf{p}$, Sparse Grid $\mathcal{G}$, Feature Particles $\mathbf{c}$, Feature particle functions f, Feature particle radius h
Variable : Each Grid Average fixing Position $\mathbf{x}^*_{\mathcal{G}}$, Fixing sample particles in each grid n

Activate $\mathcal{G}$ ▷ Algorithm 1
foreach *activated* $\mathcal{G}$ **do**
  Generate feature particles $\mathbf{c} = \mathbf{c}_{\mathcal{G}}, h = \alpha h_{\mathcal{G}}$
end
foreach *Feature particle* $P_{\mathcal{F}}$ **do**
  while $f_i(\mathbf{c}_i) < 0.01\Delta \mathbf{c}_i$ **do**
    Approximate f_i
    $\mathbf{c}_i \leftarrow \mathbf{c}_i - f_i(\mathbf{c}_i)\frac{\nabla f_i(\mathbf{c}_i)}{|\nabla f_i(\mathbf{c}_i)|}$
  end
  Approximate f_i
end
foreach *Sample Particles* $\mathbf{p}$ **do**
  if p_i *not contained by any feature particles* **then**
    Get finest $\mathcal{G}_{\mathbf{p}_i}$ *contains* $\mathbf{p}_i$
    mark $\mathcal{G}_{\mathbf{p}_i}$ *as re-generated*
    $\mathbf{x}^*_{\mathcal{G}_{\mathbf{p}_i}} \leftarrow \mathbf{x}^*_{\mathcal{G}_{\mathbf{p}_i}} + \mathbf{p}_i$
    $n_{\mathcal{G}_{\mathbf{p}_i}} \leftarrow n_{\mathcal{G}_{\mathbf{p}_i}} + 1$
  end
end
foreach $\mathcal{G}$ *needs re-generated* **do**
  Generate feature particle $c_i = \frac{\mathbf{x}^*_{\mathcal{G}_i}}{n_i}, h_i = \alpha h_{\mathcal{G}_i}$ ▷ *Particle Fixing*
end

where $\mathbf{V}_i$ denotes the velocity value at a nearby point $\mathbf{X}$, and we use gradients on these position as velocity $\mathbf{V}_i \leftarrow \frac{\partial \mathcal{L}_{task}}{\partial X_i}$. Given that points $\mathbf{X}$, sample particles, and feature particles typically reside around the implicit surface, we accurately estimate the velocity $\mathbf{V}$ for each sample particle and feature particle. Consequently, we finalize the remaining backpropagation process through particle advection and polynomial approximation.

In addition to tackling gradient flow, our method can also handle explicit velocity fields such as interface advection or mean curvature flow (see Sec. 9.4). In these applications, we can propagate gradients to our representation via feature particle and sample particle advection, thereby updating the quadratic parameter $\boldsymbol{\beta}$ through polynomial approximation.

7 DYNAMIC INTERFACE

Next, we will detail the process of utilizing velocity to advect the underlying implicit surface, thereby evolving its geometry, topology, and adaptivity.

7.1 Geometric Evolution

Once flow field $\mathbf{V}$ is obtained on each sample particle and feature particle, we advect particles using the forward-Euler scheme: $\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t \mathbf{V}$. For optimization, Δt is equivalent to the learning rate. The normal of each sample particle is updated following the normal advection in Ianniello and Di Mascio [2010]. In an arbitrary velocity field, e.g. velocity field derived from backpropagation, the normal is advected by: $\frac{D\mathbf{n}}{Dt} = -(\nabla \mathbf{V})^\top \mathbf{n} + (\mathbf{n}^\top (\nabla \mathbf{V})^\top \mathbf{n})\mathbf{n}$, and for divergence-free field $\frac{D\mathbf{n}}{Dt} = -(\nabla \mathbf{V})^\top \mathbf{n}$. To calculate $\nabla \mathbf{V}$, we derive from Eq.

(10):

$$\nabla \mathbf{V}(\mathbf{x}) = \sum_{i=1}^{k} \nabla \varphi(\mathbf{x}) \mathbf{V}_i. \tag{11}$$

Velocity derived from the optimization task could be noise (e.g., in inverse rendering). We apply a weighted principal component analysis (PCA) used in Nicolet et al. [2021] and Yu and Turk [2013] to update the particle normals for robustness.

7.2 Topological Transition

After advecting particles, we must deal with interface merging and non-manifold cases by particle deletion like in the PLS method [Enright et al. 2002].

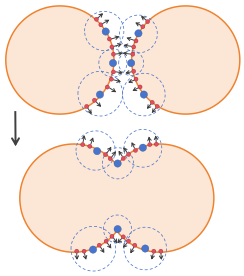

Fig. 11. Particles with opposite orientations and in close distance are deleted to handle the topological transition.

We conduct sample particle deletion when two sample particles with opposite orientations are too close to each other, in particular, we delete particles if their distance $|p_i - p_j| < 0.01r$, where r is the current smallest feature particle radius.

In Fig. 11, two level-set spheres within our representation converge toward each other. Since we're utilizing signed distance level sets (as delineated in contrast to Sec. F), we eliminate sample particles upon collision of the two interfaces. This approach facilitates the transition to the correct topology during evolution.

7.3 Dynamic Adaptivity

After advection and sample particle deletion, we perform a refinement step to feature particle first and then sample particle. For feature particles, we will first conduct the projection step and fixing step, which we discussed in Sec. 5.2 and Sec. 5.2. Upon generating new feature particles, we project them onto the implicit surface. We iterate through this generalization and projection process until all sample particles are encompassed.

Then, after feature particle refinement and obtaining the corrected feature particles and the narrow band implicit surface, we processed them to refine sample particles. Advecting sample particles to their positions and normals occasionally makes the surface noisier and more error-prone. However, this happens infrequently enough that we perform sample particle refinement every 20 steps. First, we perform a projection step to sample particles in which the angle between its normal n_i and its gradient $\nabla f(p_i)$ is significant when $n_i \cdot \frac{\nabla f(p_i)}{|\nabla f(p_i)|} < 0.95$, we use the same Newton projection step discussed in Sec. 5.2 until $f(p_i) < 10^{-6}$, and we set a stop iteration 20 if the projection hasn't met the condition, we delete these sample particles. After the projection step, we would correct sample particles that are selected in the previous step to correct their normal as $n_i = \frac{\nabla f(p_i)}{|\nabla f(p_i)|}$. Also, we randomly resample, add, and delete sample particles during this process.

ALGORITHM 3: Differentiable moving particle

Input : Sample Particles $\mathbf{p}$, Sample particle normal $\mathbf{n}$, Sparse Grid $\mathcal{G}$, Feature Particles $\mathbf{c}$, Feature particle functions f, Smallest feature particle radius r_0, refine step i_{step}, Task Ground truth
Variable : Loss function $\mathcal{L}$, Task oriented output $\mathbf{X}$, Velocity field $\mathbf{V}$
while $\mathcal{L}$ *not converged* **do**
  Export $\mathbf{X}$ from feature particles
  $\mathcal{L} \leftarrow$ compare $\mathbf{X}$ to ground truth
  Compute $\frac{\partial \mathcal{L}_{task}}{\partial X}$
  Obtain $\mathbf{V}$ using PoU over $V_i \leftarrow \frac{\partial \mathcal{L}_{task}}{\partial \mathbf{X}_i}$ ▷ Eq. (10)
  foreach p **do**
    $Advect(\mathbf{p}_i, \mathbf{n}_i, \mathbf{V}(\mathbf{p}_i))$
  end
  foreach c **do**
    $Advect(\mathbf{c}_i, \mathbf{V}(\mathbf{c}_i))$
  end
  foreach p **do**
    $\mathbf{p}^* \leftarrow$ Neighbor search on $\mathbf{p}$
    foreach $\mathbf{p}^*$ **do**
      if $|\mathbf{p}_i^*| < 0.01 r_0$ **then**
        mark $\mathbf{p}_i^*$ and $\mathbf{p}$ as need deletion
      end
    end
  end
  Delete every $\mathbf{p}$ marked as need deletion
  Implicit surface reconstruction ▷ Algorithm 2
  $AddAndDecrease(\mathbf{p}, \mathbf{c})$
  if $iteration \equiv 0 \pmod{i_{step}}$ **then**
    Project $\mathbf{p}$
    Re-activate $\mathcal{G}$
    Poisson reconstruct on $\mathbf{p}$ and $\mathbf{n}$
    Sample new $\mathbf{p}$ and $\mathbf{n}$
  end
end

7.4 Resampling

To maintain our structure's adaptivity, we will reinitialize the sparse grid every n iteration (in our experiments, ranging from 8 to 20), depending on application and learning rate. And within the context of differentiable applications, to mitigate the issue of the least square being underfitted and the sample particle's position becoming excessively noisy following multiple gradient velocity advections, we apply the Poisson reconstruction method [Kazhdan and Hoppe 2013] [Kazhdan et al. 2006] to surface after every 20 steps. This approach effectively diminishes the occurrence of local minima and reduces noise within the differentiable application. We show our pseudo-code in Algorithm 3. And in Fig. 12, we show a sphere deform by vortex field until $t = 10$, which will be further discussed in Sec. 9.4.1.

We implement a resampling strategy to mitigate the risk of falling into the local minima for differentiation tasks like Chamfer-loss SDF optimization (Sec. 9.2) and inverse rendering (Sec. 9.3). The resampling step involves removing sample particles with large gradients, akin to the 3D Gaussian method [Kerbl et al. 2023], and randomly seeding new sample particles along the extensions of nearby feature

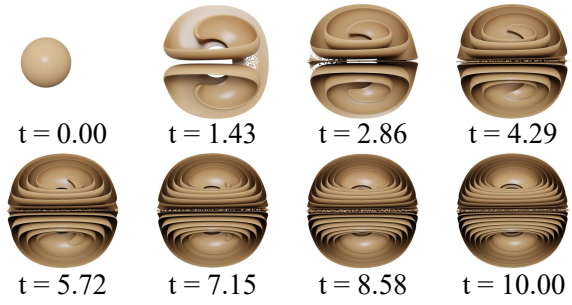

Fig. 12. 3D deformation field advection experiments until $t = 10$. We advect an initial sphere by a given velocity field, showing the stability and superiority of our method in dynamic interface tracking.

particle surfaces. For this purpose, we select feature particles that do not include deleted sample particles within the same coarsest grid as the deleted ones. Subsequently, we multiply their radius by a factor (typically 2.3 in our experiments), scatter random sample particles in this coarsest grid, and then project them onto the extension feature particles' surface.

8 IMPLEMENTATION

We implemented our pipeline based on Taichi [Hu et al. 2019], pytorch3D [Ravi et al. 2020] and Nvdiffrast [Laine et al. 2020]. Taichi is a High-performance parallel programming language we use it for high-speed GPU optimization and we leverage its bitmasked SNode to construct our sparse grid and manage the particle system. We use pytorch3D for fast neighbor search and sample points from meshes, and we use Nvdiffrast for inverse rendering pipeline. In differentiable applications, we set the learning rate to 5×10^{-4} for the first 600 iterations, then reduce it to 1×10^{-4} thereafter. And the activation threshold ε_0, we set it to 85 in our experiments. We use workstations with NVIDIA RTX A4000 and NVIDIA RTX A6000 to run all our code and comparison code.

9 EXPERIMENTS

In this section, we evaluate the performance of our representation in various differentiable optimization and surface evolution tasks. First, in Sec. 9.1, we compare our implicit surface reconstruction approach with the neural implicit method in terms of parameter usage and surface quality.

Next, we demonstrate various differentiable tasks, including differentiable SDF optimization in Sec. 9.2, inverse rendering in Sec. 9.3, and explicit flow-based surface evolution in Sec. 9.4. These examples highlight the robust differentiability and surface-tracking capabilities. Besides, we include runtime performance analysis in Sec. 9.6 and ablation studies in Sec. 9.5.

9.1 Direct SDF Reconstruction

In this section, we focus on demonstrating the efficiency and accuracy of our representation in implicit surface reconstruction. To achieve this, we extract sample particles and their normals from various intricate input shapes, such as meshes. Subsequently, we generate feature particles based on the methodology outlined in Sec. 4 and 5, as described in Algorithm 2. We compare the resulting implicit shape with those obtained using other methods, specifically NGLOD [Takikawa et al. 2021] and Instant-NGP [Müller et al. 2022]. For these comparisons, we utilize the source code provided by the respective authors to ensure consistency and accuracy in our evaluation.

Metric. We assess the quality of reconstruction by computing the Chamfer-L distance and the Intersection-over-Union (IoU) metric. Additionally, we compare the number of parameters, which represents the sum of the training parameters for each model. This includes the combined total of the decoder and representation storage parameters. In our model, parameters encompass the positions, polynomial coefficients, and radii of feature particles. Consequently, the total number of parameters is equal to 14 times the number of feature particles (3 for positions, 10 for coefficients, and 1 for radius).

Dataset. The dataset consists of models sourced from The Stanford 3D Scanning Repository, Thingi10K [Zhou and Jacobson 2016], and TurboSquid, encompassing shapes with intricate topologies and geometries. For testing, we impartially select 8 models from TurboSquid, 50 from Thingi10K, and all models from Stanford Models as our test shapes.

In Fig. 13, we present the reconstruction results of challenging shapes, accompanied by a metric comparison with NGLOD, Instant NGP, and our method. It's clear that, with an equivalent or fewer number of parameters, our model showcases smoother surfaces and superior geometric quality. The figure also illustrates the distribution of our final feature particles and their radius sizes, effectively demonstrating the adaptive nature of our representation in handling surface co-dimension.

Moreover, Table 1 presents the average results of models sourced from three distinct datasets, facilitating comparison across various metrics. The findings indicate that, with an equivalent or fewer number of parameters, our model surpasses the neural-based methods in both IoU and NAE metrics, suggesting that our feature particles can encode shapes more smoothly and accurately. Additionally, our model generally exhibits superior performance in Chamfer Loss across most shapes. Interestingly, I-NGP achieves a comparable Chamfer Loss to ours when both reconstructed shapes are nearly perfect (i.e., IoU close to 1).

9.1.1 non-manifold unsigned distance surface. As described in Appendix F, our method could represent non-manifold junctions in unsigned distance surfaces which are widely used to represent foam and bubbles in nature. In Fig. 14, we showed our non-manifold reconstruction results, we selected ground truth surface from bubbles simulations and used our method to represent the surface, we also showed the particle view for the reconstruction, and this indicates our method's ability to handle non-manifold junctions.

9.2 Differentiable SDF Optimization

In this section, our focus is primarily on presenting the task related to differentiable SDF optimization, aiming to showcase the differentiability inherent in our framework. In our experiments, we consistently start by initializing our model as a sphere. We then

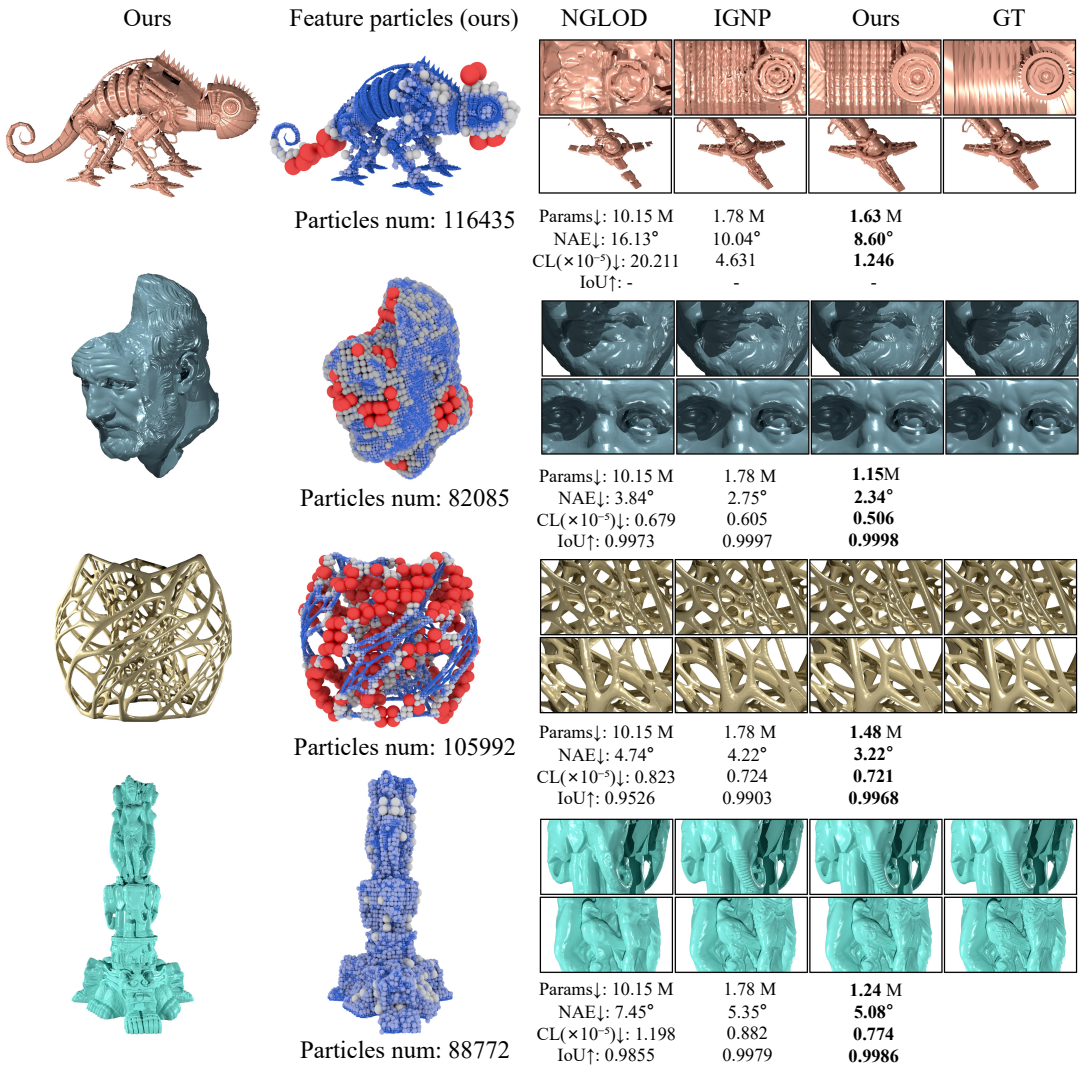

Fig. 13. Comparison of different implicit representation methods for direct SDF reconstruction. We compare our method with NGLOD [Takikawa et al. 2021] and Instant-NGP [Müller et al. 2022] using several metrics: Params (parameters), NAE (normalized angular error), CL (Chamfer loss^2), and IoU (intersection over union). The first two columns display a full view of our reconstructed shape and feature particles structure, while the subsequent columns provide zoomed-in views comparing different methods and the ground truth.

Direct SDF Reconstruction										
Method	Parameters↓	StF Models			Thingi10K			TurboSquid		
		IoU ↑	Chamfer-L^2 ↓	NAE ↓	IoU↑	Chamfer-L^2 ↓	NAE ↓	IoU ↑	Chamfer-L^2 ↓	NAE ↓
NGOLD5	10.15M	0.9947	0.701×10^{-5}	3.56°	0.9523	1.675×10^{-5}	5.08°	0.8574	6.470×10^{-5}	14.04°
I-NGP	1.77M	0.9995	0.575×10^{-5}	2.56°	0.9935	0.950×10^{-5}	3.89°	0.9367	1.971×10^{-5}	8.91°
Ours	**0.84M-1.68M**	**0.9997**	$\mathbf{0.505 \times 10^{-5}}$	**2.23°**	**0.9966**	$\mathbf{1.036 \times 10^{-5}}$	**2.83°**	**0.9538**	$\mathbf{1.319 \times 10^{-5}}$	**7.42°**

Table 1. In direct shape reconstruction, we conduct a quantitative comparison with the NGLOD5 and INGP models. We select 8 models from TurboSquid, 50 from Thingi10K, and all models from Stanford Models as our test shapes. The findings show that our model outperforms neural-based methods in both IoU and NAE metrics, with an equal or fewer number of parameters. This indicates that our feature particles encode shapes more smoothly and accurately.

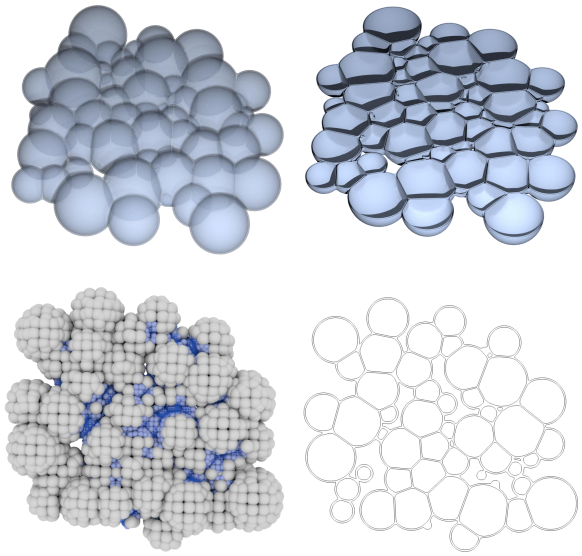

Fig. 14. Non-manifold junctions reconstructed by our representation. **Left top**: 3D reconstructed mesh. **Left bottom**: feature particles view. **Right top**: 3D cross section showing the non-manifold structures. **Right bottom**: 2D slice curves showing the junctions. We used a threshold of 0.003 to extract mesh surfaces (with Marching Cubes) on both sides of a non-manifold interface for visualization purposes.

use the task gradient, detailed in Sec. 6, to guide the evolution of our model shape towards the target output. For these experiments, we set the particle advection time step to $\Delta t = 0.1$. This approach allows us to observe the movement of feature particles and evaluate the robustness of our model.

9.2.1 Chamfer Loss based surface reconstruction. In this task, we employ the Chamfer distance loss for the differential reconstruction of the target shape, initially set as a sphere. To achieve this, we sample a set of surface points ${\mathbf{p}_g}^{N_g}$ from the ground truth mesh and calculate the Chamfer distance loss between these ground truth points $\mathbf{p}_g$ and the sample particles $\mathbf{p}_s$: $\mathcal{L}_{Chamfer} = \sum_{i=1}^{N_s} min_{j=1}^{N_g} |\mathbf{p}_s^i - \mathbf{p}_g^j|_2 + \sum_{j=1}^{N_g} min_{i=1}^{N_s} |\mathbf{p}_g^j - \mathbf{p}_s^i|_2$. This loss quantifies the alignment between the points in the predicted set and those in the ground truth set. Minimizing this loss during training optimizes our model towards the target shape.

As shown in Fig. 15, we compare our method to Point2Mesh [Hanocka et al. 2020], NIE [Mehta et al. 2022], and DMTet [Shen et al. 2021], we present various reconstruction results initiated from a differentiable process that begins with a sphere and utilizes the gradient of Chamfer loss for reconstruction. In these experiments, our approach employs a minimal radius equivalent to a 512^3 resolution, utilizing a 4-layer radius. Similarly, for other methods, we adopt their configurations set to a 512^3 resolution. For instance, NIE utilizes a 512^3 marching cube, and Point2Mesh specifies a maximal 512^2 triangle face count. As for DMTet, we utilize a 400^3 resolution, which is the maximum allowable resolution on our 32GB GPU. In

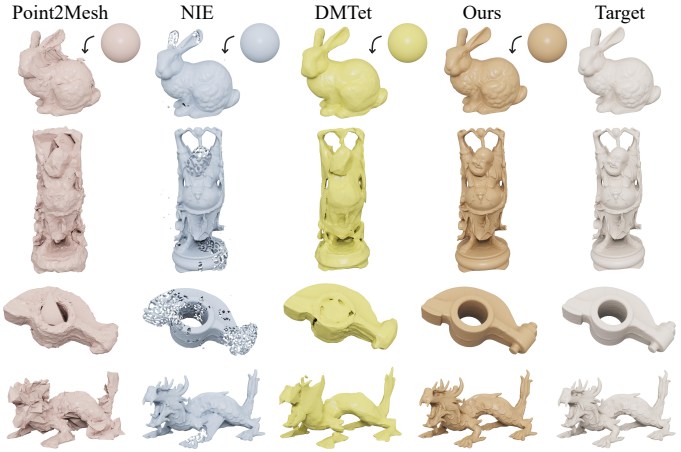

Fig. 15. We compare our method against Point2Mesh, NIE, and DMTet by evolving an initial sphere to a given target shape via differentiable SDF optimization. Our results outperform others regarding its shape completeness (IoU, see Table2) and geometry smoothness.

these experiments, we maintain consistency by utilizing the same number of sample points ($500K$) on the ground truth mesh. We train each model until convergence to ensure a fair comparison.

Differentiable Reconstruction Average			
Method	learning iterations	IoU ↑	Chamfer-L %↓
Point2Mesh	6000	\	4.14×10^{-3}
NIE	3000	0.8741	3.12×10^{-4}
DMTet	10000	0.9455	9.17×10^{-5}
Ours	**1500**	**0.9948**	$\mathbf{1.28 \times 10^{-5}}$

Table 2. We quantitatively compare our method with Point2Mesh, NIE, DMTet in differentiable SDF optimization experiments. The learning iterations column shows the steps until convergence. Our model demonstrates better shape reconstruction completeness (IoU) and improved convergence (Chamfer-L).

Due to the absence of a resampling strategy, methods like NIE are susceptible to fitting local minima for chamfer loss, in certain cases, the chamfer loss generates a gradient perpendicular to the surface normal on some sections, leading to the potential occurrence of local minima. The implementation of a resampling strategy is necessary to mitigate this issue. Methods like Point2Mesh introduce difficulty in altering topology through mesh optimization and dependence on remesh strategies. This often leads to the final result preserving the same topology as the sphere, it shows a strong reliance on the initial shape. For DMTet, in our experiments, achieving a high-precision mesh with their approach proves challenging, and they occasionally encounter local minima in the loss function. We demonstrate the IoU and Chamfer-L^2 metrics for these experiments in the Table. 2, and our method outperforms others in this comparison.

9.2.2 Under-Sampling SDF Reconstruction. We apply our method to under-sampling SDF reconstruction, utilizing sparsely sampled

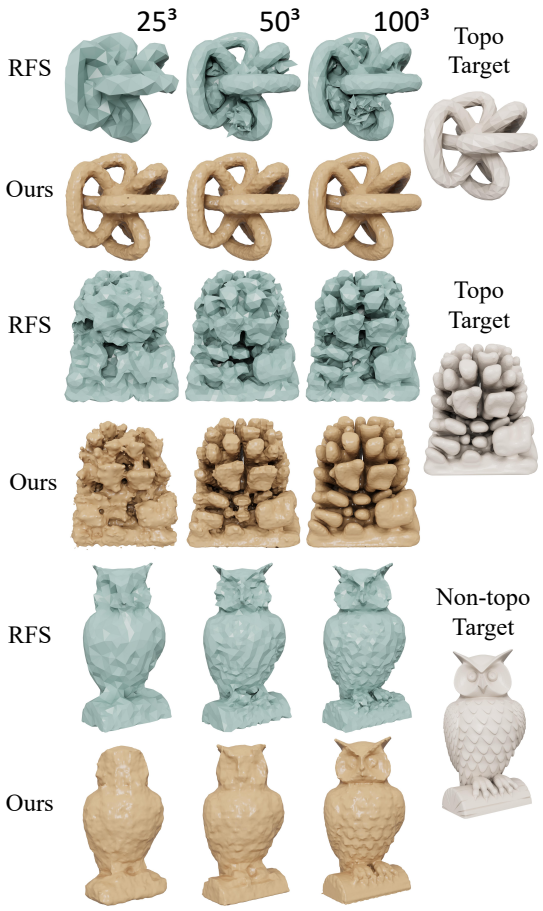

Fig. 16. Under-sampling SDF grid reconstruction comparison with the work RFS by Sellán et al. [2023]. We use sparsely sampled SDF input on a voxel grid and optimize our structure to represent the underlying surface continuously. The first row displays the resolution of the input voxel grids (i.e., 25^3). The other rows show the results of RFS optimization and our method. Our approach effectively handles topological changes while RFS usually fails on these Topo targets.

SDF grids (e.g., 10^3) or a limited number of sample points with SDF values in space. Our optimization aims to reveal a more faithful representation of the underlying surface from the sparse input data, contrasting with methods like direct marching cubes. For a reference, we apply the same Loss function/energy in Reach for sphere [Sellán et al. 2023] method, assume we are given a set of sparse points in space $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n \in \mathbb{R}^3$ with SDF value $s_1, s_2, \ldots, s_n \in \mathbb{R}$, the task is to reconstruct a valid surface Ω expressed the constrain $\phi(\mathbf{p}_i, \Omega) = s_i, \forall i \in \{1, \ldots, n\}$. Every point $\mathbf{p}_i$ here are considered as a small ball with a radius s_i, and our SDF must be tangent to it. This approach is referred to as tangency-aware surface reconstruction. The loss function is expressed as: $\mathcal{L}_{sdf} = \sum_i^n (\phi(\mathbf{p}_i, \Omega) - s_i)^2$.

In our methodology, we incorporate tangency-aware flow for both feature particles and sample particles. This indicates our aim is to guarantee that the surfaces of feature particles and sample particles, along with their normals, are tangent to the spheres. Consequently, the loss function becomes

$$\mathcal{L}_{tangency} = \sum_j^{N_f} (s_{min} - (\mathbf{p}_{min} - \mathbf{p}_f^j) \cdot \frac{\nabla f(\mathbf{p}_f^j)}{|\nabla f(\mathbf{p}_f^j)|}) + \sum_j^{N_s} (s_{min} - (\mathbf{p}_{min} - \mathbf{p}_s^j) \cdot n_s^j). \tag{12}$$

Here, s_{min} and $\mathbf{p}_{min}$ represent the nearest input points for each sample particle and feature particle, respectively.

Under-Sampling SDF Reconstruction Reconstruction Average					
Method	Grid Sizes	Topo		Non-topo	
		IoU ↑	Chamfer-L %↓	IoU ↑	Chamfer-L %↓
RFS	25^3	0.6618	3.06×10^{-3}	0.9326	4.36×10^{-4}
RFS	50^3	0.75	1.71×10^{-3}	0.9643	1.41×10^{-4}
RFS	100^3	0.8692	9.73×10^{-4}	0.9775	7.77×10^{-5}
Ours	25^3	**0.9308**	$\mathbf{5.27 \times 10^{-4}}$	**0.9706**	$\mathbf{1.78 \times 10^{-4}}$
Ours	50^3	**0.9567**	$\mathbf{1.29 \times 10^{-4}}$	**0.9763**	$\mathbf{1.18 \times 10^{-4}}$
Ours	100^3	**0.9799**	$\mathbf{3.72 \times 10^{-5}}$	**0.9858**	$\mathbf{3.77 \times 10^{-5}}$

Table 3. Under-sampling SDF experiments, the second column displays the sizes of the input sparse sampled data grids. The "Topo" column indicates that the target shape's topology differs from the initial sphere, while the "Non-topo" column shows that the target shapes have the same topology as the initial sphere. IoU and Chamfer-L show the average quality of the optimized shapes for the two methods..

We compare our result with reach for sphere (RFS) [Sellán et al. 2023], which is the original paper that proposed the tangency-aware flow. As shown in Fig. 16, given that RFS operates on a remeshing strategy, a notable constraint is its inability to alter topology during optimization. In contrast, our approach, as an implicit surface optimization method, offers the capability to modify topology dynamically throughout the optimization process. Observing the figure, it becomes evident that our method reconstruct a more precise surface under the same resolution of SDF grid input. Additionally, our approach demonstrates experimentally that implicit surfaces reconstructed from undersampled Signed Distance Function (SDF) grids display superior performance in terms of evolution metrics, as in Table. 3.

9.3 Inverse Rendering

For task of inverse rendering, we use a similar method to NIE [Mehta et al. 2022] for inverse rendering, which uses a differentiable renderer designed for triangle meshes to optimize geometry defined using parametric level sets. We focus on geometry recovery from synthetic scenes with known reflectance, and we choose Nvdiffrast [Laine et al. 2020] as a differentiable rasterizer. Like NIE, we extract triangle mesh using a marching cube every time step and pass mesh to Nvdiffrast to get the gradients for minimizing a rendering photometric error $\mathcal{E}$. After getting the gradients, we define the flow field on vertices as $\mathbf{V}^t(x_i) = -\frac{\partial \mathcal{E}}{\partial \mathbf{x}_i} - \lambda \mathbf{L}\mathbf{x}_i$, where $\mathbf{L}$ is the Laplacian smooth term which has been proved that useful for reducing floating and surface artifacts in Nicolet et al. [2021] and NIE. After obtaining the flow field on vertices, we perform a partition of unity

Fig. 17. Inverse rendering experiments compare our method with D-SDF and NIE across 8 different shapes. The first four columns display geometrically complex shapes, while the last four show topologically complex shapes. This demonstrates that our method surpasses the compared approaches in detail preservation and topology correctness.

Inverse Rendering									
Method	Metric	Shapes							
		armadillo	lion	lucy	roal	vbunny	vine	lightbulb	bukcy
D-SDF	Chamfer↓	9.152×10^{-5}	1.414×10^{-4}	$\mathbf{1.289 \times 10^{-4}}$	1.537×10^{-4}	1.622×10^{-2}	3.166×10^{-4}	$\mathbf{8.659 \times 10^{-5}}$	1.622×10^{-2}
NIE		5.774×10^{-5}	1.519×10^{-4}	6.981×10^{-4}	6.981×10^{-4}	1.172×10^{-4}	3.063×10^{-4}	0.126 (fail)	1.219×10^{-4}
Ours		$\mathbf{3.134 \times 10^{-5}}$	$\mathbf{9.176 \times 10^{-5}}$	5.939×10^{-4}	$\mathbf{1.290 \times 10^{-4}}$	$\mathbf{1.012 \times 10^{-4}}$	$\mathbf{2.945 \times 10^{-4}}$	1.381×10^{-4}	$\mathbf{1.153 \times 10^{-4}}$
D-SDF	PSNR↑	29.6193	29.4887	30.4397	31.0695	19.8149	25.2570	30.5000	19.9519
NIE		34.5570	32.6136	31.3545	32.7217	29.2792	32.9315	18.5219 (fail)	28.8182
Ours		**41.0505**	**36.1631**	**33.1622**	**36.4942**	**31.1530**	**34.9583**	**35.0729**	**31.5995**

Table 4. Inverse Rendering comparison. We quantitatively compare our method with D-SDF [Vicini et al. 2022] and NIE [Mehta et al. 2022] using Chamfer loss and PSNR. Our results achieve up to a 4-point increase in PSNR compared to other models and enhance stability (NIE fails in the penultimate case).

on velocity to get a continuous narrow-band velocity described in Sec. 6.1 to update our representation.

In our experiments, we select various shapes characterized by diverse genus and geometric intricacies. We conduct comparative analyses between our approach and the methods NIE and D-SDF[Vicini et al. 2022]. Specifically, we employ a grid resolution of 175^3 for marching cubes in NIE and an sdf grid resolution of 175^3 for D-SDF. For our method, we utilize a resolution comprising a 3-layer sparse grid, with the finest grid width set to match the 175^3 resolution. Additionally, we set our finest particle radius to 1.15 times the size of this cell. In both methods, we randomly generated 100 viewpoints for each shape. Throughout the optimization process, we initialized both shapes as spheres with a radius of 0.3 within the space defined by $[-1, 1]$. As illustrated in Figure 17, each method demonstrates the capability to reconstruct shapes effectively, particularly those exhibiting minimal topological deviation from a sphere, but D-SDF encounters challenges in accurately optimizing shapes with high genus, such as the vbunny, NIE may encounter situations where it ends up with an SDF field lacking an iso-surface (for instance, in the case of a lightbulb, the final optimized SDF network outputs are above 0 throughout the space, and we present the last third iteration in our figure). Under the same resolution, the optimized shapes produced by NIE and D-SDF methods exhibit less detail compared to ours. This discrepancy arises from the inherent smoother tendency of the networks utilized in NIE and the grids employed in D-SDF, in contrast to our quadratic particles approach.

In Table 4, We assess the Chamfer Loss and PSNR of the rendered images. Our method outperforms the other two methods in terms of the PSNR metric and achieves superior performance in most of the Chamfer Loss evaluations. However, D-SDF exhibits better Chamfer

Loss in some shapes, potentially attributed to the optimization of the grid to maintain a smoother average and prevent the generation of off-surface artifacts. In contrast, our method and NIE may occasionally exhibit inner floating artifacts due to the lack of visible information about the inner space of a shape.

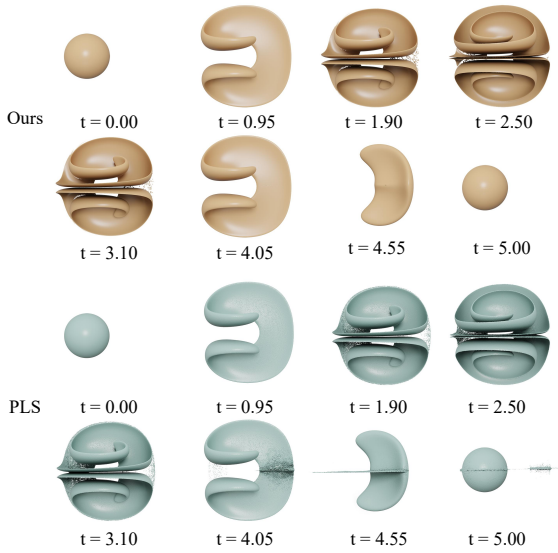

Fig. 18. We compare our method with PLS (Particle Level Set) in a 3D advection test. The initial sphere is advected by a given velocity field and reversed at $t = 2.5$. Ours is able to better preserve the volume with fewer artifacts than the PLS method.

9.4 Explicit Velocity Evolution

In this section, we demonstrate the use of pre-defined velocity fields, such as the vortex field or velocity fields derived from explicit computations like curvature flow, for the advection of our representation. This is in contrast to utilizing the gradient flow derived from automatic differentiation in the previous section. For the advection tests in this section, we use the fourth-order Runge-Kutta method to achieve a more accurate solution and adhere to established practices in advection testing.

9.4.1 Level Set Evolution Tests. We employ a three-dimensional incompressible flow field advection test used in PLS [Enright et al. 2002]. The velocity field is given by $u = 2\,sin^2(\pi x)$ $sin(2\pi y)\,sin(2\pi z), v = -sin(2\pi x)\,sin^2(\pi y)\,sin(2\pi z), w = -sin(2\pi x)$ $sin(2\pi y)\,sin^2(\pi z)$, and the flow field is reversed at $t = 3$, and we choose sub-step $\Delta t = 0.01$ for experiments. We placed a sphere with a radius 0.15 at $(0.35, 0.35, 0.35)$ within a unit computational domain. This flow makes the sphere to be entrained by two rotating vortices, the sphere deformation reaches the maximum at $t = 3$ and should be flowed back to the original shape at $t = 6$. We employ 4 layer initialization sparse grid beginning at 128^3 and end with the finest resolution 512^3 in this test, as a comparison, we employ the PLS method with 512^3 resolution to perform the same evolution.

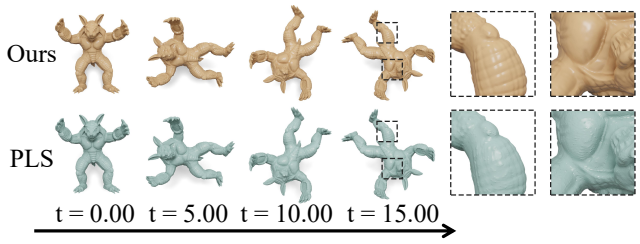

Fig. 19. Rigid body rotation comparison with PLS method. The armadillo is rotated at a constant speed and we show that our method better preserves the details than the PLS method with zoomed-in views of the final advected frames.

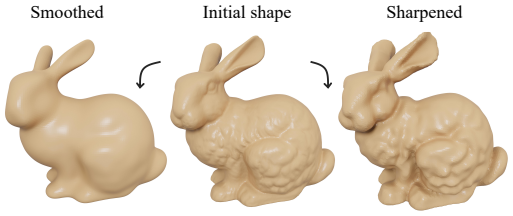

Fig. 20. Smoothed and sharpened bunny using mean curvature flow.

As depicted in Fig. 18, our approach effectively restores the original spherical shape during this evolution process. Moreover, we assess the volume lost in the restored shape compared to the original one, as presented in Table 5, demonstrating our method outperforms the PLS method. Our method can also deform for a longer version for example till $t = 10$. A longer version was proposed in the Level Set Review by Gibou et al. [2018]. They deformed the sphere with a resolution of 4096^3 using a supercomputer until $t = 9$. We are able to achieve similar results, as demonstrated in Fig. 12, employing a 2-layer sparse grid with the finest grid being 1024^3.

We also use the rigid body rotation velocity field $u = 0.5 - y$, $v = x - 0.5$, $w = 0$ to evaluate 3D diffusion error under evolution. We use bunny and armadillo as our test shapes, the shapes are unit normalized in bounding box $[1, 1, 1]$ and centering at $(0.5, 0.5, 0.5)$. Similarly to the previous test, we compare our method with the PLS method under 512^3 resolution. Fig. 19 shows the rotation process and table 5 we compare the lost volume for each method at the final frame, and they indicate our method preserves better shape during this process.

9.4.2 Mean Curvature Flow. We employ an intrinsic flow that is self-generated from the implicit surface, known as mean curvature flow. The velocity is given by:

$$\mathbf{V}(\mathbf{x}) = \lambda \Delta \mathbf{x} = -2\lambda\kappa(\mathbf{x})\mathbf{n}(\mathbf{x}), \tag{13}$$

where $\mathbf{x}$ is a point on the implicit surface, λ is a diffusion scalar, Δ is the Laplace-Beltrami operator, and κ and $\mathbf{n}$ is the mean curvature and normal. Calculating the mean curvature κ becomes straightforward with our polynomial-based Partition of Unity. From Eq. (4) for each feature particle we calculate curvature inside its supporting radius

Explicit Velocity Evolution							
Method	Resolution	3D Deformation Field			Rigid Body Rotation		
		Deform Iterations	IoU ↑	Volume Loss %↓	Rotation Iterations	IoU ↑	Volume Loss %↓
PLS	128^3	500	0.95224	3.83628	1500	0.97005	1.77448
PLS	512^3	500	0.97543	0.45876	1500	0.99392	0.20883
Ours	128^3	500	0.99878	−0.14774	1500	0.99660	−0.01213
Ours	512^3	500	**0.99893**	**0.05751**	1500	**0.99868**	**0.00059**

Table 5. Deformation Evaluation. The "Resolution" column displays the finest grid resolution used in the experiments, while the "Iterations" column shows the number of advection steps performed. We evaluate IoU and Volume Loss between the final advected frame and the initial frame.

as:

$$\kappa_i = \nabla \cdot \frac{\nabla f_i}{|\nabla f_i|} = \nabla \cdot \frac{\nabla \mathbf{b_i}^\top \beta_i}{|\nabla \mathbf{b_i}^\top \beta_i|}. \tag{14}$$

We derive a detailed curvature formulation in Appendix D. Then we get the global curvature function by $\kappa(\mathbf{x}) = \sum_{i=1}^{N_p} \varphi_i(\mathbf{x})\kappa_i(\mathbf{x})$. In mean curvature experiments, our set 4-layer radius for our particle, the finest feature particle are generated from the cell with 128^3 resolution and the coarsest from 16^3.

We apply the flow described in Eq. (13) to our representation following the method outlined in Sec. 6. Consequently, the implicit surface represented by our model undergo smoothing (for $t > 0$) or sharpening (for $t < 0$) during the mean curvature flow evolution. Fig. 20 illustrates a smoothed and sharpened bunny achieved using our method.

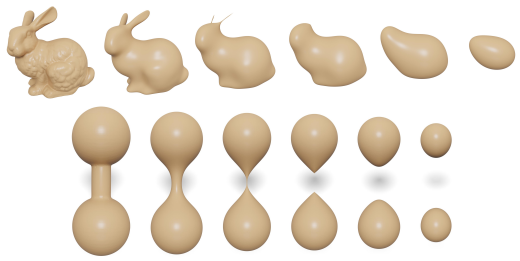

Fig. 21. Mean curvature flow evolution of bunny surface . And Dumbbell surface illustrating the changes in topology throughout the process.

In Fig. 21, we depict the evolution process of a topological genus 0 bunny into a point. Noticeably, the rabbit's ears are in the process of evolving into singularity shapes. Our implicit surface evolution method adeptly handles these singularities, a capability that other methods (such as mesh-based approaches) may lack. Furthermore, we demonstrate the evolution of a classic example the dumbbell in Fig. 21. Initially, it pinches off, creating two connected components. Subsequently, each component collapses to a point, eventually forming small spheres just before that. The topology of the dumbbell undergoes changes during this evolution. We effectively manage topology changes and remove non-manifold regions due to the method outlined in Sec. 7.2.

9.5 Ablation Study

Reconstruction	Params	IoU	CL
MPU	1.32M	0.9996	0.641×10^{-5}
Our	1.55M	0.9997	0.505×10^{-5}

Table 6. Moving v.s. Fixed Feature Particles in Reconstruction ablation study. We use StF Models in Sec. 9.1 for comparsion. The moving-particle version outperforms the fixed-particle version (original MPU) by 20% in Chamfer loss, though it requires 14% more parameters to enable particle motion.

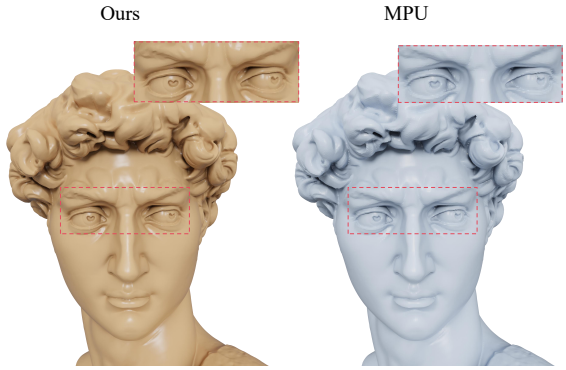

Fig. 22. Moving v.s. Fixed Feature Particles in Reconstruction. It shows that our reconstruction results are smoother and of higher quality around the statue's eyes.

Moving v.s. Fixed Feature Particles in Reconstruction. We compare the algorithm efficacy in direct shape reconstruction (see Sec.9.1) between using moving feature particles and fixed feature particles (i.e., traditional MPU [Ohtake et al. 2005]). This ablation test aims to demonstrate the efficacy of moving particles in the reconstruction pipeline. In Figure 22, we show comparison reconstruction on David's statue, it shows that our reconstruction results are smoother and of higher quality at statue's eyes. And as shown in Table 6, the moving-particle version outperforms the fixed-particle version (i.e., original MPU) by 20% in Chamfer loss, at the expense of 14% extra parameter usage to enable particle motion in reconstruction.

Fixed v.s. Moving Feature Particles in Inverse Rendering. Next, we further demonstrate the role of moving feature particles in differentiable optimization tasks. We carry out our ablation test within an inverse rendering pipeline. As shown in Figure 23 (b), the algorithm produces an output shape with noticeable artifacts on the bunny's

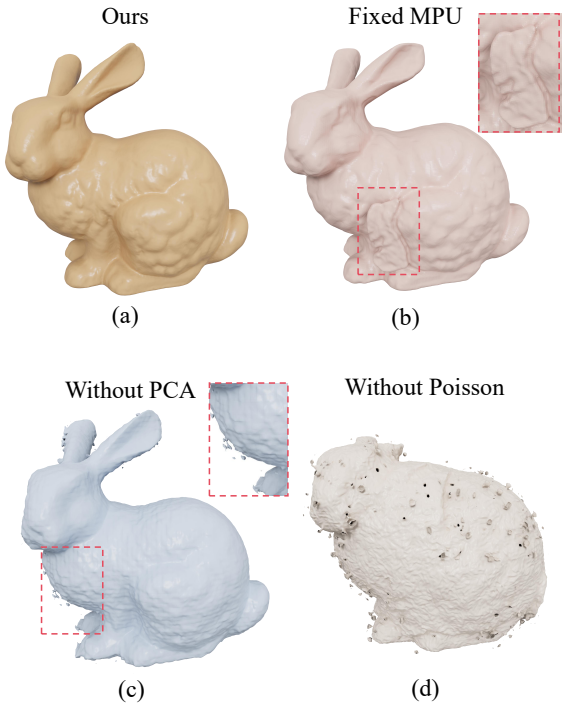

Fig. 23. Robustness Operation ablation study in inverse rendering. Figure (b) shows the fixed particle version of our method. Figure (c) displays particle advection without PCA for normal correction, which results in noticeable artifacts. Figure (d) illustrates advection without Poisson reconstruction for reinitialization, which fails midway through optimization due to excessive artifacts, leading to invalid gradients.

face and around its feet. In implementing the algorithm with fixed feature particles, we carry out the reinitialization step for every iteration to match the MPU representation with the current shape. We conjecture that this frequent reinitialization causes less accurate rendering and the loss of surface details.

PCA and Poisson Reconstruction. To demonstrate the necessity of our additional operations of PCA for particle normal calculation and Poisson reconstruction for reinitialization, we conduct ablation tests on these two operations for evolving a sphere to a bunny shape using inverse rendering. As shown in Fig. 23, without these operations, the algorithm will produce results with noticeable artifacts and errors. In particular, as shown in Fig. 23 (c), the shape output exhibits over 20 visually noticeable artifacts and loses details without the PCA normal calculation. As shown in Fig. 23 (d), the global shape diverges from the ground truth after 70 iterations without a Poisson reconstruction due to the invalid gradients.

9.6 Runtime Performance

Training Time for Reconstruction. As shown in Table 7, our method's training time is comparable to INGP (7 seconds slower in total training) and significantly faster than NGLOD ($x10$ speedup) for direct reconstruction tasks (i.e., producing SDF using ground truth input

Querying time(s)		DeepSDF	NGLOD	I-NGP	Ours
Voxel size	1024^3	2824.92	125.75	11.16	19.23
	512^3	756.43	8.75	1.89	2.14
	128^3	15.18	0.23	0.094	0.083
Training time(s)		3600.00	439.24	8.40	16.17

Table 7. Runtime performance: Querying times for different models after training and average training time in Sec. 9.1. "Voxel size" denotes the size of the test querying positions used to measure these times. We trained DeepSDF for only one hour due to its lengthy training time and lower quality compared to other models.

sample particles, see Sec. 9.1 for task details). We further provide a breakdown of the time spent on each step as shown in Table 8, indicating that the sparse grid initialization is the bottleneck of the entire process.

Training Time for Inverse Rendering. Next, we further demonstrate the timing performance for the training step in inverse rendering tasks. As shown in Table 8, our approach is 50% faster than NIE in total time with the same number of iterations. This is because, for each iteration, NIE trains the neural network, which takes about 6 seconds, while we advect particles and update their MPU parameters, which takes about one second, along with particle reinitialization every twenty iterations.

Query Time. Last, we evaluate our model's querying speed after the training process is finished. In particular, We measure the time required to input voxel position data and retrieve the SDF value at each position. As shown in Table 7, the average querying time is comparable to that of INGP and ten times faster than NGLOD and DeepSDF [Park et al. 2019b].

10 CONCLUSIONS

We have introduced a novel differentiable moving particle representation using the multi-level partition of unity for dynamic implicit geometries. Our adaptive particle representation effectively expresses implicit surfaces and is enhanced by a multi-level background grid for particle adaptivity. We have developed a fully differentiable framework capable of inferring and evolving highly detailed implicit geometries. We demonstrate that our pipeline is versatile, applicable to various optimization and dynamic tracking problems, and demonstrates lower memory consumption, fewer training iterations, and higher accuracy across different applications.

Limitation. While our approach has strengths, it also has limitations. The reliance on quadratic functions can lead to outliers, causing instability in surface representation and optimization tasks. Additionally, maintaining an adaptive sparse grid requires extra memory. While our use of weighted least squares offers speed advantages, alternative methods like gradient descent, seen in works such as 3D Gaussian[Kerbl et al. 2023], could offer similar efficiency. Hence, replacing this step with gradient descent could balance speed and stability, improving our framework.

Future work. Looking forward, our framework offers several promising avenues for future development. We plan to incorporate the feature particle radius into the optimization process, eliminating

Direct reconstruction	reinitialization	solving	projection + fixing	-
Avg Time(s)	2.17(only beginning)	0.32	0.29	-
Inverse rendering	reinitialization	compute gradient	advection	refinement
Avg Time(s)	8.42(every 20 step)	0.46	0.12	0.53

Table 8. Runtime performance breakdown for direct reconstruction and inverse rendering of our method, analyzed using the NVIDIA RTX A6000.

the need for an adaptive sparse grid and enhancing robustness. Additionally, we aim to use more expressive feature vectors beyond quadratic polynomials to better capture mixed-codimensional geometric features. Integrating our non-manifold representation into the optimization pipeline will enable effective optimization on non-manifold surfaces and any unsigned distance field, as demonstrated by meshUDF[Guillard et al. 2022], NeuralUDF[Long et al. 2023]. Finally, given the versatility of our feature particles in fitting various physical quantities, there is potential for applications in physical simulations, particularly in surface simulation.

ACKNOWLEDGMENTS

This project was supported by Toyota Central RD Labs., Inc.. Georgia Tech authors also acknowledge NSF IIS 2433322, ECCS 2318814, CAREER 2433307, IIS 2106733, OISE 2433313, and CNS 1919647 for funding support. We credit the Houdini education license for video animations.

REFERENCES

Jad Abou-Chakra, Feras Dayoub, and Niko Sünderhauf. 2023. ParticleNeRF: A Particle-Based Encoding for Online Neural Radiance Fields. arXiv:2211.04041 [cs.CV]

Samir Akkouche and Eric Galin. 2001. Adaptive Implicit Surface Polygonization Using Marching Triangles. *Computer Graphics Forum* 20, 2 (2001), 67–80. https://doi.org/10.1111/1467-8659.00479 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00479

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T Silva. 2001. Point set surfaces. In *Proceedings Visualization, 2001. VIS'01.* IEEE, 21–29.

Sergei Azernikov and Anath Fischer. 2005. Anisotropic meshing of implicit surfaces. In *International Conference on Shape Modeling and Applications 2005 (SMI'05).* IEEE, 94–103.

Markus Becker and Matthias Teschner. 2007. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 209–217.

Nathan Bell, Yizhou Yu, and Peter J Mucha. 2005. Particle-based simulation of granular materials. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 77–86.

Jules Bloomenthal and Chandrajit Bajaj. 1997. *Introduction to implicit surfaces.* Morgan Kaufmann.

Zhong Chen, Zhiwei Hou, Quanquan Yang, and Xiaobing Chen. 2018. Adaptive Meshing Based on the Multi-level Partition of Unity and Dynamic Particle Systems for Medical Image Datasets. *International Journal Bioautomation* 22, 3 (2018), 229.

Zhang Chen, Zhong Li, Liangchen Song, Lele Chen, Jingyi Yu, Junsong Yuan, and Yi Xu. 2023. Neurbf: A neural fields representation with adaptive radial basis functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 4182–4194.

Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. 2021. Differentiable surface rendering via non-differentiable sampling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 6088–6097.

Y. Deng, M. Wang, X. Kong, S. Xiong, Z. Xian, and B. Zhu. 2022. A Moving Eulerian-Lagrangian Particle Method for Thin Film and Foam Simulation. *ACM Trans. Graph.* 41, 4 (2022).

Tamal K Dey and Jian Sun. 2005. . An Adaptive MLS Surface for Reconstruction with Guarantees.. In *Symposium on Geometry processing.* 43–52.

Francçois Duranleau, Philippe Beaudoin, and Pierre Poulin. 2008. Multiresolution point-set surfaces. In *Proceedings of Graphics Interface 2008.* 211–218.

Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. 2002. A hybrid particle level set method for improved interface capturing. *Journal of Computational physics* 183, 1 (2002), 83–116.

Richard Franke and Greg Nielson. 1980. Smooth interpolation of large sets of scattered data. *International journal for numerical methods in engineering* 15, 11 (1980), 1691–1704.

Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* 249–254.

Frederic Gibou, Ronald Fedkiw, and Stanley Osher. 2018. A review of level-set methods and some recent applications. *J. Comput. Phys.* 353 (2018), 82–109.

Gaël Guennebaud and Markus Gross. 2007. Algebraic point set surfaces. In *ACM siggraph 2007 papers.* 23–es.

Benoit Guillard, Federico Stella, and Pascal Fua. 2022. MeshUDF: Fast and Differentiable Meshing of Unsigned Distance Field Networks. In *European Conference on Computer Vision.*

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084* (2020).

Simone E Hieber and Petros Koumoutsakos. 2005. A Lagrangian particle level set method. *J. Comput. Phys.* 210, 1 (2005), 342–367.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM transactions on graphics* 38, 6 (2019), 1–16.

Yuan Hu, Jingqi Yan, Wei Li, and Pengfei Shi. 2010. A novel facial localization for three-dimensional face using multi-level partition of unity implicits. In *2010 20th International Conference on Pattern Recognition.* IEEE, 682–685.

Xianping Huang, Qing Tian, Jianfei Mao, Li Jiang, and Ronghua Liang. 2010. Adaptive moving least squares for scattering points fitting. *WSEAS Transactions on Computers* 9, 6 (2010), 664–673.

Zhiyang Huang, Nathan Carr, and Tao Ju. 2019. Variational implicit point set surfaces. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–13.

Sandro Ianniello and Andrea Di Mascio. 2010. A self-adaptive oriented particles Level-Set method for tracking interfaces. *J. Comput. Phys.* 229, 4 (2010), 1353–1380.

Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 1251–1261.

Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, et al. 2024. VR-GS: A Physical Dynamics-Aware Interactive Gaussian Splatting System in Virtual Reality. *arXiv preprint arXiv:2401.16663* (2024).

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7.

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 1–13.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Dan Koschier, Crispin Deul, Magnus Brand, and Jan Bender. 2017. An hp-adaptive discretization algorithm for signed distance field generation. *IEEE transactions on visualization and computer graphics* 23, 10 (2017), 2208–2221.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular Primitives for High-Performance Differentiable Rendering. *ACM Transactions on Graphics* 39, 6 (2020).

David Levin. 2004. Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization.* Springer, 37–49.

Weitao Li, Yuanfeng Zhou, Caiming Zhang, and Xuemei Li. 2014. Robust multi-level partition of unity implicits from triangular meshes. *Computer Animation and Virtual Worlds* 25, 2 (2014), 115–127.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 7708–7717.

Shi-Lin Liu, Hao-Xiang Guo, Hao Pan, Peng-Shuai Wang, Xin Tong, and Yang Liu. 2021. Deep implicit moving least-squares functions for 3D reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 1788–1797.

Xiaoxiao Long, Cheng Lin, Lingjie Liu, Yuan Liu, Peng Wang, Christian Theobalt, Taku Komura, and Wenping Wang. 2023. Neuraludf: Learning unsigned distance fields for

multi-view reconstruction of surfaces with arbitrary topologies. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20834–20843.
Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2024. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis. In *3DV*.
Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2022. A Level Set Theory for Neural Implicit Evolution under Explicit Flows. *arXiv preprint arXiv:2204.07159* (2022).
Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2023. A Theory of Topological Derivatives for Inverse Rendering of Geometry. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 419–429.
Corentin Mercier, Thibault Lescoat, Pierre Roussillon, Tamy Boubekeur, and Jean-Marc Thiery. 2022. Moving level-of-detail surfaces. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–10.
Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127
Andrew Nealen. 2004. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. *URL: http://www. nealen. com/projects* 130, 150 (2004), 25.
Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large Steps in Inverse Rendering of Geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40, 6 (Dec. 2021). https://doi.org/10.1145/3478513.3480501
Tiago Novello, Vinicius Da Silva, Guilherme Schardong, Luiz Schirmer, Helio Lopes, and Luiz Velho. 2023. Neural Implicit Surface Evolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14279–14289.
Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2003. Multi-Level Partition of Unity Implicits. *ACM Trans. Graph.* 22, 3 (jul 2003), 463–470. https://doi.org/10.1145/882262.882293
Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. 2005. Multi-level partition of unity implicits. In *Acm Siggraph 2005 Courses*. 173–es.
Stanley Osher and James A Sethian. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of computational physics* 79, 1 (1988), 12–49.
Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019a. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 165–174.
Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019b. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
Mark Pauly, Leif P Kobbelt, and Markus Gross. 2006. Point-based multiscale surface representation. *ACM Transactions on Graphics (TOG)* 25, 2 (2006), 177–193.
Songyou Peng, Chiyu Jiang, Yiyi Liao, Michael Niemeyer, Marc Pollefeys, and Andreas Geiger. 2021. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems* 34 (2021), 13032–13044.
Marie-Julie Rakotosaona, Noam Aigerman, Niloy J Mitra, Maks Ovsjanikov, and Paul Guerrero. 2021. Differentiable surface triangulation. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–13.
Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. 2020. Accelerating 3D Deep Learning with PyTorch3D. *arXiv:2007.08501* (2020).
Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. 2020. Meshsdf: Differentiable iso-surface extraction. *Advances in Neural Information Processing Systems* 33 (2020), 22468–22478.
A. Ricci. 1973. A constructive geometry for computer graphics. *Comput. J.* 16, 2 (01 1973), 157–160. https://doi.org/10.1093/comjnl/16.2.157 arXiv:https://academic.oup.com/comjnl/article-pdf/16/2/157/1060001/160157.pdf
Naohisa Sakamoto, Jorji Nonaka, Koji Koyamada, and Satoshi Tanaka. 2007. Particle-based volume rendering. In *2007 6th International Asia-Pacific Symposium on Visualization*. IEEE, 129–132.
Marc Alexander Schweitzer. 2009. An adaptive hp-version of the multilevel particle–partition of unity method. *Computer methods in applied mechanics and engineering* 198, 13-14 (2009), 1260–1272.
Marc Alexander Schweitzer. 2011. Multilevel particle-partition of unity method. *Numer. Math.* 118, 2 (2011), 307–328.
Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers*. 1–11.
Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–12.
Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems* 34 (2021), 6087–6101.
Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–16.
Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. (2021).
Satoshi Tanaka, Kyoko Hasegawa, Yoshiyuki Shimokubo, Tomonori Kaneko, Takuma Kawamura, Susumu Nakata, Saori Ojima, Naohisa Sakamoto, Hiromi T Tanaka, and Koji Koyamada. 2012. Particle-Based Transparent Rendering of Implicit Surfaces and its Application to Fused Visualization.. In *EuroVis (Short Papers)*. 35–29.
Gokul Varadhan, Shankar Krishnan, TVN Sriram, and Dinesh Manocha. 2004. Topology preserving surface extraction using adaptive subdivision. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. 235–244.
Magnus Vartdal and Arne Bøckmann. 2013. An oriented particle level set method based on surface coordinates. *Journal of computational physics* 251 (2013), 237–250.
Delio Vicini, Sébastien Speierer, and Wenzel Jakob. 2022. Differentiable Signed Distance Function Rendering. *Transactions on Graphics (Proceedings of SIGGRAPH)* 41, 4 (July 2022), 125:1–125:18. https://doi.org/10.1145/3528223.3530139
Chun-Xia Xiao. 2011. Multi-level partition of unity algebraic point set surfaces. *Journal of Computer Science and Technology* 26, 2 (2011), 229–238.
Tianyi Xie, Zeshun Zong, Yuxin Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. 2023. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *arXiv preprint arXiv:2311.12198* (2023).
Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. 2019. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.
Jihun Yu and Greg Turk. 2013. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Transactions on Graphics (TOG)* 32, 1 (2013), 1–12.
Lanhao Zhao, Hongvan Khuc, Jia Mao, Xunnan Liu, and Eldad Avital. 2018. One-layer particle level set method. *Computers & Fluids* 170 (2018), 141–156.
Zichun Zhong, Xiaohu Guo, Wenping Wang, Bruno Lévy, Feng Sun, Yang Liu, Weihua Mao, et al. 2013. Particle-based anisotropic surface meshing. *ACM Trans. Graph.* 32, 4 (2013), 99–1.
Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

A CODE

Our code is available at https://jinjinhe2001.github.io/diffmpu-page/index.html.

B TRAINING DATA

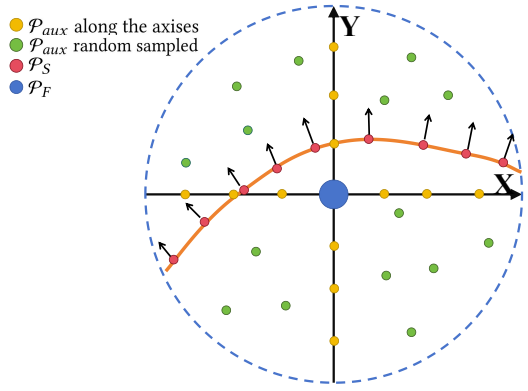

Fig. 24. Our feature particles incorporate various auxiliary points for polynomial approximation.

For the optimizing purpose, we generate our training data points with signed distance values interpolated from sample particles. As

shown in Fig. 24 these training points include: 1) all sample particles' positions with 0 distance value, 2) auxiliary points that are generalized at distance from $\pm\frac{1}{4}, \pm\frac{1}{2}, \pm\frac{3}{4}$ along the three axes of the supporting radius, and 3) randomly generalized points in supporting radius with distance value interpolated from its nearby sample particle. To calculate each auxiliary point $\mathbf{q}$'s distance to the surface, we find its k nearest sample particle neighbors $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_k$ with their normal vectors $\mathbf{n}_1, \mathbf{n}_2, \cdots, \mathbf{n}_k$, k here we select ranging from 1 to 10 in experiments, then the distance $D(\mathbf{q})$ can be interpolated as,

$$D(\mathbf{q}) = \frac{1}{k}\sum_{i=1}^{k} \mathbf{n}_i(\mathbf{q} - \mathbf{p}_i). \tag{15}$$

To obtain a reliable signed distance function, we discard auxiliary points whose nearest k signed distance values exhibit divergent signs.

C WEIGHTED LEAST SQUARE

In weighted least square optimization, we conduct optimization to minimize the error function from Eqution 6:

$$\min_{\beta} \mathcal{L} = \frac{1}{\sum w(\mathbf{p}_i)} \sum w(\mathbf{p}_i) f(\mathbf{p}_i)^2 + \frac{1}{m}\sum_{i=1}^{m} \left(f(\mathbf{q}_i) - D(\mathbf{q}_i)\right)^2, \tag{16}$$

According to Eq. 3, the error function can be written as:

$$\mathcal{L} = \frac{1}{\sum w(\mathbf{p}_i)} \sum w(\mathbf{p}_i)(\mathbf{b}(\mathbf{p}_i - \mathbf{c})^\top \beta)^2 + \frac{1}{m}\sum_{i=1}^{m} \left(\mathbf{b}(\mathbf{q}_i - \mathbf{c})^\top \beta - D(\mathbf{q}_i)\right)^2, \tag{17}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \beta} &= \frac{1}{\sum w(\mathbf{p}_i)} \sum 2w(\mathbf{p}_i)\mathbf{b}(\mathbf{p}_i - \mathbf{c})\mathbf{b}(\mathbf{p}_i - \mathbf{c})^\top \beta \\ &+ \frac{1}{m}\sum_{i=1}^{m} 2(\mathbf{b}(\mathbf{q}_i - \mathbf{c})(\mathbf{b}(\mathbf{q}_i - \mathbf{c})^\top \beta - D(\mathbf{q}_i)),\end{aligned} \tag{18}$$

To minimize the error function, we need to get β when $\frac{\partial \mathcal{L}}{\partial \beta} = 0$, then:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \beta} &= \frac{1}{\sum w(\mathbf{p}_i)} \sum 2w(\mathbf{p}_i)\mathbf{b}(\mathbf{p}_i - \mathbf{c})\mathbf{b}(\mathbf{p}_i - \mathbf{c})^\top \beta \\ &+ \frac{1}{m}\sum_{i=1}^{m} 2(\mathbf{b}(\mathbf{q}_i - \mathbf{c})(\mathbf{b}(\mathbf{q}_i - \mathbf{c})^\top \beta - D(\mathbf{q}_i)) \\ &= 0,\end{aligned} \tag{19}$$

next we have

$$\begin{aligned}&[\frac{1}{\sum w(\mathbf{p}_i)} \sum 2w(\mathbf{p}_i)\mathbf{b}(\mathbf{p}_i - \mathbf{c})\mathbf{b}(\mathbf{p}_i - \mathbf{c})^\top \\ &+ \frac{1}{m}\sum_{i=1}^{m} 2\mathbf{b}(\mathbf{q}_i - \mathbf{c})(\mathbf{b}(\mathbf{q}_i - \mathbf{c})^\top]\beta \\ &= \frac{1}{m}\sum_{i=1}^{m} 2\mathbf{b}(\mathbf{q}_i - \mathbf{c})D(\mathbf{q}_i),\end{aligned} \tag{20}$$

and solve we can solve coefficients by:

$$\mathbf{c} = \mathbf{A}^{-1}\mathbf{B}, \tag{21}$$

where

$$\begin{aligned}\mathbf{A} &= \frac{1}{\sum w(\mathbf{p}_i)} \sum 2w(\mathbf{p}_i)\mathbf{b}(\mathbf{p}_i - \mathbf{c})\mathbf{b}(\mathbf{p}_i - \mathbf{c})^\top \\ &+ \frac{1}{m}\sum_{i=1}^{m} 2\mathbf{b}(\mathbf{q}_i - \mathbf{c})\mathbf{b}(\mathbf{q}_i - \mathbf{c})^\top, \\ \mathbf{B} &= \frac{1}{m}\sum_{i=1}^{m} 2\mathbf{b}(\mathbf{q}_i - \mathbf{c})D(\mathbf{q}_i),\end{aligned}$$

Since both $\mathbf{A}$ and $\mathbf{B}$ are low-dimensional matrices and vectors(10×10 and 10×1 for 3D), we can optimize the local shape at a low computational cost least square for each feature particle.

D CURVATURE CALCULATION ON QUADRATIC FUNCTION

To compute the curvature from our feature particles, we start from Eq. 3 and Eq. 14,

$$\begin{aligned}\kappa &= \nabla \cdot \frac{\nabla \mathbf{b}^\top \beta}{|\nabla \mathbf{b}^\top \beta|} \\ &= \frac{\partial(\frac{\nabla \mathbf{b}^\top \beta}{|\nabla \mathbf{b}^\top \beta|})_x}{\partial x} + \frac{\partial(\frac{\nabla \mathbf{b}^\top \beta}{|\nabla \mathbf{b}^\top \beta|})_y}{\partial y} + \frac{\partial(\frac{\nabla \mathbf{b}^\top \beta}{|\nabla \mathbf{b}^\top \beta|})_z}{\partial z}\end{aligned} \tag{22}$$

, where $\partial(\frac{\nabla \mathbf{b}^\top \beta}{|\nabla \mathbf{b}^\top \beta|})_x$ is the component along the x-axis, and y,z is the same. In the 3D case, let's assume $\beta = (\beta_0, \beta_1, \ldots, \beta_9)$. set $\nabla \mathbf{b}^\top \beta = \mathbf{G} = (G_x, G_y, G_z)$ which actually is the local quadratic function's gradient, we can get

$$\begin{aligned}G_x &= 2\beta_0 \cdot x + \beta_3 \cdot y + \beta_5 \cdot z + \beta_6 \\ G_y &= 2\beta_1 \cdot y + \beta_3 \cdot x + \beta_4 \cdot z + \beta_7 \\ G_z &= 2\beta_2 \cdot z + \beta_4 \cdot y + \beta_5 \cdot x + \beta_8,\end{aligned} \tag{23}$$

and $|\nabla \mathbf{b}^\top \beta| = \sqrt{G_x^2 + G_y^2 + G_z^2} = |\mathbf{G}|$, next we expansion Eq. 23,

$$\begin{aligned}\kappa &= \frac{\partial(\frac{G_x}{|\mathbf{G}|})}{\partial x} + \frac{\partial(\frac{G_y}{|\mathbf{G}|})}{\partial y} + \frac{\partial(\frac{G_z}{|\mathbf{G}|})}{\partial z} \\ &= \frac{\partial G_x}{\partial x} \cdot \frac{1}{|\mathbf{G}|} - \frac{G_x}{|\mathbf{G}|^2} \cdot \frac{\partial |\mathbf{G}|}{\partial x} \\ &+ \frac{\partial G_y}{\partial y} \cdot \frac{1}{|\mathbf{G}|} - \frac{G_y}{|\mathbf{G}|^2} \cdot \frac{\partial |\mathbf{G}|}{\partial y} \\ &+ \frac{\partial G_z}{\partial z} \cdot \frac{1}{|\mathbf{G}|} - \frac{G_z}{|\mathbf{G}|^2} \cdot \frac{\partial |\mathbf{G}|}{\partial z}\end{aligned} \tag{24}$$

Then we get

$$\begin{aligned}\kappa = \frac{1}{|\mathbf{G}|^2}(&\frac{\partial G_x}{\partial x}|\mathbf{G}| + \frac{\partial G_y}{\partial y}|\mathbf{G}| + \frac{\partial G_z}{\partial z}|\mathbf{G}| \\ &- \frac{\partial |\mathbf{G}|}{\partial x}G_x - \frac{\partial |\mathbf{G}|}{\partial y}G_y - \frac{\partial |\mathbf{G}|}{\partial z}G_z)\end{aligned} \tag{25}$$

Given that κ in Eq. (25) is entirely expressed by $\mathbf{G}$, and combined with Eq. (23), we can derive the local curvature.

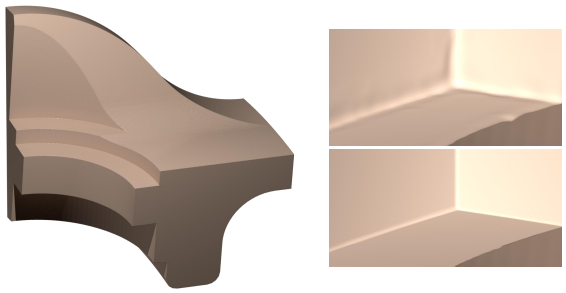

Fig. 26. Sharp features characterized by our representation. **Left**: fandisk reconstructed by our method. **Right top**: result without group partition. **Right bottom**: result with group partition.

E SHARP FEATURES

As the polynomial quadratic approach typically smooths the presence of sharp features on the implicit surface, we adopt a similar strategy employed by Ohtake et al. [2003] to tackle this concern. This involves the automatic recognition of faces, edges, and corners among the feature particles with the finest layer radius. For each feature particle in the finest layer, given the neighbor sample particles $\{\mathbf{p}\}_{i=1}^{k}$ with normals $\{\mathbf{n}\}_{i=1}^{k}$ in its supporting radius, we identify the local implicit surface of the feature particle as having a sharp feature if $\min_{i,j}(\mathbf{n}_i \cdot \mathbf{n}_j) < \varepsilon_1$, where ε_1 is a user-defined threshold that we set to $\varepsilon_1 = 0.9$ in our experiments. By recording the two normals $\mathbf{n}_1$ and $\mathbf{n}_2$ of the minimal product, the sharp feature is identified as a corner if $\max_i |\mathbf{n}_i \cdot (\mathbf{n}_1 \times \mathbf{n}_2)| > \varepsilon_2$, where ε_2 is a threshold set to 0.7; otherwise, it is identified as an edge.

We then partition the sample particles within the radius of this feature particle into two groups for edges or three groups for corners, using spherical Voronoi subsets as illustrated in Figure 25. Each group is fitted to a polynomial parameter. Non-smooth implicit surfaces are generated using intersections, as described by Ricci [1973]. Fig. 26 shows the comparison results. We delay applying the partition in the initial stages of differentiable optimization applications because the shape isn't finalized. Such partitioning could introduce discontinuities in surface normals or curvature, harming the differentiable application. We postpone this step until the final iterations when the shape approaches the target.

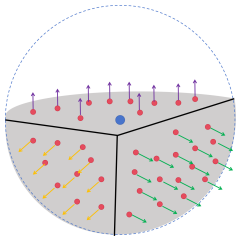

Fig. 25. Our sharp group partition, with different colored normals indicating different groups.

F NON-MANIFOLD FEATURES

Our moving particle representation can handle non-manifold shapes such as unsigned distance functions and junctions. For the unsigned distance function, converting the original signed function Eq. (3) is straightforward using an absolute operation,

$$\mathcal{F}(\mathbf{x}) = |\mathbf{b}(\mathbf{x}-\mathbf{c})^{\top}\beta|. \tag{26}$$

To solve the correct polynomial vector of the non-manifold shape, we capitalize on the advantages offered by the group partitioning method of sharp feature recognition and sample particles. Regarding the junctions depicted in Fig. 27, where the normals of sample particles may point in opposite directions on the junction edge and corner, we adapt previous edge detection to recognize $\mathbf{n}_i$ and $\mathbf{n}_j$ within the same group for unsigned distance implicit surface if they exhibit nearly opposite directions, $min_{i,j}(\mathbf{n}_i \cdot \mathbf{n}_j) < -\varepsilon_1$, and if $|\mathbf{p}_i - \mathbf{p}_j| < 0.01 r_0$, where r_0 is the smallest radius of feature particles.

In polynomial approximation, for each group of sample particles, we designate one sample particle's normal as the principal direction for the group. Then, each sample particle within this group exhibiting an opposite direction utilizes the opposite normal in Eq. (3) and Eq. (6), and thus feature particles cloud approximate correct unsigned distance value for non-manifold junctions.

We solely utilize Eq. 26 to represent unsigned distances, as illustrated in Experiments 9.1.1, demonstrating how our representation can accommodate non-manifold junctions if required. However, in other experiments, such junctions are considered errors to be addressed. We will discuss methods for detecting and rectifying non-manifold shapes in Sec. 7.3.

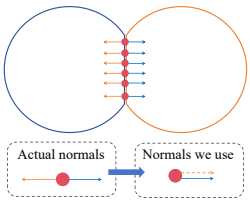

Fig. 27. unsigned distance representation principal sample orientation scheme used on the non-manifold face.