A Neural Particle Level Set Method for Dynamic Interface Tracking

DUOWEN CHEN, Georgia Institute of Technology, Atlanta, USA
JUNWEI ZHOU, University of Michigan-Ann Arbor, Ann Arbor, USA
BO ZHU, Georgia Institute of Technology, Atlanta, USA

We propose a neural particle level set (Neural PLS) method to accommodate tracking and evolving dynamic neural representations. At the heart of our approach is a set of oriented particles serving dual roles of interface trackers and sampling seeders. These dynamic particles are used to evolve the interface and construct neural representations on a multi-resolution grid-hash structure to hybridize coarse sparse distance fields and multi-scale feature encoding. Based on these parallel implementations and neural-network-friendly architectures, our neural particle level set method combines the computational merits on both ends of the traditional particle level sets and the modern implicit neural representations, in terms of feature representation and dynamic tracking. We demonstrate the efficacy of our approach by showcasing its performance surpassing traditional level-set methods in both benchmark tests and physical simulations.

CCS Concepts: • **Computing methodologies** → **Physical simulation;**

Additional Key Words and Phrases: Neural representation, level set, interface tracking, physically-based simulation

ACM Reference Format:
Duowen Chen, Junwei Zhou, and Bo Zhu. 2025. A Neural Particle Level Set Method for Dynamic Interface Tracking. *ACM Trans. Graph.* 44, 3, Article 23 (May 2025), 21 pages. https://doi.org/10.1145/3730399

1 Introduction

Dynamic interface tracking has been a critical research problem in many visual and scientific computing applications. Examples include tracking complex fluid surfaces, generating structural designs, and segmenting medical images, just to name a few. Level set methods serve as a fundamental tool in accommodating these applications. Since its inception in [Osher and Sethian 1988], level sets have proven to be highly effective in tackling dynamic interfaces exhibiting complex geometrical and topological characteristics. The key concept behind a level set representation is to describe a codimension-k interface using a **signed distance field** (**SDF**) defined in a codimension-(k-1) space. In a typical interface tracking scenario, this SDF information is discretized on a Cartesian grid and evolved with its background velocity by adhering to the distance field constraints. Traditional level set methods manifest two aspects of limitation.

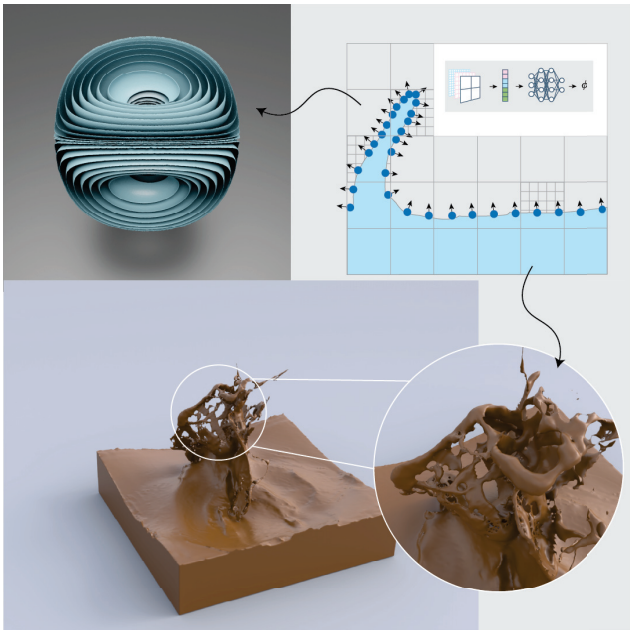


Fig. 1. We introduce a novel neural-discrete hybrid method for dynamic interface tracking that combines Instant-NGP with a sparse grid (the top-right inset picture). By utilizing this approach, we are able to accurately track and preserve surface details even in the presence of high-distortion velocity fields (the top-left inset picture). To showcase the effectiveness of our method, we demonstrate its performance in fluid simulation scenarios. Specifically, we employ an incompressible fluid solver operating on a 256^3 velocity field and illustrate the successful recovery of fine fluid structures represented by our level set representation (the bottom inset picture). Through the integration of Instant-NGP with a sparse grid, our method offers a robust and efficient solution for dynamic interface tracking, making it applicable to various fluid simulation applications and related domains.

First, enhancing the *geometry resolution* of a level set is expensive. On a grid discretization, the highest level of detail that a level set model can capture is restricted by the size of its grid cell. Naively refining the cell size of a uniform grid to refine the level set will drastically boost the computational cost. Second, *feature preservation* when tracking a dynamic interface is challenging. Grid-based advection and reinitialization cause feature smear-out during an SDF's temporal evolution. Maintaining auxiliary structures near the interface, e.g., the **particle level set** (**PLS**) method [Enright et al. 2002], comes with extra implementation complexities and nontrivial computational expenses.

The recent advances in **implicit neural representations** (**INRs**) in visual computing communities (see [Xie et al. 2022] for

Georgia Tech authors also acknowledge NSF IIS #2433322, ECCS #2318814, CAREER #2433307, IIS #2106733, OISE #2433313, and CNS #1919647 for funding support.
Author's Contact Information: Duowen Chen, Georgia Institute of Technology, Atlanta, Georgia, USA; e-mail: dchen322@gatech.edu; Junwei Zhou, University of Michigan-Ann Arbor, Ann Arbor, Michigan, USA; e-mail: zjw330501@gmail.com; Bo Zhu, Georgia Institute of Technology, Atlanta, Georgia, USA; e-mail: bo.zhu@gatech.edu.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2025 Copyright held by the owner/author(s).
ACM 0730-0301/2025/05-ART23
https://doi.org/10.1145/3730399

a comprehensive survey) open up a new door for characterizing complicated implicit geometry. The core idea of a neural representation is to use a neural network to encode a geometry's implicit **signed distance function** (**SDF**). Specifically, the neural network takes a position vector (x, y, z) as input and produces a scalar ϕ for its SDF value. The neural network is trained by taking sufficient point samples with correct ϕ values sampled around the interface. The most salient advantage of a neural SDF representation is its decoupling of spatial discretization and representative resolution. By encoding a level set function in a network directly, an INR can characterize shapes with a (theoretically) infinite resolution by nature, without resorting to complex adaptive structures such as octrees or sparse grids (e.g., see recent progress in [Martel et al. 2021; Sitzmann et al. 2020; Takikawa et al. 2021; Tancik et al. 2020]).

Despite the inspiring progress made in static geometry representation, tracking highly dynamic interfaces with an INR remains challenging. On one side, it is difficult to advect an implicit neural field without an explicit spatial discretization. Traditional advection schemes such as semi-Lagrangian [Stam 1999] and BFECC [Dupont and Liu 2003] rely on a grid structure to traverse the space and update the SDF values around the moving interface. Such discretization is unavailable for a neural representation. On the other side, preserving interface features during the spatiotemporal evolution of a level set is difficult. Directly transporting the level set's implicit field with an advection loss term [Chen et al. 2023b] ignores its SDF nature and is, therefore, inefficient in keeping its geometric features. Moreover, updating the network structures and parameters every time when the interface moves is expensive. It is costly to generate new sample points, update network weights, and enforce the distance constraints within a plausible period of time. These difficulties jointly render the neural representations less attractive in tackling dynamic interface problems when compared with their grid-based cousins and restrict their applications mostly within the scope of tackling static geometries.

To tackle these challenges, we propose a neural PLS method to enable highly dynamic interface tracking for INRs. The key idea of our approach is inspired by the renowned PLS method [Enright et al. 2002], in which a group of marker particles is tracked around the moving interface to enhance its feature preservation on the background grid. Motivated by the role particles play in PLS, we maintain a set of oriented particles carrying normal information on the interface to simultaneously facilitate dynamic tracking and network training of a moving neural interface. These oriented particles play a *dual role* in our representation, both as *particle trackers* to characterize local geometric features and as *training samplers* to seed training samples, which jointly capture the complicated geometric and topological features during the evolution of a neural interface. On the neural representation side, we devised a *multi-resolution hash structure* to encode neural distance values near the interface, augmented by a *sparse grid structure* to maintain distance information far from the interface on a coarse level aiming at reducing the training cost. Regarding implementation, our method can seamlessly integrate the Taichi sparse structures [Hu et al. 2019] and the **Instant Neural Graphics Primitives** (**I-NGP**) framework [Müller et al. 2022] to leverage their optimal memory access and rapid network training capabilities on GPU.

Our neural PLS combines the computational merits of both traditional PLSs and modern INRs. On one hand, our method fully inherits the capability to track and preserve thin and sharp features, thanks to the Lagrangian nature of our particle trackers. On the other hand, our framework extends the adaptive expressiveness of neural representations to characterize dynamic implicit surface evolution, without requiring complex adaptivity mechanisms. We demonstrated the capabilities of our method in terms of high-resolution tracking and feature preservation by comparing it against traditional PLS methods in various benchmark tests and physics simulation scenarios. To the best of our knowledge, our method has proven, for the first time, that a neural representation can rival the traditional level set method in terms of both representation accuracy and dynamic tracking capability.

We summarize the main contributions of our method as follows:

- A novel hybrid neural-discrete data structure based on multi-resolution hash encoding and sparse grids to represent dynamic level set functions with adaptivity.
- An efficient oriented particle representation that decouples interface location and orientation representation to facilitate fast network training.
- A unified neural PLS framework to accommodate large-scale dynamic interface tracking applications.

2 Related Work

Level set methods [Osher and Fedkiw 2005; Osher and Sethian 1988] address the spatiotemporal evolution of an implicit function. This approach can be computationally expensive due to the need to store abundant spatial samples near the interface and to reinitialize the distance field. Researchers have put forward a multitude of works for acceleration, which can be grouped into three main categories exploring *adaptive and sparse grids*, *particle methods*, and *high-order augmentations*. The recent trends on *INRs* and their *dynamic representations* add a new dimension to the problem. We will survey these works as below.

Adaptive and sparse grids. Adaptive signed distance field representations were introduced to graphics in [Frisken et al. 2000], which has later been extended by a plethora of novel data structures such as Octree [Losasso et al. 2004], sparse grids [Museth 2013; Setaluri et al. 2014], hybrid height fields [Chentanez and Müller 2011; Irving et al. 2006], Power diagram [Aanjaneya et al. 2017], hybrid grid-particle structures [Gao et al. 2017], and the very recent INRs [Kim et al. 2024; Müller et al. 2022], to name just a few examples. These data representations have been adapted to accommodate high-resolution interface tracking applications (e.g., fluid simulation) characterized by very detailed interfacial geometry and dynamics (e.g., [Bojsen-Hansen and Wojtan 2013; Goldade et al. 2016]). However, traditional adaptive data structures usually suffer from their implementation complexities and parallelization difficulties.

Particle methods. Pioneered by the PLS method [Enright et al. 2002], adding particles to enhance the numerical accuracy of interface tracking has been one of the common practices in both computer graphics and computational physics. Various kinds of PLS extensions have been investigated, exemplified by the single-layer

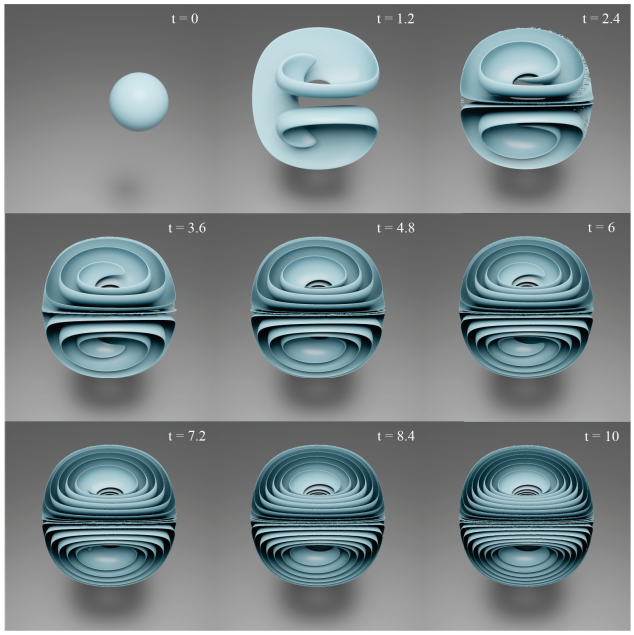


Fig. 2. A sphere is evolved in a 3D deformation field for a long duration to obtain extremely thin films using our neural PLS method. Similar results have been previously obtained using a supercomputer in [Gibou et al. 2018], and we have achieved comparable results using a workstation equipped with a single Nvidia Quadro RTX8000 GPU.

PLS [Ianniello and Di Mascio 2010; Vartdal and Bøckmann 2013], pure Lagrangian method [Hieber and Koumoutsakos 2005], and hybrid grid-particle structures [Ferstl et al. 2016; Leung et al. 2011; Leung and Zhao 2009a, b; Sato et al. 2018; Zhao et al. 2018]. The one that is most relevant to our approach is the **self-adaptive oriented particles level set** (**AOPLS**) method [Ianniello and Di Mascio 2010], in which oriented particles were adopted to accommodate accurate interface tracking. In computer graphics, pure Lagrangian representations, such as **Moving Least-Squares** (**MLS**) particles [Wang et al. 2020], **Smoothed Particle Hydrodynamics** (**SPH**) surface [Wang et al. 2021], and **Moving Eulerian–Lagrangian Particle** (**MELP**) Method [Deng et al. 2022], have been explored to track interface without evolving a distance field, reminiscent of the traditional Lagrangian front-tracking methods [Brochu and Bridson 2009; Da et al. 2014; Wojtan et al. 2011] in fluid simulation.

High-order representations. Besides discrete particles, researchers also leverage extra geometric fields or high-order polynomials to enhance the interface representation. Gradient fields are one of these common choices (e.g., see [Bøckmann and Vartdal 2014; Nave et al. 2010]). These approaches evolve gradient fields of the level set to compensate for the advection error. Similar concepts have been recently introduced to computer vision by Sommer et al. [2022], which combines implicit voxel-based SDFs and explicit gradient information to improve the accuracy of surface normal estimation. High-order polynomials (e.g., Hermite interpolation in [Nave et al. 2010]) are typically used within these gradient-augmented schemes to enhance the local subcell geometric representation. Examples of using these polynomials include [Saye 2014] and [Heo and Ko 2010], which employed high-order polynomials to feature distance fields in local grid cells. Recently, polynomial methods have also been explored to enforce divergence-free subcell velocity field [Chang et al. 2022; Schroeder et al. 2022]. Our method was inspired by these high-order methods (in particular [Saye 2014]) to maintain a continuous representation (in our case lightweight neural network) to enhance local geometric features.

INR. Neural implicit representations, pioneered by a series of neural SDF works [Chen and Zhang 2019; Mescheder et al. 2019; Park et al. 2019], leverage neural networks to represent implicit shapes through regression over directly sampled points. In recent years, there has been a proliferation of research projects exploring the diverse applications of neural representations in visual computing [Chen et al. 2023b; Hertz et al. 2021; Park et al. 2021; Sitzmann et al. 2020; Tancik et al. 2020] (for a survey, see [Xie et al. 2022]). One important focus in this area has been on accelerating algorithms through sparsity, adaptivity, and signal encoding. Notably, several recent works [Chabra et al. 2020; Jiang et al. 2020; Müller et al. 2022; Peng et al. 2020; Takikawa et al. 2021] have made significant contributions in this direction. Of particular relevance is the work by Müller et al. [2022], which introduced a multi-resolution spatial hashing encoding and a lightweight neural network to represent complex fields. The recent proposition of adopting particle structures to enhance scene encoding and training is noteworthy. The concept of Gaussian Splatting was initially integrated into implicit representation in the work of Kerbl et al. [2023], demonstrating that oriented information from 3D Gaussians could be directly employed to encode high-fidelity scenes. This approach has been further developed for dynamic settings by Luiten et al. [2024]. Additionally, the combination of particles with **Radial Basis Functions** (**RBFs**) has been proven to aid in the convergence and accuracy of I-NGP, as shown in [Chen et al. 2023].

Building upon these approaches, we propose our neural PLS method, which incorporates particles to address the challenges associated with complex interface tracking.

Dynamic neural representation. While significant progress has been made in static representations using neural networks, there has been a relatively scarce exploration of dynamic tracking. Some pioneering research in this area includes the work by Atzmon et al. [2019], which uses samples attached to model parameters for level set deformation, and the recent article by Novello et al. [2023], which proposes learning implicit surface movements over a continuous interval of time using a single network. Recently, Chen et al. [2023b] proposed a generic framework to evolve implicit fields by incorporating PDE losses in network training, which share similar ideas to [Chu et al. 2022]. Among these efforts, we highlight an inspiring recent work by Mehta et al. [2022], which allows applying deformation operations onto a neural implicit surface. They leveraged a Marching Cubes algorithm to generate an explicit surface mesh, enabling the use of differential operators that drive the interface evolution, and demonstrated applications in differentiable geometry processing and rendering. However, this approach has limitations when dealing with challenging scenarios involving highly

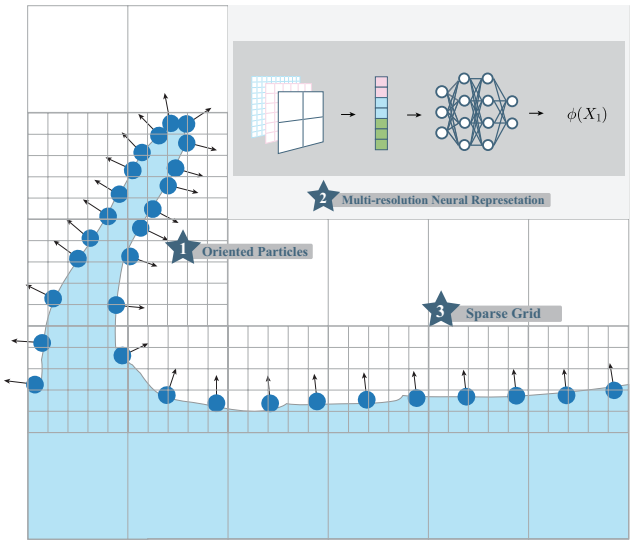


Fig. 3. Our representation consists of three parts: a group of oriented particles, a multi-resolution neural representation, and a sparse grid.

distorted interfaces, substantial advection, or frequent topological transitions, which are the focus areas of our proposed method.

3 Neural Level Set Representation

Problem description. We aim at devising a data structure by hybridizing particles, a sparse grid, and an INR to describe a narrow-band, high-resolution signed distance field. Given a query position $\boldsymbol{x}$, the data structure returns the accurate signed distance value $\phi(\boldsymbol{x})$ if $\boldsymbol{x}$ is within the narrow band specified by distance ϵ around the zero level set; and it returns $+\epsilon$ or $-\epsilon$ with the correct sign to distinguish the positive/negative side if $\boldsymbol{x}$ is outside the narrow band.

Data structure overview. As shown in Figure 3, our data structure comprises three main components: a set of oriented particles on the tracked interface, a multi-resolution hash structure implemented with I-NGP [Müller et al. 2022], and a sparse grid structure for far-field information. These components work together to support distance query operations (discussed in this section) and dynamic tracking of an evolving interface (covered in Section 4). The oriented particles serve as both dynamic interface trackers and training sample seeders. The multi-resolution hash structure efficiently encodes fine distance values, while the sparse grid structure maintains information on a coarser level, extending further from the interface. By combining these three components, we achieve effective distance queries and enable accurate tracking of the evolving interface.

3.1 Oriented Particles

Our approach involves the maintenance of a group of particles precisely located on the interface. Each particle stores its position and the corresponding normal vector.These oriented particles play two important roles in our method: dynamic interface trackers and network training seeders. As dynamic interface trackers, these particles move in accordance with the background velocities, ensuring

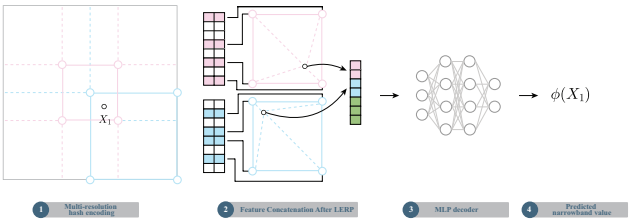


Fig. 4. Detailed illustration of I-NGP structure which consists of three parts: a multi-resolution hash table lookup, a linear interpolation of feature vectors, and an MLP decoder.

accurate tracking of the interface's position as it evolves over time. Simultaneously, these particles, along with their updated normals, serve as key seeding inputs for network training, enabling the generation of an implicit level set representation (see Section 3.4). By effectively combining the functionalities of dynamic interface tracking and network training, our method leverages the information carried by these oriented particles to achieve an accurate and adaptive implicit level set representation.

Unlike the traditional level set method [Enright et al. 2002], which maintains a narrow band of marker particles on both sides of the interface, our method employs a single layer of particles, augmented by their normals, to track the interface's evolution. This design choice is motivated by a fundamental principle in level set theory—the Lagrangian nature of a material interface. According to this principle, all material points initially located on the interface will remain on the same interface throughout its evolution, expressed mathematically as $D\boldsymbol{X}/Dt = 0$, where $\boldsymbol{X}$ denotes the position of a material point on the interface. Leveraging this Lagrangian property, we utilize these material particles as markers that continuously move with the interface. These markers will further serve as seeders to generate points near the interface with different distance values to accommodate network training, which effectively facilitates network training and seamlessly integrates with our neural representation. It is important to note that the idea of using a single layer of oriented particles to improve the traditional PLS method originates from the work of [Ianniello and Di Mascio 2010] in computational physics. Subsequent research [Vartdal and Bøckmann 2013; Zhao et al. 2018] has further extended this idea to improve its numerical robustness and geometric expressiveness. Our approach follows this thread of work. In contrast to previous methods (e.g., Ianniello et al. [Ianniello and Di Mascio 2010]), which often relied on local polynomial fitting or particle correction for updating level set values, our approach utilizes oriented particles as sample seeders to establish a connection between discrete trackers and the continuous functions of our hybrid representation.

3.2 Multi-resolution Neural Representation

To describe the level set function based on oriented particles, we adopt an implicit multi-resolution neural representation inspired by the I-NGP method proposed by Müller et al. [2022]. This neural representation employs a multi-resolution architecture that allows for sparse and adaptive feature encoding. It operates by taking sampling points primarily seeded by the oriented particles and training an implicit SDF.

Network architecture. Following Müller et al. [2022], our network architecture consists of a multi-resolution spatial hash-table encoder and a **Multi-Layer Perceptron** (**MLP**) decoder. As illustrated in Figure 4, the multi-resolution structure consists of L levels of spatial hashing tables to store the multi-resolution features of the target field in a local and sparse manner. On each level, the spatial hashing entries are hashed voxel node coordinates, and their values are the encoded feature vectors. For a given point $\mathbf{x}$, it first finds each of the 8 (for 3D) or 4 (for 2D) node coordinates and then converts them to a hash-table entry. The feature vector for point $\mathbf{x}$ on a specific level is then acquired through a linear interpolation of feature vectors of the voxel nodes. Feature vectors on different levels are concatenated together and passed to the MLP to obtain the predicted signed distance value.

3.3 Sparse Grid

To complement the neural representation and augment distance queries that are far from the interface (namely, for query points $\boldsymbol{x}$ with $|\phi(\boldsymbol{x})| > \epsilon$), we maintain a two-layer sparse grid structure in the background. The purpose of this grid is to store the correct side information (i.e., interior or exterior) for regions that are not within the narrow band of the neural representation. In particular, for a point that is outside the narrow band, indicating that our neural representation cannot predict an accurate distance value, the sparse grid should replace the network prediction by $-\epsilon$ if it is inside the domain and $+\epsilon$ if it is outside. We implement this feature with our two-layer sparse grid. The code infrastructure for sparse memory access is based on the Taichi Programming Language [Hu et al. 2019]. Next, we will briefly describe the grid structure's high-level design and how it complements the SDF query for I-NGP for points that are far from the interface.

Grid structure. Our grid structure consists of two levels—a coarse level and a fine level. We maintain a uniform grid on the coarse level. Each cell on this layer is marked by one of the three tags: *interface*, *exterior*, and *interior*. A cell is marked as an *interface* cell if it contains at least one oriented particle. Otherwise, it will be marked *exterior* if it is outside the level set domain and *interior* if it is inside. An interface cell on the coarse level will be further divided into 8^3 fine cells. For each fine cell, if it is within the narrow band of the interface, it is marked as *interface*; otherwise, it is marked as *exterior* or *interior* according to whether it is outside the level set domain or not.

SDF query. For a query point $\boldsymbol{x}$, we first check its tag on the coarse level: if the point is in an *exterior/interior* coarse cell, we directly return $(\pm)\epsilon$ as its SDF value. Otherwise, we further check the point's tag on the fine level: if it is in an *exterior/interior* fine cell, we directly return $(\pm)\epsilon$; otherwise, we return $\phi(\boldsymbol{x})$ by querying from I-NGP.

Storage. We store (1) an SDF value as a floating-point number within $[-\epsilon, +\epsilon]$), and (2) a bitmask as a Taichi built-in variable [Hu et al. 2019], which is used to mark if a cell's type is *interface*. Based on these two fields, we implement the following mechanism to judge a cell's type: A cell is an *interface* cell if its bit mask is true; it is judged as an *exterior/interior* cell if its bit mask is false and the sign of its SDF value is $+/-$. We note that the motivation of directly storing a sparse SDF field on the sparse grid is particularly beneficial for dynamic tracking, in which case the SDF values (at least their signs) need to be updated according to the background velocity, which is beyond the capability of the narrow-band neural representation. In this sense, storing a floating-point number can seamlessly accommodate existing advection schemes essential for dynamic interface updates. The stored SDF values will be corrected using the trained I-NGP module to guarantee consistency. Because of the sparse nature of the grid structure and its notably low resolutions (in our examples, we used 32^3 for the coarse grid and 256^3 for the sparse one), the memory consumption of the grid is negligible when compared with the neural-network counterpart.

3.4 Data Sampling and Network Training

Sampling strategy. For each training iteration, our training point samples are composed of three parts: (1) randomly selected oriented particles, (2) sample points obtained from selected oriented particles, and (3) sample points randomly generated in the fined cells as shown in Figure 6. The sampling proportion between the three types of samples is 48%:48%:4%. For Part (1), we randomly select 10% of the total number of oriented particles. We note that this proportion could be a hyperparameter depending on factors such as the GPU memory size. For Part (2), we perturb the point location by $\mathbf{x}_p + d\mathbf{n}$ along each seeder's normal, where $\mathbf{x}_p$ is the seeding location, $\mathbf{n}$ is its normal direction, and d is a random distance to the interface in the range $[-\epsilon, +\epsilon]$. In the actual training, these distance values are normalized between $[0, 1]$ and will be mapped back to their actual values after training.

For Part (3), for each coarse grid being marked as *interface*, we randomly select fine grid cells it contains, allowing possible duplicate selections, equal to 1% of the total number of oriented particles maintained. A sample point is randomly placed inside each selected cell and uses the sign of the cell as the target.

The three groups of particles are collected to train the I-NGP network.

Network training. Points and their SDF values are encoded following the procedure described in Section 3.2. We pass the final prediction through a shifted Sigmoid function to clamp the result into the area of interest. We use a simple **mean absolute percentage error** (**MAPE**) [Müller et al. 2022] as our loss function, defined as $|prediction - target|/(|target| + 0.01)$. The termination criterion is when the loss stops dropping for 60 iterations or exceeds a maximum of 1000 iterations in 2D or 1,500 iterations in 3D. Such a criterion is also considered as hyperparameters that can be adjusted based on specific use cases.

4 Dynamic Neural Interface Tracking

After introducing the static geometric representation, we next present our dynamic interface tracking method on Neural PLS.

As shown in Figure 7, our dynamic tracking pipeline consists of five steps, including

(1) oriented particle advection, (2) sparse grid advection, (3) particle deletion, (4) sampling and training, and (5) particle reseeding and normal correction. The whole algorithm is presented with pseudo-code provided in Algorithm 1. We will elaborate on each of them as follows.

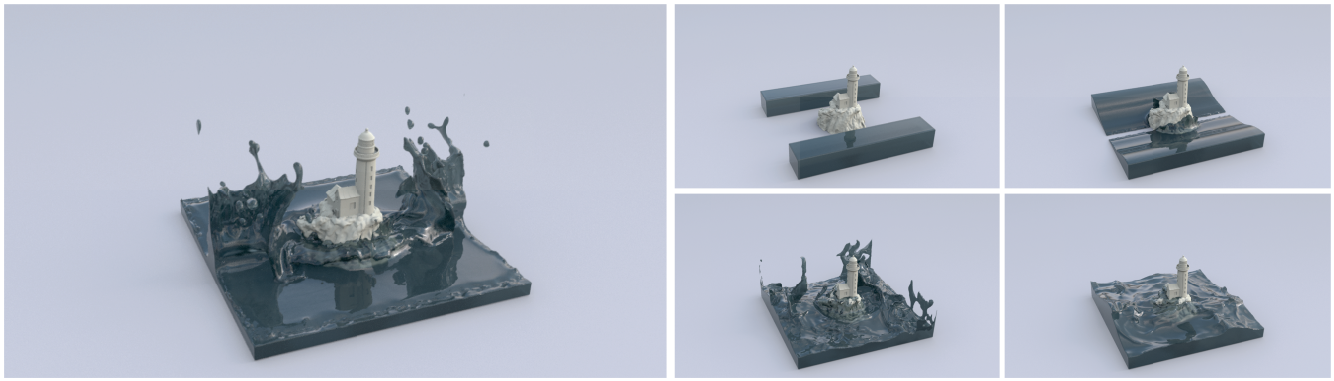


Fig. 5. We utilize our Neural PLS method to simulate the collision of two dam breaches on a lighthouse.

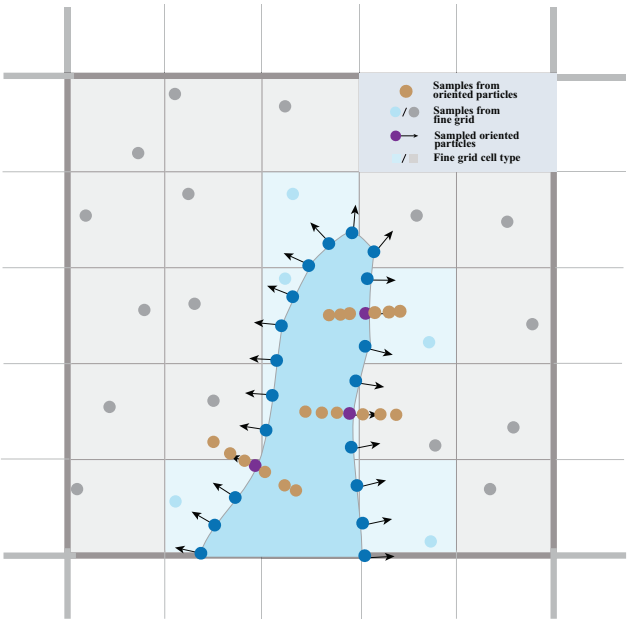


Fig. 6. We show an illustration of our sampling strategy in network training. In this picture, three types of samples are shown: (1) sampled oriented particles, (2) samples generated from sampled oriented particles, (3) samples in a fine grid.

4.1 Oriented Particle Advection

We advect particle positions using the **fourth-order Runge–Kutta (RK4)**.

The normal of each particle is updated by following the normal advection in [Ianniello and Di Mascio 2010; Nave et al. 2010]. In an arbitrary velocity field, it is given by $D\mathbf{n}/Dt = -(\nabla \mathbf{u})^T\mathbf{n} + (\mathbf{n}^T(\nabla \mathbf{u})^T\mathbf{n})\mathbf{n}$ and for divergence-free field, $D\mathbf{n}/Dt = -(\nabla \mathbf{u})^T\mathbf{n}$. Here, $\nabla \mathbf{u}_p$ is calculated as $\nabla u_p = \sum_i u_i^{n+1}((\partial K_i/\partial \mathbf{x})(\mathbf{x}_p))^T$.

Here, K_i is the interpolation kernel function, which is chosen to be bi/trilinear interpolation in our case. A similar formula can be seen in the literature to calculate gradients on a hybrid Eulerian–Lagrangian scheme, such as [Ianniello and Di Mascio 2010; Jiang et al. 2016]. We include the pseudo-code for our RK4 oriented particle advection in Algorithm 5.

4.2 Grid Advection

In order to maintain a far-field SDF field that is valid in terms of its sign (i.e., for a query point, it needs to return a correct sign) and our stored sparse SDF field (see Section 3.3), we employ a grid-based advection scheme to update the SDF values on both levels of our sparse grid. Initially, we employ a forward RK4 semi-Lagrangian advection method to advect the fine level grid cells from the previous timestep and mark the fine level grid cells in the current timestep that contain these locations in the bitmask. Additionally, we mark all the fine level grid cells that contain advected oriented particles in the current timestep, as well as their neighboring cells. For each marked fine cell, we will change the tag of the coarse level grid that contains it to *interface* and mark all the fine cells within the coarse level grid. Subsequently, on these identified fine cells, we perform a backward RK4 time integration to update their SDF values. We also update the sign value of the coarse grid using backward RK4 time integration for their tags.

4.3 Particle Deletion and Cell Masking

Like the PLS method, we need to address interface merging explicitly on the particle level by deleting particles that are colliding. We have designed a routine for particle deletion that only deletes particles if particles with different orientations are too close to each other. In particular, these criteria are:

- Two particles have a relative distance $|\mathbf{x}_p^i - \mathbf{x}_p^j|_2^2 < 2\epsilon$, where ϵ is the narrow band width.
- Two particles have different orientations, $\mathbf{n}_p^i \cdot \mathbf{n}_p^j < 0$.

Due to particle deletion and interface merging, not all fine-level cells will require correction from the network-predicted SDF value. Therefore, we maintain a mask M denoting the fine-level cells that will be corrected for their stored SDF value using the criteria of the number of oriented particles contained in the cell. The mask M is updated after the particle deletion step. The cell in M will be marked if it contains no less than 4 oriented particles or it's in the neighbor of such cell. If a cell in M contains less than 4 oriented particles, only itself will be marked but not its neighbors. Such marked cells will be noted as requiring correction after network training and their stored SDF value will be updated (See Algorithm 3).

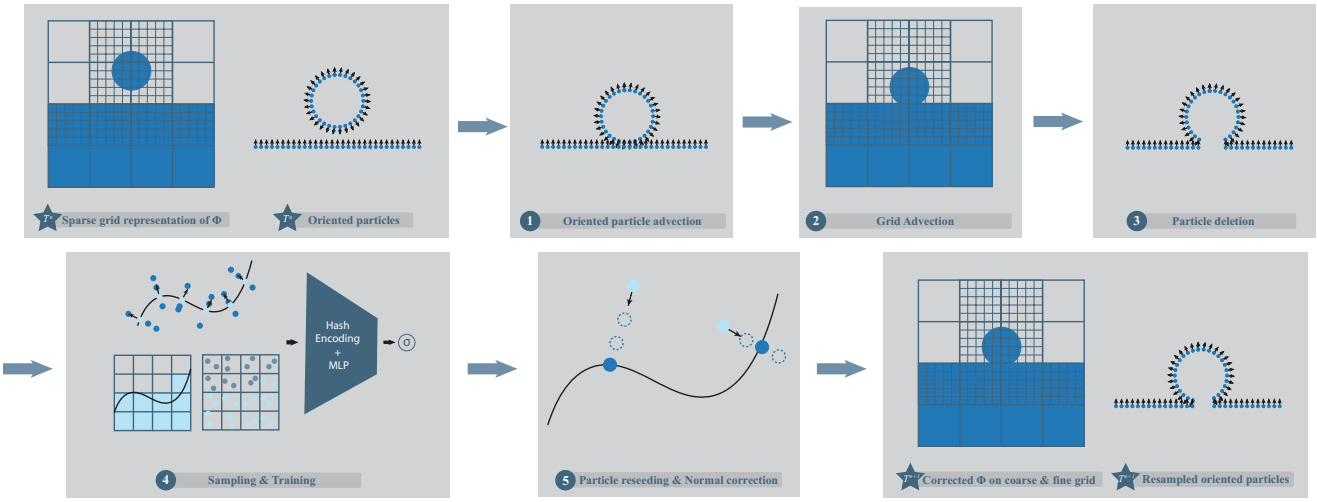


Fig. 7. The interface tracking pipeline comprises five essential components. First, we advect particles along with their associated normals (Step 1). Next, we update the ϕ values on the fine grid and the cell types on the coarse grid (Step 2). Then, we merge interfaces by removing surface particles (Step 3). Subsequently, utilizing the sampling criteria outlined in Section 3.4, we reconstruct the narrow band representation of ϕ through training the I-NGP model (Step 4). Finally, we perform normal correction based on the newly trained narrow band ϕ and resample points through an iterative projection using ϕ (Step 5).

4.4 Sampling and Training

Following the procedure described in Section 3.4, we train a narrow band representation of the level set value. After training the model, we apply it to predict the ϕ value for each center of the fine level grid cell. If a cell is marked in M, we replace its value with the corresponding prediction from the network and use the level set advected value if the cell is not marked (See Algorithm 4). Notice we only use cells marked in M for training since only those cells are needed for correction and are contained in the narrowband of interest. For each *interface* cell in the coarse level of the sparse grid, we examine whether its children contain absolute level set values that differ by less than $0.1\Delta x$ where Δx is the grid spacing. If this condition is satisfied, we remove the *interface* tag of the coarse grid and use the tag *exterior/interior* instead.

4.5 Particle Reseeding and Normal Correction

After obtaining the corrected SDF values from the training step, we proceed to correct the normal vectors carried by the particles. This correction is achieved using finite differences by dividing the gradient of ϕ by its magnitude, resulting in $\nabla\phi/|\nabla\phi|$.

Particle reseeding is then conducted based on two criteria. Firstly, if the number of particles in a fine level grid cell is fewer than 32 in 3D or 16 in 2D, and the cell is marked in M, reseeding takes place. Secondly, if the carried value in the fine level grid cell is less than $c\Delta x$, where $c = 0.1$, reseeding is performed. Reseeding will stop if the maximum allowed number of particles is exceeded. Particles seeded in cells of interest are then projected to the interface using a Newton-style procedure. For a given point $\mathbf{x}$, we iterate $\mathbf{x}_{i+1} = x_i - \phi(x_i)\nabla\phi(x_i)/|\nabla\phi(x_i)|$ until the sampled particle has $\phi \leq 1\%\Delta x$, which serves as the stopping criterion [Saye 2014]. If a particle satisfies $\phi \leq 1\%\Delta x$ and is not located inside a solid or outside a wall boundary, we compute its normal vector and add it to the maintained group of oriented particles.

5 Validation

In this section, we present 11 classical test cases that are commonly used to evaluate level set schemes. The computation domain for all examples is confined to a unit hypercube, i.e., $[0, 1]^2$ for 2D examples and $[0, 1]^3$ for 3D examples. Shapes within these test cases have been normalized to accommodate the domain size. We show the **hyperparameter settings** in Table 1 and the timing comparison between the PLS method, **narrow-band FLIP** (**NB-FLIP**), and ours in Tables 1 and 2. To ensure a fair comparison in terms of accuracy and performance, we implemented our version of PLS following [Enright et al. 2002] and NB-FLIP following [Ferstl et al. 2016], both using Taichi on GPU.

To obtain the necessary information for our method, we either need an implicit function that delineates the shape, such as a function representing a circle in 2D or a sphere in 3D, or a mesh that effectively demonstrates the isosurface. For instances where implicit functions are used, points are uniformly sampled from the function, followed by the calculation of the normal vector at each point. When working with meshes, points are uniformly sampled from each triangular patch, and the normal of the triangle is used as the normal for the respective points.

For the subsequent test cases, the auxiliary sparse grid structure is omitted, and a pure-oriented particle representation is exclusively employed. This approach is justified by the implicit provision of velocity fields in advection tests, thus, eliminating the need for a grid structure. The time steps within our simulation examples are purposefully varied, thereby demonstrating that the stability of our method is unaffected by the time step. All experiments were conducted on an Nvidia Quadro RTX8000 GPU.

5.1 Diffusion Test

We demonstrate the efficacy of our method with a constant velocity field $u = -\pi(y - 0.5)$ and $v = \pi(x - 0.5)$ and employing

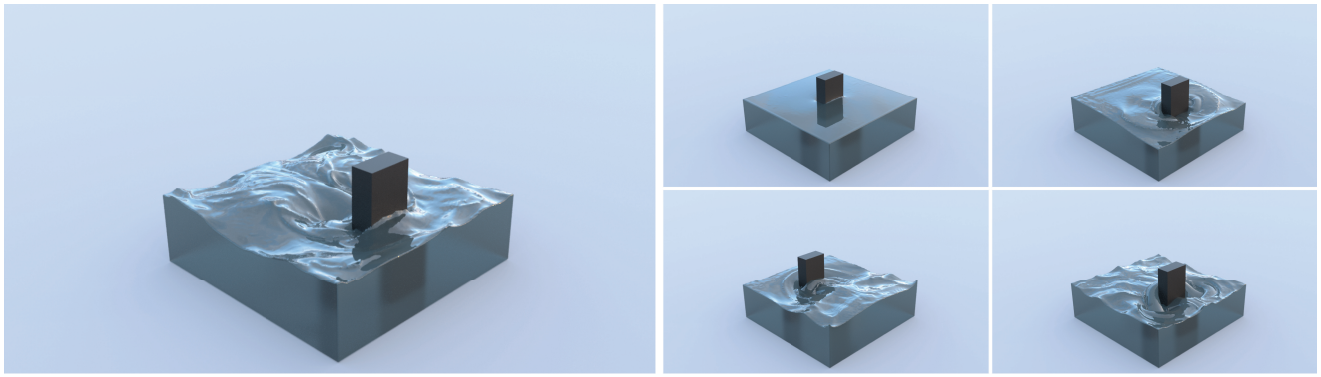


Fig. 8. Simulation of the moving paddle in a water tank. Solid and fluid are one-way coupled with solid velocity driving fluid movement.

ALGORITHM 1: Neural Level Set Tracking

Initialize: Oriented Particle $\mathbf{x_p}^t$, Fine level set grid ϕ_f, Coarse level set grid ϕ_c
while *simulation continues* **do**
  Advect $\mathbf{x_p}^t$ and $\mathbf{u}^t$ using Algorithm 5 /* Section 4.1 */;
  Set ϕ_c^{t+1} containing ϕ_f^t after advection using RK4 as interface;
  Set ϕ_c^{t+1} containing $\mathbf{x_p}^{t+1}$ as interface;
  for *each cell in* ϕ_c^{t+1} *as interface* **do** /* Section 4.2 */
    update ϕ_f^{t+1} contained by ϕ_c^{t+1} with ϕ^t using RK4;
  end
  Update label on ϕ_c^{t+1} for exterior/interior using RK4;
  Delete oriented particles based on Algorithm 3;
  Train NGP with Algorithm 4;
  for *each marked grid cell in* ϕ_f^{t+1} **do**
    Correct ϕ_f^{t+1} with NGP using its cell center location as input
  end
  for *each cell in* ϕ_c^{t+1} *as interface* **do**
    if *all fine grid cells having* $||\phi_f^{t+1}| - \epsilon| \leq 0.1\Delta x$ **then**
      Set ϕ_c^{t+1} to exterior/interior based on sign of ϕ_f^{t+1};
    end
  end
  Add external force and solve for pressure using ϕ^{t+1};
  Correct normal carried on $\mathbf{x_p}$ /* Section 4.5 */;
  Resample $\mathbf{x_p}$ as needed with Algorithm 2.
end

ALGORITHM 2: Resample Oriented Particles

Variable: P_{max} maximum number of particles allowed, fine level set grid ϕ_f, oriented particles $\mathbf{x_p}$, newly seeded particle $\tilde{\mathbf{x}}_\mathbf{p}$
for *each grid cell in* ϕ_f^{t+1} **do**
  if *(cell contains* $\mathbf{x_p} \vee \phi_f^{t+1} \leq 0.1\Delta x) \wedge Num(\mathbf{x_p}) \leq P_{max}$ **then**
    Randomly seed an oriented particle $\tilde{\mathbf{x}}_\mathbf{p}$ in cell;
    while *less than maximum iteration* **do**
      if $|\phi| \leq 1\%\Delta x$ **then**
        Add particle $\tilde{\mathbf{x}}_\mathbf{p}$ to oriented particle set;
        break;
      end
      Project particle $\tilde{\mathbf{x}}_\mathbf{p}$ with $\tilde{\mathbf{x}}_\mathbf{p}^{i+1} = \tilde{\mathbf{x}}_\mathbf{p}^i - \phi(\tilde{\mathbf{x}}_\mathbf{p}^i)\nabla\phi(\tilde{\mathbf{x}}_\mathbf{p}^i)/|\nabla\phi(\tilde{\mathbf{x}}_\mathbf{p}^i)|$
    end
  end
end

ALGORITHM 3: Delete Oriented Particles

for *each pair of oriented particle* $\mathbf{x_p}^{t+1}$ *namely* $\mathbf{x_p}_i^{t+1}$ *and* $\mathbf{x_p}_j^{t+1}$ /* Section 4.3 */ **do**
  if $|\mathbf{x_p}_i^{t+1} - \mathbf{x_p}_j^{t+1}|_2^2 < 2\epsilon$ *and* $\mathbf{n_p}_i^{t+1} \cdot \mathbf{n_p}_j^{t+1} < 0$ **then**
    Delete $\mathbf{x_p}_i^{t+1}$ and $\mathbf{x_p}_j^{t+1}$;
  end
end
for *each fine grid cell in* ϕ_c^{t+1} *as interface* /* Section 4.4 */; **do**
  if ϕ_f^{t+1} *contains more than 4 particles* **then**
    Mark ϕ_f^{t+1} and neighbor cells in $[-\epsilon, \epsilon]^d$ as require update;
  else
    Mark ϕ_f^{t+1} as require update;
  end
end

a shape represented by $x = \sin(\theta)(1 + 0.25\cos(20\theta))/4 + 0.5$ and $y = \cos(\theta)(1 + 0.25\cos(20\theta))/4 + 0.5$. We uniformly sample θ from $[0, 2\pi]$ and calculate the location of each point using the aforementioned function. The normal vector of each point is computed as $n_x = (\cos(20\theta) + 4)\sin(\theta)/16 + 5\cos(\theta)\sin(20\theta)/4$ and $n_y = (\cos(20\theta) + 4)\cos(\theta)/16 - 5\sin(\theta)\sin(20\theta)/4$.

This shape, replete with spiky features that could be readily smoothed out, necessitates a high-resolution grid to accurately capture the geometry's spikes. However, using our method, we only need to sample points on the surface, reducing the number of points required to capture the shape's detail. For this example, we use $\Delta t = 0.002$ and execute marching squares on an 800^2 field. Figure 10 presents the results derived from our method. The results obtained using PLS method with a 800^2 resolution and the level set method with $1,600^2$, $3,200^2$ resolutions are included for comparison. Table 3 provides a measure of area loss for these methods. The results demonstrate that our method significantly outperforms the two level set methods in capturing the spikes and surpasses all three methods in terms of area loss.

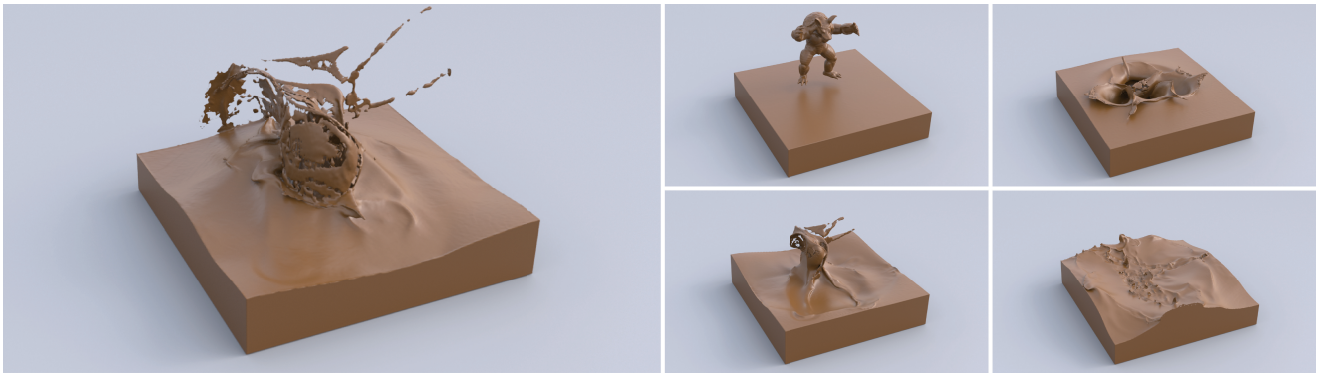


Fig. 9. We simulate Armadillos dropping into a tank using our Neural PLS method.

ALGORITHM 4: Train I-NGP

Variable : oriented particles $\mathbf{x_p}$, fine level set grid ϕ_f, coarse level set grid ϕ_c

Initialize C and V as empty storage for training, where C will store cells location range in computational grid and V will store binary level set values, i.e. $-\epsilon$ or ϵ.;

for *each fine grid cell in* ϕ_c^{t+1} *as interface* **do**

  if ϕ_f^{t+1} *is marked as require update* **then**

    Collect such cells in C and store $-\epsilon$ or ϵ in V based on its sign;

  end

end

while *not exceed maximum iteration* **do**

  Sample T_i in ${\mathbf{x_p}}^{t+1}$ with $\phi_i = 0$;

  Sample T_b using $T_i + d\mathbf{n}$ where $d \leftarrow rand(-\epsilon, \epsilon)$ with $\phi_b = d$;

  Sample T_g in C with value ϕ_g in V;

  Train NGP with $[T_i, T_b, T_g]$ with value $[\phi_i, \phi_b, \phi_g]$ using loss: $\frac{|prediction-target|}{(|target|+0.01)}$;

end

ALGORITHM 5: Oriented Particle Advection

Variable : coordinate $\mathbf{x}$, velocity $\mathbf{u}$, normal $\mathbf{n}$

$(\mathbf{u}_1, \nabla\mathbf{u}_1) \leftarrow$ **Interpolate**$(\mathbf{u}, \mathbf{x})$;

$\delta\mathbf{n}_1 = -\nabla{\mathbf{u}_1}^T\mathbf{n}$;

$\mathbf{x}_1 \leftarrow \mathbf{x} + 0.5 \cdot \Delta t \cdot \mathbf{u}_1$;

$\mathbf{n}_1 \leftarrow \mathbf{n} + 0.5 \cdot \Delta t \cdot \delta\mathbf{n}_1$;

$(\mathbf{u}_2, \nabla\mathbf{u}_2) \leftarrow$ **Interpolate**$(\mathbf{u}, \mathbf{x}_1)$;

$\delta\mathbf{n}_2 = -\nabla{\mathbf{u}_2}^T\mathbf{n}_1$;

$\mathbf{x}_2 \leftarrow \mathbf{x} + 0.5 \cdot \Delta t \cdot \mathbf{u}_2$;

$\mathbf{n}_2 \leftarrow \mathbf{n} + 0.5 \cdot \Delta t \cdot \delta\mathbf{n}_2$;

$(\mathbf{u}_3, \nabla\mathbf{u}_3) \leftarrow$ **Interpolate**$(\mathbf{u}, \mathbf{x}_2)$;

$\delta\mathbf{n}_3 = -\nabla{\mathbf{u}_3}^T\mathbf{n}_2$;

$\mathbf{x}_3 \leftarrow \mathbf{x} + \cdot\Delta t \cdot \mathbf{u}_3$;

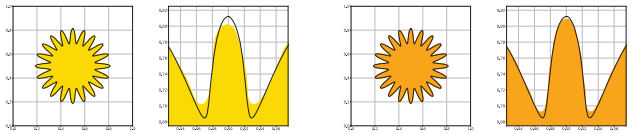
$\mathbf{n}_3 \leftarrow \mathbf{n} + \cdot\Delta t \cdot \delta\mathbf{n}_3$;

$(\mathbf{u}_4, \nabla\mathbf{u}_4) \leftarrow$ **Interpolate**$(\mathbf{u}, \mathbf{x}_3)$;

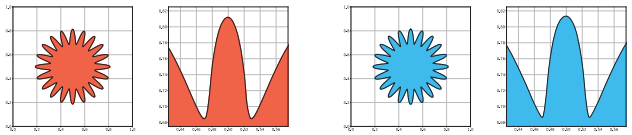
$\delta\mathbf{n}_4 = -\nabla{\mathbf{u}_4}^T\mathbf{n}_3$;

$\mathbf{x}_{\text{next}} = \mathbf{u} + \frac{1}{6} \cdot \Delta t \cdot (\mathbf{u}_1 + 2 \cdot \mathbf{u}_2 + 2 \cdot \mathbf{u}_3 + \mathbf{u}_4)$;

$\mathbf{n}_{\text{next}} = \mathbf{n} + \frac{1}{6} \cdot \Delta t \cdot (\delta\mathbf{n}_1 + 2 \cdot \delta\mathbf{n}_2 + 2 \cdot \delta\mathbf{n}_3 + \delta\mathbf{n}_4)$;



(a) 1600^2 level set method (b) 3200^2 level set method

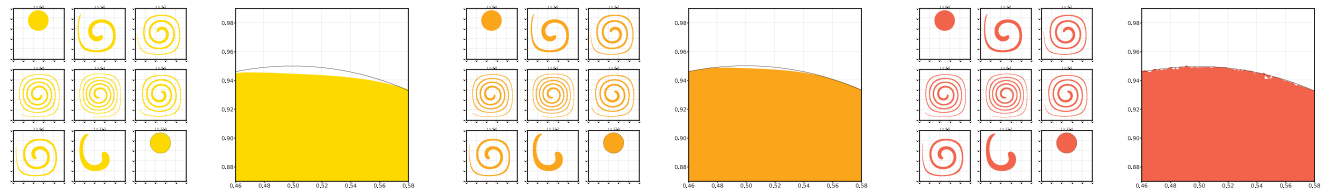


(c) 800^2 PLS method (d) 800^2 ours

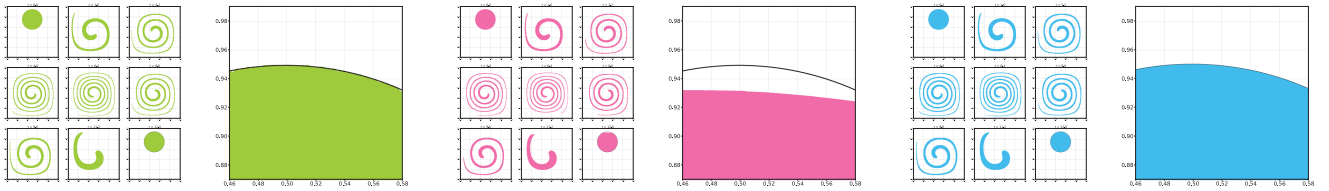
Fig. 10. The Spike Disk is evolved in an implicit rigid rotation velocity field. In each method, the simulation results are depicted in the left figure, with the zoomed-in results showcased in the right figure. The black outline represents the isosurface at $t = 0$, while the colored shape represents the isosurface at $t = 4$.

5.2 Single Vortex

In addition to demonstrating our method's capability to overcome diffusion errors generated by the level set method, we also showcase its ability to handle level set evolution under conditions of stretching and tearing flow. For this purpose, we utilize the benchmark test case suggested in [Enright et al. 2002]. A circle with a radius of 0.2 is centered at $[0.5, 0.75]$, which is deformed by a divergence-free velocity field given by $u = 2\sin^2(\pi x)\sin(\pi y)\cos(\pi y)$ and $v = -2\sin(\pi x)\sin^2(\pi y)\cos(\pi x)$. We reverse this velocity field at $t = 6$ such that the deformation reaches its peak at $t = 6$ and subsequently returns to its original shape at $t = 12$. To derive the initial surface points, we uniformly sample the circle and compute their normals as $normalize(\mathbf{x} - \mathbf{c})$, where $\mathbf{c}$ represents the circle's center and $\mathbf{x}$ represents the sample point. Employing a time step of $\Delta t = 0.004$, we execute marching squares on a field with a resolution of 800^2 to display the results. The original shape of the circle is plotted at the end of the simulation to enable a comparison of area loss. We also perform the same evolution using the PLS method with a 800^2 resolution and level set method with resolutions of $1,600^2$ and $3,200^2$. As presented in Figure 11, our method successfully restores the original



(a) 1600^2 level set method (b) 3200^2 level set method (c) 800^2 PLS method



(d) 800^2 FLIP method (e) 800^2 NB-FLIP method (f) 800^2 ours

Fig. 11. A disk is evolved in a single vortex velocity field. The left figure of each method illustrates the results of the evolution, while the right figure highlights the zoomed-in results in the final frame. The black circle in the final frame represents the isosurface of the disk at $t = 0$.

shape at the conclusion of the simulation, whereas the other three methods experience some degree of distortion from the original form. Furthermore, our method demonstrates the lowest area loss among the other methods, including level set method, PLS, FLIP, and NB-FLIP, as shown in Table 3. Here, NB-FLIP only serves as a surface tracker in 2D where the velocity field is analytically given. This result attests to the capability of our method in tracking an implicit interface under 2D scenarios with stretching and tearing background flow fields.

5.3 Mean Curvature Flow

In the context of a self-generated velocity field like mean-curvature flow, the velocity is defined as $\mathbf{u} = -b\kappa\mathbf{n}$ and the transport equation $\partial\phi/\partial t + \mathbf{u}\cdot\nabla\phi = 0$ then become:

$$\frac{\partial\phi}{\partial t} = b\kappa\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla\phi = b\kappa|\nabla\phi|. \tag{1}$$

For each time step, we compute the curvature κ using $\nabla_\Gamma \cdot \mathbf{n}$. We demonstrate our method on two shapes. One is defined by $x = \sin(\theta)(1 + 0.25\cos(5\theta))/4 + 0.5$ and $y = \cos(\theta)(1 + 0.25\cos(5\theta))/4 + 0.5$, the other is defined by $x = \sin(\theta)(1 + 0.5\cos(6\theta))/4 + 0.5$ and $y = \cos(\theta)(1 + 0.5\cos(6\theta))/4 + 0.5$.

The surface divergence ∇_Γ is obtained by taking the finite difference over the normal vector in the surface tangential direction. For a given point $\mathbf{x}$, we first obtain its surface tangent $\mathbf{t_x}$ from $\mathbf{n_x}$. Surface divergence is then computed using $(-\mathbf{n_{x-\Delta x}}\cdot\mathbf{t_x} + \mathbf{n_{x+\Delta x}}\cdot\mathbf{t_x})/2$.

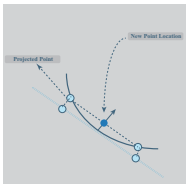


Fig. 12. Illustration of normal and location calibration.

We calculate the mean curvature for a small kernel with a radius of 0.005 and use it as the curvature for point $\mathbf{x}$. The velocity field derived from this curvature is used to compute $\nabla\mathbf{u}$ using finite differences in order to advect the normal with the same Δx. However, $\nabla\mathbf{u}$ can pose numerical issues and cause the normal $\mathbf{n}$ to become noisy after several simulation steps. To tackle this issue, we calibrate the normal vector and surface point as follows:

- Average the normal vector of a surface point in its normal direction in the narrow band.
- For this surface point, sample two points at distances of $-\Delta x$ and Δx along its tangent direction, and project the two points onto the surface using their normal vector and the kernel function value in the narrow band (Figure 12).
- Calibrate the surface point location by averaging the locations of the two projected points.
- Remove the calibrated point if its location is not close enough to the isosurface predicted by the network.

The calibration process is performed every 10 simulation steps. In the two examples, we use $\Delta t = 0.0005$ and $\Delta t = 0.0001$, respectively. To visualize the results, we plot the surface using the marching squares algorithm on an 800^2 grid (see Figures 13 and 14).

5.4 Crazy Rotation

To further demonstrate the capabilities of our method, we apply it to a rotating velocity field where a circle with a radius of 0.2 centered at $[0.3, 0.3]$ is positioned. The velocity field is defined as $u = (0.5 - y)/(\sqrt{(x-0.5)^2 + (y-0.5)^2})$ and $v = (x - 0.5)/(\sqrt{(x-0.5)^2 + (y-0.5)^2})$. We simulate this rotation until $t = 35$, depicting the resulting shape using a marching square on a field of $5,000^2$ resolution in Figure 15. The corresponding PLS simulation result with a resolution of $5,000^2$ is displayed in Figure 16. The simulation output of our method closely resembles that of the PLS method. To showcase the ability of our method to capture fine details, we also provide a zoomed-in isosurface in Figure 17, where the resolution ranges from 800 to 500,000.

5.5 Rigid Body Rotation of Armadillos

Armadillos are a common test case for evaluating the ability of mesh processing and reconstruction methods to represent fine details. We opted to use an Armadillo as our test object to evaluate

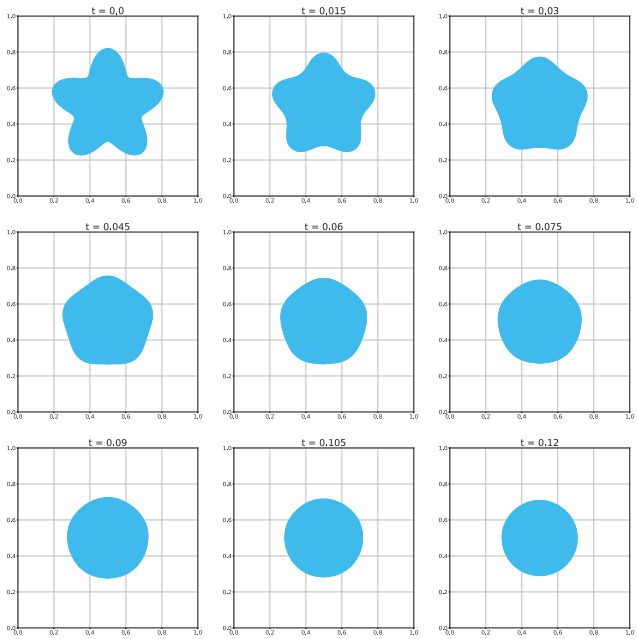


Fig. 13. A star is evolved under mean curvature flow.

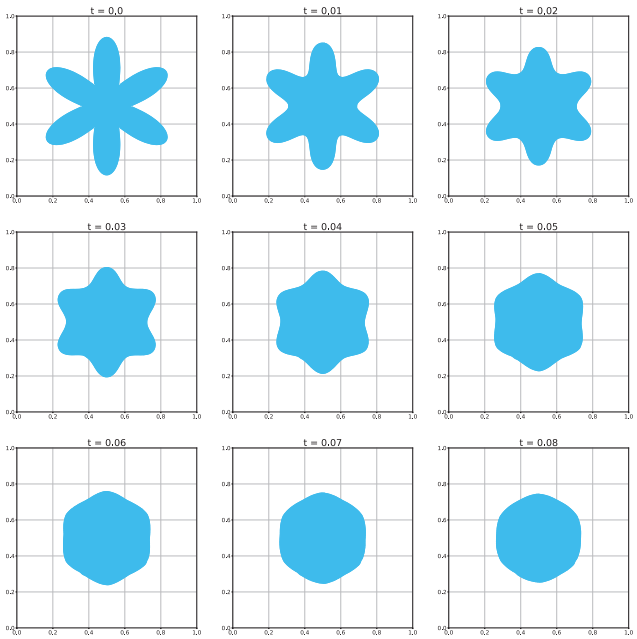


Fig. 14. A hexagonal star is evolved under mean curvature flow.

diffusion error in 3D, as opposed to a 3D Zalesak's sphere, which we consider less suitable for this purpose. We normalized the Armadillo to a unit cube and scaled it down by a factor of 2, before positioning the center of its bounding box at $[0.5, 0.5, 0.5]$. The velocity field is given by $u = 0.5 - y$, $v = x - 0.5$, and $w = 0$. To visualize the isosurface, we performed marching cubes on a $1,200^3$ grid, which enabled us to exhibit fine details and demonstrate that the surface did not suffer from any smoothing effects, as shown in Figure 19.

5.6 3D Deformation Field

We employ the classical benchmark presented in [Enright et al. 2002] to evaluate the effectiveness of our method. The velocity field is defined as $u = 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)$, $v = -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)$ and $w = -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)$. We position a sphere with a radius of 0.15 centered at $[0.35, 0.35, 0.35]$. In Figure 21, we reverse the velocity field at $t = 2$ such that the deformation reaches its maximum at $t = 2$, and should recover the original shape at the end of the simulation. In addition, we employed the PLS method at a resolution of 500^3—the maximum allowed by the computational resources available on the same GPU used in our experiments—and conducted the NB-FLIP method at the same resolution for a fair comparison during the evolution process. In this scenario, the background advection velocity fields are analytically prescribed, and thus all methods presented here function solely as implicit interface trackers. As shown in Figures 21–23, our method successfully restores the original shape, whereas the PLS method exhibits some inaccurately advected volumes by the end of the recovery process, and the NB-FLIP method loses a significant amount of volume along the sphere's midline. We attribute the superior performance of PLS over NB-FLIP to the fact that NB-FLIP is not inherently designed for interface tracking and operates similarly to single-sided PLS. Furthermore, our method outperforms the PLS method by effectively restoring the original shape and demonstrating improved volume retention, benefiting from the use of normal information and the adaptivity provided by the instant-NGP backend. This test case has been further explored in [Gibou et al. 2018] by using an adaptive forest of octrees proposed in [Mirzadeh et al. 2016] and simulating until $t = 9$ using a supercomputer with the finest resolution of $4,096^3$. Our method can also produce similar results, as demonstrated in Figure 2. We used marching cubes with a $1,500^3$ resolution to visualize the isosurface. In this test, to achieve better representation ability, we disabled the early stopping mechanism and chose longer optimization iterations.

6 Fluid Simulations

In this section, we present the 3D results of applying our method to fluid simulations. We use Taichi [Hu et al. 2019] in our fluid simulation because of compatibility and its support for GPU and sparse grid. Our fluid simulation is based on the standard grid-based advection-projection incompressible fluid solver [Foster and Fedkiw 2001] with RK4 advection scheme. We also used a **Ghost-Fluid Method (GFM)** [Gibou et al. 2002] for pressure solving for our method, PLS and NB-FLIP to prevent stair-case artifacts and explicit surface tension handling [Enright et al. 2003]. We resample all the particles and their normals every 25 steps to prevent particle clustering due to drastic fluid movement. We utilize a 256^3 resolution in our simulation for both the Poisson solver and the fine grid representation of the level set grid. The maximum total number of oriented particles allowed in the four experiments is $2,000,000$. The **network configuration** comprises 16 levels of spatial hashing resolution with 1.5 as the growth factor. We employ the hash table size of 2^{19}. We have determined that this set of parameters

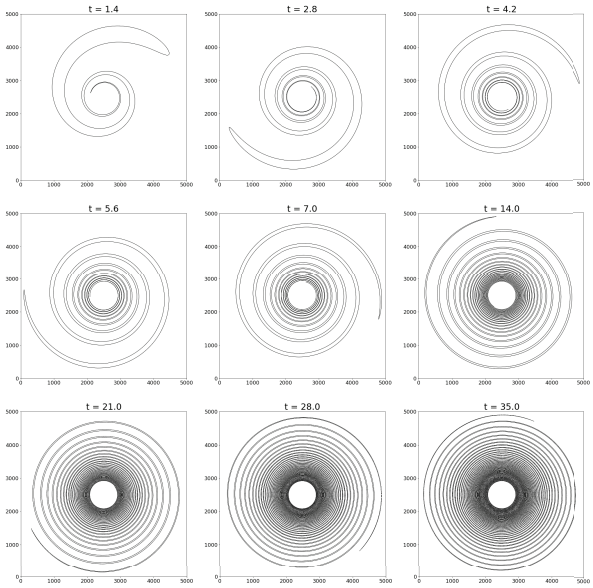


Fig. 15. A disk is evolved in a rotation velocity field using Neural PLS method.

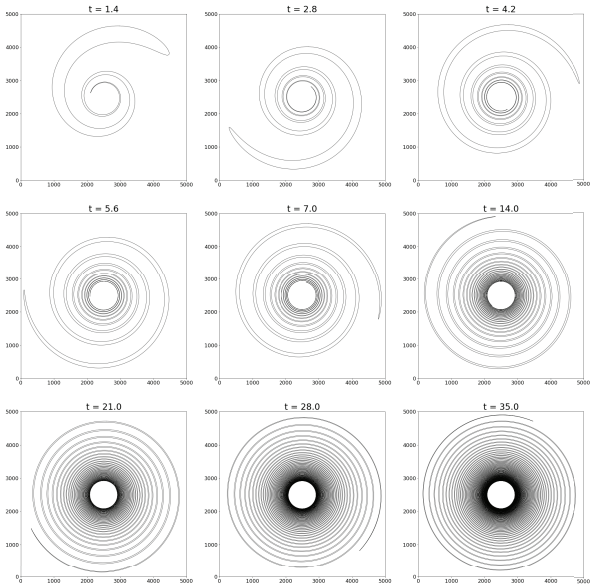


Fig. 16. A disk is evolved in a rotation velocity field using PLS method.

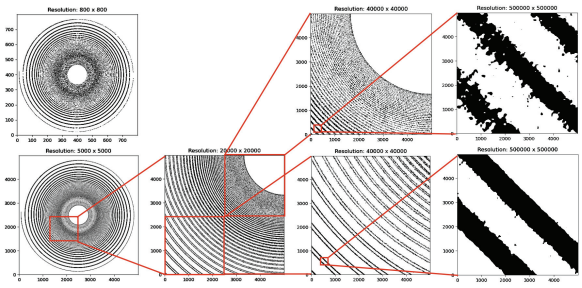


Fig. 17. We illustrate resolution independent representation ranging from 800^2 to 500, 000^2 using our neural-PLS method. As we can track up to 500,000 resolution, noise from the network does appear if the resolution goes too high.

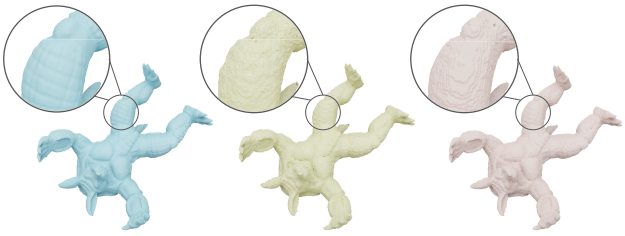


Fig. 18. Here, we present an experiment involving the rotation of Armadillos over a duration of 4.56 seconds, driven by a rigid rotation velocity field (refer to Section 5.5 for the definition of the velocity field). We compare the results obtained using our method with two other approaches. On the *left*, we show the results obtained by advecting the Armadillos using our method, which utilizes oriented particles for tracking the interface. In the *middle*, we perform the experiment using the previous timestep model as the ground truth, and we advect sample points from the current timestep backward to read their values directly. On the *right*, we conduct the same experiment as in the middle scenario, but we enforce the surface particles to carry a level set value of 0.

provides us with a sufficient level of accuracy in encoding the narrow band level set, while still maintaining good performance in terms of running time. However, it is worth noting that these settings can also be adjusted and fine-tuned based on specific requirements or preferences. The fluid experiments are conducted on an Nvidia Quadro RTX8000 GPU. We present the performance of fluid simulation at the end of this section, followed by a brief discussion.

6.1 Collision of Two Dam Breaches on a Lighthouse

In this example, we demonstrate that our method is capable of generating a dynamic water surface and energetic solid-fluid interactions. We position two fluid blocks at $x = 0.15$ and $x = 0.85$ with a height of 0.2. The simulation results are exhibited in Figure 5.

6.2 Random Sphere Drop Into a Tank

In this scenario, we aim at demonstrating that our interface merging technique can handle complex topology changes under highly dynamic fluid surface movements. We uniformly position 16 spheres on the x and z coordinates and perturb them randomly on the y coordinate. The water surface is located at $y = 0.2$. We uniformly sample points on all geometries weighted by the surface area of the geometry. To evaluate the performance of our approach,

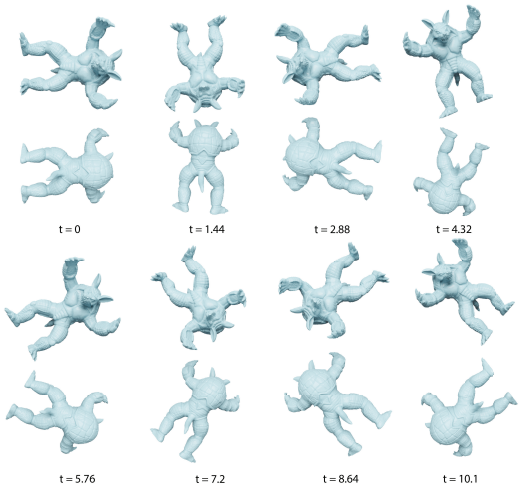


Fig. 19. The surface of the Armadillo does not experience any smoothing effect during rigid rotation.

we compared it with two other methods: PLS and NB-FLIP. For the NB-FLIP implementation, we followed the approach described in [Ferstl et al. 2016]. To ensure a fair comparison, we conducted two versions of free-surface fluid simulations using the NB-FLIP component in different ways: (1) employing NB-FLIP solely as an interface tracker, where particle velocities in the narrow band do not influence the fluid velocity field discretized on the background grid, and (2) employing NB-FLIP both as an interface tracker and as part of the simulation, where the particle velocities in the narrow band are blended with the grid velocity (as in typical PIC-FLIP fluid solvers). The motivation for including the interface-tracking-only version of NB-FLIP was to align with the roles of PLS and Neural PLS, both of which function only as interface trackers without directly influencing the fluid velocity near the interface. As shown in Figure 25, we conducted five experiments: (1) PLS as an interface tracker with a grid-based Eulerian fluid solver (PLS Tracking + Eulerian Simulation), (2) NB-FLIP as an interface tracker with a grid-based Eulerian fluid solver (NB-FLIP Tracking + Eulerian Simulation), (3) NB-FLIP as an interface tracker combined with the PIC-FLIP fluid solver in the narrow band (NB-FLIP Tracking + Lagrangian-Eulerian Simulation), (4) Neural PLS as an interface tracker with a grid-based Eulerian fluid solver at a resolution of $256 \times 256 \times 256$ (Neural PLS Tracking + Eulerian Simulation (256)), and (5) Neural PLS as an interface tracker with a grid-based Eulerian fluid solver at a resolution of $512 \times 512 \times 512$ (Neural PLS Tracking + Eulerian Simulation (512)).

From the simulation results, we observed the following:

— **PLS Tracking + Eulerian Simulation vs. NB-FLIP + Eulerian Simulation vs. NB-FLIP + Lagriangian–Eulerian Simulation (Rows 1, 2, and 3)**: The first two show notable similarities, whereas the full NB-FLIP (NB-FLIP Tracker + Lagrangian–Eulerian Simulator) achieves a less viscous and more energetic surface. This indicates that (1) when NB-FLIP is employed purely as an interface tracker and used together with a grid-based fluid solver, it behaves similarly to PLS, and (2) only when NB-FLIP is used as both an interface tracker and a background simulator (i.e., particles contribute to fluid advection) does it produce an energetic fluid surface mention, which has been commonly observed in past FLIP/NB-FLIP literature.

— **PLS/NB-FLIP Tracking + Eulerian Simulation vs. Neural PLS Tracking + Eulerian Simulation (256) (Rows 1, 2, and 4)**: In contrast to the two methods, our method maintains a more energetic free surface, attributed to utilizing a single layer of tracking particles and the interface's normal evolution.

— **NB-FLIP + Lagriangian–Eulerian Simulation vs. Neural PLS Tracking + Eulerian Simulation (256) (Rows 3 and 4):** Compared with the NB-FLIP + Lagriangian-Eulerian Simulation result, our method gives comparable behavior in terms of fluid turbulence and dynamics without letting particles influence fluid velocity at the interface. Such behavior can be attributed to the gradient information carried on particles with the adaptiveness provided by the network backend.

— **Neural PLS Tracking + Eulerian Simulation (512) (Row 5)**: We further illustrate that our method has the potential to handle large-scale surface simulation by including a 512 resolution fluid simulation.

Hence, by utilizing adaptiveness and gradient information, our method gives the most satisfying result compared to the other simulation methods provided here.

6.3 Armadillos Drop into a Tank

In this scenario, we aim at illustrating our method's ability to preserve the fine details of a level set without suffering from diffusion errors. We observe that the claw of the Armadillos maintains its sharp features throughout the process of dropping into the fluid tank, and similarly, its other details on the surface remain unchanged. Specifically, we use fast marching to initialize a fine grid of level set values for the Armadillos and sample particles on the triangle mesh of the Armadillos to obtain our surface particles. After initialization, the simulation procedure follows the method stated in Section 4. The simulation results are shown in Figure 9.

We also display a zoomed-in picture before the Armadillos drop into the tank to illustrate how our method preserves the geometric details on the Armadillos (see Figure 26).

6.4 Moving Paddle in a Tank

In this case, we show our method with fluid simulation coupled with dynamic solid boundaries. Here we use a solid paddle with height of 0.5, width of 0.2, and thickness of 0.1 located at the center of the tank. We move the paddle with a fixed location function by setting its x coordinate offset as $0.25sin(2\pi t/2.5)$. Solid boundary velocity is then calculated with finite difference. We observe symmetric patterns along z axis and vortex forms at the edge of the paddle. The simulation results are shown in Figure 8.

7 Ablation Study

In this section, we conduct three ablation studies to show the necessity of the oriented information carried on particles to aid network training, the influence of I-NGP hyperparameters on the

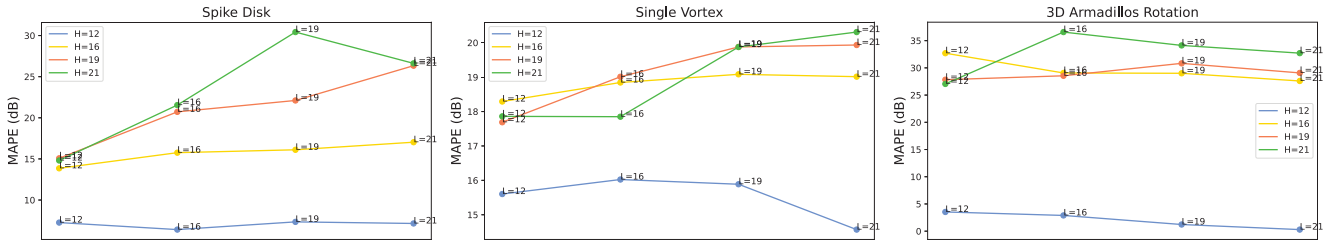


Fig. 20. Here we show an experiment performed on comparing volume loss under rigid rotation velocity field and single vortex deformation field. We aim at studying the influence of hyperparameters of I-NGP on the capacity for preserving volume under diffusion tests. Here, **H** represents hash table size and **L** represents the maximum level of resolution. We use a growing factor as 1.5 in all experiments. A fixed number of 1,500 epochs are used for 3D and 1,000 epochs are used for 2D.

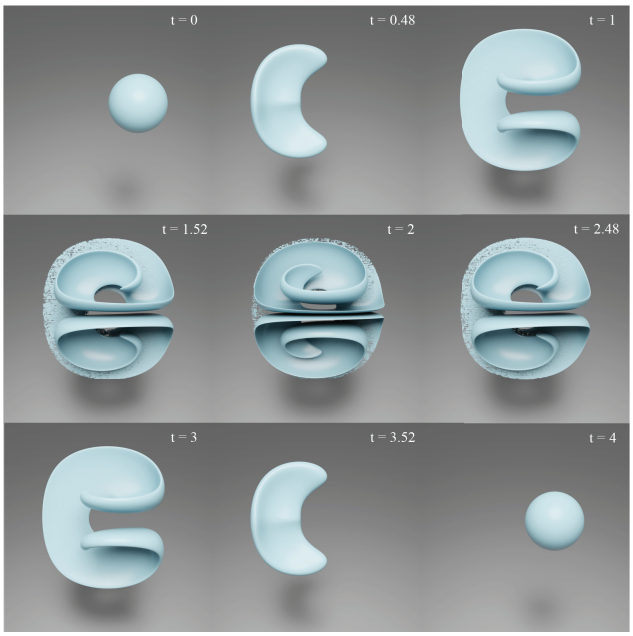


Fig. 21. A sphere is evolved in a 3D deformation field using Neural PLS method.

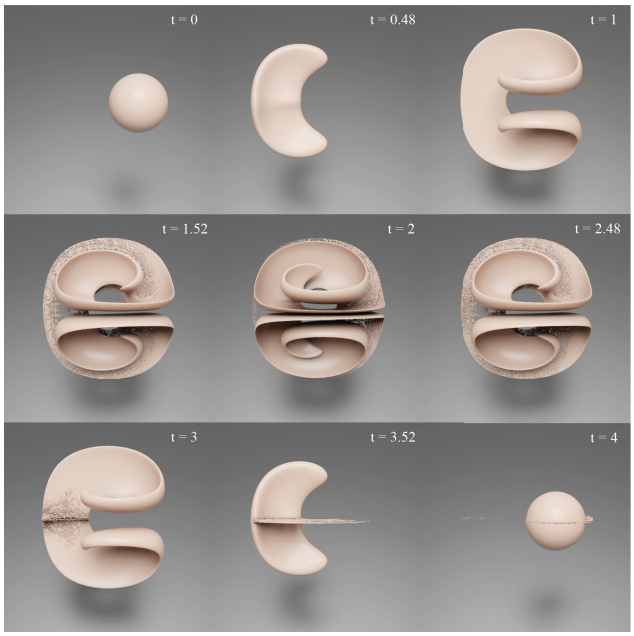


Fig. 22. A sphere is evolved in a 3D deformation field using the PLS method.

volume-preserving capacity of our method and the necessity of using second-order GFM [Gibou et al. 2002] to prevent staircase artifacts.

7.1 Orientation Information

We demonstrate the significance of orientation information carried on particles by highlighting the limitations of relying solely on the trained model from the previous timestep. It should be noted that the reason for resorting to the previous timestep model is the lack of information to reconstruct the level set at the current timestep without employing computationally intensive methods such as fast-marching [Sethian 1999] or fast-sweeping [Zhao 2005]. The experiments are performed on the 3D rotation test of Armadillos (see Figure 18). Instead of utilizing orientation information to reconstruct the level set value, we employ a high-order time integration scheme to trace back to the previous timestep and consider the model prediction from that timestep as the ground truth. Two experiments are performed: (1) in the first experiment, both surface particle locations and narrow-band locations retrieve values from the previous timestep; (2) in the second experiment, surface particle locations enforce a level set value of 0, while narrow-band values are obtained from the model in the previous timestep. Due to the accumulation of errors during advection and model prediction, the result exhibits artifacts as shown in Figure 18. To ensure a fair comparison, approximately 200,000 points are employed in these two tests, which aligns with the number used in the experiment conducted with oriented particles for reconstruction. The training time for each experiment increases to 280.74 s and 225.60 s, respectively, as each sampling step involves the use of RK4 integration and model inference, in contrast to the 9 s shown in Table 1.

7.2 Hyperparamaters

We present our study on I-NGP hyperparameters that affect volume loss. We conducted experiments with hash table sizes (H) chosen from $[2^{12}, 2^{16}, 2^{19}, 2^{21}]$ and maximum levels of resolution (L) selected from $[12, 16, 19, 21]$. The growth factor was consistently

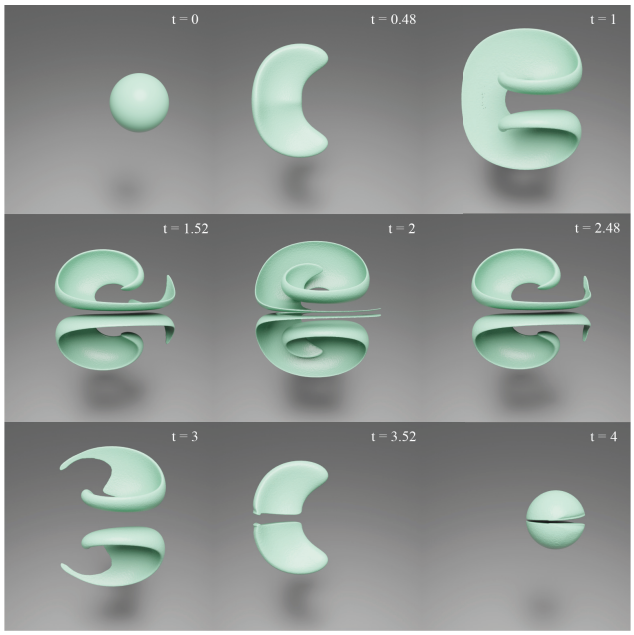


Fig. 23. A sphere is evolved in a 3D deformation field using NB-FLIP method.



Fig. 24. Ablation study on using the GFM.

set to 1.5. In these experiments, we maintained comparable volumes at both the initial and final frames. We then showcased the volume loss for three experiments conducted under different parameter settings, as depicted in Figure 20. Furthermore, we displayed the MAPE of the volume compared to the ground truth volume at the first frame. Our observations revealed a pattern similar to that demonstrated in [Müller et al. 2022]. We observe significant improvement once $H \geq 19$ and $L \geq 16$, however, the improvements are relatively small once such threshold is passed. As improvements come with a cost on performance, consequently, we opted to adhere to the default hyperparameter settings ($H = 19$, $L = 16$), which are considered suitable for general use cases and use this in our fluid simulation scenarios.

7.3 GFM

We investigate the necessity of the GFM in mitigating staircase artifacts, as in Figure 24. Previous studies, such as [Goldade et al. 2016], have demonstrated that staircase artifacts can arise when erroneous level set values outside the narrowband are used. A similar issue occurs when the GFM is replaced by a binary cell-type classification (e.g., fluid versus air). By employing the GFM, we can obtain a smooth and continuous surface, demonstrating its ability to maintain accurate level set values within the narrowband and effectively eliminate staircase artifacts.

8 Discussion

As shown in previous sections, our method contains an I-NGP module and a training step to realize high-resolution interface tracking. In this section, we discuss the rationale for incorporating the I-NGP module instead of using a traditional adaptive SDF and the influence on performance with such a decision. Following these, we will also briefly discuss the relationship between our method, PLS, and NB-FLIP in the context of interface tracking and fluid simulation. Finally, we will briefly discuss why sparse grid implementations are not included for PLS or traditional level set methods, and why we opted to use oriented particles for seeding training points.

I-NGP vs. Traditional Adaptive SDF. I-NGP [Müller et al. 2022] and adaptive SDF [Frisken et al. 2000], both capable of encoding geometry details, differ from each other in the following ways. We comprehend the I-NGP backend of our framework as an extremely adaptive spatial discretization that is easy to implement. We elaborate this in two aspects. Firstly, I-NGP establishes a hierarchical structure of hash tables to manage sparse, overlapping grids, enabling an exceptionally adaptive spatial discretization. This flexibility allows for grid resolutions to reach as high as 2^{21}, a feat that would be unattainable when employing traditional SDF methods on a single workstation (see example in Figure 2). Secondly, when traditional adaptive SDF methods are employed, they necessitate intricate implementations involving non-overlapping cell subdivision, T-junction handling, and level-set reinitialization, typically associated with a conventional adaptive structure such as an Octree. In contrast, I-NGP sidesteps the intricacies of these implementations by constructing a concatenation of feature vectors from grids and subsequently training a compact neural network to translate these vectors into SDF values (more details for I-NGP in Figure 4 and training pipeline in Section 4.4). This streamlined approach simplifies the process considerably but at the cost of non-negligible network training time.

Performance. For our fluid simulation performed on the Nvidia Quadro RTX8000 GPU, each substep of the simulation takes approximately 30 seconds to complete. The MGPCG Poisson solver, which handles the complexity of the fluid scene, takes around 1.5 to 2.5 seconds. The training process itself takes around 10 to 20 seconds. Although we note that most of the running time is dedicated to training the I-NGP component underpinning the Neural PLS representation, we mark that: First, this allows level set evolution to achieve resolutions unachievable for single workstation; Second, this significantly hides implementation complexity (e.g., Octree), especially under dynamic settings; Third, this is the first INR that can effectively accommodate physical simulations by producing results with state-of-the-art qualities.

Relationship between PLS, NB-FLIP, and Neural PLS. In the context of interface tracking, our method differs from PLS and NB-FLIP by utilizing a single layer of particles carrying orientation information, combined with an adaptive spatial discretization approach. Unlike NB-FLIP, which calculates level-set values using

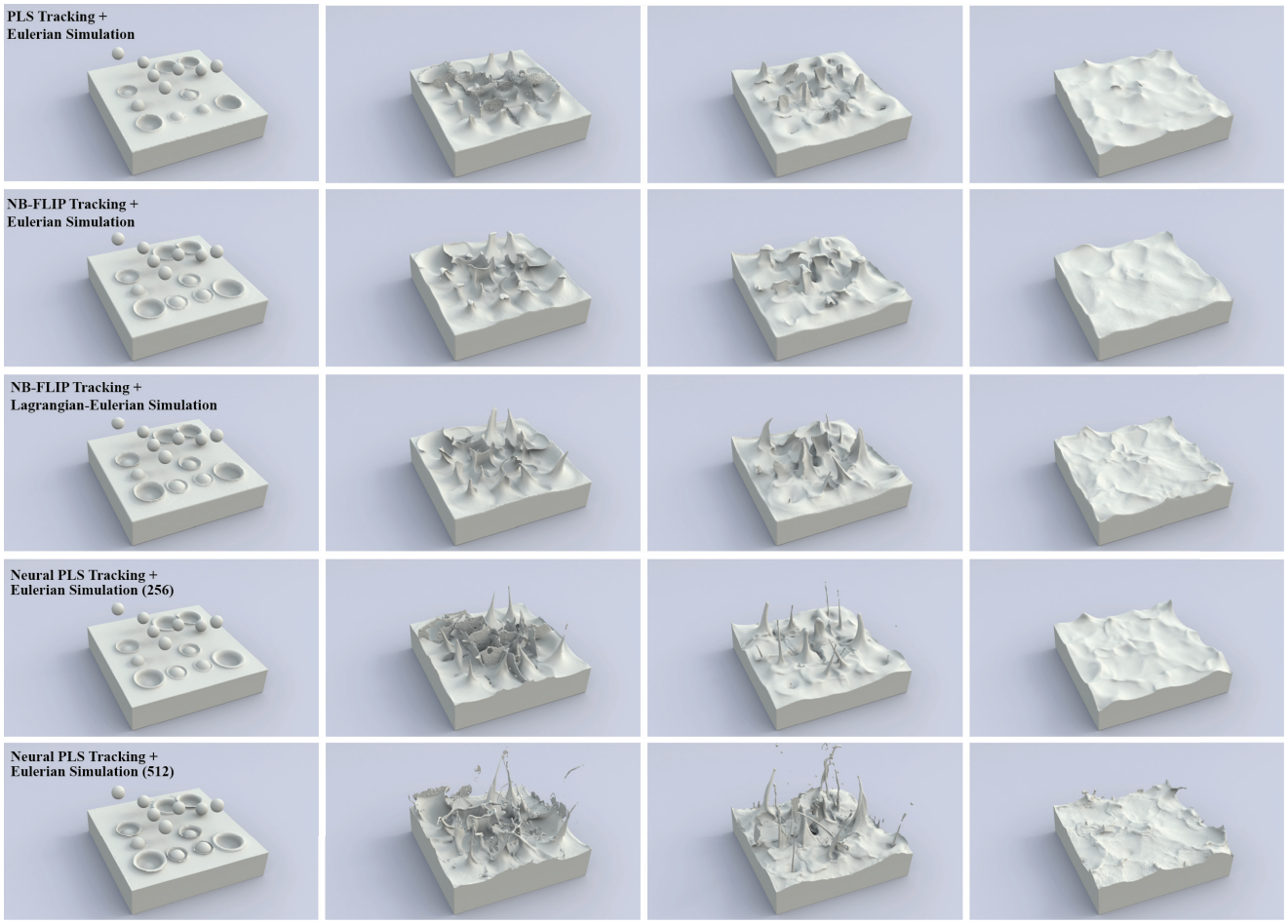


Fig. 25. We compare the simulation results of random sphere dropping into tank using PLS (first row), NB-FLIP method with Eulerian fluid solver (second row), NB-FLIP method with Lagrangian–Eulerian fluid solver (third row), our Neural PLS method with Eulerian simulation at resolution of 256^3 (fourth row) and resolution of 512^3 (last row). Here the names are represented as interface tracking method + underlying type fluid solver.

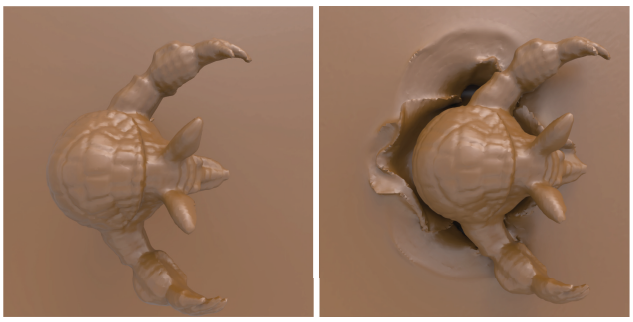


Fig. 26. We show a comparison of details on Armadillos stay unchanged when it merges into fluid (*right*) from the initial setting (*left*).

a union of spheres and requires reinitialization via fast marching or fast sweeping, or PLS, which uses a correction mechanism with similar requirements, our method directly calculates level-set values along the normal direction at each point. As previously discussed, this is facilitated by the adaptiveness inherent in the instant-NGP training process. As shown in a series of experiments (see Figures 10, 11 and 21), our Neural PLS method serves as an effective interface-tracking algorithm (the same role as the PLS method yet with improved tracking accuracy), indicating its potential for large-scale applications. Conversely, the NB-FLIP algorithm, while highly effective as a fluid simulation tool for generating dynamic free-surface flows, has inherent limitations in serving as a dedicated interface-tracking tool like PLS or Neural PLS. This limitation is partially attributed to its single-sided particle layout, making it less effective in tracking smooth surface evolution compared to PLS or Neural PLS. When it comes to simulation, NB-FLIP produces energetic free surfaces when it is used as a fluid simulator and as an interface tracker simultaneously. When used solely for interface tracking, it exhibits behavior similar to PLS. Our Neural PLS method, despite not feeding particle velocities to the grid, yields results that surpass those of PLS and are comparable to those obtained with the full NB-FLIP approach. We attribute this to the particle's gradient information and the adaptivity provided by Instant-NGP, which helps reduce error and dissipation during interface advection. In conclusion, as a dynamic interface tracking tool, our method outperforms PLS, and both our method

and PLS show better performance than NB-FLIP. Within a fluid simulation framework, our method provides results comparable to the conventional NB-FLIP method (when used as both tracker and simulator) and outperforms PLS.

Relation between sparse LS, sparse PLS, and neural PLS. We discuss the relationship between our method with sparse LS and sparse PLS method. On one hand, the high memory footprint of PLS stems from the large number of particles needed per grid cell for accurate level-set advection, typically 64 or more, to densely sample a narrowband around the interface [Enright et al. 2002]. For instance, as shown in Figure 22, PLS requires about 153 M particles to achieve comparable results with our method using only 5.7 M oriented particles (plus 0.63 M temporary particles every several training iterations). This reduction is due to differences in sampling: PLS densely populates a $\pm 2\Delta x$ narrowband, while our approach tracks only interface particles (i.e., those with zero level-set value), eliminating the need for extensive sampling across the entire narrowband. Such results can also be observed in Figures 11 and 10. Since 61% of PLS's memory cost is tied to narrowband particles, reducing sparsity outside this region does little to mitigate its overall memory demand. On the other hand, the expressiveness of the level-set representation depends on particle usage. As shown in Figure 11, a PLS representation on a low-resolution grid (e.g., $\Delta x = 1/800$) captures interface details as effectively as a standard level set on a high-resolution grid ($\Delta x = 1/3200$), despite a 4x resolution gap. Therefore, we prefer utilizing particles over increasing grid resolution (and its sparse accelerations) to enhance interface accuracy in dynamic settings.

Oriented Particles for Training. We discuss the rationale for using oriented particles to sample training points. As outlined in Section 7.1, one alternative method for calculating a point's level-set value is to trace the current point back to the previous timestep's encoded SDF and use its value as the ground truth. However, this method does not produce satisfactory results. As shown in Figure 18, the advected level-set value does not remain precise, resulting in an unsmooth surface and blurred details. Another alternative is randomly sampling points around the surface particles within a small narrowband and computing each point's distance to the surface for the current time step. However, employing K-nearest neighbor searches or Octree spatial hashing methods to locate the closest surface points for each sampled narrowband point and compute the corresponding distance would introduce substantial computational overhead. In particular, approximately 2,000,000 surface particles are used in the training loop, and training points must be re-sampled within the narrowband to train the I-NGP every few iterations using a portion of the surface particles, both render this strategy impractical for a high-resolution setting (e.g., Figures 2 and 25). In contrast, our method can directly give the sampled particle an SDF value using carried orientation information which allows a faster performance.

9 Conclusion

In this article, we introduce a novel hybrid neural-discrete data structure that combines multi-resolution hash encoding and sparse grids to represent dynamic level set functions with adaptivity. Our key contribution lies in the development of an efficient neural-discrete data structure that enables accurate interface tracking and facilitates complex physics simulations. This representation decouples the location and orientation of the interface, enabling accurate tracking and reconstruction of dynamic implicit interface with efficient neural network training. Our method demonstrates, for the first time, the potential use of INRs to accommodate dynamic interface tracking applications with complex geometric and topological characteristics.

To authenticate the efficacy of our proposed method, we undertake exhaustive advection tests, comparing our results against those obtained using the PLS method, NB-FLIP method, and the high-resolution level set method. The experimental outcomes underscore the effectiveness of our method in conserving sharp and thin features comparable to those derived from super-resolution level set methods on large-scale clusters, all within a reasonable computational resource budget. Notably, all our simulations were executed on a single workstation. Moreover, we employ our method in conducting physical fluid simulations, successfully capturing the complex motions of free-surface fluids while preserving fine fluid details. These simulations illustrate the practicality and efficiency of our proposed neural PLS framework in real-world simulation scenarios by supporting geometric and topological evolution.

Limitation and future work. Firstly, our method currently employs the I-NGP structure as our neural representation. Despite its known rapid convergence, we acknowledge that the hash collision characteristic of its design might not be ideal for our specific use case of encoding level set values. Since each level set value is unique to a spatial location, hash collisions on a high-resolution hash grid could potentially affect our method's performance. An anticipated improvement involves replacing spatial hashing with a sparse grid in the I-NGP backend, though this may slow training speed.

Secondly, we aspire to broaden our method to accommodate non-manifold level set tracking in future research. In non-manifold applications, distinct regions with different labels are retained to calculate simulation velocity and enable visualization. Our current method, while capable of differentiating between two types of regions, cannot dynamically track multiple regions bearing different labels. The challenge arises from oriented particles not carrying region labels, making it difficult to transform the unsigned distance field to regions with different labels in non-manifold applications. This issue is targeted for resolution in our future work.

Thirdly, our current method struggles with tracking tiny splashes in fluid simulations. This is due to our sampling method necessitating non-overlapping sampled narrowbands. As a result, we currently remove oriented particles to prevent overlap, which compromises our ability to track small splashes. One potential approach to address this problem is to adopt the solution presented in [Losasso et al. 2008], where splashes are artificially created and deleted using escaped particles. Furthermore, we aim at enhancing our sampling method to naturally handle such situations.

In addition to the aforementioned limitations, we are also interested in further exploring the use of neural representations in physical simulations. We aim at developing neural-discrete hybrid

Table 1. **Results: Details of Experiments**

Advection Test	Maximum Number of Particles	Network Param	Training Time	Advection Time
Zale's Disk	28,000	L:16 H:2^{16} S:1.5	2.17 sec/frame	0.0003 sec/step
Spike Disk	28,000	L:16 H:2^{16} S:1.5	2.19 sec/frame	0.0003 sec/step
Single Vortex	200,000	L:16 H:2^{16} S:1.5	3.74 sec/frame	0.0018 sec/step
Crazy Rotation	600,000	L:16 H:2^{19} S:1.5	9.43 sec/frame	0.0028 sec/step
3D Armadillos Rotation	200,000	L:16 H:2^{19} S:1.5	9.11 sec/frame	0.0024 sec/step
3D Deformation Field	6,000,000	L:16 H:2^{19} S:1.5	22.03 sec/frame	0.0795 sec/step
3D Deformation Field (Longer Version)	50,000,000	L:21 H:2^{21} S:1.5	131.35 sec/frame	0.445 sec/step

In the column of *Network Param*, L represents the level of resolution, H represents the hash table size, and S represents the growing factor between levels of resolution. Notice in *Network Param*, the hyperparameters can be tuned to achieve a balance between accuracy and running time.

Table 2. **Results: Experiment Details of Compared Methods**

Advection Test	Method	Resolution	Total Time	Advection Time
Zale's Disk	Level Set Method	1/3,000	0.0236 sec/frame	0.0027 sec/step
Zale's Disk	Level Set Method	1/6,000	0.0563 sec/frame	0.0072 sec/step
Zale's Disk	PLS Method	1/1,500	0.0175 sec/frame	0.0002 sec/step
Spike Disk	Level Set Method	1/1,600	0.0126 sec/frame	0.0012 sec/step
Spike Disk	Level Set Method	1/3,200	0.0269 sec/frame	0.0041 sec/step
Spike Disk	PLS Method	1/800	0.0105 sec/frame	0.0002 sec/step
Single Vortex	Level Set Method	1/1,600	0.0011 sec/frame	0.0004 sec/step
Single Vortex	Level Set Method	1/3,200	0.0011 sec/frame	0.0004 sec/step
Single Vortex	NB-FLIP Method	1/800	0.0288 sec/frame	0.0010 sec/step
Single Vortex	PLS Method	1/800	0.0051 sec/frame	0.0003 sec/step
Crazy Rotation	PLS Method	1/5,000	0.0208 sec/frame	0.0005 sec/step
3D Deformation Field	PLS Method	1/500	0.3160 sec/frame	0.0005 sec/step
3D Deformation Field	NB-FLIP Method	1/500	2.7295 sec/frame	0.0010 sec/step

We list the method names, resolution, and timing from other methods we compared our method with.

Table 3. **Results: Area/Volume Loss** of Our Result Compared Over Level Set Method, PLS Method, and NB-FLIP Method in Different Advection Tests

Advection Test	Method	Resolution	% Area/Volume Loss
Zale's Disk	Level Set Method	1/3,000	0.2666
Zale's Disk	Level Set Method	1/6,000	0.0271
Zale's Disk	PLS Method	1/1,500	**0.0073**
Zale's Disk	Ours	1/1,500 (Visualization)	0.0074
Spike Disk	Level Set Method	1/1,600	-0.3298
Spike Disk	Level Set Method	1/3,200	-0.2831
Spike Disk	PLS Method	1/800	0.0165
Spike Disk	Ours	1/800 (Visualization)	**-0.0150**
Single Vortex	Level Set Method	1/1,600	0.6685
Single Vortex	Level Set Method	1/3,200	0.1371
Single Vortex	PLS Method	1/800	0.0610
Single Vortex	FLIP Method	1/800	-0.0661
Single Vortex	NB-FLIP Method	1/800	2.229
Single Vortex	Ours	1/800 (Visualization)	**0.0264**
3D Deformation Field	PLS Method	1/500	0.7677
3D Deformation Field	NB-FLIP Method	1/500	12.3052
3D Deformation Field	Ours	1/1,500 (Visualization)	**-0.0008**

As our method uses oriented particles with neural representation, grid resolution for comparison is not well-defined. Therefore, *Visualization* in the table represents the resolution we perform marching cube/square.

methods that leverage the advantages of both approaches, bridging the gap between physical simulations and neural representations. Our goal is to achieve computational efficiency while retaining high fidelity in the simulation results.

Acknowledgements

We express our gratitude to the anonymous reviewers for their insightful feedback. We also thank Yuchen Sun, Diyang Zhang, Fan Feng, Yitong Deng and Mengdi Wang for their insightful

discussion. We credit the Houdini education license for video animations.

References

Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Transactions on Graphics* 36, 4 (2017), 1–12.

Matan Atzmon, Niv Haim, Lior Yariv, Ofer Israelov, Haggai Maron, and Yaron Lipman. 2019. Controlling neural level sets. In *33rd Conference on Neural Information Processing Systems*. 2034–2043.

Arne Bøckmann and Magnus Vartdal. 2014. A gradient augmented level set method for unstructured grids. *J. Comput. Phys.* 258, C (2014), 47–72.

Morten Bojsen-Hansen and Chris Wojtan. 2013. Liquid surface tracking with error compensation. *ACM Transactions on Graphics* 32, 4 (2013), 1–13.

Tyson Brochu and Robert Bridson. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing* 31, 4 (2009), 2472–2493.

Rohan Chabra, Jan E. Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. 2020. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *Proceedings of the 16th European Conference on Computer Vision–ECCV 2020, Glasgow, UK, August 23–28, 2020, Part XXIX 16*. Springer, 608–625.

Jumyung Chang, Ruben Partono, Vinicius C. Azevedo, and Christopher Batty. 2022. Curl-Flow: Boundary-respecting pointwise incompressible velocity interpolation for grid-based fluids. *ACM Transactions on Graphics* 41, 6 (2022), 1–21.

Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2023b. Implicit neural spatial representations for time-dependent pdes. In *International Conference on Machine Learning*. PMLR, 5162–5177.

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.

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5939–5948.

Nuttapong Chentanez and Matthias Müller. 2011. Real-time eulerian water simulation using a restricted tall cell grid. In *Proceedings of the ACM Siggraph 2011 Papers*. 1–10.

Mengyu Chu, Lingjie Liu, Quan Zheng, Erik Franz, Hans-Peter Seidel, Christian Theobalt, and Rhaleb Zayer. 2022. Physics informed neural fields for smoke reconstruction with sparse data. *ACM Transactions on Graphics* 41, 4 (2022), 1–14.

Fang Da, Christopher Batty, and Eitan Grinspun. 2014. Multimaterial mesh-based surface tracking. *ACM Transactions on Graphics* 33, 4 (2014), 112–1.

Yitong Deng, Mengdi Wang, Xiangxin Kong, Shiying Xiong, Zangyueyang Xian, and Bo Zhu. 2022. A moving eulerian-lagrangian particle method for thin film and foam simulation. *ACM Transactions on Graphics* 41, 4 (2022), 1–17.

Todd F Dupont and Yingjie Liu. 2003. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *Journal of Computational Physics* 190, 1 (2003), 311–324.

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.

Doug Enright, Duc Nguyen, Frederic Gibou, and Ron Fedkiw. 2003. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proceedings of the Fluids Engineering Division Summer Meeting*. 337–342.

Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. 2016. Narrow band FLIP for liquid simulations. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 225–232.

Nick Foster and Ronald Fedkiw. 2001. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. 23–30.

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.

Ming Gao, Andre Pradhana Tampubolon, Chenfanfu Jiang, and Eftychios Sifakis. 2017. An adaptive generalized interpolation material point method for simulating elastoplastic materials. *ACM Transactions on Graphics* 36, 6 (2017), 1–12.

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

Frederic Gibou, Ronald P. Fedkiw, Li-Tien Cheng, and Myungjoo Kang. 2002. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics* 176, 1 (2002), 205–227.

Ryan Goldade, Christopher Batty, and Chris Wojtan. 2016. A practical method for high-resolution embedded liquid surfaces. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 233–242.

Nambin Heo and Hyeong-Seok Ko. 2010. Detail-preserving fully-eulerian interface tracking framework. *ACM Transactions on Graphics* 29, 6 (2010), 176.

Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2021. Sape: Spatially-adaptive progressive encoding for neural optimization. *Advances in Neural Information Processing Systems* 34 (2021), 8820–8832.

Simone E. Hieber and Petros Koumoutsakos. 2005. A lagrangian particle level set method. *Journal of Computational Physics* 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), 201.

Sandro Ianniello and Andrea Di Mascio. 2010. A self-adaptive oriented particles level-set method for tracking interfaces. *Journal of Computational Physics* 229, 4 (2010), 1353–1380.

Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *Proceedings of the ACM SIGGRAPH 2006 Papers*. 805–811.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The material point method for simulating continuum materials. In *Proceedings of the ACM SIGGRAPH 2016 Courses*. 1–52.

Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser 2020. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6001–6010.

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 (2023), 1–14.

Doyub Kim, Minjae Lee, and Ken Museth. 2024. Neuralvdb: High-resolution sparse volume representation using hierarchical neural networks. *ACM Transactions on Graphics* 43, 2 (2024), 1–21.

Shingyu Leung, John Lowengrub, and Hongkai Zhao. 2011. A grid based particle method for solving partial differential equations on evolving surfaces and modeling high order geometrical motion. *Journal of Computational Physics* 230, 7 (2011), 2540–2561.

Shingyu Leung and Hongkai Zhao. 2009a. A grid based particle method for evolution of open curves and surfaces. *Journal of Computational Physics* 228, 20 (2009), 7706–7728.

Shingyu Leung and Hongkai Zhao. 2009b. A grid based particle method for moving interface problems. *Journal of Computational Physics* 228, 8 (2009), 2993–3024.

Frank Losasso, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating water and smoke with an octree data structure. In *Proceedings of the ACM SIGGRAPH 2004 Papers*. 457–462.

Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. 2008. Two-way coupled SPH and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics* 14, 4 (2008), 797–804.

Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2024. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *2024 International Conference on 3D Vision (3DV)*. IEEE, 800–809.

Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. 2021. Acorn: Adaptive coordinate networks for neural scene representation. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–13.

Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2022. A level set theory for neural implicit evolution under explicit flows. In *Proceedings of the 17th European Conference on Computer Vision–ECCV 2022, Tel Aviv, Israel, October 23–27, 2022 Part II*. Springer, 711–729.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4460–4470.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. SymPy: Symbolic computing in python. *PeerJ Computer Science* 3 (2017), e103. DOI: https://doi.org/10.7717/peerj-cs.103

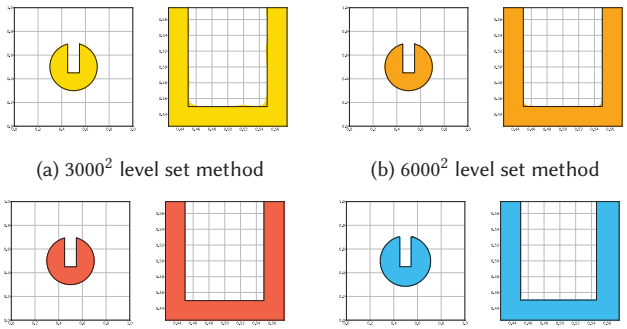
Mohammad Mirzadeh, Arthur Guittet, Carsten Burstedde, and Frederic Gibou. 2016. Parallel level-set methods on adaptive tree-based grids. *Journal of Computational Physics* 322, C (2016), 345–364.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics* 41, 4 (2022), 1–15.

Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics* 32, 3 (2013), 1–22.

Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. 2010. A gradient-augmented level set method with an optimally local, coherent advection scheme. *Journal of Computational Physics* 229, 10 (2010), 3802–3827.

Tiago Novello, Vinicius Da Silva, Guilherme Schardong, Luiz Schirmer, Helio Lopes, and Luiz Velho. 2023. Neural implicit surface evolution. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 14233–14243.
Stanley Osher and Ronald P. Fedkiw. 2005. *Level Set Methods and Dynamic Implicit Surfaces*. Springer New York.
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. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 165–174.
Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B. Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. 2021. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5865–5874.
Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional occupancy networks. In *Proceedings of the 16th European Conference on Computer Vision–ECCV 2020, Glasgow, UK, August 23–28, 2020, Part III 16*. Springer, 523–540.
Takahiro Sato, Christopher Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. 2018. Extended narrow band FLIP for liquid simulations. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 169–177.
Robert Saye. 2014. High-order methods for computing distances to implicitly defined surfaces. *Communications in Applied Mathematics and Computational Science* 9, 1 (2014), 107–141.
Craig Schroeder, Ritoban Roy Chowdhury, and Tamar Shinar. 2022. Local divergence-free polynomial interpolation on MAC grids. *Journal of Computational Physics* 468, C (2022), 111500.
Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. 2005. A vortex particle method for smoke, water and explosions. In *Proceedings of the ACM SIGGRAPH 2005 Papers*. 910–914.
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* 33, 6 (2014), 1–12.
James A. Sethian. 1999. Fast marching methods. *SIAM Review* 41, 2 (1999), 199–235.
Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems* 33 (2020), 7462–7473.
Christiane Sommer, Lu Sang, David Schubert, and Daniel Cremers. 2022. Gradient-sdf: A semi-implicit surface representation for 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6280–6289.
Jos Stam. 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. 121–128.
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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11358–11367.
Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems* 33 (2020), 7537–7547.
Magnus Vartdal and Arne Bøckmann. 2013. An oriented particle level set method based on surface coordinates. *Journal of Computational Physics* 251 (2013), 237–250.
Hui Wang, Yongxu Jin, Anqi Luo, Xubo Yang, and Bo Zhu. 2020. Codimensional surface tension flow using moving-least-squares particles. *ACM Transactions on Graphics* 39, 4 (2020), 42–1.
Mengdi Wang, Yitong Deng, Xiangxin Kong, Aditya H Prasad, Shiying Xiong, and Bo Zhu. 2021. Thin-film smoothed particle hydrodynamics fluid. *ACM Transactions on Graphics* 40, 4 (2021), 1–16.
Chris Wojtan, Matthias Müller-Fischer, and Tyson Brochu. 2011. Liquid simulation with mesh-based surface tracking. In *Proceedings of the ACM SIGGRAPH 2011 Courses*. 1–84.
Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. In *Proceedings of the Computer Graphics Forum*. Wiley Online Library, 641–676.
Wang Yifan, Shihao Wu, Cengiz Oztireli, and Olga Sorkine-Hornung. 2021. Iso-points: Optimizing neural implicit surfaces with hybrid representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 374–383.
Steven T. Zalesak. 1979. Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of Computational Physics* 31, 3 (1979), 335–362.
Hongkai Zhao. 2005. A fast sweeping method for eikonal equations. *Mathematics of Computation* 74, 250 (2005), 603–627.
Lanhao Zhao, Hongvan Khuc, Jia Mao, Xunnan Liu, and Eldad Avital. 2018. One-layer particle level set method. *Computers and Fluids* 170 (2018), 141–156.



(a) 3000^2 level set method (b) 6000^2 level set method

(c) 1500^2 particle level set method (d) 1500^2 ours

Fig. 27. The Zale Disk is evolved in an implicit rigid rotation velocity field. In each method, the simulation results are depicted in the left figure, with the zoomed-in results showcased in the right figure. The black outline represents the isosurface at $t = 0$, while the colored shape represents the isosurface at $t = 4$.

Supplement

1 Zalesak's Disk

Consider a Zalesak's Disk, centered at $[0.5, 0.5]$ with a radius of 0.2, as an example to demonstrate the diffusion error inherent in the level set method [Zalesak 1979]. The constant velocity field is defined as $u = -\pi(y-0.5)$ and $v = \pi(x-0.5)$. The initial isosurface is represented by a boolean field with a resolution of 400^2, with the isosurface and the rest denoted by 1 and 0, respectively. The reconstructed level set is initially obtained via a standard fast marching method. Thereafter, we employ the Newton iteration method in conjunction with the method described in [Yifan et al. 2021] to project randomly seeded points onto the isosurface. The normal vector at each point is then computed using finite differences with normalization. We use $\Delta t = 0.002$ in this test. Figure 27 presents the results derived from performing marching squares on a $1,500^2$ field. The same three methods as in the Spike Disk example are employed to provide results for comparison, as displayed in Figure 27 and Table 3. Our method effectively retains the disk shape after two rotation cycles, with an area loss nearly identical to that simulated through the PLS method under the same visualization resolution. Additionally, our method outperforms the level set method at 2x and 4x higher resolutions (See Table 3 for numerical comparison). This demonstrates the effectiveness of our method in overcoming diffusion errors that arise from interpolation during the advection process of the level set method.

2 Deformation Field

To demonstrate our method's capacity to recover the original shape in the face of a severely distorted velocity field, we employ a field defined in [Enright et al. 2002] as $u = \sin(\pi(-4x+6))\sin(\pi(-4y+6))$ and $v = \cos(\pi(-4x+6))\cos(\pi(-4y+6))$. We consider a circle with a radius of 0.15 centered at $[0.5, 0.75]$ and reverse the velocity field at $t = 1$, with periodic boundary conditions imposed. The surface points are again uniformly sampled on the circle, and their normals are computed in the same way as Section 5.2 as $normalize(\mathbf{x} - \mathbf{c})$. We conduct marching squares on $1,500^2$ grids and compare the

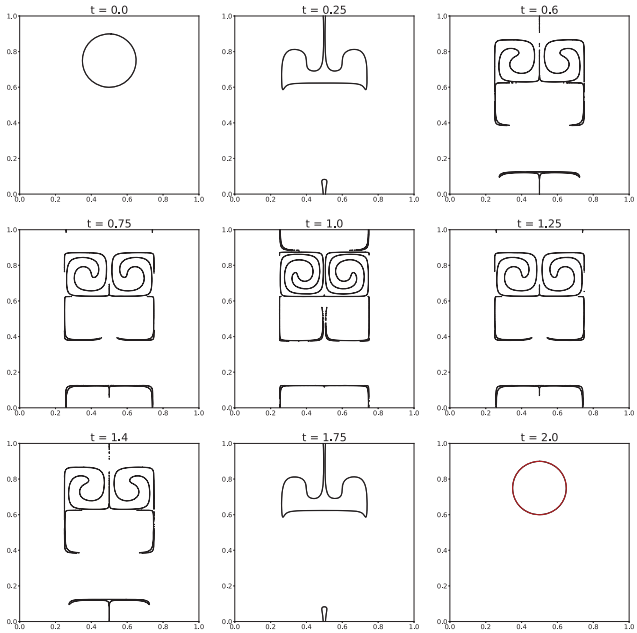


Fig. 28. A disk is evolved in a deformation field [Enright et al. 2002]. Red circle in the final frame represents the circle at $t = 0$ and black circle is the isosurface at the end of the simulation.

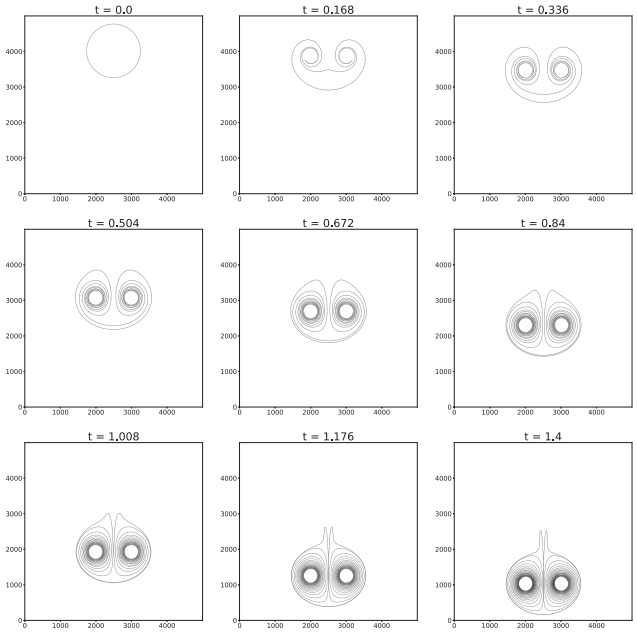


Fig. 29. A disk is evolved in a velocity field simulated with vortex particle method [Selle et al. 2005] using two vortex particles with opposite vorticity.

original shape with the final shape obtained from our simulation. The results are presented in Figure 28.

3 2D Vortex Particle

In order to further probe the robustness of our method in implicit time-varying velocity fields, we adopted the concept of vortex particles to construct a velocity field that simulates vortex movement, as delineated in [Selle et al. 2005]. This method generates a divergence-free velocity field using fictitious vortex particles by applying Biot–Savart's law to a Green's function that provides an analytical solution to a Poisson equation. To compute the velocity of a particle of interest located at $\mathbf{x}$, we consider the i^{th} vortex particle denoted as $\mathbf{v}_{\mathbf{p}_i}$ with vorticity ω_i. The velocity of the particle of interest at each timestep is then given by $\mathbf{r}_i = \mathbf{x} - \mathbf{v}_{\mathbf{p}_i}$, $c_i = (1 - \exp(-|\mathbf{r}_i|^2/\epsilon^2))/(2\pi|\mathbf{r}_i|^2 + s)$, $u = \sum_i -\mathbf{r}_{iy}\omega_i c_i$ and $v = \sum_i \mathbf{r}_{ix}\omega_i c_i$. To prevent numerical issues, we use $s = 1^{-10}$ and $\epsilon = 0.001$. In our demonstration, we seeded two vortex particles with vorticities of 4 and -4 at locations $[0.4, 0.85]$ and $[0.6, 0.85]$, respectively. We positioned a circle with a radius of 0.15 and the center located at $[0.5, 0.8]$. At each timestep, the vortex particles were advected through a straightforward explicit Euler scheme, and surface particles were advected as delineated in Section 4.1. We calculated $\nabla\mathbf{u}$ using Sympy [Meurer et al. 2017]. The results are displayed in Figure 29.

Received 15 November 2023; revised 12 February 2025; accepted 25 March 2025