Fluid Simulation on Neural Flow Maps

YITONG DENG, Dartmouth College
HONG-XING YU, Stanford University
DIYANG ZHANG, Dartmouth College
JIAJUN WU, Stanford University
BO ZHU, Georgia Institute of Technology, Dartmouth College

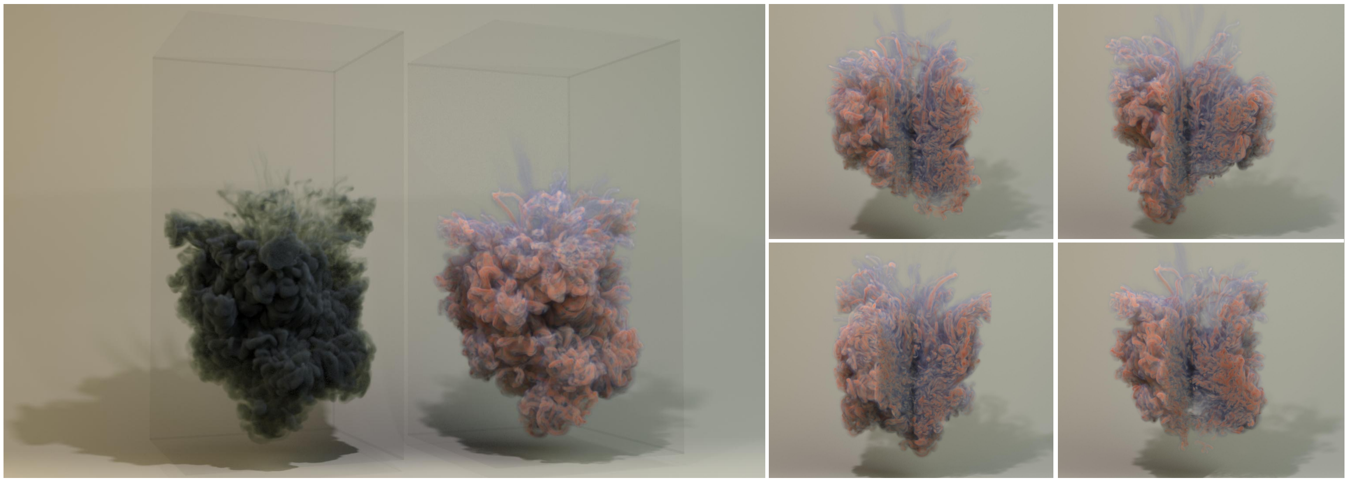

Fig. 1. Inkdrop simulated by Neural Flow Maps, a novel, high-fidelity simulation system built upon implicit neural representations. The left panel juxtaposes the ink (left) and the underlying bundle of vortex filaments (right). The right panel shows the vorticity (magnitude) field from four different angles, with the camera-facing sections culled to reveal the sophisticated internal structures.

We introduce Neural Flow Maps, a novel simulation method bridging the emerging paradigm of implicit neural representations with fluid simulation based on the theory of flow maps, to achieve state-of-the-art simulation of inviscid fluid phenomena. We devise a novel hybrid neural field representation, Spatially Sparse Neural Fields (SSNF), which fuses small neural networks with a pyramid of overlapping, multi-resolution, and spatially sparse grids, to compactly represent long-term spatiotemporal velocity fields at high accuracy. With this neural velocity buffer in hand, we compute long-term, bidirectional flow maps and their Jacobians in a mechanistically symmetric manner, to facilitate drastic accuracy improvement over existing solutions. These long-range, bidirectional flow maps enable high advection accuracy with low dissipation, which in turn facilitates high-fidelity incompressible flow simulations that manifest intricate vortical structures. We demonstrate the efficacy of our neural fluid simulation in a variety of challenging simulation scenarios, including leapfrogging vortices, colliding vortices, vortex reconnections, as well as vortex generation from moving obstacles and density differences. Our examples show increased performance over existing methods in terms of energy conservation, visual complexity, adherence to experimental observations, and preservation of detailed vortical structures.

CCS Concepts: • **Computing methodologies → Modeling and simulation**.

Additional Key Words and Phrases: Implicit neural representation, neural fluid simulation, incompressible flow, vortical flow, flow maps

ACM Reference Format:
Yitong Deng, Hong-Xing Yu, Diyang Zhang, Jiajun Wu, and Bo Zhu. 2023. Fluid Simulation on Neural Flow Maps. *ACM Trans. Graph.* 42, 6 (December 2023), 21 pages. https://doi.org/10.1145/3618392

1 INTRODUCTION

Implicit neural representations (INR) have emerged as a remarkable new class of data primitives in visual computing, as an attractive alternative to traditional, explicit representations such as grids, meshes, and particles, demonstrating its distinct advantages in representing various categories of continuous fields in appearance modeling, geometry processing, and 3D reconstruction, just to name a few. Instead of representing a field by storing a large number of samples at discrete locations, an INR employs a continuous query function in the form of neural networks that evaluates the field at arbitrary input coordinates in the domain. Such an alternative representation has proven to be uniquely advantageous in a variety

---

Authors' addresses: Yitong Deng, Dartmouth College, f003mzt@dartmouth.edu; Hong-Xing Yu, Stanford University, koven@cs.stanford.edu; Diyang Zhang, Dartmouth College, f006h07@dartmouth.edu; Jiajun Wu, Stanford University, jiajunwu@cs.stanford.edu; Bo Zhu, Georgia Institute of Technology, Dartmouth College, bo.zhu@gatech.edu.

0730-0301/2023/12-ART $15.00
https://doi.org/10.1145/3618392

of scenarios for being spatially adaptive, differentiable, agnostic to topology and dimensionality, and memory efficient; and it has empowered many state-of-the-art advances in 3D computer vision (e.g., Instant NGP [Müller et al. 2022]). One fundamental reason for this is that INRs are highly efficient and adaptive in memory, as it scales up with the *complexity* of the signals instead of the *resolution* [Sitzmann et al. 2019], hence allowing the high-fidelity representation of high-dimensional signals such as 5D Neural Radiance Fields [Mildenhall et al. 2020]. In spite of its typical association with radiance fields in rendering and reconstruction tasks, INRs are developing into a fundamentally general field representation paradigm whose virtues promise to be leveraged in a plethora of other disciplines.

Physical simulation, on the other hand, is a discipline in which fields are first-class citizens. When devising a simulation algorithm, a foundational first step is to design an appropriate representation for the dynamical fields: *e.g.*, velocity, pressure, density, and temperature, on which the numerical solving of PDEs can take place. While the options are traditionally different flavors of meshes and particles, the rapid development of INR elicits the curiosity of whether the adoption of such a novel implicit paradigm can fundamentally unlock new horizons of what physical simulation algorithms can achieve. Unfortunately, the *drop-in* incorporation of INRs in the simulation pipeline has not yet been lucrative. A major reason for this is that the principal advantage of INRs, *i.e.*, their spatially adaptive, dimensionality-agnostic, high memory efficiency, does not exhibit clear advantages in a typical physical simulation pipeline, which operates on dense fields with only spatial dimensions. Therefore, dynamical simulation using INRs has so far remained a “workable concept” that nevertheless shows no concrete advantage over existing paradigms. From our perspective, this conundrum is fundamentally caused by the current *mismatch* between the simulation algorithm’s demands and the INR’s capacities. As with grids, meshes, and particles, an INR’s advantages and drawbacks need to be carefully aligned with the simulation scheme to fulfill its full potential.

In this work, we consolidate INRs’ potency in encoding spatiotemporal signals at a small memory footprint with flow map-based fluid advection, an accurate and non-dissipative advection scheme that has been hindered by its intractable memory requirement [Sato et al. 2018]. To fulfill the full potential of flow map advection at a viable memory cost, we propose Neural Flow Maps (NFM), a novel simulation method that realizes accurate, long-range flow maps with neural representations. The core component of NFM is the Spatially Sparse Neural Fields (SSNF), a hybrid INR employing a pyramid of multi-resolution, spatially sparse feature grids to maintain long-range buffers of spatiotemporal velocity fields. We show that for this purpose, our SSNF offers improved accuracy, training speed, and memory efficiency compared to state-of-the-art representations (*e.g.*, Instant NGP [Müller et al. 2022], K-Planes [Fridovich-Keil et al. 2023], and SIREN [Sitzmann et al. 2020]), reducing the fitting error by over 70%. Leveraging this effective neural structure, we compute high-quality bidirectional flow maps by marching the SSNF forward and backward in time, and consolidate them with the impulse-based fluid simulation method to drastically reduce simulation errors by over 90% with respect to analytical solutions. In terms of practicality, our neural method is still significantly slower than traditional methods like Stable Fluids [Stam 1999], typically by an order of magnitude, rendering it unsuited for real-time applications, albeit realistic for quality-focused applications, a realm in which our method brings substantial advantages. These advantages are experimentally validated in a variety of challenging scenarios including leapfrogging vortices, vortex reconnections, and vortex generation from obstacles and density difference, in which our method consistently achieves the state-of-the-art level of energy conservation, visual complexity, and recreation of real-world phenomena. Since the cost of our cutting-edge results hinges on the efficiency of neural techniques, our method effectively reformulates a long-standing simulation problem into a machine-learning one, thereby opening up brand new avenues for the former that promise to leverage the latter’s revolutionary advancements of late.

We summarize our main contributions as follows:

1. We present Neural Flow Maps (NFM), the first INR-based dynamic simulation method to achieve state-of-the-art performance in terms of both visual quality and physical fidelity.
2. We propose Spatially Sparse Neural Fields (SSNF), a novel hybrid INR that is fast to train, compact, and collision-free. Our SSNF achieves higher fitting accuracy than the state-of-the-art methods at a lower computational cost.
3. We introduce a novel scheme for learning neural buffers of long-term spatiotemporal signals, from *streams* of input frames with variable durations.
4. We introduce a novel flow map advection scheme based on the forward and backward marching of velocity buffers to represent accurate and consistent bidirectional mappings and their Jacobians to enable high-quality fluid simulations.

2 RELATED WORKS

2.1 Machine Learning for Fluids

Machine learning has been successfully applied to the study of fluids in a number of directions, as we will briefly survey. Readers can refer to comprehensive surveys [Brunton et al. 2020; Chen et al. 2021; Majchrzak et al. 2023; Sharma et al. 2023] for in-depth discussions.

Synthesis. Machine learning can synthesize visually appealing details from coarse simulations. For instance, Chu and Thuerey [2017] adopt convolutional neural networks to enhance coarse smoke simulations with pre-computed patches; Kim et al. [2019b] synthesize divergence-free fluid motion in a reduced space; Xie et al. [2018] and Werhahn et al. [2019] use generative adversarial networks to perform fluid super-resolution, an approach adapted by Chu et al. [2021] to enable user control. Roy et al. [2021] upsample particle-based simulations by learning a deformation field using an architecture adapted from FlowNet3D [Liu et al. 2019]. Physics-inspired syntheses of *stylized* fluid details have also been accomplished in the Eulerian [Kim et al. 2019a] and Lagrangian [Kim et al. 2020] settings.

Inference. Another thread of research infers fluid properties from captured videos. Earlier methods [Gregson et al. 2012; Hasinoff and Kutulakos 2007; Ihrke and Magnor 2004] infer density volumes from videos without temporal treatments. More recent works further infer velocity using physical priors [Gregson et al. 2014; Okabe et al. 2015], and allow joint optimization for velocity and density [Eckert et al. 2018, 2019; Zang et al. 2020]. Franz et al. [2021] introduce a differentiable simulator to promote temporal consistency. Physics-informed neural networks use soft constraints to infer velocity from density [Raissi et al. 2019, 2020], to which Chu et al. [2022] combine differentiable rendering to *end-to-end* infer flow from videos.

Inverse and Control Problems. Learned fluid models can also benefit computational design and control tasks. Wandel et al. [2020] control the vortex shedding frequency based on a learned physics-informed network, which is extended by Nava et al. [2022] to control a 2D soft swimmer. Li et al. [2018] learn particle-based simulators with gradient-based trajectory optimization for fluid manipulation, while Li et al. [2021a] learn 3D fluid scenes based on 2D observations to enable gradient-free model-predictive control.

Acceleration. Data-driven simulators can emulate traditional simulators at a reduced time cost, by replacing expensive iterative solvers with more efficient operations like convolution [Tompson et al. 2017] and message passing [Li and Farimani 2022], capturing fine-grain details on coarse resolutions [Kochkov et al. 2021; Stachenfeld et al. 2021], learning reduced-order latent spaces [Kim et al. 2019b; Wiewel et al. 2019, 2020], or lifting the timestep restriction by learning long-term state transitions across multiple simulation timesteps with graph neural networks [Pfaff et al. 2020; Sanchez-Gonzalez et al. 2020], spatial convolution [Ummenhofer et al. 2020], or regression forests [Ladický et al. 2015]. These methods enable numerical accuracy to be traded for computational savings.

Physical Accuracy. Machine learning can also enhance the accuracy of traditional methods, using data to supplement the insufficiently modeled dynamics in, for instance, the Reynolds Averaged Navier–Stokes (RANS) method [Majchrzak et al. 2023], correcting the errors caused by the closure coefficients and Reynolds stress anisotropy with extended kernel regression [Tracey et al. 2013], random forest [Wang et al. 2017], field inversion and machine learning [Duraisamy et al. 2015], and neural networks [Singh et al. 2017].

2.2 Implicit Neural Representation

INRs have been used widely in visual computing applications including geometry processing [Mescheder et al. 2019; Park et al. 2019], image-based rendering [Sitzmann et al. 2019], and inverse rendering [Yu et al. 2023; Zhang et al. 2021], for being spatially adaptive, resolution-agnostic, and memory efficient. One seminal work is Neural Radiance Fields (NeRF) [Mildenhall et al. 2020] which represents a 5D radiance field with a neural network that only occupies a few tens of megabytes. However, pure neural representations suffer from their considerable time cost, and follow-up works have focused on hybrid INRs featuring classical data structures like dense [Sun et al. 2022] and sparse voxel grids [Chabra et al. 2020; Kim et al. 2022; Liu et al. 2020; Martel et al. 2021; Takikawa et al. 2021; Yu et al. 2021]. Recently, plane-based data structures [Chen et al. 2022; Fridovich-Keil et al. 2023] have also been leveraged to good effects. Most relevant to our work is Instant NGP [Müller et al. 2022], which uses multi-resolution hashing for fast training, but lacks both the explicit treatment of hash collisions and temporal modeling. Recent works have also used INRs to represent dynamic scenes, by learning deformation [Park et al. 2021a,b; Pumarola et al. 2021] and scene flow fields [Du et al. 2021; Li et al. 2022, 2021b; Xian et al. 2021].

Notation	Type	Definition
X	vector	material point position at initial state
x	vector	material point position at current state
t	scalar	time
ϕ	vector	forward map
ψ	vector	backward map
$\mathcal{F}$	matrix	forward map gradients
$\mathcal{T}$	matrix	backward map gradients
$\boldsymbol{u}$	vector	velocity
$\boldsymbol{m}$	vector	impulse
$\mathcal{N}$	SSNF	neural buffer
S	scalar	sizing field
n	scalar	reinitialization steps
σ	scalar	activation threshold
λ	scalar	time multiplier

Table 1. Symbols and notations used in this paper.

2.3 Fluid Simulation

The pioneering works by Foster and Metaxas [1997] and Stam [1999] lay the groundwork for fluid simulation in computer graphics with the employment of uniform Marker-and-Cell (MAC) grids [Harlow and Welch 1965], Chorin's projection method [Chorin 1968], and the semi-Lagrangian advection scheme [Robert 1981; Sawyer 1963]. Since then, researchers have developed upon this paradigm on multiple fronts to improve on the efficiency and accuracy.

Hierarchical Representation. Compared to uniform discretization, hierarchical or multi-resolution discretizations can capture the intricate details more efficiently by biasing the computational resources towards regions of interest, which has motivated researchers to design computational systems using nested layers of uniform grids [Johansen and Colella 1998; Martin et al. 2007; Martin and Cartwright 1996; Minion 1994; Setaluri et al. 2014], Chimera grids [English et al. 2013; Henshaw 1994], and non-uniform grids like quadtrees and octrees [Ando and Batty 2020; Batty 2017; Losasso et al. 2006a, 2004; Popinet 2003]. In addition, multi-resolution modeling has also been achieved with wavelet [Deriaz and Perrier 2006; Kevlahan and Vasilyev 2005; Schneider and Vasilyev 2010] and model-reduced methods [Mercier and Nowrouzezahrai 2020].

Numerical Dissipation Reduction. The Stable Fluids method [Stam 1999] yields a significant amount of numerical viscosity causing the results to appear blurred and damped. Researchers have addressed this artifact with error correction schemes [Kim et al. 2006; Selle et al. 2008], higher-order interpolation [Losasso et al. 2006b; Nave et al. 2009], improved backtracking schemes [Cho et al. 2018; Jameson et al. 1981], particle-based advection [Fu et al. 2017; Jiang et al. 2015; Zhu and Bridson 2005], energy-preserving integration, [Mullen et al. 2009], *a posteriori* vorticity correction [Fedkiw et al. 2001; Zhang et al. 2015], reflection [Narain et al. 2019; Strang 1968; Zehnder et al. 2018] and Lie advection [Nabizadeh et al. 2022]. Flow map methods offer another alternative option, as elaborated below.

Flow Map Methods. The method of characteristic mapping (MCM) proposed by Wiggert and Wylie [1976] pioneers in solving the fluid equations on flow maps, a technique introduced to the graphics community by Tessendorf and Pelfrey [2011]. Tracking long-term flow maps instead of fluid quantities, MCM drastically reduces the number of interpolations and hence the resulting numerical diffusion. We refer to the work by Tessendorf [2015] for more in-depth analyses and derivations on MCM. Earlier methods [Hachisuka 2005; Sato et al. 2018, 2017; Tessendorf and Pelfrey 2011] compute flow maps by tracing virtual particles which leads to a demanding time and memory cost. Qu et al. [2019] trade off accuracy for efficiency by advecting the backward map in a semi-Lagrangian-like manner; and bring in the forward map to realize the BFECC error compensation [Kim et al. 2006]. Nabizadeh et al. [2022] effectively combine bidirectional flow maps with impulse-based fluid dynamics [Buttke 1992; Feng et al. 2022; Oseledets 1989], using the mappings *along with their Jacobians* to perform circulation-preserving advection.

3 PHYSICAL MODEL

3.1 Mathematical Foundation

Flow Map Preliminaries. We start by defining a spatiotemporal velocity field $\boldsymbol{u}(\boldsymbol{q}, \tau)$ in the fluid domain Ω to specify the velocity vector at location $\boldsymbol{q}$ and time τ. For a material point $\boldsymbol{X} \in \Omega$, the corresponding forward flow map $\phi : \Omega_0 \to \Omega_t$ is defined as:

$$\begin{cases} \dfrac{\partial \phi(\boldsymbol{X}, \tau)}{\partial \tau} = \boldsymbol{u}(\phi(\boldsymbol{X}, \tau), \tau), \\ \phi(\boldsymbol{X}, 0) = \boldsymbol{X}, \\ \phi(\boldsymbol{X}, t) = \boldsymbol{x}, \end{cases} \tag{1}$$

in which $\boldsymbol{X}$ represents the material point's position at time 0, and $\boldsymbol{x}$ specifies the position of the same material point at time t. The forward mapping from $\boldsymbol{X}$ to $\boldsymbol{x}$ is determined by the trajectory of $\boldsymbol{X}$, which is calculated by integrating $\boldsymbol{u}$ within the time interval $[0, t]$.

Similarly, we define the backward flow map $\psi : \Omega_t \to \Omega_0$ as:

$$\begin{cases} \dfrac{\partial \psi(\boldsymbol{x}, \tau)}{\partial \tau} = \boldsymbol{u}(\psi(\boldsymbol{x}, \tau), \tau), \\ \psi(\boldsymbol{x}, t) = \boldsymbol{x}, \\ \psi(\boldsymbol{x}, 0) = \boldsymbol{X}. \end{cases} \tag{2}$$

Consequently, integrating $\boldsymbol{u}$ reversely from time t to time 0 on the temporal axis, starting from $\boldsymbol{x}$ in the spatial domain, yields the backward mapping from $\boldsymbol{x}$ to $\boldsymbol{X}$. Without causing confusion, in the following discussion, we will abbreviate $\phi(\boldsymbol{X}, t)$ as ϕ and $\psi(\boldsymbol{x}, 0)$ as ψ. We can also use ϕ and ψ interchangeably with $\boldsymbol{x}$ and $\boldsymbol{X}$.

As seen in Figure 2, intuitively, the mapping ϕ sends a material point to its current position at time t given its initial position; the mapping ψ backtracks a material from its current position at time t to its initial position.

We denote the spatial gradients (Jacobians) of ϕ and ψ with $\mathcal{F}$ and $\mathcal{T}$ as:

$$\begin{aligned} \mathcal{F} &= \frac{\partial \phi}{\partial \boldsymbol{X}}, \\ \mathcal{T} &= \frac{\partial \psi}{\partial \boldsymbol{x}}. \end{aligned} \tag{3}$$

Symbolically, $\mathcal{F}$ and $\mathcal{T}$ represent the deformation between the initial and current reference frames. In continuum mechanics, $\mathcal{F}$ is typically known as the deformation gradient. The equations of the temporal evolution of $\mathcal{F}$ and $\mathcal{T}$ are given by:

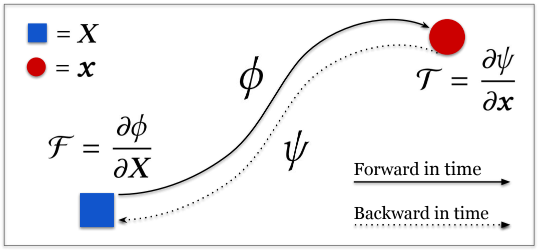

Fig. 2. The forward map ϕ sends $\boldsymbol{X}$ (blue square) to $\boldsymbol{x}$ (red circle) by traveling forward in time; the backward map ψ sends $\boldsymbol{x}$ to $\boldsymbol{X}$ by traveling backward in time. The Jacobians $\mathcal{F}$ and $\mathcal{T}$ are calculated at $\boldsymbol{X}$ and $\boldsymbol{x}$ respectively.

$$\begin{aligned} \frac{D\mathcal{F}}{Dt} &= \nabla \boldsymbol{u} \, \mathcal{F}, \\ \frac{D\mathcal{T}}{Dt} &= -\mathcal{T} \, \nabla \boldsymbol{u}. \end{aligned} \tag{4}$$

Both equations are written in matrix form. We refer readers to the book by Gonzalez and Stuart [2008] for a more detailed derivation.

Flow Maps for Fluid Simulation. Since the four flow map quantities: ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$ can fully prescribe material transport, they can serve as essential ingredients for accurate fluid simulations. To illustrate, we consider the impulse form of the Euler equations for inviscid flow [Cortez 1995; Feng et al. 2022; Nabizadeh et al. 2022]:

$$\begin{cases} \dfrac{D\boldsymbol{m}}{Dt} = -\left(\nabla \boldsymbol{u}\right)^T \, \boldsymbol{m}, \\ \nabla^2 \varphi = \nabla \cdot \boldsymbol{m}, \\ \boldsymbol{u} = \boldsymbol{m} - \nabla \varphi, \end{cases} \tag{5}$$

with $\boldsymbol{m}$ and $\boldsymbol{u}$ being fluid impulse and velocity, and φ an intermediate variable used only for projecting $\boldsymbol{m}$ to the divergence-free $\boldsymbol{u}$. The impulse $\boldsymbol{m}$ is considered to contain more information than $\boldsymbol{u}$, as $\boldsymbol{u}$ can be reconstructed from $\boldsymbol{m}$ by solving the Poisson equation.

A flow map-based solver for Equation 5 is introduced to the graphics community by Nabizadeh et al. [2022], which solves the equation by first computing the backward flow map ψ and its Jacobian $\mathcal{T}$, and then reconstructing the impulse $\boldsymbol{m}$ by evaluating:

$$\boldsymbol{m}(\boldsymbol{x}, t) = \mathcal{T}^T \, \boldsymbol{m}(\psi(\boldsymbol{x}), 0). \tag{6}$$

Intuitively, this equation computes $\boldsymbol{m}$ by 1) mapping the current point $\boldsymbol{x}$ back to its initial location $\boldsymbol{X} = \psi(\boldsymbol{x})$, 2) reading the initial impulse at $\boldsymbol{X}$, and 3) applying the deformation by multiplying with $\mathcal{T}^T$. The mathematical derivation can be found in the work by Cortez [1995] (see Proposition 3). The forward flow map ϕ and its Jacobian $\mathcal{F}$ are also leveraged in their system in a similar fashion. In particular, they evaluate:

$$\bar{\boldsymbol{m}}(\boldsymbol{X}, 0) = \mathcal{F}^T \, \boldsymbol{m}(\phi(\boldsymbol{X}), t), \tag{7}$$

with $\bar{\boldsymbol{m}}(\boldsymbol{X}, 0)$ then being compared to $\boldsymbol{m}(\boldsymbol{X}, 0)$ for BFECC error compensation [Kim et al. 2006]. Their work's successful employment of ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$ in a computer graphics simulation pipeline well-demonstrates the promise of flow map-based methods in building high-fidelity fluid solvers.

3.2 The Perfect Flow Map, and its Numerical Fallibilities

Although the incorporation of $\mathcal{F}$ and $\mathcal{T}$ along with ϕ and ψ has already led to a high level of simulation fidelity, the prevalent method used for solving these flow map variables [Nabizadeh et al. 2022; Qu et al. 2019; Yin et al. 2023] is error-prone and invites reconsideration, as we will elaborate below.

The Perfect Flow Maps. In the absense of numerical inaccuracies, ψ, ϕ, $\mathcal{F}$ and $\mathcal{T}$ should always have the following qualities:

Remark. A point undergoing, in sequence, a backward map ψ and then a forward map ϕ should return to its original position. This also holds for the reverse direction. In other words,

$$\begin{cases} \boldsymbol{X} = \psi \circ \phi(\boldsymbol{X}), \\ \boldsymbol{x} = \phi \circ \psi(\boldsymbol{x}). \end{cases} \tag{8}$$

Remark. A coordinate frame deformed, in sequence, by the backward map Jacobian $\mathcal{F}$ and then by the forward map Jacobian $\mathcal{T}$ should remain identical. This also holds for the reverse direction. In matrix notation,

$$\begin{cases} \boldsymbol{I} = \mathcal{F}\mathcal{T}, \\ \boldsymbol{I} = \mathcal{T}\mathcal{F}. \end{cases} \tag{9}$$

Numerical Fallibilities. Equations 8 and 9 prescribe two consequential properties of flow maps which are challenging to satisfy numerically. Qu et al. [2019] and Nabizadeh et al. [2022] employ two separate numerical paradigms to evolve the backward and forward mappings. On one hand, the backward map ψ is advected using a semi-Lagrangian-based method, which involves a diffusive grid interpolation at each step. On the other hand, the forward map ϕ is accurately solved by marching the map with a high-order Runge-Kutta method, which is void of interpolation errors. The asymmetry between both schemes leads to an *inaccurate* backward mapping and an *accurate* forward mapping, which are bound to diverge (become inconsistent) and fail to satisfy Equations 8 and 9 numerically.

3.3 Alternative: Bidirectional March

Driven by this observation, we propose a simple and natural alternative for solving ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$: **bidirectional marching**. Since the forward and backward flow maps essentially describe the temporal integration of the velocity field forward and backward in time, we can adopt the same, interpolation-free, Runge-Kutta scheme for both the computation of $(\phi, \mathcal{F})$ and $(\psi, \mathcal{T})$, using $+\Delta t$ for the former and $-\Delta t$ for the latter. In particular, we solve for ϕ with Equation 1, $\mathcal{F}$ with Equation 4; and solve for ψ and $\mathcal{T}$ using the exact same procedures but with time reversed.

A Motivational Experiment. As shown in Figure 3, we conduct a concrete numerical experiment for both strategies to vividly illustrate the importance of symmetry in the computation of bidirectional flow maps. We define a steady velocity field with a single vortex in the center, as shown on the left panel. The upper figure plots the streamlines, and the lower figure plots the angular velocity against the radial distance. We use this steady velocity field to construct flow map variables ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$ for both methods, and compare their adherence to Equations 8 and 9.

The two figures on the top-right correspond to the solution adopted by Qu et al. [2019] and Nabizadeh et al. [2022], where a semi-Lagrangian-based scheme is used for solving ψ (we use the dual-mesh characteristics [Cho et al. 2018] for backtracking as suggested by Qu et al. [2019]), and the 4^{th} order Runge-Kutta scheme for solving ϕ. Both $\mathcal{T}$ and $\mathcal{F}$ are computed from ψ and ϕ using finite difference. The two figures on the bottom-right correspond to our proposed approach, where we solve for ϕ and ψ with Equation 1, $\mathcal{F}$ and $\mathcal{T}$ with Equation 4 (top) — all with 4^{th} order Runge-Kutta.

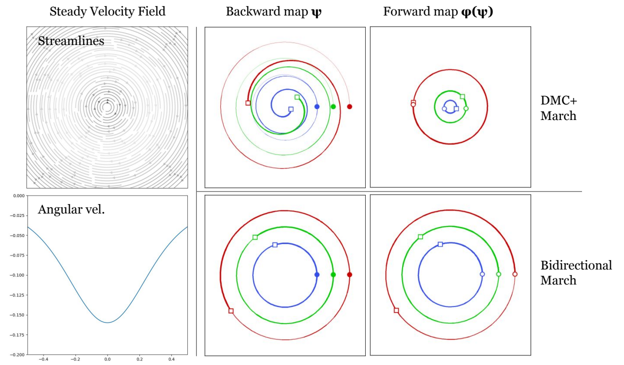

Fig. 3. Under the steady velocity field shown on the left panel, a point transported by its flow map should 1) retain in its original orbit, and 2) return to its original position after the "roundtrip" of $\phi \circ \psi(x)$. This means that 1) the trajectories of the points should always be perfect circles, and 2) the solid circles should coincide with the hollow circles. It can be seen that *Bidirectional March* satisfies these properties nicely, while *DMC+March* yields significant errors.

The middle column depicts the backward mapping ψ, where the (input) current positions $\boldsymbol{x}$ are highlighted with **solid circles**, and the (backtracked) initial positions $\boldsymbol{X} = \psi(\boldsymbol{x})$ are highlighted with **hollow squares**. The upper and lower plots in this column exhibit significantly different behaviors: the lower one traces out streamlines that are perfectly circular, whereas the upper one traces out streamlines that erroneously spiral inward, which are inconsistent with the ground-truth streamlines shown on the top-left.

The right column depicts the forward mapping ϕ originated from the (backtracked) initial positions $\boldsymbol{X} = \psi(\boldsymbol{x})$. These are highlighted with **hollow squares**, and the roundtrip positions $\hat{\boldsymbol{x}} = \phi \circ \psi(\boldsymbol{x})$ are highlighted with **hollow circles**. As prescribed by Equation 8, $\hat{\boldsymbol{x}} = \phi \circ \psi(\boldsymbol{x}) = \boldsymbol{x}$, so the hollow circles ($\hat{\boldsymbol{x}}$) should in theory coincide with the solid circles ($\boldsymbol{x}$). On the upper plot, this is clearly not the case with the presence of the radial drift that has previously occurred in the solving of ψ. On the lower plot, this requirement is well-fulfilled, due to that the computations of the backward and forward flow maps are mechanistically symmetric.

The visual evidence is also supported numerically. The positional error of $\phi \circ \psi(\boldsymbol{x}) - \boldsymbol{x}$ is **0.3581** for *DMC+March*, and **1.904E-06** for *Bidirectional March*, showing that our proposed strategy better satisfies Equation 8 than the existing strategy [Nabizadeh et al. 2022; Qu et al. 2019] by 5 orders of magnitude. We further gauge both strategies' adherence to Equation 9 by calculating the Frobenius norm of $\mathcal{F}\mathcal{T} - \mathbf{I}$. *DMC+March* yields an error of **6.987**, while *Bidirectional March* yields an error of **1.937E-3**, showing that our strategy better satisfies Equation 9 by 3 orders of magnitude.

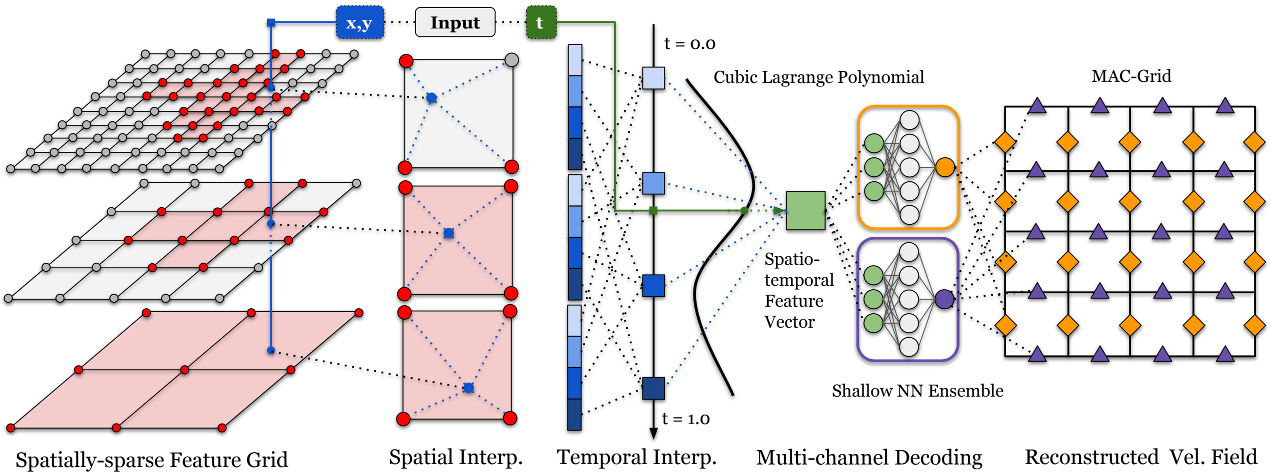

Fig. 4. Illustration of our Spatially Sparse Neural Fields (SSNF) in 2D. In our NFM simulation framework, we use SSNF to represent a continuous spatiotemporal velocity field. To fetch the velocity given coordinates (x, y, t), we first interpolate the multi-resolution, spatially sparse feature grid with (x, y) to obtain one feature vector for each resolution (the left two columns). We reorganize these vectors into 4 temporal anchor vectors, and interpolate them with t to obtain the final feature vector (middle column). Finally, we decode the feature vector with neural networks to get the velocity components (the right two columns).

3.4 Towards Perfect Flow Maps with INR

Current Conundrum. With the current flow map-based methods, the numerical inaccuracies of the flow maps undermine the simulation quality with the added 1) transportation error (*e.g.*, warping the value from the wrong initial location), and 2) interpolation error by the requirement for frequent reinitializations. To fulfill the full potential of low-diffusion, long-range flow map advection, a more accurate and symmetric method for computing the flow maps such as bidirectional marching would be desirable as it is proven to radically increase the numerical accuracy and consistency of the computed ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$. However, the key hindrance to the adoption of the bidirectional marching scheme is the intractable memory requirements for storing the spatiotemporal velocity buffer, which is necessitated for marching the backward variables ψ and $\mathcal{T}$, as the solution from previous steps cannot be reused.

Our Solution. Bridging such a conundrum faced by traditional fluid simulation with the state-of-the-art developments of fast-to-train, accurate, and memory-compact INRs in the machine learning and computer vision community, we design **Neural Flow Maps** (NFM), an impulse and flow map-based fluid simulation method which implements the accurate bidirectional marching scheme for solving ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$, at a small additional memory cost thanks to our novel hybrid INR structure: **Spatially Sparse Neural Fields** (SSNF), which is able to maintain long-range spatiotemporal velocity buffers at a smaller memory footprint than a single velocity field stored on a dense, uniform grid.

In Section 4, we first discuss the design and implementation of SSNF. Then, in Section 5, we introduce our NFM simulation algorithm which incorporates SSNF at its core. Then, in Section 6, we validate the efficacy of both SSNF as a high-performance neural representation, and NFM as a state-of-the-art simulation algorithm.

4 SPATIALLY SPARSE NEURAL FIELDS

In this section, we introduce Spatially Sparse Neural Fields (SSNF), which is the core component of our Neural Flow Maps simulator. As illustrated in Figure 4, SSNF is a novel, general-purpose hybrid INR for representing spatiotemporal signals. Specifically, in our application of fluid simulation, SSNF represents a continuous spatiotemporal velocity field. SSNF consists of 1) a multi-resolution, spatially sparse data structure that maintains trainable feature vectors, and 2) an ensemble of lightweight neural networks that decodes the feature vectors to velocity fields discretized on MAC grids.

Reading Figure 4 from left to right: with input coordinates (x, y, t), we first query the feature grid with spatial coordinates (x, y) to fetch one feature vector at each resolution level. Each feature vector is split into 4 segments which are associated with 4 timestamps. The segments are concatenated across all resolution levels to form 4 temporal anchor vectors, which will be interpolated with t to obtain the final feature vector. After that, we use our ensemble of shallow neural networks to decode this feature vector into the staggered x, y (and z in 3D) velocity components.

4.1 Spatially Sparse Feature Grid

As shown on the left of Figure 4, we store feature vectors on a pyramid of multi-resolution regular grids, whose spatial sparsity is implemented with a bitmask. Our design is inspired by SPGrid [Setaluri et al. 2014], a shallow, sparse structure for physical simulation that is found to be more efficient and convenient than deep, tree-based structures. Unlike SPGrid which maintains a *unique* discretization of the domain, our feature grids are *overlapping*, therefore allowing a point in space to be included by cells on multiple levels and combine information from multiple spatial frequencies. This overlapping, multi-resolution design has been used to good effects by recent methods [Müller et al. 2022; Takikawa et al. 2021].

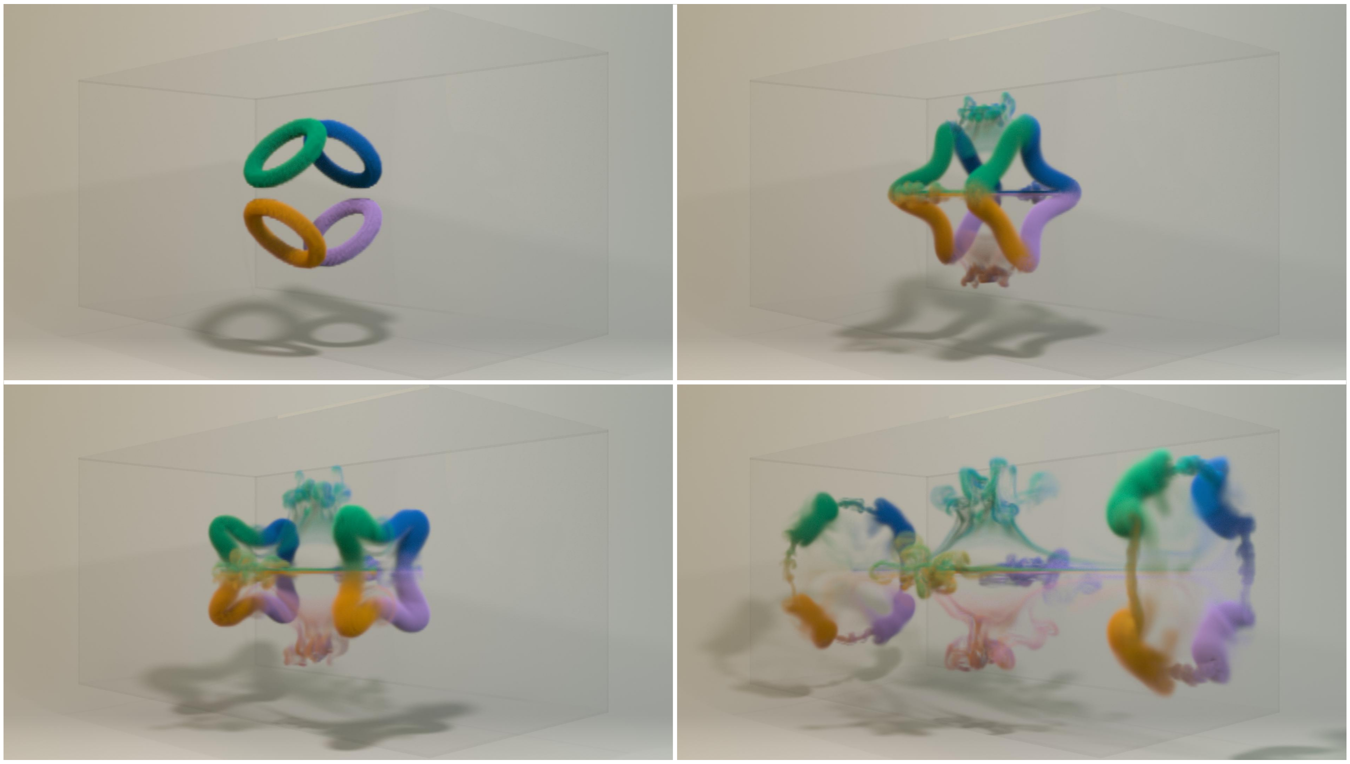

Fig. 5. Collision and reconnection of four vortices each forming right angles with its neighbors. Upon collision, the four vortices reconnect to form two larger vortices shaped like four-pointed stars. They roam towards the left and right walls until each splits into four vortex tubes.

Cell Activation. Our spatial sparsity is explicitly controlled with a strategy similar to the octree division strategy proposed by Ando and Batty [2020]. By default, the coarsest level is densely activated, and a cell on a finer level is activated only when the *complexity* of the flow at the region is found to exceed the characteristic *expressiveness* of that level. The complexity is expressed with the sizing value S computed from the Jacobian of the velocity field, as elaborated in Appendix B; and the characteristic expressiveness is given by $\sigma \cdot \frac{1}{dx}$ with *activation threshold* σ being a scalar hyperparameter. Each cell on each level l computes the max sizing $S_{\max}$ among all the voxels it controls, and if $S_{\max} > \sigma \cdot \frac{1}{dx_l}$, the cell becomes activated. As seen in Figure 4, we activate the cell by activating all 4 (or 8 in 3D) geometric nodes of the cell (the red dots). This process is carried out independently on all levels, and it guarantees that a voxel activated on one level is also activated on all the coarser levels.

Feature Lookup. On each level, we linearly interpolate a feature vector from the 4 (or 8 in 3D) enclosing nodes, and concatenate across all levels. If the interpolated location has no active nodes nearby, a zero vector is returned. If the enclosing nodes are a blend of active and inactive ones, we carry out the interpolation as usual while assuming the inactive ones contain zero vectors. Such an activation-agnostic lookup scheme requires no explicit case switching, and leads to the continuous transition between active and inactive regions, which improves the fitting quality. Quadratic interpolation has also been tested, but shows no significant improvement.

Memory Management. We use the *bitmasked SNode* offered by Taichi [Hu et al. 2019] to compose our data structure. Each bitmasked entry contains a dense block of 4^2 (or 4^3 in 3D) nodes to amortize the sparsity overhead. Each node stores in the AoS manner 1) a feature vector of length 8 (or 16 in 3D), and 2) a serialized index used for grid raveling. The gradients of the feature vectors are stored in a juxtaposed array, so the overall layout can be considered as SoAoS.

Comparison with Existing Methods. Our SSNF is a collision-free analog of Instant NGP [Müller et al. 2022], as both methods employ multiple layers of multi-resolution, *virtual* dense grids, and concatenate the interpolated features. However, Instant NGP saves memory by allowing multiple voxels to share the same parameters as assigned by a hashing function, and compete for dominance through their contributions to the loss function. While this is reasonable for neural rendering applications, where surface cells dominate far-field and inner cells in the loss computation, it fails to disambiguate in our application where each velocity vector contributes equally to the loss. In comparison, we explicitly assign the parameters to the high-frequency regions, giving them exclusive access. As shown in the comprehensive numerical experiments in Subsection 6.1, such explicit control acts to significantly reduce noise and leads to better fitting accuracy at lower parameter numbers and reduced training time. In this sense, SSNF is also similar to NGLOD [Takikawa et al. 2021], a method for fitting SDFs that also uses a collision-free sparse structure. In comparison, our overlapping grid-based design offers

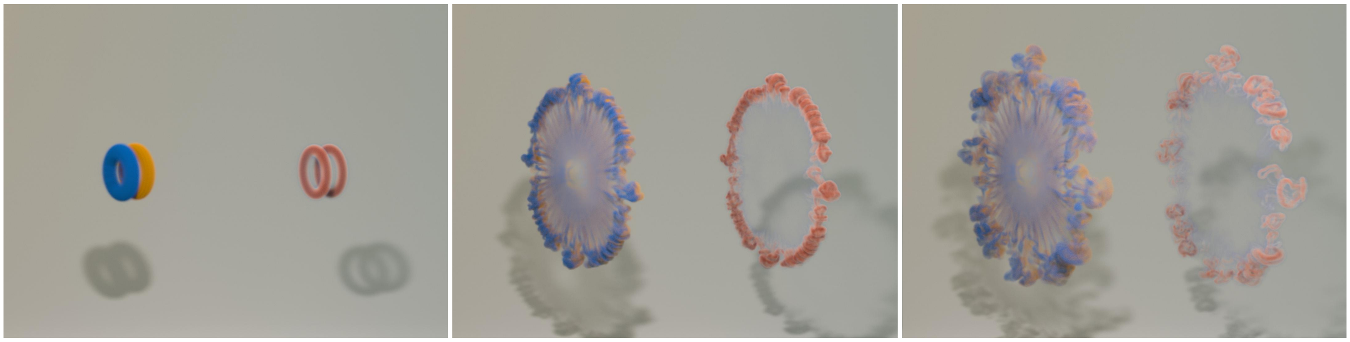

Fig. 6. The evolution of a pair of vortex rings colliding head-on. Upon collision, the two rings stretch in the yz-plane while thinning in the x-direction. The structure then destabilizes and splits into a set of smaller vortex rings reminiscent of a Ferris wheel.

implementational simplicity and the alleviation of the restriction from surface octree to general-purpose, full-domain fields. SSNF is also comparable to NeuralVDB [Kim et al. 2022] as both employ shallow structures rather than deep octrees for sparse representation, and both focus on *compressing* volumetric input data instead of *inferring* from images, although unlike SSNF, NeuralVDB does not currently support temporal information. Both methods' design intents are also fundamentally different: NeuralVDB compresses readily-generated simulation data, whereas SSNF is embedded in the simulation loop to facilitate high-quality simulation.

4.2 Temporal Dimension

Our SSNF handles the temporal dimension by interpolating multiple *anchor* feature vectors in time, similar to the method by Park et al. [2023]. Each feature vector looked up in space is split into 4 segments associated with timestamps 0, $\frac{1}{3}$, $\frac{2}{3}$, and 1 respectively. We use the cubic Lagrange polynomial to interpolate these vectors given any $t \in [0, 1]$. As validated in Subsection 6.1, we find that cubic interpolation (instead of the linear interpolation found in existing methods) is consequential to the fitting accuracy.

Dynamic Timestamp Normalization. The interpolation scheme requires that the time of the stored sequence is within $[0, 1]$. For typical dynamic-NeRF applications, this is trivially fulfilled by normalizing the *actual* timestamps by the total timelapse. In our case, this is challenging as the sequence is a stream of frames, each with a variable duration that is not known in advance. In response, we devise *dynamic timestamp normalization* to handle sequences with an indefinite number of frames or total timelapse. Essentially, we always assume that the current frames occupy the range $[0, 1]$, and normalize the timestamps with the *time multiplier* λ, which is the reciprocal of the timelapse-so-far. As more of the sequence arrives, λ decreases and the earlier frames are scaled down to accommodate the later frames. Not only does this lift the requirement for a predetermined timelapse, in Subsection 6.1, we show that our strategy is more effective even when the total timelapse is available.

4.3 Staggered Feature Decoder

After the successful interpolation of feature vectors with spatiotemporal coordinates, the SSNF will then reconstruct velocity fields on MAC grids using neural decoders. To obtain the staggered velocity components, we create one shallow neural network per spatial dimension. Based on the shared feature grid, for each spatial dimension, feature vectors are queried at its associated face centers and decoded with its proprietary neural network. Such a multi-branch design cannot be replaced with a single decoder that recovers the x, y (and z in 3D) components *en masse*, because treating the staggered vector field as a scalar field leads to non-smooth signals.

4.4 Training Scheme

As an integral part of the Neural Flow Maps (NFM) simulation loop, our SSNF is trained at each simulation step to incorporate the latest update of the spatiotemporal velocity field produced by the simulation (elaborated in Section 5). During this training process, the input frame (the most recent velocity) must be properly stored *without losing track of the previously stored frames*. However, the ground truth data for the prior frames are already lost. To address this, we maintain two copies of SSNF: $\mathcal{N}$ and $\hat{\mathcal{N}}$, where the former is our main velocity buffer and the latter is an auxiliary buffer for memorizing the past. To better illustrate, suppose $\mathcal{N}$ has already stored the sequence $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$. When $\boldsymbol{u}_4$ arrives, we begin by hard-copying $\mathcal{N}$ to $\hat{\mathcal{N}}$. To update $\mathcal{N}$ for $\boldsymbol{u}_4$, we assemble the training data *as if* we have $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{u}_4$ in hand and sample all frames with equal priority — but in practice, all the samples from $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ will be queried from $\hat{\mathcal{N}}$. We note that training recurrently in this way will indeed accumulate errors. But as shown in Figure 11 (left 4 plots), this is not significant in practice as the errors exhibit only a minor rising trend as the number of training sessions rises from 1 to 25.

Data Sampling. We train our SSNF using mini-batches sampled from a discrete probability distribution q computed as:

$$q_I = \alpha \cdot \min(\max\left(\sigma \cdot \frac{1}{dx_{\mathrm{C}}}, S_I\right), \sigma \cdot \frac{1}{dx_{\mathrm{F}}}), \tag{10}$$

with I denoting the 2 or 3D grid index, dx_C and dx_F the spacings of the coarsest and finest levels, and α the normalization factor to ensure that q sums to 1. Such a distribution will bias the training data towards regions with greater flow complexity. The clipping functions are invoked to handle the edge cases (*e.g.*, without them, a constant field with $S = 0$ everywhere will leave α undefined). More details regarding the sizing value S can be found in Appendix B.

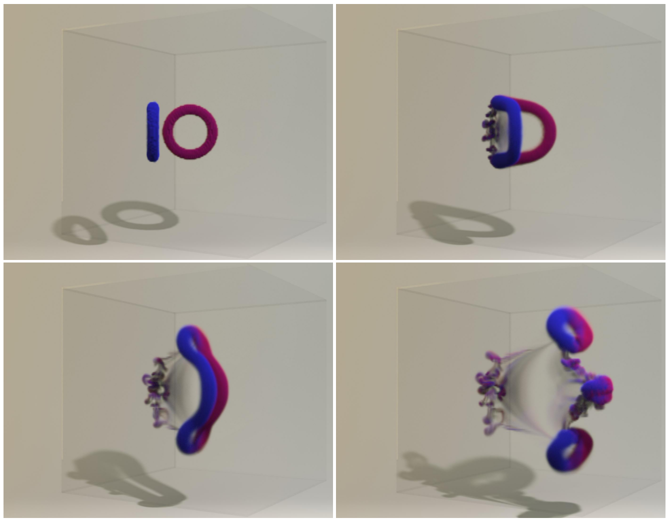

Fig. 7. The evolution of a pair of oblique vortex rings. The two vortices attach on the left side, undergo multiple topological changes, and eventually morph into three vortex rings.

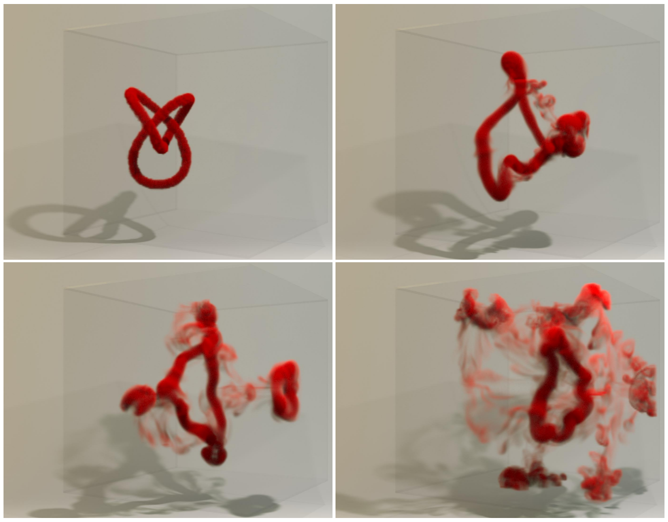

Fig. 8. The temporal evolution of a vortex trefoil knot. The knot deforms, self-collides, and breaks into two vortices of different sizes, corresponding to the experimental observation [Kleckner and Irvine 2013].

Encoder Growth. As discussed in Subsection 4.1, the encoder is activated only for regions where the flow is deemed complex enough. However, a region that is simple at the beginning of a sequence might become complex as the simulation proceeds. Hence, we *grow* the encoder upon receiving a new frame by recomputing the sizing value S and activating any additional cells as needed.

Reinitialization. As will be elaborated in Section 5, our simulation reinitializes every n steps where n is a hyperparameter whose selection should consider both SSNF's fitting capacity and NFM's accumulated errors from long-range flow map marching (further analysis can be found in Subsection 6.2). At reinitialization, we uniformly randomly reinitialize the feature grid following Müller et al. [2022], and leave the decoder networks untouched. Additionally, we also reinitialize the feature grid when the ratio of time multipliers $\frac{\lambda_{t-1}}{\lambda_t} > 1.33$ during dynamic timestamp normalization, because when the timestamp assignment for *the same* frame changes significantly between two training sessions, the previously learned anchor vectors are often sub-optimal for initializing the current session.

Details. Each decoder is a shallow MLP with one hidden layer of width 64. We use the ELU activation to avoid the dead ReLU issue [Clevert et al. 2015]. We use AdamW [Loshchilov and Hutter 2017] with $\beta = (0.9, 0.99)$ as the optimizer. The learning rate is set to 0.01, and is scheduled to exponentially decay to 0.001 at iteration 1500. We employ the Mean Squared Error (MSE) as the loss function.

5 SIMULATION ON NEURAL FLOW MAPS

The Neural Flow Maps (NFM) simulation algorithm built upon our SSNF velocity buffer is outlined in Algorithm 1.

At the start of a simulation, the SSNF buffer will be *randomly* initialized — it serves purely as a data structure and hence does not need any pre-training. Each simulation step starts by checking if the current step count is a multiple of n. If so, we reinitialize by setting the forward flow map ϕ and its Jacobian $\mathcal{F}$ to the identity mapping $\mathbf{id}_\Omega$ and identity matrix $\boldsymbol{I}$ respectively, and the initial velocity $\boldsymbol{u}_0$ to the current velocity $\boldsymbol{u}$. After that, we compute the current timestep Δt_j (j represents the step count since the last reinitialization), with $\boldsymbol{u}$ and the CFL condition. We maintain an array $[\Delta t_0, \ldots, \Delta t_{n-1}]$ to store the history of Δt. Then, we compute the midpoint velocity $\boldsymbol{u}_{\text{mid}}$ using Algorithm 3, and store it with the SSNF $\mathcal{N}$. Doing so involves the encoder growth procedure and the utilization of the auxiliary buffer $\hat{\mathcal{N}}$, which have been discussed in Subsection 4.4.

Once $\boldsymbol{u}_{\text{mid}}$ is properly stored in the SSNF buffer $\mathcal{N}$, we compute ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$ using $\mathcal{N}$. To do so, we reset ψ and its Jacobian $\mathcal{T}$ to the identity map and identity matrix, and march with the spatiotemporal velocity field stored in $\mathcal{N}$ backward in time for a total of j steps. For ϕ and $\mathcal{F}$, we reuse their solutions from the previous step and march for a single step forward in time. The marching algorithm is further elaborated in Subsection 5.1 and Algorithm 2.

Once ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$ are obtained, we compute the advected velocity $\hat{\boldsymbol{u}}$ using the impulse-based fluid advection with BFECC. The detailed scheme is given in Algorithm 4. If any external forces are present, their integrals along the fluid streamlines need to be computed and added to the velocity before projecting again by solving the Poisson equation.

5.1 Bidirectional Flow Map Marching

As described in Subsection 3.3, we solve for ϕ and ψ by marching Equation 1 forward and backward in time; and solve for $\mathcal{F}$ and $\mathcal{T}$ by marching Equation 4 (top) forward and backward in time.

All four quantities are temporally integrated with our custom 4th order Runge-Kutta (RK4) integration scheme, outlined in Algorithm 2. The algorithm uses ϕ and $\mathcal{F}$ for notation, but applies directly to ψ and $\mathcal{T}$ by reversing time. We note that, to evolve $\mathcal{F}$ and $\mathcal{T}$ using the RK4 scheme, the values $\frac{\partial \mathcal{F}}{\partial t}$ and $\frac{\partial \mathcal{T}}{\partial t}$ need to be

Algorithm 1 Neural Flow Map Simulation

```
Initialize: $\boldsymbol{u}$ to initial velocity; $\mathcal{N}$, $\hat{\mathcal{N}}$ with random weights
for $i$ in total steps do
    $j \leftarrow i \pmod{n}$;
    if $j = 0$ then
        $\phi \leftarrow \mathbf{id}_\Omega, \mathcal{F} \leftarrow \boldsymbol{I}$;
        $\boldsymbol{u}_0 \leftarrow \boldsymbol{u}$;
        Randomly initialize $\mathcal{N}$'s feature vectors;
    end if
    Compute $\Delta t_j$ with $\boldsymbol{u}$ and the CFL number;
    Estimate midpoint velocity $\boldsymbol{u}_{\text{mid}}$ according to Alg. 3;
    if $i + 1 \pmod{n} \neq 0$ then
        Compute sizing field $S$ with $\boldsymbol{u}_{\text{mid}}$;
        Grow $\mathcal{N}$'s feature grid with $S$;
        Train $\mathcal{N}$ with $\boldsymbol{u}_{\text{mid}}$ and $\hat{\mathcal{N}}$;
        $\hat{\mathcal{N}} \leftarrow \mathcal{N}$;
    end if
    $\psi \leftarrow \mathbf{id}_\Omega, \mathcal{T} \leftarrow \boldsymbol{I}$;
    March $\psi, \mathcal{T}$ with Alg. 2, using $\boldsymbol{u}_{\text{mid}}$ and $-\Delta t_j$;
    for $l$ in $j - 1 \ldots 0$ do
        Query $\boldsymbol{u}_{\text{mid},l}$ from $\mathcal{N}$ using time $\sum_{c=0}^{l-1} \Delta t_c + 0.5\Delta t_l$;
        March $\psi, \mathcal{T}$ with Alg. 2, using $\boldsymbol{u}_{\text{mid},l}$ and $-\Delta t_l$;
    end for
    March $\phi, \mathcal{F}$ according to Alg. 2, using $\boldsymbol{u}_{\text{mid}}$ and $\Delta t_j$;
    Reconstruct $\boldsymbol{u}$ with $(\boldsymbol{u}_0, \psi, \mathcal{T}, \phi, \mathcal{F})$ as in Alg. 4;
    if use external force then
        $\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u} + \int_0^t f_{\text{ext}}(\psi(\boldsymbol{x}, \tau), \tau) d\tau$;
        $\boldsymbol{u} \leftarrow \mathbf{Poisson}(\hat{\boldsymbol{u}})$;
    end if
end for
```

estimated at $t + 0.5\Delta t$ and $t + \Delta t$, which require the estimations of ϕ and ψ at these times. We manage this by evolving the maps and the Jacobians in an *interleaved* manner, so that the estimations of ϕ and ψ at $t + 0.5\Delta t$ and $t + \Delta t$ can be recycled for updating $\mathcal{F}$ and $\mathcal{T}$.

5.2 Midpoint Method

As outlined in Algorithm 3, we adopt the second-order, midpoint method to effectively reduce the truncation error of the temporal integration. To do so, we reset ψ and $\mathcal{T}$ to identity, perform a single RK4 backtrack with the current velocity $\boldsymbol{u}$ and $0.5 \cdot \Delta t$, and carry out the impulse-based advection to obtain $\boldsymbol{u}_{\text{mid}}$. We empirically find that, for estimating $\boldsymbol{u}_{\text{mid}}$, neither the BFECC error compensation nor the long-range flow map makes a significant difference to the simulation results, hence they are ablated for the estimation of $\boldsymbol{u}_{\text{mid}}$.

5.3 Error-compensated Impulse Advection

As outlined in Algorithm 4, once we have successfully solved for the quantities ϕ, ψ, $\mathcal{F}$ and $\mathcal{T}$, we transport the initial velocity $\boldsymbol{u}_0$ to find the updated velocity $\boldsymbol{u}$ using the impulse-based scheme [Cortez 1995; Nabizadeh et al. 2022], which is used in combination with the BFECC error compensation [Kim et al. 2006; Qu et al. 2019].

Additional implementation details of our NFM simulation method can be found in Appendix B.

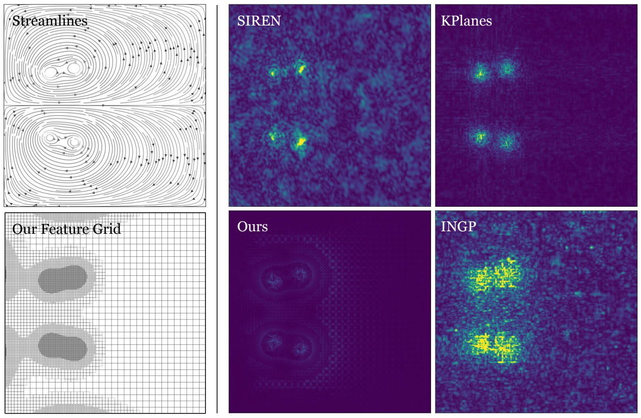

Fig. 9. Fitting error in 2D. The benchmarks are noisy around the vortical regions; our method achieves high fitting accuracy in these regions at a smaller memory footprint due to its explicitly managed spatial sparsity.

6 VALIDATION

We validate the effectiveness of NFM in two parts. First, we conduct a thorough comparison of SSNF with existing INR benchmarks in terms of memory efficiency, training speed, and fitting accuracy. Then, we validate the advantage of employing SSNF for simulation, by comparing NFM quantitatively and qualitatively with traditional simulation methods, showcasing improved adherence to analytical solutions, energy preservation, and phenomenological fidelity.

6.1 Validation of SSNF

We compare SSNF to three benchmarks: Instant NGP (INGP) [Müller et al. 2022], KPlanes [Fridovich-Keil et al. 2023], and SIREN [Sitzmann et al. 2020], in fitting spatiotemporal fields in both 2D and 3D (spatial-wise). These benchmarks are carefully selected to cover a wide spectrum of INR paradigms. INGP and KPlanes are both hybrid INRs combining feature vectors stored on traditional data structures with small neural networks, a design adopted by our method. Between them, INGP achieves data compression by leveraging spatial sparsity, as does our approach, whereas KPlanes is spatially dense and achieves compression by spatial decomposition. SIREN, on the other hand, is a pure INR that is fully implicit featuring much larger neural networks. Detailed descriptions of each benchmark along with our testing methodology can be found in Appendix C.

The testing results are presented in Table 2 and Figure 11. On the left of Table 2, the computational cost of the 2D and 3D versions of all 4 methods are given. On the right, the averaged errors are given. Since the errors are natively computed on the MAC grid, we report two errors: a scalar (component-wise) error with the Root Mean Squared Error (RMSE), and a vector error with the Average End Point Error (AEPE). The velocity vectors are reconstructed by aggregating the face-centered velocity components at the cell centers. For each metric, we present three numbers: the initial error, the final error, and the averaged error. To elucidate the difference between these errors, we remind that each test consists of 25 *training sessions*. The initial session learns a single, static frame, while the final session learns the entire sequence of 25 frames. Hence the initial error

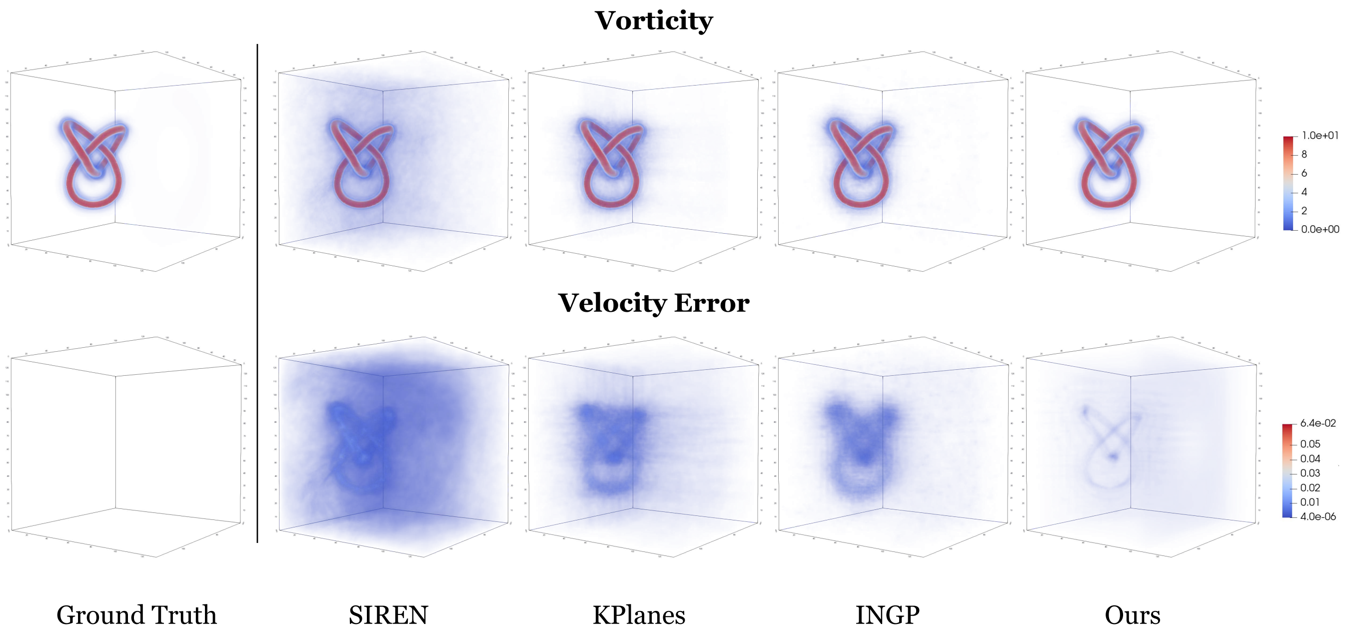

Fig. 10. Visualization of the fitting quality of a spatiotemporal sequence in 3D. Displayed is the 8th frame from the 24th training session. The top row shows the vorticity field. It can be seen that our method yields the highest-quality compression with the lowest amount of smearing around the vortex tubes. The bottom row shows the velocity error compared to the ground truth, which confirms the improved accuracy of our method, especially near the vortical regions.

isolates the system's capacity to resolve spatial details, while the final error gauges its capacity to handle the long-range temporal evolution. The average error measures the system's consistency throughout the 25 training sessions.

The left of Table 2 shows that, in 2D, SSNF is the most memory efficient and only trails INGP in terms of time cost by less than 10%. In 3D, SSNF is the most time efficient and only trails SIREN in terms of memory cost, albeit for a favorable tradeoff since SIREN trains about 6× slower. The right of Table 2 shows that SSNF has a clear advantage in terms of fitting accuracy. For instance, in 2D, our method reduces the average RMSE by 73.7% from the best benchmark (KPlanes) and 91.1% from the worst benchmark (INGP). In 3D, our method reduces the error by 73.5% from the best benchmark (INGP) and 87.3% from the worst benchmark (SIREN). Similarly, in Figure 11, it can be observed that our method consistently outperforms the benchmarks in all metrics.

Discussion. The reason behind our method's effectiveness can be probed with Figure 9, in which the ground truth velocity is plotted on the top left, and the fitting errors of all four methods are plotted on the right. The feature grid discretization of our method is illustrated on the lower left. We can see that for all the benchmarks, the error is concentrated in the vortical regions with large velocity gradients. These methods either lack the ability to adapt the DoFs towards these regions (KPlanes), or rely on black-box schemes (INGP, SIREN) which underperform for our application. In comparison, our method explicitly assigns its DoFs to the regions where they are the most needed, according to the domain-specific prior knowledge encoded in the sizing function (Equation 12 of Appendix B). Despite SSNF using the fewest parameters, the lower left of Figure 9 shows that most of the parameters are clustered to densely cover the vortical regions, which leads to a much greater local effective resolution and therefore much smaller fitting errors.

The analogous 3D error visualization is given in Figure 10, in which the vorticity fields computed from the fitted velocity fields are depicted on the top row, and the fitting errors are visualized on the bottom row. It can be observed that SSNF yields the highest-fidelity fitting result by a considerable margin, sharply carving out high-frequency details near the trefoil knot, while the benchmarks are visibly diffused around the knot, a sign of insufficient storage resolution. Our method achieves this by explicitly assigning most of its parameters to express the knot structure, so that the effective resolution is largely increased at a manageable cost.

In summary, experimental evidence suggests that our SSNF representation is generally advantageous for cases where:

1. The accuracy of compression is important, *e.g.*, in scientific computing;
2. The memory budget is constrained and needs to be efficiently managed;
3. The spatial sparsity is significant and domain-specific knowledge is available.

Therefore, suitable scenarios for SSNF *include* high-fidelity fluid simulations for computer graphics and computational fluid dynamics, but they are also not confined to this application.

Ablation Studies. As shown in Table 3 and Figure 12, we conduct 2D and 3D comparisons with two ablated versions: [A] without dynamic timestamp normalization (instead, normalize full timelapse to [0, 1]), and [B] with linear time interpolation instead of the cubic

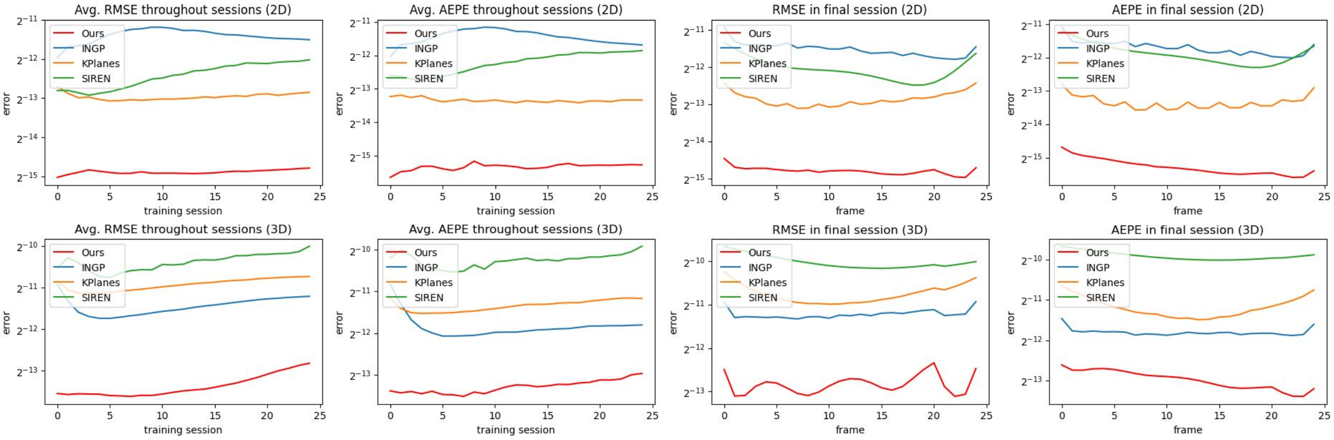

Fig. 11. The fitting errors (RMSE and AEPE) for experiments in 2D (top) and 3D (bottom). Our method consistently yields the lowest fitting error in comparison to existing methods.

Computational Cost			**Performance**					
	Num. Params.	Time (s)	RMSE (init.)	AEPE (init.)	RMSE (avg.)	AEPE (avg.)	RMSE (final)	AEPE (final)
			XY + T					
INGP	90,562	**0.49**	24.91E-5	23.66E-5	36.89E-5	35.82E-5	34.33E-5	30.25E-5
KPlanes	105,262	0.64	15.00E-5	10.42E-5	12.55E-5	9.659E-5	13.51E-5	9.679E-5
SIREN	98,822	1.19	13.95E-5	16.11E-5	18.53E-5	21.16E-5	24.02E-5	26.89E-5
Ours	**83,138 - 87,522**	0.53	**2.999E-5**	**1.960E-5**	**3.299E-5**	**2.439E-5**	**3.533E-5**	**2.536E-5**
			XYZ + T					
INGP	2,148,147	4.26	51.36E-5	54.98E-5	36.12E-5	27.59E-5	42.13E-5	27.97E-5
KPlanes	2,335,107	2.99	54.43E-5	43.75E-5	51.48E-5	38.66E-5	58.60E-5	43.42E-5
SIREN	**462,595**	11.60	67.10E-5	84.89E-5	75.06E-5	82.54E-5	96.94E-5	103.6E-5
Ours	1,969,779 - 2,129,891	**1.94**	**8.356E-5**	**9.323E-5**	**9.554E-5**	**10.01E-5**	**13.84E-5**	**12.46E-5**

Table 2. Errors of our method compared to those of the three benchmarks in spatiotemporal signal fitting in 2D and 3D. The time reported reflects the training time per 100 training iterations. Our method consistently yields the lowest fitting error at a highly competitive memory and time cost.

one. To facilitate [A], we create test sequences with a fixed per-step duration, so that the total timelapse is known ahead of time. It can be seen that both ablated versions are less effective than the full version, which validates that our proposed techniques are well-motivated.

Influence of Encoder's Size. As shown in Figure 16 (right), we analyze the influence of the activation threshold σ and the encoder's effective resolution, which together control the encoder's number of parameters. We perform 2 experiment sets: 1) fixing $\sigma = 0.01$ with varying *base* resolutions $(8, 32)$, $(16, 64)$, $(32, 128)$, and $(64, 256)$ (with 4 levels of refinement), and 2) fixing base resolution $(32, 128)$ with varying σ = 0.01, 0.02, 0.03, 0.04, and 0.05. For 1), a higher resolution leads to more parameters, and for 2), a lower σ leads to more parameters. For both tests, we observe that the accuracy quickly saturates once the encoder reaches a certain size, and a larger model can even worsen the fitting accuracy, due to our limited training iterations.

6.2 Validation of NFM Simulation

In this section, we validate the efficacy of our NFM neural simulation method in comparison with existing benchmarks. We begin with a simple, 2D steady flow to compare our method to the benchmarks in retaining the steady state. We move on to complex 3D scenarios without analytical solutions and compare in terms of the conservation of energy. Finally, we qualitatively compare our method to the benchmarks in terms of visual intricacy and the recreation of real-world phenomena.

2D Analysis: Steady Point Vortex. An isolated point vortex in the absence of viscosity or external forces is a steady state where the velocity field remains constant over time. This is nevertheless challenging to satisfy for simulation methods due to numerical dissipation. Here, we compare NFM to BiMocq [Qu et al. 2019], CF+BiMocq [Nabizadeh et al. 2022], and MC+R [Zehnder et al. 2018], three of the advanced simulation methods recently proposed, in terms of their

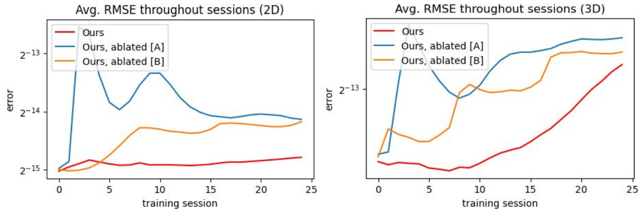

Fig. 12. Errors of the ablation tests corresponding to Table 3.

	2D RMSE (avg.)	2D RMSE (final)	3D RMSE (avg.)	3D RMSE (final)
Ours	**3.299E-5**	**3.533E-5**	**9.554E-5**	**13.84E-5**
Ablated [A]	7.258E-5	5.562E-5	13.94E-5	15.91E-5
Ablated [B]	4.555E-5	5.408E-5	12.12E-5	14.78E-5

Table 3. Time-averaged ablation results. The reduced errors of our full method validate that our design choices are well-motivated.

adherence to the steady state after long simulations. The mean absolute error between the simulated velocity and the steady velocity over 1300 frames is plotted in Figure 17 (right), and the final error image is depicted in Figure 13 (right). It can be seen that CF+BiMocq and MC+R both yield much reduced errors compared to BiMocq, with MC+R being slightly better than CF+BiMocq. Our method outperforms these two methods by an order of magnitude.

It is particularly interesting to note that, with the three methods all built upon the premise of reduced interpolation errors with long-range mapping, BiMocq and CF+BiMocq yield less accurate results than the single-step MC+R method, while NFM yields a much more accurate one than MC+R. This corroborates the argument that the potential of flow map-based advection can only be realized when the flow maps are simultaneously 1) long-range, and 2) accurate, which reiterates the necessity of our bidirectional marching scheme.

3D Analysis: Vortex Rings. As shown in Figure 15, we set up the leapfrogging vortex rings experiment and compare NFM to 4 benchmarks: BFECC [Kim et al. 2006], MC+R [Narain et al. 2019], CF and CF+BiMocq [Nabizadeh et al. 2022; Qu et al. 2019]. The time-varying kinetic energy for all five methods are plotted in Figure 17 (left), and it can be observed that our method conserved energy better than its counterparts. It is known that in the conservative case, the parallel vortex rings will remain separated and leap around each other indefinitely. As shown in Figure 15, our improved energy conservation indeed translates to visibly better fulfillment of this phenomenon, as NFM yields vortex rings that remain separate after the fifth leap, while those from the other methods merge after at most three leaps. As shown in Figure 13 (left), we also conduct an analogous experiment in 2D and achieve similar results.

Our method's reduction of numerical dissipation noticeably enhances the visual beauty of turbulent fluid simulation. For example, in Figure 14, we compare the vortex ring reconnection simulated by NFM to that by the benchmarks. It can be observed that, the most diffusive method: BFECC creates damped, viscous bridges between the rings that keep them from reconnecting. The modern solvers (CF, MC+R and CF+BiMocq) lessen the numerical diffusion to effectively thin these viscous bridges. NFM offers another step up with its clean reconnection without the viscous bridges, which is more in line with the reference simulation [Matsuzawa et al. 2022].

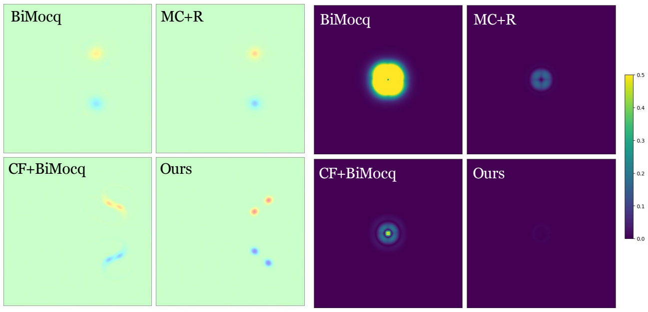

Fig. 13. Left: leapfrogging vortices in 2D. Our method best preserves the vortical structures. Right: absolute errors *w.r.t.* the steady-state field after 1300 frames. The mean absolute errors are 1.589E-4, 21.56E-4, 30.92E-4, and 488.9E-4 for NFM (ours), MC+R, CF+BiMocq, and BiMocq respectively.

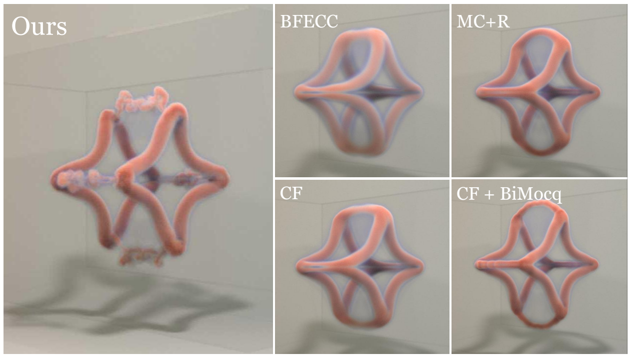

Fig. 14. The vortex reconnecting instant of the "four vortices" example. Our method does not suffer from the numerical diffusion which manifests in the *viscous bridges* between the two reconnected vortices.

Influence of Reinitialization Steps. As shown in Figure 16 (left), we analyze the influence of the reinitialization step n on the simulation accuracy by performing the aforementioned 2D steady-state test with $n = 1$, 5, 9, 11, 13, 15, 17, 21, 25, 29, and 34. It can be seen that the error exhibits a u-shaped trend, and minimizes at $n = 17$. A small choice of n can suffer from the frequent interpolations, while a large choice tends to suffer from the errors in long-range flow map marching. Currently, n is empirically selected for each simulation.

7 EXAMPLES

With our high-performance NFM simulator, we tackle a range of complex simulation scenarios, including vortex ring reconnections, vortex shedding from moving obstacles, and vortex development from fluid density difference. A catalog of our examples can be found in Table 5. In our examples, we assume the shorter edge to have the unit length, and the setups are reported accordingly. We use a workstation with AMD Ryzen Threadripper 5990X and NVIDIA RTX A6000 to compute our examples.

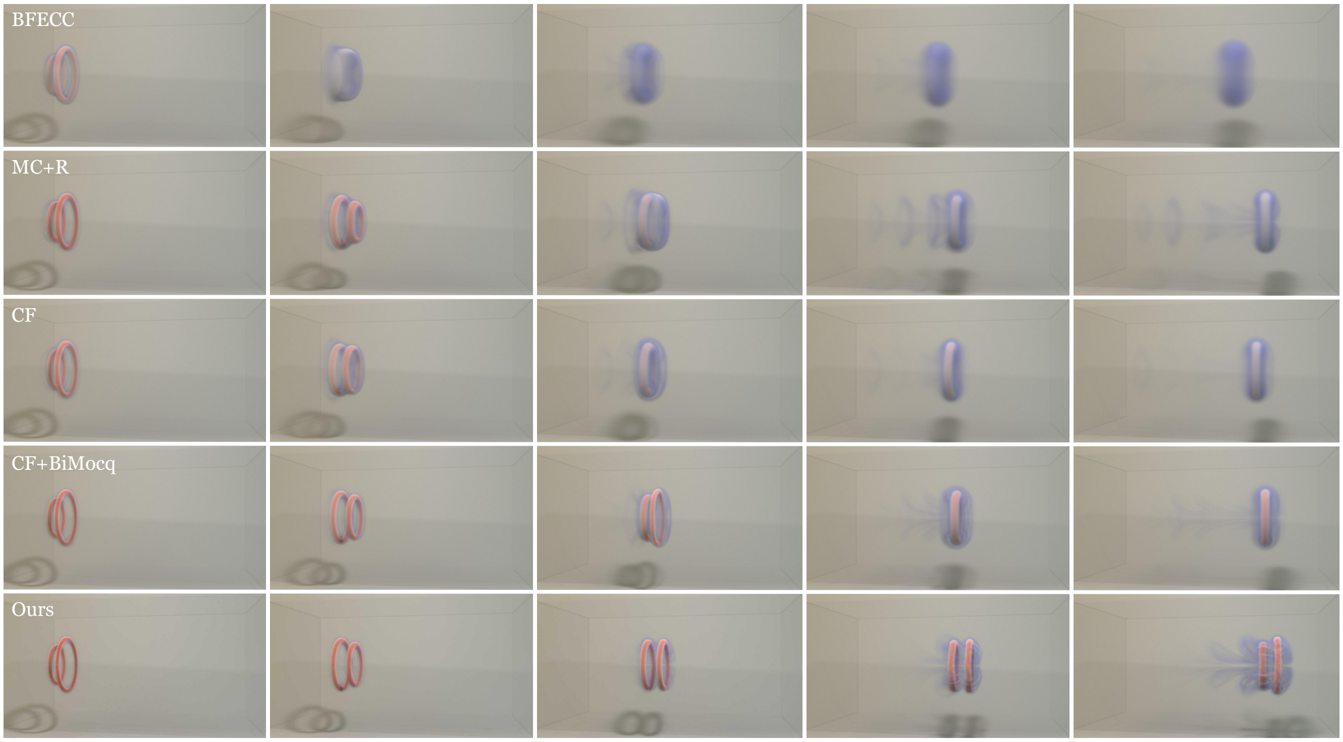

Fig. 15. Comparison of 3D leapfrogging vortices to the benchmarks. Our method simulates vortex rings that remain separate even after the fifth leap, while those simulated by the benchmarks merge after at most the third leap, showing improved correspondence to the expected phenomenon.

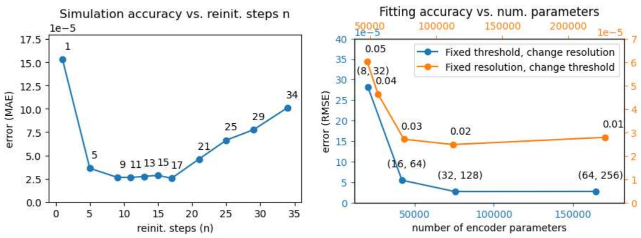

Fig. 16. Left: simulation errors from different choices of n. Right: fitting errors for different encoder sizes.

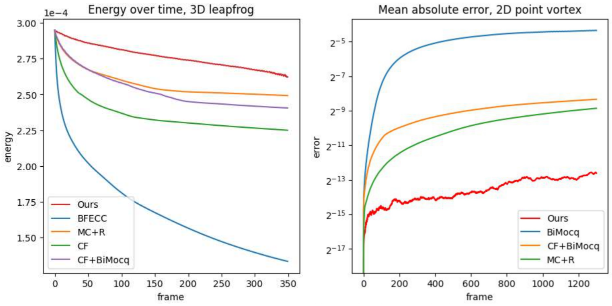

Fig. 17. Left: the 3D kinetic energy plotted over time, which showcases the improved energy conservation property of NFM. Right: mean absolute error of NFM over time compared to the benchmarks in a steady 2D flow.

Leapfrogging Vortices (2D). As shown in Figure 13, we conduct the classic 2D leapfrog experiment by placing four point vortices centered at $x = 0.25$ and $y = 0.26$, 0.38, 0.62, and 0.74. The vortices have the same strength (magnitude) of 0.005, with the upper two being negative and the lower two being positive. The velocity fields are obtained from the point vortices using a mollified Biot-Savart kernel with support 0.02. The leapfrogging vortices will hit the right wall and separate into two vortex *pairs* that will return to the left. Once these two pairs hit the left wall, they reassemble into the initial configuration and repeat the process. Our method is able to repeat this cycle 3 times with the vortices still separated.

Leapfrogging Vortices (3D). As shown in Figure 19, we initialize two parallel vortex rings with $x = 0.16$ and 0.29125. The major radius is 0.21; the minor radius (the mollification support of the vortices) is 0.0168. Our method remains separate after the 5th leap, while existing benchmarks diffuse and merge after at most the 3rd one.

Oblique Vortex Collision. As shown in Figure 7, we initialize two vortex rings initially facing each other at the right angle. The center of the vortex rings are offset by 0.3 along the x-axis. The major radius of both is 0.13; the minor radius is 0.02. Upon collision, the vortices attach on the left side to form a single vortex ring, which gets catapulted to the right and divides into three smaller vortices.

Headon Vortex Collision. As shown in Figure 6, we initialize two opposing vortex rings that are offset by 0.3 along the x-axis. The major radius is 0.065; the minor radius is 0.016. Upon collision, the

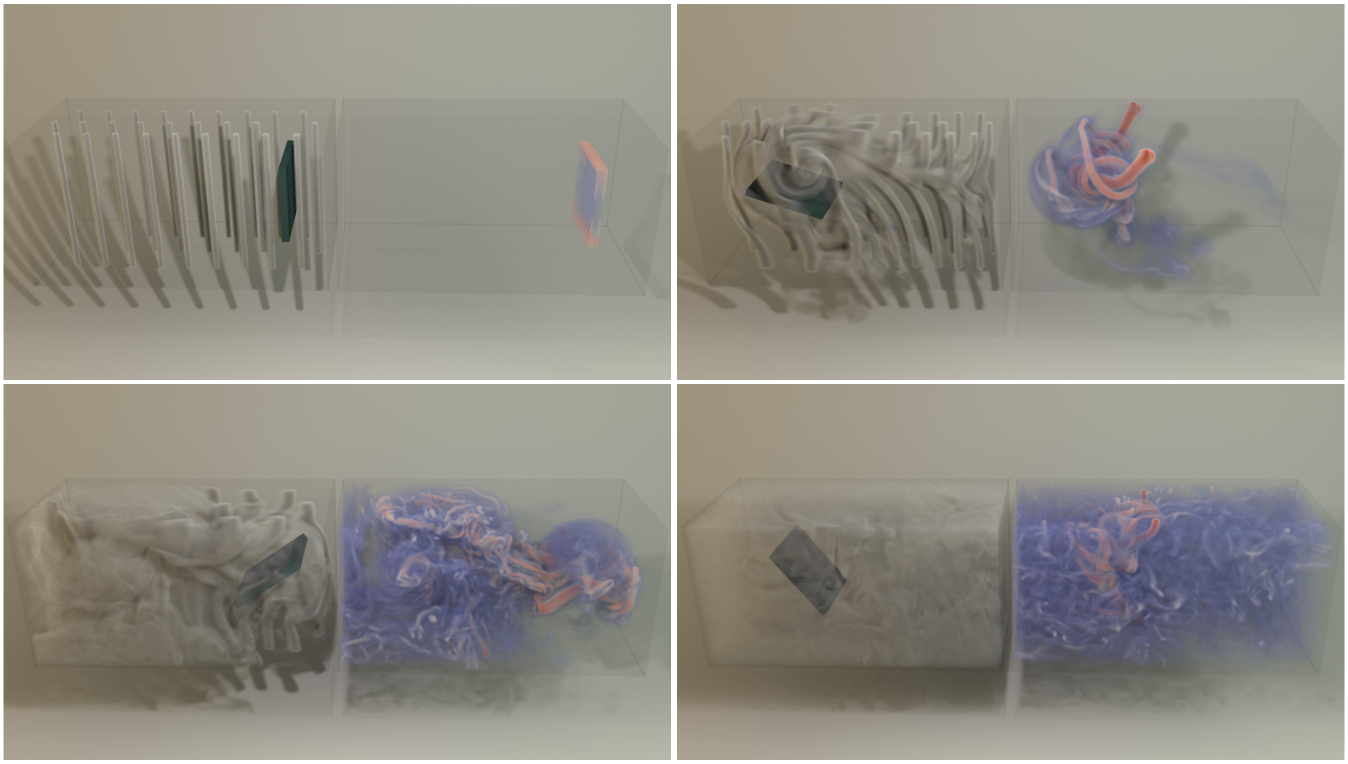

Fig. 18. Turbulent flow induced by a rotating and translating paddle. The smoke is juxtaposed with the vorticity field that shows intricate vortex filaments.

two vortices stretch rapidly along the yz-plane while thinning along the x-axis, causing the structure to destabilize and break into a ring of small, secondary vortices facing radially outward, which resembles the experimental results by Lim and Nickels [1992].

Trefoil Knot. As shown in Figure 8, we reproduce with NFM the classic trefoil knot setup by Kleckner and Irvine [2013]. We simulate with the initialization file open-sourced by Nabizadeh et al. [2022], and observe that the knot structure correctly breaks into one larger vortex and one smaller vortex.

Four Vortices Collision. As shown in Figure 5, we initialize four colliding vortices following Matsuzawa et al. [2022] by placing four vortex rings that each form right angles with its neighbors, essentially outlining a square in the yz-plane. The major radius is 0.13; the minor radius is 0.02. The collision causes the four vortex rings to reconnect into two vortices shaped like four-pointed stars, leaving behind intricate, turbulent patterns. Both vortex rings then "crawl" towards the left and right walls, morphing their shapes. After hitting the walls, the two vortices separate into four vortex tubes.

Moving Paddle. As shown in Figure 18, we initialize arrays of smoke columns with zero initial velocity. A kinematic boundary rotates around the z-axis while translating along the x-axis. The translated position linearly interpolates between $x = 0.5$ and $x = 1.7$, with a time-dependent fraction $0.5 \cdot (1 + \cos(0.5 \cdot t))$. The rotation angle relates to time as $0.75 \cdot t$. The paddle is a cube with a width and height of 0.54 and thickness of 0.05. The velocity difference between the paddle and the surrounding fluid creates vortex sheets that roll up into intricate vortex filaments. The vortical velocity field disintegrates the initial smoke columns into a turbulent mixture.

Inkdrop. As shown in Figure 1, we initialize a vortex ring with a major radius of 0.06 and minor radius of 0.016, centered at $y = 1.85$ facing the $-y$-direction. Without density difference, the vortex would simply translate downward. However, we give the smoke (which is similarly shaped as the vortex) a relative density of 1.7, and apply a gravitational force of $[0, -2, 0]$, which together elicits the Rayleigh-Taylor instability that deforms the vortex ring and evolves it into intricate vortex filaments.

8 DISCUSSION AND FUTURE WORKS

In this work, we propose Neural Flow Maps (NFM), an effective simulation method based on our novel SSNF neural representation, which bridges the mathematical models of characteristic mapping and impulse fluid mechanics with the efficacy of neural networks. Leveraging SSNF's cutting-edge accuracy, speed, and memory efficiency, we compute highly accurate bidirectional flow maps at a viable memory cost, to facilitate NFM's exceptional computational capabilities as demonstrated in a variety of challenging simulation scenarios, showcasing state-of-the-art energy conservation, visual complexity, and accurate recreation of real-world phenomena.

Our work represents a significant step towards harnessing the power of machine learning for high-fidelity, first-principled simulation. Leveraging the virtues of neural networks, our work presents

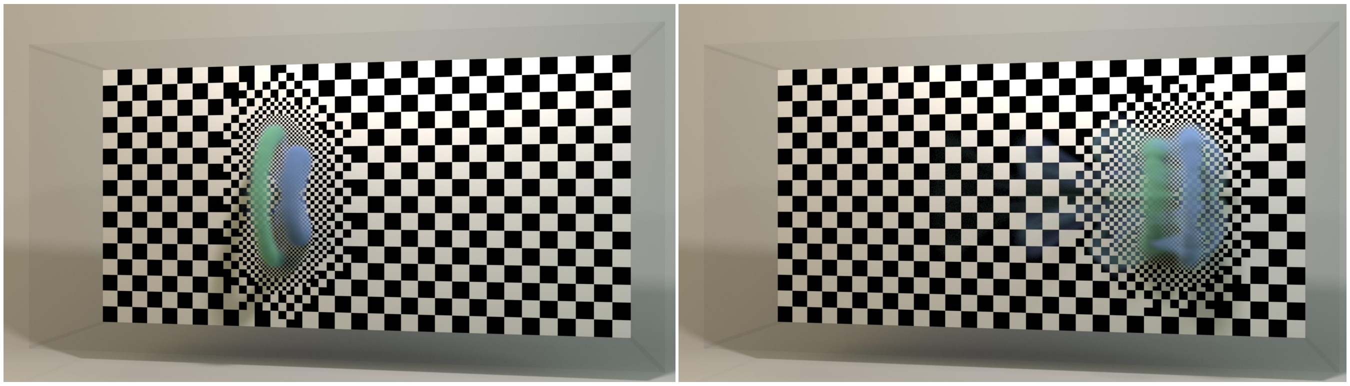

Fig. 19. The evolving spatial discretization of our neural buffer $\mathcal{N}$'s spatially sparse feature grid, as the simulation proceeds.

Average Time Cost per Step					
Method	BFECC	MC+R	CF	CF+BiMocq	Ours
Time cost	0.509s	0.539s	0.525s	0.539s	9.01s
Timing Breakdown of a Step					
Project	Advect	Advect-T	Advect-N	Train	I/O & etc.
0.312s	3.67s	0.381s	3.29s	4.83s	0.196s

Table 4. The top compares our method's wall time to those of the benchmarks; the bottom breaks down the time cost into the different subroutines. Advect-T and Advect-N correspond to the *traditional* and *neural* aspects of the advection. The timings are obtained with a laptop with Intel Core i7-12700H and NVIDIA RTX 3070 Ti.

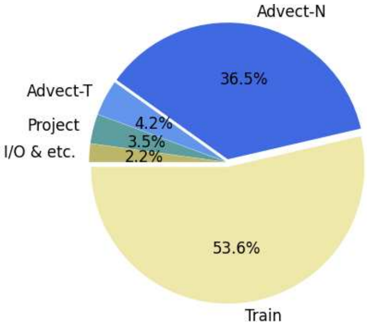

Fig. 20. Time cost breakdown.

a general approach for efficiently storing high-dimensional fields, which unlocks a myriad of algorithmic designs that were previously unattainable, and highlights the immense potential of using INRs as data primitives in physics-based simulation. On a higher level, our method differentiates from prior neural simulation methods that focus on the emulation of non-neural methods, by leveraging neural techniques to *extend the frontier* attained by existing schemes into new, unknown territories. It thereby offers a new perspective on the incorporation of machine learning in numerical simulation research for computer graphics and computational sciences alike.

Our work is subject to several limitations. First, despite SSNF's state-of-the-art efficiency, our neural advection scheme still incurs a major performance bottleneck compared to traditional methods. As shown in Table 4 and Figure 20, compared to BFECC, MC+R, and CF, our neural method increases the overall wall time by one order of magnitude, with over 90% spent on neural-related operations. To better handle larger-scale simulations, future research might further reduce the INR's training time, or devise time integration schemes that train more sporadically. Furthermore, our algorithm currently only treats smoke simulation. To extend it for water simulation, additional buffering techniques and advanced interface representations are called for to handle viscosity and surface tension respectively. Beyond fluid systems, our flow map method's potential in simulating solids and multi-physics systems might also be investigated to open up an even broader range of applications.

ACKNOWLEDGMENTS

We thank Shiying Xiong, Yuchen Sun, and Duowen Chen for the valuable discussions. Georgia Tech and Dartmouth authors acknowledge NSF IIS #2313075, ECCS #2318814, CAREER #2144806, IIS #2106733, OISE #2153560, and CNS #1919647 for funding support. Stanford authors are supported in part by NSF RI #2211258 and Google. We credit the Houdini education license for producing the video animations.

REFERENCES

Ryoichi Ando and Christopher Batty. 2020. A practical octree liquid simulator with adaptive surface resolution. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 32–1.

Christopher Batty. 2017. A cell-centred finite volume method for the Poisson problem on non-graded quadtrees with second order accurate gradients. *J. Comput. Phys.* 331 (2017), 49–72. https://api.semanticscholar.org/CorpusID:18861069

Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. 2020. Machine learning for fluid mechanics. *Annual review of fluid mechanics* 52 (2020), 477–508.

Tomas F. Buttke. 1992. Lagrangian Numerical Methods Which Preserve the Hamiltonian Structure of Incompressible Fluid Flow. https://api.semanticscholar.org/CorpusID:125992412

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 *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*. Springer, 608–625.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. Tensorf: Tensorial radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*. Springer, 333–350.

Qian Chen, Yue Wang, Hui Wang, and Xubo Yang. 2021. Data-driven simulation in fluids animation: A survey. *Virtual Reality & Intelligent Hardware* 3, 2 (2021), 87–104.

Name	Figure	Resolution	CFL	Reinit. Steps n	Encoder Min. Res	Encoder Max Res	Max Num. Params.	Avg. Num. Params.	Max Training Iters	Batch Size
2D Leapfrog	Figure 13	256 × 256	1.0	20	32 × 32	256 × 256	108,272	75,848	3,000	25,000
3D Leapfrog	Figure 19	256 × 128× 128	0.5	20	32 × 16 × 16	256 × 128× 128	4,721,104	2,905,837	3,000	240,000
3D Oblique	Figure 7	128 × 128× 128	0.5	20	16 × 16 × 16	128 × 128× 128	1,828,192	1,359,180	3,000	80,000
3D Headon	Figure 6	128 × 256× 256	0.5	12	16 × 32 × 32	128 × 256× 256	16,623,504	10,387,849	3,000	240,000
3D Trefoil	Figure 8	128 × 128× 128	0.5	20	16 × 16 × 16	128 × 128× 128	2,587,280	1,693,634	3,000	80,000
3D Four Vortices	Figure 5	256 × 128× 128	0.5	20	32 × 16 × 16	256 × 128× 128	4,306,160	1,725,884	1,500	240,000
3D Paddle	Figure 18	256 × 128× 128	0.5	8	32 × 16 × 16	256 × 128× 128	6,036,240	2,714,730	500	240,000
3D Inkdrop	Figure 1	256 × 512× 256	0.5	12	64 × 32 × 32	256 × 512× 256	30,044,000	18,193,506	1,000	320,000

Table 5. The catalog of all our 2D and 3D simulation examples. The exact numbers of simulation steps and training iterations are determined dynamically based on early-termination and the CFL condition. Typically, each example consists of 300-500 frames, a frame requires $5-10$ steps, and each step requires $100-500$ training iterations.

Chung-Ki Cho, Byungjoon Lee, and Seongjai Kim. 2018. Dual-Mesh Characteristics for Particle-Mesh Methods for the Simulation of Convection-Dominated Flows. *SIAM Journal on Scientific Computing* 40, 3 (2018), A1763–A1783.

Alexandre Joel Chorin. 1968. Numerical solution of the Navier-Stokes equations. *Mathematics of computation* 22, 104 (1968), 745–762.

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 (TOG)* 41, 4 (2022), 1–14.

Mengyu Chu and Nils Thuerey. 2017. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–14.

Mengyu Chu, Nils Thuerey, Hans-Peter Seidel, Christian Theobalt, and Rhaleb Zayer. 2021. Learning meaningful controls for fluids. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–13.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

Ricardo Cortez. 1995. *Impulse-based particle methods for fluid flow.* University of California, Berkeley.

Erwan Deriaz and Valérie Perrier. 2006. Divergence-free and curl-free wavelets in two dimensions and three dimensions: application to turbulent flows. *Journal of Turbulence* (2006). https://api.semanticscholar.org/CorpusID:52245593

Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B Tenenbaum, and Jiajun Wu. 2021. Neural radiance flow for 4d view synthesis and video processing. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV).* 14304–14314.

Karthikeyan Duraisamy, Ze J Zhang, and Anand Pratap Singh. 2015. New approaches in turbulence and transition modeling using data-driven techniques. In *53rd AIAA Aerospace sciences meeting.* 1284.

M-L Eckert, Wolfgang Heidrich, and Nils Thuerey. 2018. Coupled fluid density and motion from single views. *Computer Graphics Forum* 37, 8 (2018), 47–58.

Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. 2019. ScalarFlow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–16.

R. Elliot English, Linhai Qiu, Yue Yu, and Ronald Fedkiw. 2013. An adaptive discretization of incompressible flow using a multitude of moving Cartesian grids. *J. Comput. Phys.* 254 (2013), 107–154. https://api.semanticscholar.org/CorpusID:9179791

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* 15–22.

Fan Feng, Jinyuan Liu, Shiying Xiong, Shuqi Yang, Yaorui Zhang, and Bo Zhu. 2022. Impulse fluid simulation. *IEEE Transactions on Visualization and Computer Graphics* (2022).

Nick Foster and Dimitris N. Metaxas. 1997. Modeling the motion of a hot, turbulent gas. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997). https://api.semanticscholar.org/CorpusID:1776988

Erik Franz, Barbara Solenthaler, and Nils Thuerey. 2021. Global transport for fluid reconstruction with learned self-supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 1632–1642.

Sara Fridovich-Keil, Giacomo Meanti, Frederik Warburg, Benjamin Recht, and Angjoo Kanazawa. 2023. K-planes: Explicit radiance fields in space, time, and appearance. *arXiv preprint arXiv:2301.10241* (2023).

Chuyuan Fu, Qi Guo, Theodore F. Gast, Chenfanfu Jiang, and Joseph Teran. 2017. A polynomial particle-in-cell method. *ACM Transactions on Graphics (TOG)* 36 (2017), 1 – 12. https://api.semanticscholar.org/CorpusID:10182674

Oscar Gonzalez and Andrew Stuart. 2008. *A First Course in Continuum Mechanics.* Cambridge University Press.

James Gregson, Ivo Ihrke, Nils Thuerey, and Wolfgang Heidrich. 2014. From capture to simulation: connecting forward and inverse problems in fluids. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

James Gregson, Michael Krimerman, Matthias B Hullin, and Wolfgang Heidrich. 2012. Stochastic tomography and its applications in 3D imaging of mixing fluids. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–10.

Toshiya Hachisuka. 2005. Combined Lagrangian-Eulerian approach for accurate advection. In *ACM SIGGRAPH 2005 Posters.* 114–es.

Francis H Harlow and J Eddie Welch. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids* 8, 12 (1965), 2182–2189.

Samuel W Hasinoff and Kiriakos N Kutulakos. 2007. Photo-consistent reconstruction of semitransparent scenes by density-sheet decomposition. *IEEE transactions on pattern analysis and machine intelligence* 29, 5 (2007), 870–885.

William D. Henshaw. 1994. A Fourth-Order Accurate Method for the Incompressible Navier-Stokes Equations on Overlapping Grids. *J. Comput. Phys.* 113 (1994), 13–25. https://api.semanticscholar.org/CorpusID:120503158

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 (TOG)* 38, 6 (2019), 1–16.

Ivo Ihrke and Marcus Magnor. 2004. Image-based tomographic reconstruction of flames. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 365–373.

Antony Jameson, Wolfgang Schmidt, and Eli Turkel. 1981. Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes. https://api.semanticscholar.org/CorpusID:6948802

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–10.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The material point method for simulating continuum materials. In *Acm siggraph 2016 courses*. 1–52.

Hans Johansen and Phillip Colella. 1998. A Cartesian Grid Embedded Boundary Method for Poisson's Equation on Irregular Domains. *J. Comput. Phys.* 147 (1998), 60–85. https://api.semanticscholar.org/CorpusID:11900366

Nicholas K.-R. Kevlahan and Oleg V. Vasilyev. 2005. An Adaptive Wavelet Collocation Method for Fluid-Structure Interaction at High Reynolds Numbers. *SIAM J. Sci. Comput.* 26 (2005), 1894–1915. https://api.semanticscholar.org/CorpusID:2070583

Byungsoo Kim, Vinicius C Azevedo, Markus Gross, and Barbara Solenthaler. 2019a. Transport-based neural style transfer for smoke simulations. *arXiv preprint arXiv:1905.07442* (2019).

Byungsoo Kim, Vinicius C Azevedo, Markus Gross, and Barbara Solenthaler. 2020. Lagrangian neural style transfer for fluids. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 52–1.

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019b. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 59–70.

ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. 2006. Advections with significantly reduced dissipation and diffusion. *IEEE transactions on visualization and computer graphics* 13, 1 (2006), 135–144.

Doyub Kim, Minjae Lee, and Ken Museth. 2022. NeuralVDB: High-resolution Sparse Volume Representation using Hierarchical Neural Networks. *arXiv preprint arXiv:2208.04448* (2022).

Dustin Kleckner and William TM Irvine. 2013. Creation and dynamics of knotted vortices. *Nature physics* 9, 4 (2013), 253–258.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. 2021. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences* 118, 21 (2021), e2101784118.

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–9.

Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. 2022. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5521–5531.

Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 2021a. 3D Neural Scene Representations for Visuomotor Control. *arXiv preprint arXiv:2107.04004* (2021).

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. 2018. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566* (2018).

Zijie Li and Amir Barati Farimani. 2022. Graph neural network-accelerated Lagrangian fluid simulation. *Computers & Graphics* 103 (2022), 201–211.

Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. 2021b. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6498–6508.

TT Lim and TB Nickels. 1992. Instability and reconnection in the head-on collision of two vortex rings. *Nature* 357, 6375 (1992), 225–227.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural sparse voxel fields. *Advances in Neural Information Processing Systems* 33 (2020), 15651–15663.

Xingyu Liu, Charles R Qi, and Leonidas J Guibas. 2019. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 529–537.

Frank Losasso, Ronald Fedkiw, and S. Osher. 2006a. Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids* 35 (2006), 995–1010. https://api.semanticscholar.org/CorpusID:12862571

Frank Losasso, Ronald Fedkiw, and Stanley Osher. 2006b. Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids* 35, 10 (2006), 995–1010.

Frank Losasso, Frédéric Gibou, and Ronald Fedkiw. 2004. Simulating water and smoke with an octree data structure. *ACM SIGGRAPH 2004 Papers* (2004). https://api.semanticscholar.org/CorpusID:6233788

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

Maciej Majchrzak, Katarzyna Marciniak-Lukasiak, and Piotr Lukasiak. 2023. A Survey on the Application of Machine Learning in Turbulent Flow Simulations. *Energies* 16, 4 (2023), 1755.

Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. 2021. Acorn: Adaptive coordinate networks for neural scene representation. *arXiv preprint arXiv:2105.02788* (2021).

Daniel F. Martin, Phillip Colella, and Daniel T. Graves. 2007. A cell-centered adaptive projection method for the incompressible Navier-Stokes equations in three dimensions. *J. Comput. Phys.* 227 (2007), 1863–1886. https://api.semanticscholar.org/CorpusID:2257685

Daniel P. Martin and Keith L. Cartwright. 1996. Solving Poisson's Equation using Adaptive Mesh Renemen t. https://api.semanticscholar.org/CorpusID:12179247

Takumi Matsuzawa, Noah P. Mitchell, Stéphane Perrard, and William T.M. Irvine. 2022. Video: Turbulence through sustained vortex ring collisions. *75th Annual Meeting of the APS Division of Fluid Dynamics - Gallery of Fluid Motion* (2022). https://api.semanticscholar.org/CorpusID:252974545

Olivier Mercier and Derek Nowrouzezahrai. 2020. Local Bases for Model-reduced Smoke Simulations. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 9–22.

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.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *2003.08934* (2020).

Michael L. Minion. 1994. Two methods for the study of vortex patch evolution on locally refined grids. https://api.semanticscholar.org/CorpusID:118328125

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Y. Tong, and Mathieu Desbrun. 2009. Energy-preserving integrators for fluid animation. *ACM SIGGRAPH 2009 papers* (2009). https://api.semanticscholar.org/CorpusID:2870112

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

Mohammad Sina Nabizadeh, Stephanie Wang, Ravi Ramamoorthi, and Albert Chern. 2022. Covector fluids. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Rahul Narain, Jonas Zehnder, and Bernhard Thomaszewski. 2019. A second-order advection-reflection solver. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019), 1–14.

Elvis Nava, John Z Zhang, Mike Yan Michelis, Tao Du, Pingchuan Ma, Benjamin F Grewe, Wojciech Matusik, and Robert Kevin Katzschmann. 2022. Fast aquatic swimmer optimization with differentiable projective dynamics and neural network hydrodynamic models. In *International Conference on Machine Learning*. PMLR, 16413–16427.

Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. 2009. A gradient-augmented level set method with an optimally local, coherent advection scheme. *J. Comput. Phys.* 229 (2009), 3802–3827. https://api.semanticscholar.org/CorpusID:5668273

Makoto Okabe, Yoshinori Dobashi, Ken Anjyo, and Rikio Onai. 2015. Fluid volume modeling from sparse multi-view images by appearance transfer. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–10.

Valery Oseledets. 1989. COMMUNICATIONS OF THE MOSCOW MATHEMATICAL SOCIETY: On a new way of writing the Navier-Stokes equation. The Hamiltonian formalism. *Russian Mathematical Surveys* (1989). https://api.semanticscholar.org/CorpusID:116983811

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. 2021a. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5865–5874.

Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. 2021b. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228* (2021).

Sungheon Park, Minjung Son, Seokhwan Jang, Young Chun Ahn, Ji-Yeon Kim, and Nahyup Kang. 2023. Temporal Interpolation Is All You Need for Dynamic Neural Radiance Fields. *arXiv preprint arXiv:2302.09311* (2023).

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. 2020. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409* (2020).

Stéphane Popinet. 2003. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *J. Comput. Phys.* 190 (2003), 572–600. https://api.semanticscholar.org/CorpusID:16058065

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. 2021. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10318–10327.

Ziyin Qu, Xinxin Zhang, Ming Gao, Chenfanfu Jiang, and Baoquan Chen. 2019. Efficient and conservative fluids using bidirectional mapping. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. 2020. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* 367, 6481 (2020), 1026–1030.

André Robert. 1981. A stable numerical integration scheme for the primitive meteorological equations. *Atmosphere-ocean* 19 (1981), 35–46. https://api.semanticscholar.org/CorpusID:122755378

Bruno Roy, Pierre Poulin, and Eric Paquette. 2021. Neural upflow: A scene flow learning approach to increase the apparent resolution of particle-based liquids. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 4, 3 (2021), 1–26.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. 2020. Learning to simulate complex physics with graph networks. In *International conference on machine learning*. PMLR, 8459–8468.

Takahiro Sato, Christopher Batty, Takeo Igarashi, and Ryoichi Ando. 2018. Spatially adaptive long-term semi-Lagrangian method for accurate velocity advection. *Computational Visual Media* 4, 3 (2018), 6.

Takahiro Sato, Takeo Igarashi, Christopher Batty, and Ryoichi Ando. 2017. A long-term semi-lagrangian method for accurate velocity advection. *SIGGRAPH Asia 2017 Technical Briefs* (2017). https://api.semanticscholar.org/CorpusID:12292146

John Stanley Sawyer. 1963. A semi-Lagrangian method of solving the vorticity advection equation. *Tellus A* 15 (1963), 336–342. https://api.semanticscholar.org/CorpusID:123686485

Kai Schneider and Oleg V. Vasilyev. 2010. Wavelet Methods in Computational Fluid Dynamics. *Annual Review of Fluid Mechanics* 42 (2010), 473–503. https://api.semanticscholar.org/CorpusID:53006945

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *Journal of Scientific Computing* 35 (2008), 350–371.

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–12.

Pushan Sharma, Wai Tong Chung, Bassem Akoush, and Matthias Ihme. 2023. A Review of Physics-Informed Machine Learning in Fluid Mechanics. *Energies* 16, 5 (2023), 2343.

Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. 2017. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA journal* 55, 7 (2017), 2215–2227.

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.

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems* 32 (2019).

Kimberly Stachenfeld, Drummond B Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. 2021. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275* (2021).

Jos Stam. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 121–128.

Gilbert Strang. 1968. On the Construction and Comparison of Difference Schemes. *SIAM J. Numer. Anal.* 5 (1968), 506–517. https://api.semanticscholar.org/CorpusID:62800567

Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5459–5469.

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.

Jerry Tessendorf. 2015. Advection Solver Performance with Long Time Steps, and Strategies for Fast and Accurate Numerical Implementation. (2015).

Jerry Tessendorf and Brandon Pelfrey. 2011. The characteristic map for fast and efficient vfx fluid simulations. In *Computer Graphics International Workshop on VFX, Computer Animation, and Stereo Movies. Ottawa, Canada*.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*. PMLR, 3424–3433.

Brendan Tracey, Karthik Duraisamy, and Juan Alonso. 2013. Application of supervised learning to quantify uncertainties in turbulence and combustion modeling. In *51st AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*. 259.

Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. 2020. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*.

Nils Wandel, Michael Weinmann, and Reinhard Klein. 2020. Learning Incompressible Fluid Dynamics from Scratch–Towards Fast, Differentiable Fluid Models that Generalize. *arXiv preprint arXiv:2006.08762* (2020).

Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. 2017. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. *Physical Review Fluids* 2, 3 (2017), 034603.

Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. 2019. A multi-pass GAN for fluid flow super-resolution. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019), 1–21.

Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 71–82.

Steffen Wiewel, Byungsoo Kim, Vinicius C Azevedo, Barbara Solenthaler, and Nils Thuerey. 2020. Latent space subdivision: stable and controllable time predictions for fluid flow. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 15–25.

DC Wiggert and EB Wylie. 1976. Numerical predictions of two-dimensional transient groundwater flow by the method of characteristics. *Water Resources Research* 12, 5 (1976), 971–977.

Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. 2021. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9421–9431.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.

Xi-Yuan Yin, Kai Schneider, and Jean-Christophe Nave. 2023. A Characteristic Mapping Method for the three-dimensional incompressible Euler equations. *J. Comput. Phys.* (2023), 111876.

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5752–5761.

Hong-Xing Yu, Michelle Guo, Alireza Fathi, Yen-Yu Chang, Eric Ryan Chan, Ruohan Gao, Thomas Funkhouser, and Jiajun Wu. 2023. Learning object-centric neural scattering functions for free-viewpoint relighting and scene composition. *Transactions on Machine Learning Research* (2023).

Guangming Zang, Ramzi Idoughi, Congli Wang, Anthony Bennett, Jianguo Du, Scott Skeen, William L Roberts, Peter Wonka, and Wolfgang Heidrich. 2020. Tomofluid: Reconstructing dynamic fluid from sparse view videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1870–1879.

Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski. 2018. An advection-reflection solver for detail-preserving fluid simulation. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–8.

Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. 2021. Physg: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5453–5462.

Xinxin Zhang, Robert Bridson, and Chen Greif. 2015. Restoring the missing vorticity in advection-projection fluid solvers. *ACM Transactions on Graphics (TOG)* 34 (2015), 1 – 8. https://api.semanticscholar.org/CorpusID:15172469

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.

A ADDITIONAL PSEUDOCODE

In this section, we provide additional pseudocodes to supplement Section 5.

Algorithm 2 details the procedure for our custom RK4 integration scheme for evolving the flow maps and flow map Jacobians, Algorithm 3 outlines the second-order, midpoint method, and Algorithm 4 describes the error-compensated impulse advection scheme.

Algorithm 2 Interleaved RK4 for ϕ and $\mathcal{F}$

```
Input: $\boldsymbol{u}, \phi, \mathcal{F}, \Delta t$
Output: $\phi_{\text{next}}, \mathcal{F}_{\text{next}}$
$(\boldsymbol{u}_1, \nabla\boldsymbol{u}|_1) \leftarrow$ Interpolate$(\boldsymbol{u}, \phi)$;
$\frac{\partial\mathcal{F}}{\partial t}|_1 \leftarrow \nabla\boldsymbol{u}|_1\mathcal{F}$;
$\phi_1 \leftarrow \phi + 0.5 \cdot \Delta t \cdot \boldsymbol{u}_1$;
$\mathcal{F}_1 \leftarrow \mathcal{F} + 0.5 \cdot \Delta t \cdot \frac{\partial\mathcal{F}}{\partial t}|_1$;
$(\boldsymbol{u}_2, \nabla\boldsymbol{u}|_2) \leftarrow$ Interpolate$(\boldsymbol{u}, \phi_1)$;
$\frac{\partial\mathcal{F}}{\partial t}|_2 \leftarrow \nabla\boldsymbol{u}|_2\mathcal{F}_1$;
$\phi_2 \leftarrow \phi + 0.5 \cdot \Delta t \cdot \boldsymbol{u}_2$;
$\mathcal{F}_2 \leftarrow \mathcal{F} + 0.5 \cdot \Delta t \cdot \frac{\partial\mathcal{F}}{\partial t}|_2$;
$(\boldsymbol{u}_3, \nabla\boldsymbol{u}|_3) \leftarrow$ Interpolate$(\boldsymbol{u}, \phi_2)$;
$\frac{\partial\mathcal{F}}{\partial t}|_3 \leftarrow \nabla\boldsymbol{u}|_3\mathcal{F}_2$;
$\phi_3 \leftarrow \phi + \Delta t \cdot \boldsymbol{u}_3$;
$\mathcal{F}_3 \leftarrow \mathcal{F} + \Delta t \cdot \frac{\partial\mathcal{F}}{\partial t}|_3$;
$(\boldsymbol{u}_4, \nabla\boldsymbol{u}|_4) \leftarrow$ Interpolate$(\boldsymbol{u}, \phi_3)$;
$\frac{\partial\mathcal{F}}{\partial t}|_4 \leftarrow \nabla\boldsymbol{u}|_4\mathcal{F}_3$;
$\phi_{\text{next}} \leftarrow \phi + \Delta t \cdot \frac{1}{6} \cdot (\boldsymbol{u}_1 + 2 \cdot \boldsymbol{u}_2 + 2 \cdot \boldsymbol{u}_3 + \boldsymbol{u}_4)$;
$\mathcal{F}_{\text{next}} \leftarrow \mathcal{F} + \Delta t \cdot \frac{1}{6} \cdot (\frac{\partial\mathcal{F}}{\partial t}|_1 + 2 \cdot \frac{\partial\mathcal{F}}{\partial t}|_2 + 2 \cdot \frac{\partial\mathcal{F}}{\partial t}|_3 + \frac{\partial\mathcal{F}}{\partial t}|_4)$;
```

Algorithm 3 Midpoint Method

```
Input: $\boldsymbol{u}$
Output: $\boldsymbol{u}_{\text{mid}}$
Reset $\psi, \mathcal{T}$ to identity;
March $\psi, \mathcal{T}$ with $\boldsymbol{u}$ and $-0.5\Delta t$ using Alg. 2;
$\boldsymbol{m}_{\text{mid}} \leftarrow \mathcal{T}^T\boldsymbol{u}(\psi)$;
if use external force then
    $\boldsymbol{m}_{\text{mid}} \leftarrow \boldsymbol{m}_{\text{mid}} + 0.5 \cdot \Delta t \cdot \boldsymbol{f}_{\text{ext}}$;
end if
$\boldsymbol{u}_{\text{mid}} \leftarrow$ Poisson$(\boldsymbol{m}_{\text{mid}})$;
```

Algorithm 4 Error-compensated Impulse Advection

```
Input: $\boldsymbol{u}_0, \psi, \mathcal{T}, \phi, \mathcal{F}$
Output: $\boldsymbol{u}$
$\bar{\boldsymbol{m}} \leftarrow \mathcal{T}^T\boldsymbol{u}_0(\psi)$;
$\bar{\boldsymbol{m}}_0 \leftarrow \mathcal{F}^T\bar{\boldsymbol{m}}(\phi)$;
$\boldsymbol{e} \leftarrow 0.5 \cdot (\bar{\boldsymbol{m}}_0 - \boldsymbol{u}_0)$;
$\bar{\boldsymbol{e}} \leftarrow \mathcal{T}^T\boldsymbol{e}(\psi)$;
$\hat{\boldsymbol{m}} \leftarrow \bar{\boldsymbol{m}} - \bar{\boldsymbol{e}}$;
$\boldsymbol{m} \leftarrow$ Clamp$(\hat{\boldsymbol{m}})$;
$\boldsymbol{u} \leftarrow$ Poisson$(\boldsymbol{m})$;
```

B IMPLEMENTATION DETAILS FOR NFM

MAC Grid Stencil. We use the standard MAC grid [Harlow and Welch 1965] for storing the velocity components, and we evolve flow maps and flow map Jacobians on face centers. To carry out impulse-based advection, the matrix multiplication $\mathcal{T}^T\boldsymbol{u}$ needs to be evaluated, which can be rewritten as $[\frac{\partial\psi}{\partial x} \cdot \boldsymbol{u}, \frac{\partial\psi}{\partial y} \cdot \boldsymbol{u}, \frac{\partial\psi}{\partial z} \cdot \boldsymbol{u}]^T$. As a result, for faces storing $\boldsymbol{u}_x$, instead of storing the full Jacobian matrix $\mathcal{T}$, we only need to store its first column $\frac{\partial\psi}{\partial x}$. Similarly, for faces with $\boldsymbol{u}_y$, we store $\frac{\partial\psi}{\partial y}$; for faces with $\boldsymbol{u}_z$, we store $\frac{\partial\psi}{\partial z}$. The situation is analogous for ϕ and $\mathcal{F}$ as well.

As a result, in our implementation, the $\mathcal{T}$ and $\mathcal{F}$ matrices are always stored as columns at the staggered face centers. And the Algorithm 2 is applied to evolve each column individually.

Interpolating $\boldsymbol{u}$ and $\nabla\boldsymbol{u}$. We approximate the velocity $\boldsymbol{u}$ and velocity Jacobian $\nabla\boldsymbol{u}$ at subgrid points using the MPM interpolation scheme [Jiang et al. 2016] with the quadratic kernel:

$$N(x) = \begin{cases} \frac{3}{4} - |x|^2 & 0 \le |x| < \frac{1}{2}, \\ \frac{1}{2}(\frac{3}{2} - |x|)^2 & \frac{1}{2} \le |x| < \frac{3}{2}, \\ 0 & \frac{3}{2} \le |x|. \end{cases} \tag{11}$$

Empirically, we find the quadratic kernel to perform the best, as the linear kernel leads to numerical instabilities due to its discontinuous gradient and the cubic kernel leads to numerical diffusion due to its larger support size. Alternatively, automatic differentiation can be used to directly query velocity gradients from the SSNF buffer $\mathcal{N}$. In practice, however, this leads to significant noise in the gradient computation, and makes the simulation unstable.

Sizing Function. We compute the sizing value S as the Frobenius norm of the velocity Jacobian:

$$S = \sqrt{\sum_{i=1}^{d}\sum_{j=1}^{d}\left|\frac{\partial\boldsymbol{u}_i}{\partial\boldsymbol{e}_j}\right|^2}. \tag{12}$$

Activating the feature grid of $\mathcal{N}$ using S computed in this way can lead to sharp level transitions. Our computational scheme handles these sharp transitions naturally, but they nevertheless undermine the fitting accuracy. To encourage a smooth transition between levels, we dilate the sizing function for 1024 iterations as follows:

$$S_{i,j}^{k+1} = \max(S_{i,j}^k, 0.25 \cdot (S_{i-1,j}^k + S_{i+1,j}^k + S_{i,j-1}^k + S_{i,j+1}^k)). \tag{13}$$

In other words, we allow the sizing value in a voxel to diffuse to neighboring voxels without decreasing its original value.

Gravity and Buoyancy. For the realization of density difference-driven effects, we apply a Boussinesq buoyancy force $\boldsymbol{f}_{\text{ext}} = c \cdot \rho \cdot \boldsymbol{g}$. We assume ρ remains constant during flow paths, so the integration for this term can be trivially computed by multiplying with the total timelapse. We note that such a force integral should be added to the velocity $\boldsymbol{u}$ instead of the impulse $\boldsymbol{m}$. The raw velocity after this addition will be projected again to obtain the final velocity.

Solid Boundaries and Obstacles. Our NFM simulation system naturally supports moving, voxelized boundaries. Since our neural buffer $\mathcal{N}$ stores spatiotemporal velocity fields that respect the moving boundary's geometry and kinematics, a path obtained by marching

$\mathcal{N}$ will be valid and will not penetrate the boundary as long as $\mathcal{N}$ has stored the velocity accurately, which ensures that our advection scheme will work in this case just like in the boundary-free cases.

With this being said, it is indeed more challenging for $\mathcal{N}$ to learn the velocity field when obstacles are present, as they lead to sharp transitions in the velocity profile near the edges. We opt to smooth out these sharp velocity transitions by extrapolating the fluid velocity into the boundary, and letting $\mathcal{N}$ learn the extrapolated version, which significantly improves the convergence performance. We note that for the purpose of flow map-marching, modifying the solid velocity does not sacrifice the physical integrity, because a *valid* path makes no use of the solid velocity anyways, so it would not be affected by such a modification; a path that penetrates into the boundary would be *invalid* to begin with, and in that case, it can only benefit from such a modification as the marched flow map would be more smooth, which makes the simulation more stable. For Poisson solving, we make no modifications to the velocity.

Additionally, with a moving, voxelized boundary inside all Neumann walls, the total numerical divergence can deviate from 0, causing the Poisson solver to not converge. We compensate for this deviation by offsetting the divergence of each fluid voxel by a constant value so that the divergence field sums to 0.

C SETUP DETAILS FOR SUBSECTION 6.1

Experimental Setup. We set up the benchmarking experiment by generating offline a 25-frame sequence of MAC grid velocity fields for both 2D and 3D. In 2D, we use the leapfrog setup; in 3D, we use the trefoil knot setup. For all four methods, we adjust the hyperparameters so that the parameter count is about the same. For fairness of comparison, all three benchmarks are extended with the multi-branch decoder in our method, one for each spatial dimension. All methods are trained for 2000 iterations with the same optimizer and learning rate scheduler. In 2D, all methods are trained with a batch size of 25000; in 3D, all methods are trained with a batch size of 120000.

INGP. For the 2D test, INGP uses 16 scales from min resolution [16, 16, 2] to max resolution [256, 256, 24] (the last dimension is time). The max number of parameters for each level is set to 2600. For the 3D test, INGP uses 16 scales from min resolution [8, 8, 8, 2] to max resolution [256, 256, 256, 24] with the max number of parameters being 87880 per level. In both 2D and 3D, we use an ensemble of MLP decoders with a depth of 2 and a width of 64.

KPlanes. For the 2D test, KPlanes uses 3 scales from min resolution [16, 16, 8] to max resolution [64, 64, 8] (the last dimension is time), using no multi-resolution for the time axis as suggested by Fridovich-Keil et al. [2023]. The feature length is set to 12, which is smaller than the paper's suggested number to stay within a comparable budget. For the 3D test, KPlanes uses 3 scales from [32, 32, 32, 8] to [128, 128, 128, 8], and a feature length of 32. The decoders used for KPlanes are the same as those for INGP.

SIREN. For the 2D test, SIREN uses an MLP with a depth of 4 and a width of 180. For the 3D test, SIREN uses an MLP with a depth of 8 and a width of 256. Both MLPs have sinusoidal activation functions. Following the approach by Sitzmann et al. [2020], we employ a frequency multiplier of $\omega_0 = 30$ to boost up the networks' characteristic frequency. We perform this for the spatial axes only to ensure temporal smoothness.

In order to fairly compare the computation time, we implement all four methods using PyTorch for neural networks and optimization, and Taichi [Hu et al. 2019] for efficient feature vector storage.

The training procedure in this test mimics the actual simulation scenario, where our "dynamic scene", a 25-frame velocity sequence, is not presented to the INRs all at once, but rather as a stream of frames. Each model is only presented with the newest frame, and the prior frames must be read from its auxiliary buffer $\hat{\mathcal{N}}$.