# *EDGE*: Epsilon-Difference Gradient Evolution for Buffer-Free Flow Maps

ZHIQI LI*, Georgia Institute of Technology, USA
RUICHENG WANG*, Georgia Institute of Technology, USA
JUNLIN LI*, Georgia Institute of Technology, USA
DUOWEN CHEN, Georgia Institute of Technology, USA
SINAN WANG, Georgia Institute of Technology, USA
BO ZHU, Georgia Institute of Technology, USA

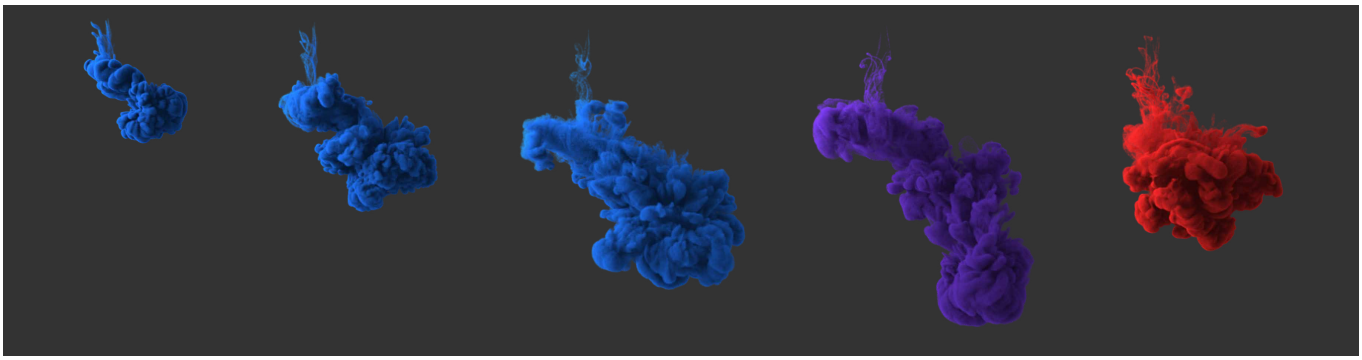


Fig. 1. Dye drift simulation using flow map methods. Our novel strategy for flow map evolution, Epsilon-Difference Gradient Evolution (EDGE) (blue dye) and Four-Point Epsilon Difference (ED4) (purple dye), achieves vorticity conservation on par with the original Eulerian Flow Map (EFM) method (red dye), but with significantly reduced memory usage. While EFM requires 37.89 GB of overall simulation memory, EDGE cuts this down to 10.79 GB, and ED4 further reduces it to 8.54 GB—a performance leap without compromising accuracy.

We propose the Epsilon Difference Gradient Evolution (EDGE) method for accurate flow-map calculation on grids via Hermite interpolation without using velocity buffers. Our key idea is to integrate Gradient Evolution for accurate first-order derivatives and a tetrahedron-based Epsilon Difference scheme to compute higher-order derivatives with reduced memory consumption. EDGE achieves $O(1)$ memory usage, independent of flow map length, while maintaining vorticity preservation comparable to buffer-based methods. We validate our methods across diverse vortical flow scenarios, demonstrating up to 90% backward map memory reduction and significant computational efficiency, broadening the applicability of flow-map methods to large-scale and complex fluid simulations.

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

Additional Key Words and Phrases: Flow Map Methods, Gradient Evolution, Epsilon Difference, Incompressible FLow

*joint first authors

Authors' Contact Information: Zhiqi Li, zli3167@gatech.edu, Georgia Institute of Technology, USA; Ruicheng Wang, wrc0326@outlook.com, Georgia Institute of Technology, USA; Junlin Li, jli3518@gatech.edu, Georgia Institute of Technology, USA; Duowen Chen, dchen322@gatech.edu, Georgia Institute of Technology, USA; Sinan Wang, swang3081@gatech.edu, Georgia Institute of Technology, USA; Bo Zhu, bo.zhu@gatech.edu, Georgia Institute of Technology, USA.


ACM 1557-7368/2025/8-ART96
https://doi.org/10.1145/3731193

**ACM Reference Format:**
Zhiqi Li, Ruicheng Wang, Junlin Li, Duowen Chen, Sinan Wang, and Bo Zhu. 2025. *EDGE*: Epsilon-Difference Gradient Evolution for Buffer-Free Flow Maps. *ACM Trans. Graph.* 44, 4, Article 96 (August 2025), 11 pages. https://doi.org/10.1145/3731193

## 1 Introduction

Flow-map methods have garnered increasing attention for their exceptional ability to preserve spatiotemporal vortical structures. The fundamental mechanism underlying this capability lies in constructing a *perfect* long-range flow map between frames. As demonstrated in [Deng et al. 2023] and subsequent works, a perfect flow map must satisfy two key geometric properties: (1) a point undergoing a forward map followed by a backward map returns to its original position, and (2) the product of the forward and backward Jacobians equals identity. To date, two representations have been established for constructing a perfect flow map: a purely Eulerian representation on grids, through the use of velocity buffers (e.g., storing a sequence of previous frames or training a neural network), and a Lagrangian representation, through the use of moving particles carrying the gradients of flow maps. The core idea behind both mechanisms is the same: each flow map is represented by a Lagrangian trajectory in the spatiotemporal domain (a backward map for the grid representation and a forward map for the particle representation) that allows points to move forward or backward in an exactly symmetric manner along the temporal axis.

The computation and memory costs for both mechanisms are significant. In an Eulerian setting, a reverse advection step is required for every grid node to construct its long-range trajectory, necessitating $O(n)$ velocity buffers and evolution steps for the $n$-th time step. Compressing such storage into implicit representations, such as training a neural network [Deng et al. 2023], can reduce the memory cost to $O(1)$, but it incurs significant computational and training overhead, as the training must be performed at every time step. On the Lagrangian side, particles are used to represent the flow map [Zhou et al. 2024], allowing trajectories to start from arbitrary points rather than grid nodes. This approach achieves $O(1)$ memory per particle without requiring velocity buffers. However, as a typical particle-grid method, particle flow maps demand 8–16 particles per grid cell in 3D, leading to substantial memory consumption and a less coherent memory layout due to the use of particles. To date, there is no perfect solution that enables a flow-map method to retain its vorticity-preserving capability while maintaining computational and memory costs comparable to standard grid-based (e.g., advection-projection) or particle-grid (PIC/FLIP) fluid solvers.

This paper aims to explore a third pathway to reduce the computational cost of flow-map methods, moving beyond stacking velocity buffers or utilizing dynamic particles. The key idea is to construct an "imperfect" flow map, rather than a "perfect" one, by reverting to a semi-Lagrangian-style one-step advection scheme to eliminate the velocity buffer and employing Hermite interpolation to enhance advection accuracy. At first glance, this seemingly old-fashioned idea appears unworkable, as the distortion of flow maps could rapidly degrade the interpolated values and gradients from grid nodes, causing the flow-map trajectory to quickly deviate from the ground truth. One potential avenue for this scheme to succeed lies in the ability to accurately compute up-to-third-order gradients of the flow map, which are required by Hermite interpolation, a task that is computationally challenging or even impractical on a grid discretization.

We explored two strategies existing in the literature of computational physics and computer graphics to address this challenge: (1) **Gradient Evolution** (GE) [Li et al. 2023], which evolves gradients along the flow map instead of relying on finite-difference or finite-element stencils on grid nodes. While effective for first-order derivatives, higher-order derivatives are still computed using finite differences due to the lack of explicit evolution equations, limiting the method's accuracy. (2) **Epsilon Difference** (ED) [Chidyagwai et al. 2011; Seibold et al. 2011], which calculates high-order derivatives by placing eight sample points to form an $\epsilon$-sized cubic element around each target point. However, the accuracy of the $\epsilon$-difference method is constrained by $\epsilon$, which is limited by machine precision, and the efficiency of storing eight sample points per grid node is also a concern, making the balance between stability, accuracy, and efficiency a persistent challenge.

We propose a new flow-map advection method that combines the merits of both Gradient Evolution and Epsilon Difference while addressing their inherent weaknesses in stability and accuracy. The key idea is to use Gradient Evolution to maintain accurate first-order derivatives and then apply the Epsilon Difference method on top of these first-order derivatives to compute higher-order mixed derivatives, enabling a balance between precision and stability for $\epsilon$. To further reduce memory consumption and computational cost, we introduce a four-point tetrahedron Epsilon Difference method, which requires fewer sample points compared to the standard cubic element approach. We demonstrate the efficacy of our methods across a variety of vortical flow simulation scenarios, achieving up to a 90% reduction in backward map memory consumption compared to previous buffer-based flow-map methods, while still producing vortical structures comparable to the traditional flow map methods. As our method integrates elements from both Epsilon Difference and Gradient Evolution, we name it **Epsilon Difference Gradient Evolution** (EDGE).

We summarize our key contributions as follows:

(1) We propose a buffer-free flow map method with $O(1)$ memory consumption, independent of the flow map length, while preserving vortices comparable to buffer-based methods.
(2) We develop a novel high-order advection scheme by evolving first-order derivatives and computing high-order mixed derivatives using the epsilon difference method, addressing the weaknesses of both approaches in flow map settings.
(3) We introduce a tetrahedron-based epsilon element scheme to further reduce computational cost for flow map methods.

## 2 Related Work

*Advection Schemes.* The advection term plays a pivotal role in fluid dynamics [Stam 1999]. However, high diffusion errors caused by repeated interpolations need to be addressed. A range of solutions has been extensively explored within both the computational physics and graphics communities to address this challenge. Notable methods include RK4 [Jameson et al. 1981], HJ-WENO [Losasso et al. 2006], Hermite Interpolation [McGregor and Nave 2019; Nave et al. 2010; Ni et al. 2020], Jet Scheme [Seibold et al. 2011], energy conservative semi-lagrangian method [Lentine et al. 2011b], the MacCormack method [Selle et al. 2008], and the Back and Forth Error Compensation and Correction (BFECC) method [Kim et al. 2006], among others. These improved advection schemes have led to remarkable advancements in simulating various phenomena, with smoke simulation [Fedkiw et al. 2001; Kim et al. 2006; Lentine et al. 2011a; Mullen et al. 2009] standing out as a particularly significant application, requiring exacting accuracy to capture its intricate vortical behavior.

Recently, advection techniques utilizing flow map methods [Deng et al. 2023; Nabizadeh et al. 2022; Taylor and Nave 2023] have achieved state-of-the-art results in simulating incompressible flows. The progress and development of these techniques will be further reviewed in a subsequent section.

*Impulse and Vortex Methods.* Flow map methods are highly related to the impulse model. Initially introduced by [Buttke 1992], this concept provides an alternative formulation of the incompressible Navier-Stokes equations using a gauge variable and gauge transformation [Buttke and Chorin 1993; Oseledets 1989; Roberts 1972]. Following research explored its application to surface turbulence [Buttke 1993; Buttke and Chorin 1993], fluid-structure interactions [Cortez 1996; Summers 2000], and numerical stability [Weinan and Liu 2003]. More recently, Saye [2016, 2017] utilized gauge freedom to handle interfacial discontinuities in density and viscosity for free surface flows and fluid-structure coupling. In computer graphics,

this concept of gauge freedom was revised in recent works [Deng et al. 2023; Feng et al. 2022; Li et al. 2024a; Nabizadeh et al. 2022].

Vortex methods, on the other hand, use a different way to reformulate the incompressible Navier-Stokes equations by treating vorticity as a gauge variable for velocity. Because vorticity is explicitly advected in this approach, fluid circulation is preserved naturally. To represent vorticity, some chose to use Lagrangian methods, including particles [Angelidis 2017; Cottet et al. 2000; Park and Kim 2005; Zhang and Bridson 2014], filaments [Angelidis and Neyret 2005; Padilla et al. 2019; Weißmann and Pinkall 2010], segments [Xiong et al. 2021], sheets [Brochu et al. 2012; Pfaff et al. 2012], and Clebsch level sets [Chern et al. 2016; Xiong et al. 2022; Yang et al. 2021]. While these representations improve the preservation of vortex structures, they are less straightforward to implement compared to the Eulerian vortex method, especially when dealing with solid boundaries. Others [Ando et al. 2015; Yin et al. 2023] adopted vorticity-streamfunction formulations to handle boundaries and solved a potential equation. Wang et al. [2024] proposed a new scheme based on flow maps for vorticity-to-velocity reconstruction that is both efficient and able to accurately handle boundaries.

*Flow Map Methods.* The origins of flow map methods can be traced back to the characteristic map approach in computational fluid dynamics (CFD). Wiggert and Wylie [1976] first applied this method to fluid simulation to mitigate numerical dissipation through velocity field-advected long-range mapping. Subsequently, the technique was adopted in computer graphics by Hachisuka [2005] and Tessendorf and Pelfrey [2011]. Later research [Sato et al. 2018, 2017; Tessendorf 2015] commonly relied on computationally intensive virtual particle methods to track flow maps. Inspired by Kim et al. [2006], Qu et al. [2019] introduced a Semi-Lagrangian-like scheme for bidirectional flow map advection, significantly reducing computational cost while enhancing mapping accuracy.

Recently, Nabizadeh et al. [2022] extended this framework to the impulse fluid model [Cortez 1996]. Additionally, Neural Flow Maps (NFM) [Deng et al. 2023] presented a novel backward flow map advection scheme and leveraged neural networks to efficiently compress velocity buffers during flow map reconstruction. Wang et al. [2024] integrated the flow map concept with the vortex method to enhance numerical stability and physical interpretability. Furthermore, Zhou et al. [2024] introduced long-short flow maps, enabling the transport of impulses on Lagrangian particles using the APIC scheme [Jiang et al. 2015], achieving state-of-the-art results.

## 3 Mathematical Foundation

### 3.1 Flow Map Method

Flow maps are used to describe the correspondence between spatial points in the domain of a flowing fluid at different times. For any two domains $\mathbb{U}_{t_1}$ and $\mathbb{U}_{t_2}$ at times $t_1, t_2 \geq s$ of a fluid flowing according to the velocity field $\mathbf{u}(\mathbf{x}, t)$ from initial time $s$, the forward flow map $\boldsymbol{\Phi}_{t_1 \to t_2} : \mathbb{U}_{t_1} \to \mathbb{U}_{t_2}$ and the backward flow map $\boldsymbol{\Psi}_{t_2 \to t_1} : \mathbb{U}_{t_2} \to \mathbb{U}_{t_1}$ are defined as mappings that satisfy $\boldsymbol{\Phi}_{t_1 \to t_2}(\mathbf{x}_p(t_1)) = \mathbf{x}_p(t_2)$ and $\boldsymbol{\Psi}_{t_2 \to t_1}(\mathbf{x}_p(t_2)) = \mathbf{x}_p(t_1)$ for any particle $p$ moving with the fluid as $\frac{d\mathbf{x}_p(t)}{dt} = \mathbf{u}(\mathbf{x}_p(t), t)$, with position $\mathbf{x}_p(t)$ at time $t$. Forward and backward flow maps satisfy the evolution equations:

$$\begin{aligned} \frac{\partial \boldsymbol{\Phi}_{s \to t}(\mathbf{x})}{\partial t} &= \mathbf{u}(\boldsymbol{\Phi}_{s \to t}(\mathbf{x}), t), \quad & \boldsymbol{\Phi}_{s \to s}(\mathbf{x}) &= \mathbf{x}, \\ \frac{D\boldsymbol{\Psi}_{t \to s}(\mathbf{x})}{Dt} &= 0, & \boldsymbol{\Psi}_{s \to s}(\mathbf{x}) &= \mathbf{x}. \end{aligned} \tag{1}$$

The Jacobians of forward and backward flow maps are defined as $\mathcal{F}_{t_1 \to t_2}(\mathbf{x}) = \frac{\partial \boldsymbol{\Phi}_{t_1 \to t_2}(\mathbf{x})}{\partial \mathbf{x}}, \mathbf{x} \in \mathbb{U}_{t_1}$ and $\mathcal{T}_{t_2 \to t_1}(\mathbf{x}) = \frac{\partial \boldsymbol{\Psi}_{t_2 \to t_1}(\mathbf{x})}{\partial \mathbf{x}}, \mathbf{x} \in \mathbb{U}_{t_2}$ respectively, which can be proved to satisfy the evolution equations [Deng et al. 2023]:

$$\begin{aligned} \frac{\partial \mathcal{F}_{s \to t}(\mathbf{x})}{\partial t} &= \nabla \mathbf{u}(\boldsymbol{\Phi}_{s \to t}(\mathbf{x}), t) \mathcal{F}_{s \to t}(\mathbf{x}), \quad & \mathcal{F}_{s \to s}(\mathbf{x}) &= \mathbf{I}, \\ \frac{D\mathcal{T}_{t \to s}(\mathbf{x})}{Dt} &= -\mathcal{T}_{t \to s}(\mathbf{x}) \nabla \mathbf{u}(\mathbf{x}, t), & \mathcal{T}_{s \to s}(\mathbf{x}) &= \mathbf{I}, \end{aligned} \tag{2}$$

where $\mathbf{I}$ is the identity matrix.

For incompressible fluids that satisfy the Euler equations,

$$\begin{aligned} \left( \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \right) \mathbf{u} &= -\frac{1}{\rho} \nabla p + \mathbf{f}, \\ \nabla \cdot \mathbf{u} &= 0, \end{aligned} \tag{3}$$

where $\rho$, $p$, and $\mathbf{f}$ denote the fluid's density, pressure, and fluid forces, like viscous force $\mathbf{f}_v = \nu \Delta \mathbf{u}$ with viscosity $\nu$. Flow maps can be used to describe their solutions as [Li et al. 2024b]:

$$\begin{aligned} \mathbf{u}(\mathbf{x}, t) &= \mathcal{T}_{t \to s}^{\top}(\mathbf{x}) \mathbf{u}(\boldsymbol{\Psi}_{t \to s}(\mathbf{x}), s) + \mathcal{T}_{t \to s}^{\top}(\mathbf{x}) \boldsymbol{\Gamma}_{s \to t}^{\mathbf{u}}(\boldsymbol{\Psi}_{t \to s}(\mathbf{x})), \\ \boldsymbol{\Gamma}_{s \to t}^{\mathbf{u}}(\mathbf{x}) &= \int_s^t \mathcal{F}_{s \to \tau}^{\top}(\mathbf{x}) \left( -\frac{1}{\rho} \nabla p + \frac{1}{2} \nabla |\mathbf{u}|^2 + \mathbf{f} \right) (\boldsymbol{\Phi}_{s \to \tau}(\mathbf{x}), \tau) d\tau, \end{aligned} \tag{4}$$

where the first term $\mathbf{u}_t^M(\mathbf{x}) = \mathcal{T}_{t \to s}(\mathbf{x})^{\top} \mathbf{u}(\boldsymbol{\Psi}_{t \to s}(\mathbf{x}), s)$ of $\mathbf{u}(\mathbf{x}, t)$ is referred to as the long-term mapped velocity since it is directly mapped from the initial velocity $\mathbf{u}_s$ using long-term flow maps $\boldsymbol{\Psi}_{t \to s}$, and $\boldsymbol{\Gamma}_{s \to t}^{\mathbf{u}}$ is called the path integrator because it accumulates integration along the trajectory $\mathbf{S}_{\mathbf{x}_0}(t) = \boldsymbol{\Phi}_{s \to t}(\mathbf{x}_0)$ for any material point $\mathbf{x}_0 \in \mathbb{U}_s$ on the flow map. Furthermore, for the vorticity-form Euler equations

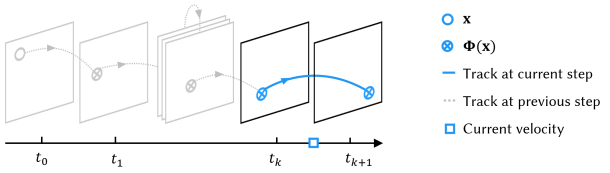
$$\begin{aligned} \left( \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \right) \boldsymbol{\omega} &= (\boldsymbol{\omega} \cdot \nabla) \mathbf{u} + \nabla \times \mathbf{f}, \\ \nabla \times \mathbf{u} &= \boldsymbol{\omega}, \end{aligned} \tag{5}$$

flow maps can also be used to describe their solutions as

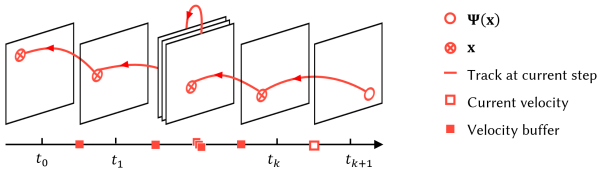
$$\begin{aligned} \boldsymbol{\omega}(\mathbf{x}, t) &= \mathcal{T}_{t \to s}^{-1}(\mathbf{x}) \boldsymbol{\omega}(\boldsymbol{\Psi}_{t \to s}(\mathbf{x}), s) + \mathcal{T}_{t \to s}^{-1}(\mathbf{x}) \boldsymbol{\Gamma}_{s \to t}^{\boldsymbol{\omega}}(\boldsymbol{\Psi}_{t \to s}(\mathbf{x})), \\ \boldsymbol{\Gamma}_{s \to t}^{\boldsymbol{\omega}}(\mathbf{x}) &= \int_s^t \mathcal{F}_{s \to \tau}^{-1}(\mathbf{x}) \left( \nabla \times \mathbf{f} \right) (\boldsymbol{\Phi}_{s \to \tau}(\mathbf{x}), \tau) d\tau, \end{aligned} \tag{6}$$

where the first term, $\boldsymbol{\omega}_t^M(\mathbf{x}) = \mathcal{T}_{t \to s}^{-1}(\mathbf{x}) \boldsymbol{\omega}(\boldsymbol{\Psi}_{t \to s}(\mathbf{x}), s)$, is similarly referred to as the long-term mapped vorticity, and $\boldsymbol{\Gamma}_{s \to t}^{\boldsymbol{\omega}}$ is the path integrator for vorticity.

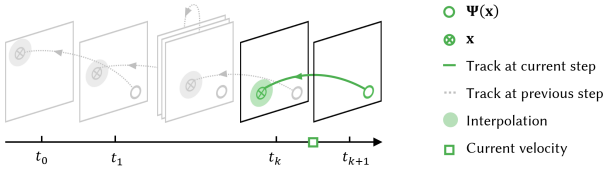
The flow map method utilizes long-term mapped velocity $\mathbf{u}_t^M(\mathbf{x})$ [Deng et al. 2023] and vorticity $\boldsymbol{\omega}_t^M(\mathbf{x})$ [Wang et al. 2024] instead of computing short-term advected velocity $\mathbf{u}_t^A(\mathbf{x}) = \mathbf{u}_{t'}(\boldsymbol{\Psi}_{t \to t'}(\mathbf{x}))$, which is calculated repeatly from the previous substep time $t'$ in semi-Lagrangian methods when solving the advection terms in Equation 3. By avoiding repeated interpolations inherent in semi-Lagrangian approaches, the flow map method prevents cumulative numerical dissipation, ensuring more accurate advection calculations and demonstrating superior vorticity preservation capabilities.




(a) Forward map evaluation in EFM




(b) Backward map evaluation in EFM




(c) Backward map evaluation with interpolation

Fig. 2. Comparison of methods for evolving flow maps. (a) Forward flow maps are evolved progressively over time steps. (b) Backward flow maps can be computed by inversely evolving forward maps from the $(k+1)$-th time step back to the initial time, which requires storing velocity fields for the previous $k$ steps. (c) When backward flow maps are progressively evolved using a semi-Lagrangian scheme, interpolation is inevitable.

## 3.2 Memory Issues for Flow Map Buffers

In the flow map method, accurate computation of $\mathbf{u}_t^M(\mathbf{x})$ and $\boldsymbol{\omega}_t^M(\mathbf{x})$ depends on the accuracy of the flow maps and their Jacobians and thus computing flow maps and their Jacobians precisely is critical. For the Eulerian method, in Equation 1 and Equation 2, the evolution equations of $\mathbf{\Phi}_{s\to t}$ and $\mathcal{F}_{s\to t}$ can be accurately computed using high-order time integration algorithms, like the 4th order Runge-Kutta (RK4) method without advection terms, while the evolution equations of $\mathbf{\Psi}_{t\to s}$ and $\mathcal{T}_{t\to s}$ contain advection terms, making their stable computation require the semi-Lagrangian method. The repeated interpolation of the semi-Lagrangian method with low-accuracy kernel-based interpolation leads to accumulated errors, making the evolution of $\mathbf{\Psi}_{t\to s}$ and $\mathcal{T}_{t\to s}$ inaccurate. Therefore, **the key issue in flow map calculation is to calculate $\mathbf{\Psi}_{t\to s}$ and $\mathcal{T}_{t\to s}$ accurately.**

Deng et al. [2023] found that at any given time $r$, $\mathbf{\Psi}_{r\to s}$ and $\mathcal{T}_{r\to s}$ can be viewed as the result of evolving $\mathbf{\Psi}_{r\to t}$ and $\mathcal{T}_{r\to t}$ backward from time $r$ to time $s$, which follow the evolution equations in this inverse fluid motion process:

$$\begin{cases} \frac{\partial \mathbf{\Psi}_{r\to t}(\mathbf{x})}{\partial t} = \mathbf{u}(\mathbf{\Psi}_{r\to t}(\mathbf{x}), t), & \mathbf{\Phi}_{r\to r}(\mathbf{x}) = \mathbf{x}, \\ \frac{\partial \mathcal{T}_{r\to t}(\mathbf{x})}{\partial t} = \nabla\mathbf{u}(\mathbf{\Psi}_{r\to t}(\mathbf{x}), t)\mathcal{T}_{r\to t}(\mathbf{x}), & \mathcal{T}_{r\to r}(\mathbf{x}) = \mathbf{I}. \end{cases} \tag{7}$$

Since Equation 7 does not contain advection terms, it can be accurately computed using the RK4 method, similar to $\mathbf{\Phi}_{s\to t}$ and $\mathcal{F}_{s\to t}$. With the precise computation of $\mathbf{\Psi}_{r\to s}$ and $\mathcal{T}_{r\to s}$, [Deng et al. 2023] achieves better results compared to directly using the semi-Lagrangian method with low-accuracy kernel-based interpolation to compute $\mathbf{\Psi}_{r\to s}$ and $\mathcal{T}_{r\to s}$ [Nabizadeh et al. 2022; Qu et al. 2019]. We refer to this approach as the Eulerian Flow Maps (EFM) method.

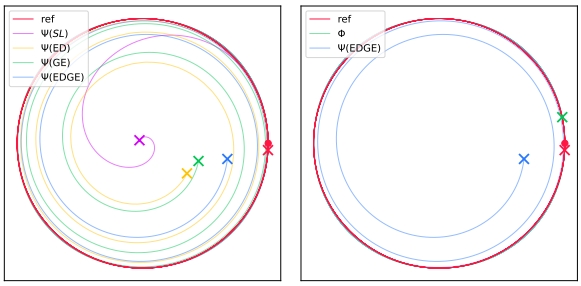



Fig. 3. A 2D intuition experiment comparing flow map advection schemes. The flow features a clockwise vorticity field centered at $(0.5, 0.5)$. The red point marks the common starting point, while crosses mark endpoints. The red line (*ref*) is the analytical advection position. $\mathbf{\Phi}$ and $\mathbf{\Psi}$ correspond to the forward and backward flow maps respectively. *ED*, *GE*, and *EDGE* are our proposed schemes, while *SL* is the classic semi-Lagrangian scheme. Notably, $\mathbf{\Phi}$ also represents the inverse backward flow map advection as in [Deng et al. 2023]. The closer a cross lies to the red reference cross, the more accurate the corresponding advection scheme is.

However, in the EFM method, for each time step $r > s$, a full evolution of $\mathbf{\Psi}_{r\to t}$ and $\mathcal{T}_{r\to t}$ from time $r$ to time $s$ must be performed, as shown in Figure 2. Specifically, at the $(k+1)$-th substep, the velocity field of the previous $k$ substeps must be stored and used to evolve flow maps through $k$ iterations from the current substep to the initial time. This results in a total computational complexity of $O(k^2)$ and requiring the storage of $k$ velocity buffers for calculating $\mathbf{\Psi}_{r\to s}$ and $\mathcal{T}_{r\to s}$ of the first $k+1$ substeps.

In the following discussions, our primary focus is on optimizing backward map memory, which dominates the memory cost in flow-map methods. Meanwhile, tracking overall simulation memory and total memory usage remains meaningful for evaluating practical performance. Here, overall simulation memory refers to the necessary memory required for running the simulation, excluding auxiliary buffers, visualization data, and and runtime overhead introduced by the implementation framework. To provide a comprehensive evaluation, we report all three metrics in Table 1.

## 4 Flow Map with Hermite Interpolation

EFM accurately computes $\mathbf{\Psi}_{r\to s}$ and $\mathcal{T}_{r\to s}$ by solving the evolution Equation 7 of the inverse process with no advection terms. Despite high accuracy, it results in a computational cost of $O(k^2)$ in time and $O(k)$ in space. Unlike EFM, we continue to compute the evolution Equation 1 and Equation 2 of $\mathbf{\Psi}_{t\to s}$ and $\mathcal{T}_{t\to s}$ with advection terms and improve the accuracy of computing advection with semi-Lagrangian method directly.

The error in the semi-Lagrangian method mainly lies in the accumulation of interpolation errors at each advection step. At each time $r$, $\mathbf{\Psi}_{r\to s}$ is obtained by interpolating from $\mathbf{\Psi}_{r'\to s}$ of the last time step $r'$, i.e. $\mathbf{\Psi}_{r\to s}(\mathbf{x}) = I[\mathbf{\Psi}_{r'\to s}](\mathbf{\Psi}_{r\to r'}(\mathbf{x}))$, where $I[f](\mathbf{x})$ denotes the interpolation function that evaluates field $f$ at position $\mathbf{x}$. Since

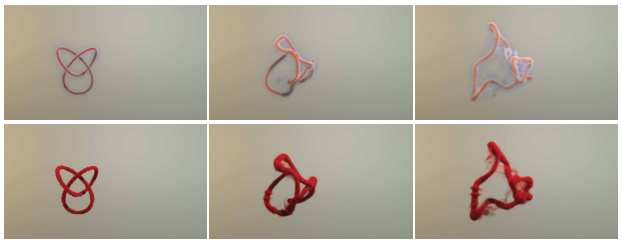


Fig. 4. The evolution of a trefoil.

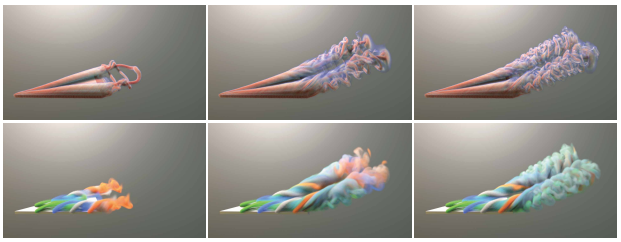


Fig. 5. Vorticity (top row) and smoke (bottom row) visualizations of an airflow passing through a delta wing.

$\Psi_{r'\to s}$ is itself obtained by interpolating from an even earlier time step $r''$, and so forth, this repeated interpolation process leads to the accumulation of errors from all previous steps when computing $\Psi_{r\to s}(\mathbf{x})$. Using a low-accuracy kernel-based interpolation function $I$ exacerbates error accumulation over multiple time steps, resulting in significant inaccuracies in $\Psi_{r\to s}(\mathbf{x})$. To mitigate this issue, we incorporate high-accuracy Hermite interpolation $H[f](\mathbf{x})$ into the semi-Lagrangian computation of $\Psi_{r\to s}$ and $\mathcal{T}_{t\to s}$, which reduces interpolation errors and, ultimately, reduces the accumulated errors when computing $\Psi_{r\to s}(\mathbf{x})$. As shown in Figure 3, we conduct a 2D intuition experiment for all different strategies to present the issue and solution to the flow map evolution.

In subsection 4.1, we introduce Hermite interpolation. In subsection 4.2, we incorporate Hermite interpolation to compute evolution of $\Psi_{t\to s}(x)$ and $\mathcal{T}_{t\to s}(\mathbf{x})$ using the semi-Lagrangian method. In our discussion, we take 3D case as an example, and the 2D approach can be derived similarly.

## 4.1 Hermite Interpolation

Consider a grid $G$ with spacing $\Delta x$, with the set of grid points represented as $\mathbb{G}$. For any field $v$, we use the subscript $_g$ to denote its value at a grid point $g \in \mathbb{G}$, i.e., $v_g = v(\mathbf{x}_g)$, where $\mathbf{x}_g$ represents the position of the grid point $g$.

For a scalar field $f(\mathbf{x})$, Hermite interpolation utilizes the values $(f_g)$ stored on the grid and its partial derivatives of orders up to one in each direction $f_g^{(d_x d_y d_z)}$, where $d_x, d_y, d_z = 0, 1$ to perform interpolation:

$$H[f; f_g, f_g^{(d_x d_y d_z)}](\mathbf{x}) = \sum_{g\in N(\mathbf{x})} \sum_{i,j,k=0}^{1} [f_g^{(ijk)} h^{(ijk)} \left(\frac{\mathbf{x}-\mathbf{x}_g}{\Delta x}\right) \Delta x^{i+j+k}], \tag{8}$$

where the terms $f_g, f_g^{(d_x d_y d_z)}$ after the semicolon represent the information required for interpolation, $N(\mathbf{x})$ represents the 8 vertex points of the cubical cell containing position $\mathbf{x}$, $\Delta x$ represents the grid edge length, and $f_g^{(d_x d_y d_z)} = \frac{\partial^{d_x+d_y+d_z} f}{\partial x^{d_x} \partial y^{d_y} \partial z^{d_z}}\Big|_{\mathbf{x}=\mathbf{x}_g}$. For example, $f_g^{(011)} = \frac{\partial^2 f}{\partial y \partial z}\Big|_{\mathbf{x}=\mathbf{x}_g}$, and $f_g^{(000)} = f_g$. The expression for the interpolation kernel $h^{(ijk)}(\mathbf{x})$ is given by:

$$\begin{aligned} h^{(ijk)}(\mathbf{x}) &= h^{(i)}(x_1)h^{(j)}(x_2)h^{(k)}(x_3), \mathbf{x} = (x_1, x_2, x_3), \\ h^{(0)}(\theta) &= \begin{cases} 2|\theta|^3 - 3|\theta|^2 + 1, |\theta| < 1, \\ 0, \text{otherwise}, \end{cases} \\ h^{(1)}(\theta) &= \begin{cases} \theta^3 - 2\theta^2 + \theta, |\theta| < 1, \\ 0, \text{otherwise}. \end{cases} \end{aligned} \tag{9}$$

The gradient of the Hermite interpolant $\nabla H[f; f_g, f_g^{(d_x d_y d_z)}](\mathbf{x})$ can serve as an interpolation of $\nabla f(\mathbf{x})$, since Hermite interpolation satisfies the property

$$\nabla H[f; f_g, f_g^{(d_x d_y d_z)}](\mathbf{x}_g) = (f_g^{(100)}, f_g^{(010)}, f_g^{(001)}). \tag{10}$$

Let $\tilde{H}[\nabla f; f_g, f_g^{(d_x d_y d_z)}](\mathbf{x}) = \nabla H[f; f_g, f_g^{(d_x d_y d_z)}](\mathbf{x})$ denote the interpolation of $\nabla f$ induced by $H[f; f_g, f_g^{(d_x d_y d_z)}](\mathbf{x})$.

For a vector field $\mathbf{v}(\mathbf{x})$, $H[\mathbf{v}; \mathbf{v}_g, \mathbf{v}_g^{(d_x d_y d_z)}](\mathbf{x})$ represents performing Hermite interpolation on each component $v_i$ of $\mathbf{v}(x) = (v_1(x), v_2(x), \dots)$ individually, where $\mathbf{v}_g = (v_{1,g}, v_{2,g}, \dots)$, $\mathbf{v}_g^{(d_x d_y d_z)} = (v_{1,g}^{(d_x d_y d_z)}, v_{2,g}^{(d_x d_y d_z)}, \dots)$, and $v_{i,g}$ represents the value of the component $v_i$ at the grid point $g$.

## 4.2 Hermite Interpolation for Flow Map Evolution

Now we incorporate Hermite interpolation with flow map evolution. At $r = r' + \Delta t$, we already have the values of $\Psi_{r'\to s,g}$ and $\Psi_{r'\to s,g}^{(ijk)}$ at the grid points . Using this information, we aim to compute $\Psi_{t\to s}$ at the grid points $\mathbb{G}$ using the semi-Lagrangian method to evaluate Equation 1. The detailed procedure is as follows:

for any grid point $g$,

(1) Calculate $\mathbf{x}'_g$ by integrating

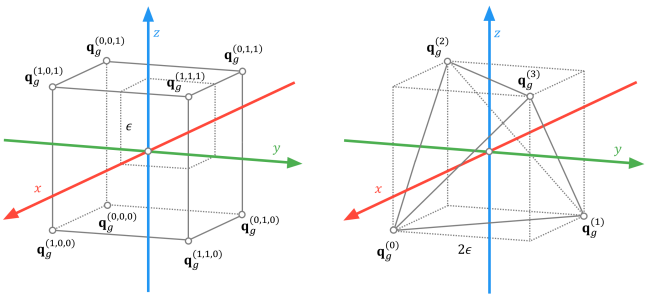
$$\frac{d\mathbf{x}}{dt} = \mathbf{u}_{r'}(\mathbf{x}) \tag{11}$$

from time $r$ to $r'$ with initial value $\mathbf{x} = \mathbf{x}_g$ by RK4.

(2) Compute $\Psi_{r\to s}(\mathbf{x}_g) = H[\Psi_{r'\to s}; \Psi_{r'\to s,g}, \Psi_{r'\to s,g}^{(i,j,k)}](\mathbf{x}'_g)$.

(3) Prepare $\Psi_{r\to s,g}^{(i,j,k)}$ for the Hermite interpolation at the next time step.

Here (1) is the backtrace process of the semi-Lagrangian method. (3) is necessary because the information of $\Psi_{r\to s,g}^{(i,j,k)}$ is required to form




(a) Eight-Point Epsilon Difference (b) Four-Point Epsilon Difference

Fig. 6. Epsilon difference element configuration. (a) Relative positions of $\mathbf{q}_g^{(l,m,n)}$, $l, m, n = 0, 1$ and $\mathbf{x}_g$ in eight-point epsilon difference. (b) Relative positions of $\mathbf{q}_g^{(l)}$, $l = 0, 1, 2, 3$ and $\mathbf{x}_g$ in four-point epsilon difference.

interpolant $H[\Psi_{r\to s}; \Psi_{r\to s,g}, \Psi_{r\to s,g}^{(i,j,k)}]$ for Hermite interpolation next time step. The remaining problem is to compute $\Psi_{r\to s,g}^{(i,j,k)}$, $i, j, k = 0, 1$.

*4.2.1 Gradient Evolution (GE).* Li et al. [2023] proposed a method for evolving flow maps and their derivatives. The first-order partial derivatives of $\Psi_{r\to s}$, i.e., $(\Psi_{r\to s}^{(1,0,0)}, \Psi_{r\to s}^{(0,1,0)}, \Psi_{r\to s}^{(0,0,1)})^\top$ actually form the Jacobian matrix $\mathcal{T}_{r\to s}$ of $\Psi_{r\to s}$, and based on the fact that $\Psi_{r\to s}(\mathbf{x}) = \Psi_{r'\to s}(\Psi_{r\to r'}(\mathbf{x}_g))$, $\mathcal{T}_{r\to s,g}$ can be converted by the chain rule as

$$\begin{aligned}\mathcal{T}_{r\to s,g} &= \frac{\partial \Psi_{r'\to s}(\mathbf{y})}{\partial \mathbf{y}}|_{\mathbf{y}=\Psi_{r\to r'}(\mathbf{x}_g)} \frac{\partial \Psi_{r\to r'}(\mathbf{x})}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_g} \\ &= \mathcal{T}_{r'\to s}(\Psi_{r\to r'}(\mathbf{x}_g))\mathcal{T}_{r\to r',g},\end{aligned} \tag{12}$$

where $\mathcal{T}_{r\to r',g}$ can be calculated by integrating $\frac{d\mathcal{T}}{dt} = \nabla\mathbf{u}(\mathbf{x})\mathcal{T}$ together with step (1). At step (3), first-order derivative $\nabla\Psi_{r\to s,g} = (\Psi_{r\to s,g}^{(1,0,0)}, \Psi_{r\to s,g}^{(0,1,0)}, \Psi_{r\to s,g}^{(0,0,1)})$ can be computed by

$$\nabla\Psi_{r\to s,g} = \tilde{H}[\nabla\Psi_{r'\to s}; \Psi_{r'\to s,g}, \Psi_{r'\to s,g}^{(d_x d_y d_z)}](\mathbf{x}_g)\mathcal{T}_{r\to r',g}. \tag{13}$$

However, since the linear chain rule holds only for first-order derivatives, there is no simple computation method for higher-order mixed derivatives of $\Psi_{t\to s}$. But according to [Li et al. 2023], these higher-order mixed derivatives can be computed using finite differences of the first-order derivatives $\Psi_{t\to s,g}^{(i,j,k)}$ on a grid without significantly compromising accuracy. For instance, $\Psi_{t\to s,g}^{(1,1,0)} = \frac{\Psi_{t\to s,gy+}^{(1,0,0)} - \Psi_{t\to s,gy-}^{(1,0,0)}}{\Delta x}$ where $g_{y\pm}$ are the grid points with position $x_g + (0, \pm\Delta x, 0)$. We refer to this method as the **Gradient Evolution (GE) Method**.

**Method 4.2.1: Gradient Evolution Method**

(1) Calculate $\mathbf{x}'_g$ and $\mathcal{T}_{r\to r',g}$ by integrating $\frac{d\mathbf{x}}{dt} = \mathbf{u}_{r'}(\mathbf{x})$ and $\frac{d\mathcal{T}}{dt} = \nabla\mathbf{u}(\mathbf{x})\mathcal{T}$ together from time $r$ to $r'$ with initial value $\mathbf{x} = \mathbf{x}_g$ and $\mathcal{T} = \mathbf{I}$ by RK4.
(2) Compute $\Psi_{r\to s}(\mathbf{x}_g) = H[\Psi_{r'\to s}; \Psi_{r'\to s,g}, \Psi_{r'\to s,g}^{(i,j,k)}](\mathbf{x}'_g)$.
(3) Calculate first-order derivative of $\Psi_{r\to s}(\mathbf{x}_g)$ by Equation 13 and higher-order mixed derivatives by finite difference on the grid $G$.

*4.2.2 Epsilon Difference (ED).* Another method for computing $\Psi_{r\to s,g}^{(i,j,k)}$, where $i, j, k = 0, 1$, is known as the epsilon difference method [Chidyagwai et al. 2011; Seibold et al. 2011]. This method utilizes the values of $\Psi_{r\to s}$ at several virtual particles surrounding the grid point $g$ to compute $\Psi_{r\to s,g}^{(i,j,k)}$. For any grid point $g$, as shown in Figure 6a, we consider the eight neighboring points $\mathbf{q}_g^{(i,j,k)}$ around $\mathbf{x}_g$:

$$\mathbf{q}_g^{(l,m,n)} = \mathbf{x}_g - ((-1)^l\epsilon\Delta x, (-1)^m\epsilon\Delta x, (-1)^n\epsilon\Delta x), \tag{14}$$

where $\epsilon$ is a tunable parameter that determines the accuracy of the spatial derivative computation using the epsilon difference method. At time $r$, the values of $\Psi_{r\to s}$ at $\mathbf{q}_g^{(l,m,n)}$ can be obtained using the same method as for $\Psi_{r\to s}(x_g)$ by interpolating $\Psi_{r'\to s}$ at $\mathbf{q'}_g^{(l,m,n)}$, where $\mathbf{q'}_g^{(l,m,n)}$ is calculated by integrating Equation 11 with initial value $\mathbf{q}_g^{(l,m,n)}$. By performing finite differences at the fine grid formed by eight points $\mathbf{q}_g^{(l,m,n)}$, $\Psi_{r\to s}^{(i,j,k)}(\mathbf{x}_g)$ can be computed as:

$$\Psi_{r\to s}^{(i,j,k)}(\mathbf{x}_g) = \frac{\sum_{l,m,n=0}^{1}(-1)^{S(i,j,k,l,m,n)}\Psi_{r\to s}(\mathbf{q}_g^{(l,m,n)})}{8(\epsilon\Delta x)^{i+j+k}}. \tag{15}$$

Here, $S(i, j, k, l, m, n) = i + l + j + m + k + n$ is the function that helps to compute the sign of each term. In this computation, the error depends on $\epsilon\Delta x$, compared to the error of directly using finite differences on the grid $G$, which depends on $\Delta x$. Since $\epsilon$ can be chosen to be a small value, the error in Equation 15 is much better than that of direct finite differences. This method is called **Eight-Point Epsilon Difference (ED8) Method**.

**Method 4.2.2: Eight-Point Epsilon Difference Method**

(1) Calculate $\mathbf{x}'_g$ and $\mathbf{q'}_g^{(l,m,n)}$ by integrating $\frac{d\mathbf{x}}{dt} = \mathbf{u}_{r'}(\mathbf{x})$ from time $r$ to $r'$ with initial value $\mathbf{x} = \mathbf{x}_g$ and $\mathbf{x} = \mathbf{q}_g^{(l,m,n)}$ respectively by RK4.
(2) Compute $\Psi_{r\to s}(\mathbf{x}_g) = H[\Psi_{r'\to s}; \Psi_{r'\to s,g}, \Psi_{r'\to s,g}^{(i,j,k)}](\mathbf{x}'_g)$.
(3) Compute $\Psi_{r\to s}(\mathbf{q}_g^{(l,m,n)})$ by Hermite interpolation similar to $\Psi_{r\to s}(\mathbf{x}_g)$ and then compute $\Psi_{r\to s}^{(ijk)}(\mathbf{x}_g)$ by Equation 15.

In ED8, a cubic element consisting of eight points is used to compute the spatial derivatives of $\Psi_{t\to s}$ at $\mathbf{x}_g$. Actually, other 3D elements can also be utilized for this computation. We propose a method based on a tetrahedral element with four points around grid point $g$ to compute $\Psi_{t\to s}$, achieving computational accuracy comparable to the ED8 with faster computation speed. For any grid point $g$, as shown in Figure 6b, we consider the four neighboring points $\mathbf{q}_g^{(l)}$, $l = 0, 1, 2, 3$ around $x_g$, with values $\mathbf{q}_g^{(0)} = \mathbf{q}_g^{(1,0,0)}$, $\mathbf{q}_g^{(1)} = \mathbf{q}_g^{(0,1,0)}$, $\mathbf{q}_g^{(2)} = \mathbf{q}_g^{(0,0,1)}$ and $\mathbf{q}_g^{(3)} = \mathbf{q}_g^{(1,1,1)}$, and the first-order derivatives can be calculated as

$$\Psi_{r\to s}^{(i,j,k)} = \frac{\Psi_{r\to s}(\mathbf{q}_g^{(3)}) + \sum_{l=0}^{2}(-1)^{S(i,j,k,l)}\Psi_{r\to s}(\mathbf{q}_g^{(l)})}{2\epsilon\Delta x}, i + j + k = 1, \tag{16}$$

where the sign function $S(i, j, k, l) = 0$ only when $(i, l) = (1, 0)$, $(j, l) = (1, 1)$, $(k, l) = (1, 2)$, otherwise $S(i, j, k, l) = 1$. The four points $\mathbf{q}_g^{(i)}$ cannot be used to compute higher-order mixed partial derivatives directly, but similar to the GE method, these derivatives can be obtained by applying finite-difference calculations to the first-order derivatives on the grid. We refer to this method as the **Four-Point Epsilon Difference (ED4) Method**.

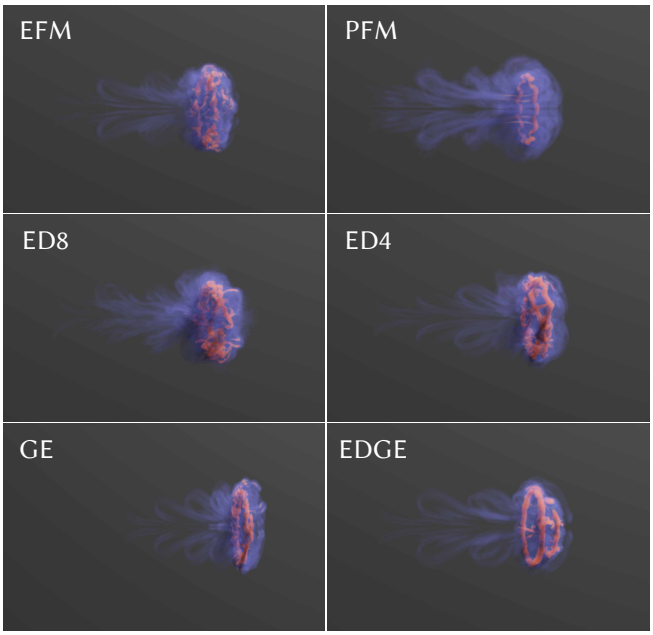



Fig. 7. Comparison of 3D leapfrogging vortices. Our methods are able to achieve the same performance or even outperform previous EFM and PFM methods.

**Method 4.2.3: Four-Point Epsilon Difference Method**

(1) Calculate $\mathbf{x}'_g$ and $\mathbf{q}'^{(l)}_g$ by integrating $\frac{d\mathbf{x}}{dt} = \mathbf{u}_{r'}(\mathbf{x})$ from time $r$ to $r'$ with initial value $\mathbf{x} = \mathbf{x}_g$ and $\mathbf{x} = \mathbf{q}^{(l)}_g$ respectively by RK4.
(2) Compute $\Psi_{r\to s}(\mathbf{x}_g) = H[\Psi_{r'\to s}; \Psi_{r'\to s,g}, \Psi^{(i,j,k)}_{r'\to s,g}](\mathbf{x}'_g)$.
(3) Compute $\Psi_{r\to s}(\mathbf{q}^{(l)}_g)$ by Hermite interpolaition similar to $\Psi_{r\to s}(\mathbf{x}_g)$ and then compute first-order derivatives by Equation 16. High-order mixed derivatives are computed by finite difference on the grid $G$.

*4.2.3 Epsilon Difference Gradient Evolution (EDGE).* In our experiments, we found that the Gradient Evolution method and the Epsilon Difference method cannot achieve the same level of accuracy as the previous EFM method, as shown in Figure 3. Analyzing these two methods, we observe that (1) while the GE method accurately computes the first-order derivatives of $\Psi_{r\to s}$, its high-order mixed derivatives calculated by finite difference approximations on the coarse grid $\mathbb{G}$ are inaccurate. (2) the ED method, on the other hand, can achieve highly accurate computations by using small $\epsilon$ to form a very fine local grid around each grid point $g$, but the value of $\epsilon$ is constrained by machine precision, and excessively small $\epsilon$ can lead to computational instability. With a stable choice of $\epsilon$, ED still can not match the accuracy of EFM. To address this issue, we propose a new method, the **Epsilon-Difference Gradient Evolution (EDGE) method**, based on both GE and ED methods. This approach achieves, and even surpasses, the accuracy of the EFM method (see Figure 7 for validation).

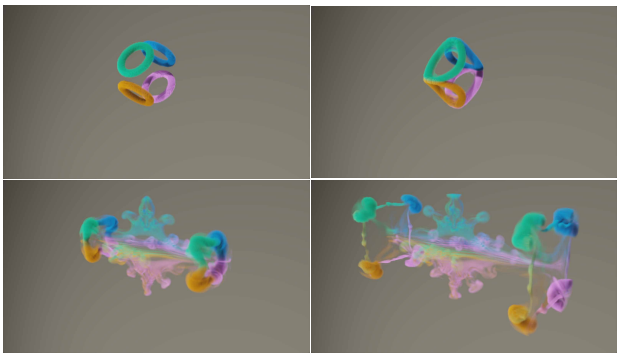


Fig. 8. The collision and reconnection of four vortices.

EDGE is based on the fact that the calculation with a fine $\epsilon$-sized grid in ED method is more accurate compared to direct finite difference calculation on the grid $\mathbb{G}$. To improve the GE method, we replace the finite difference used for computing high-order mixed derivatives in GE with the epsilon difference calculation. Since, in general, the error in high-order derivatives has a lower impact on the overall accuracy compared to first-order derivatives, we can use a relatively large and stable $\epsilon$ for computing high-order mixed derivatives with the epsilon difference method while still maintaining high overall accuracy. Taking EDGE with four-point epsilon difference as an example, the same method as in Equation 13 for evolving gradients is applied to obtain $\nabla\Psi_{r\to s}(\mathbf{q}^{(l)}_g) = (\Psi^{(1,0,0)}_{r\to s}(\mathbf{q}^{(l)}_g), \Psi^{(0,1,0)}_{r\to s}(\mathbf{q}^{(l)}_g), \Psi^{(0,0,1)}_{r\to s}(\mathbf{q}^{(l)}_g))$. Then, second-order mixed derivatives can be computed using the four-point epsilon difference similar to Equation 16. For example, $\nabla\Psi^{(1,1,0)}_{r\to s,g}$ can be calculated as:

$$\nabla\Psi^{(1,1,0)}_{r\to s,g} = \frac{\Psi^{(0,1,0)}_{r\to s}(\mathbf{q}^{(3)}_g) + \sum_{l=0}^{2}(-1)^{S(1,0,0,l)}\Psi^{(0,1,0)}_{r\to s}(\mathbf{q}^{(l)}_g)}{2\epsilon\Delta x}. \quad (17)$$

Here, we present the EDGE method with the four-point epsilon difference calculation, which can be similarly extended to EDGE with an eight-point epsilon difference calculation.

**Method 4.2.4: Epsilon-Difference Gradient Evolution Method**

(1) From time $r$ to $r'$ using RK4, integrate $\frac{d\mathbf{x}}{dt} = \mathbf{u}_{r'}(\mathbf{x})$ for $\mathbf{x}'_g$ and $\mathbf{q}'^{(l)}_g$ with initial value $\mathbf{x} = \mathbf{x}_g$ and $\mathbf{q}^{(l)}_g$ respectively and integrate $\frac{d\mathcal{T}}{dt} = \nabla\mathbf{u}(\mathbf{x})\mathcal{T}$ for $\mathcal{T}_{r\to r',g}$ and $\mathcal{T}_{r\to r'}(\mathbf{q}^{(l)}_g)$ with initial value $\mathcal{T} = \mathbf{I}$ simultaneously.
(2) Compute $\Psi_{r\to s}(\mathbf{x}_g) = H[\Psi_{r'\to s}; \Psi_{r'\to s,g}, \Psi^{(i,j,k)}_{r'\to s,g}](\mathbf{x}'_g)$.
(3) Calculate first-order derivative of $\Psi_{r\to s}(\mathbf{x}_g)$ by Equation 13, second-order mixed derivatives by Equation 17 using four-point epsilon difference and third-order mixed derivatives by finite difference on the grid $G$.

## 4.3 Summary

In summary, we present three methods for evolving the flow map based on Hermite interpolation: Gradient Evolution (subsubsection 4.2.1), Epsilon Difference (subsubsection 4.2.2), and Epsilon-Difference Gradient Evolution (subsubsection 4.2.3). Unlike EFM,

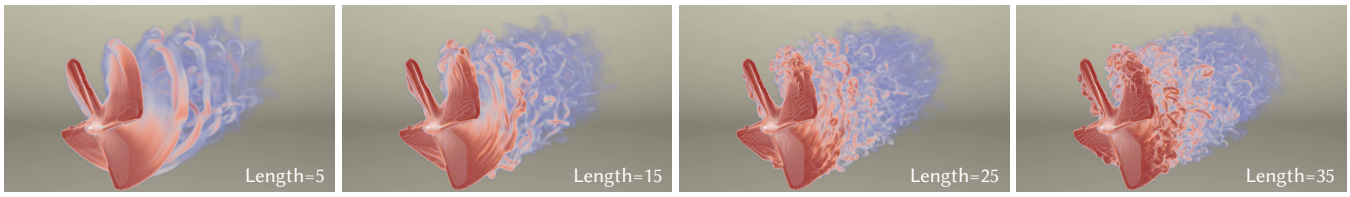



Fig. 9. Propeller with varying flow map lengths. Richer vortex details emerge as the flow map length increases.

none of these methods require storing a large velocity buffer. The intuition experiment (Figure 3) demonstrates their accuracy compared to the classic semi-Lagrangian method. The newly proposed EDGE scheme achieves accuracy comparable to the original EFM method. As shown in Table 1 and Figure 7, they achieve vorticity preservation comparable to EFM while offering superior speed and memory efficiency. Each of these three methods has its advantages and disadvantages: GE is the fastest, ED has the lowest memory consumption, and EDGE exhibits the strongest vorticity preservation, even surpassing EFM.

## 5 Implementation and Experiments

To validate the usage and effectiveness of our method, in this section, we first compare the time and space complexity of each algorithm and validate our findings with numerical experiments, and then we present several validation results for our proposed EDGE method. These results show that our method not only can achieve results comparable to previous state-of-the-art methods, but can also significantly reduce memory usage and accelerate computation, as in Table 1 and Figure 1.

### 5.1 Implementation Details

All simulations were conducted on an NVIDIA RTX−4090 GPU and implemented with Taichi [Hu et al. 2019].

*Discretization.* We adopt the standard MAC grid [Harlow and Welch 1965] for velocity discretization. Unlike EFM, which stores backward flow maps and their derivatives at face centers, we store these components at cell centers and compute face values using Hermite interpolation as needed.

*Kernel Fusion.* To reduce memory cost, we applied kernel fusion to our methods. We observed that some of the variables can be calculated on the fly, which means they do not need to be saved as permanent memory blocks. Instead, we compute them as temporary local variables, utilizing the GPU local cached memory.

*Generality Across Flow Map Frameworks.* In addition to EFM, to demonstrate the seamless integration of our approach with existing flow map frameworks, we incorporate our method into recently proposed particle-laden flows method [Li et al. 2024b] and solid-fluid interactions method [Chen et al. 2024]. In our implementation, we replace the Particle Flow Maps (PFM) advection mechanism [Zhou et al. 2024] for velocity fields with our proposed technique, while keeping the rest of these algorithms, such as the computation of dissipative forces and fluid-solid coupling, unchanged.

*Complexity Analysis.* For a flow map simulation with range $n$, EFM and our Hermite-interpolated semi-Lagrangian method exhibit different complexities in both time and space for backward map evolution. As illustrated in Figure 2b, the evolution of backward flow maps in EFM is designed to trace back to the reference frame. During an $n$-step simulation, the backtracing process is executed $O(n^2)$ times. Additionally, it requires storing the velocity history, incurring an $O(n)$ backward map memory cost. In contrast, our method evolves the backward flow maps continuously, backtracing and interpolating them only with the previous frame. This results in a significant reduction in complexity, achieving $O(n)$ time and $O(1)$ backward map memory consumption. If we define storing a floating-point number at each grid point in the domain as one memory unit, then the original EFM implementation requires $18+3n$ units for backward map evolution. In contrast, our proposed ED4 and EDGE methods reduce this memory cost to 24 and 42 units, respectively. Given that our examples use flow maps of length up to 80, our methods achieve up to a 90% reduction in backward map memory usage.

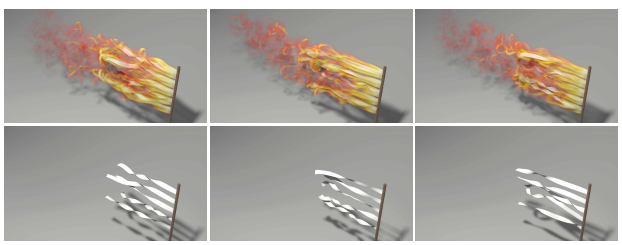


Fig. 10. Flag stripes affected by incoming flow.

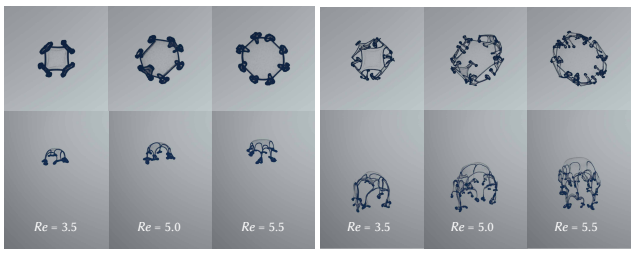



Fig. 11. Ink torus breakup at Reynolds numbers 3.5, 5.0, and 5.5. Left and right column groups show early and late frames. Top and bottom rows show top and side views. Branched blobs increase with Reynolds number.

Table 1. Detailed resource profiling for flow map methods.

| Name | Resolution | Flow-map Length | Method | Backward Map Mem (GB) | Overall Sim Mem (GB)[1] | Total Mem (GB) | Advection Time (sec/substep)[2] | Avg. Time (sec/substep) |
|---|---|---|---|---|---|---|---|---|
| Leapfrogging Vortices | 384×128×128 | 20 | EFM | 1.839 | 2.694 | 4.094 | 0.123 | 0.243 |
| | | | GE | 1.268 | 2.123 | 3.521 | 0.0455 | 0.163 |
| | | | ED8 | 1.549 | 2.404 | 3.767 | 0.0588 | 0.175 |
| | | | ED4 | 0.563 | 1.417 | 2.763 | 0.0856 | 0.201 |
| | | | EDGE | 0.984 | 1.839 | 3.207 | 0.0803 | 0.196 |
| | | | PFM | 3.799 | 7.791 | 11.992 | 0.432 | 0.568 |
| | | | NFM | 6.916 | 7.771 | 9.122 | 4.737 | 4.872 |
| Four Vortices Collision | 128×128×256 | 20 | EFM | 1.227 | 1.797 | 2.896 | 0.0907 | 0.191 |
| | | | ED4 | 0.375 | 0.945 | 2.052 | 0.0589 | 0.153 |
| | | | EDGE | 0.656 | 1.226 | 2.330 | 0.0583 | 0.155 |
| Dye Drift | 256×512×256 | 80 | EFM[3] | 32.355 | 37.894 | 42.737 | 4.766 | 4.935 |
| | | | ED4 | 3.000 | 8.539 | 13.365 | 0.541 | 0.712 |
| | | | EDGE | 5.250 | 10.789 | 15.610 | 0.496 | 0.666 |

[1] *Overall Sim Mem* refers to the necessary memory required for the simulation, excluding auxiliary buffers, visualization data, and Taichi runtime overhead.

[2] *Advection Time* refers to the time excluding Poisson equation solving, primarily spent on advection.

[3] The velocity buffer exceeds the available GPU memory and is therefore stored in CPU memory (30.098 GB), with data transferred between CPU and GPU on demand.

## 5.2 Experimental Results

In this subsection, we begin by conducting experiments and profiling memory and timing data to validate our proposed methods. We then present several examples to demonstrate the generality and effectiveness of the EDGE method.

*Validation.* To validate our analysis, we conducted simulations to evaluate the resource consumption and performance of various flow map methods, including 3D leapfrogging vortex scenes [Deng et al. 2023] within an extended domain. The resource usage statistics are summarized in Table 1. Without caching velocity buffers, our method, based on Hermite interpolation, significantly reduces memory usage. Although Hermite interpolation introduces additional computational overhead, our one-step evolution outperforms EFM in terms of speed, as EFM relies on multi-step backtraces. The visual results of the leapfrogging vortices simulation are shown in Figure 7. While our GE and ED methods achieve performance comparable to the original EFM, the EDGE method outperforms EFM by nearly one leap. In the dye drift example shown in Figure 1, ED4 and EDGE effectively capture fine-scale fluid structures on par with EFM. These results highlight the superior vorticity conservation of our methods, matching or even exceeding the performance of the original EFM.

*Flow-Map Length Experiment.* Since the EDGE method and EFM differ in memory complexity—$O(1)$ vs. $O(n)$—our approach becomes more advantageous for longer flow maps. In this experiment, we simulate a rotating propeller to validate the importance of extended flow maps. As illustrated in Figure 9, turbulence intensity increases with the number of flow map reinitializations due to interactions between incoming flow and solid boundaries, enhancing simulation detail. Since our method maintains a constant computational cost for extending flow map length, it enables improvements without additional computational overhead.

*Four Vortices Collision.* Figure 8 illustrates an experiment inspired by [Matsuzawa et al. 2022], where four vortex rings, initially arranged in a square configuration, collide within the $yz$-plane. Each ring has a major radius of 0.15 and a minor radius of 0.024. As the rings interact, they merge to form two four-pointed star-shaped vortices before eventually colliding with the boundaries.

*Trefoil.* Figure 4 uses the same configuration from [Matsuzawa et al. 2022] to replicate the trefoil knot. Our method is able to retain an accurate structure of vortices, yielding two separate vortex rings in the end.

*Delta Wing.* Figure 5 illustrates the "vortex lift" phenomenon generated by a delta wing with a 75° sweep angle at a 20° angle of attack. This behavior is in agreement with the experimental observations reported by [Délery 2003].

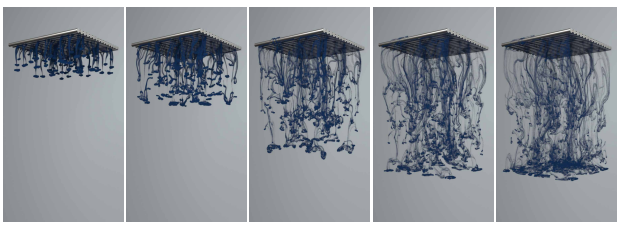


Fig. 12. Ink drops passing through a porous obstacle.

*Two-Way Coupled Thin Film.* To demonstrate the effectiveness of solid–fluid interaction in [Chen et al. 2024] using the EDGE method, several flag strips are subjected to an incoming flow. The motion of the flags and the flow field are tightly coupled, resulting in rich fluid–structure dynamics and detailed vortex structures as shown in Figure 10.

*Ink Torus Breakup.* The EDGE scheme is also applicable to particle-laden flow simulations [Li et al. 2024b]. Driven by viscous forces, an ink torus breaks into multiple blobs, which then deform into smaller tori and undergo further fragmentation, forming a cascading cycle of continuous deformation and breakup. As the Reynolds number increases ($Re$ = 3.5, 5.0, 5.5), the number of resulting blobs grows from 4 to 8, as shown in Figure 11.

*Ink Passing Porous Obstacles.* Figure 12 illustrates nine ink drops descending through the gaps between cylinders, breaking apart into numerous smaller falling droplets.

*Dye Drift.* In Figure 1, a small pinch of dye is released onto the liquid surface, allowing it to disperse freely. The image reveals rich flow details, demonstrating the method's superior ability to preserve vorticity. In this example, the combination of high resolution and long flow maps prevents EFM from storing velocity buffers on the GPU. As a result, buffers are stored in CPU memory instead, and the additional data transfer further slows down the simulation. This highlights the limitations of EFM in handling large-scale simulation scenarios.

*Comparison to More Flow Map Methods.* In the original flow map simulation work [Deng et al. 2023], the velocity buffer is designed to be compressed using neural networks. However, Zhou et al. [2024] found that this approach introduces a significant training time cost and additional overhead from extensive data transfers. Instead of relying on neural networks, our method adopts a different strategy by leveraging Hermite interpolation to eliminate the memory overhead associated with the velocity buffer, thereby improving both time and memory efficiency. Regarding the PFM proposed in [Zhou et al. 2024], which also has $O(1)$ memory complexity, we argue that it is less efficient in terms of memory usage. This is because each particle must carry flow map information, and, on average, each grid cell contains 8-16 particles in a 3D simulation. Our argument is supported by detailed memory and timing data, as shown in Table 1.

## 6 Limitations and Future Work

We proposed the Epsilon Difference Gradient Evolution method to enable buffer-free flow-map simulations at O(1) memory consumption. One of the primary limitations of our current scheme is its reliance on Hermite interpolation, which requires additional memory storage compared to other potential alternatives. Our goal is to further reduce memory consumption and computational overhead by investigating alternative interpolation schemes on a Cartesian grid, with the aim of lowering the constant factor in the $O(1)$ complexity and achieving performance comparable to standard grid-based solvers. Furthermore, we plan to explore GPU implementations and performance engineering techniques to optimize the EDGE scheme for GPU architectures. Finally, we envision extending our EDGE scheme to support adaptive grid structures, such as octrees, which presents an open challenge in addressing interpolation across cells with differing resolutions.

## Acknowledgments

We thank the anonymous reviewers for their constructive comments. We acknowledge NSF IIS #2433322, ECCS #2318814, CAREER #2420319, IIS #2433307, OISE #2433313, and CNS #1919647 for funding support. We credit the Houdini education license for video animations.

## References

Ryoichi Ando, Nils Thuerey, and Chris Wojtan. 2015. A stream function solver for liquid simulations. *ACM Trans. Graph.* 34, 4, Article 53 (July 2015), 9 pages. doi:10.1145/2766935

Alexis Angelidis. 2017. Multi-scale vorticle fluids. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–12.

Alexis Angelidis and Fabrice Neyret. 2005. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '05)*. Association for Computing Machinery, New York, NY, USA, 87–96. doi:10.1145/1073368.1073380

Tyson Brochu, Todd Keeler, and Robert Bridson. 2012. Linear-time smoke animation with vortex sheet meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Lausanne, Switzerland) *(SCA '12)*. Eurographics Association, Goslar, DEU, 87–95.

Tomas F Buttke. 1992. Lagrangian numerical methods which preserve the Hamiltonian structure of incompressible fluid flow. (1992).

Tomas F Buttke. 1993. Velicity methods: Lagrangian numerical methods which preserve the Hamiltonian structure of incompressible fluid flow. In *Vortex flows and related numerical methods*. Springer, 39–57.

Thomas F Buttke and Alexandre J Chorin. 1993. Turbulence calculations in magnetization variables. *Applied numerical mathematics* 12, 1-3 (1993), 47–54.

Duowen Chen, Zhiqi Li, Junwei Zhou, Fan Feng, Tao Du, and Bo Zhu. 2024. Solid-Fluid Interaction on Particle Flow Maps. *ACM Trans. Graph.* 43, 6, Article 267 (Nov. 2024), 20 pages. doi:10.1145/3687959

A. Chern, F. Knöppel, U. Pinkall, P. Schröder, and S. Weißmann. 2016. Schrödinger's smoke. *ACM Trans. Graph.* 35 (2016), 77.

Prince Chidyagwai, Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. 2011. A comparative study of the efficiency of jet schemes. *arXiv preprint arXiv:1104.0542* (2011).

Ricardo Cortez. 1996. An impulse-based approximation of fluid motion due to boundary forces. *J. Comput. Phys.* 123, 2 (1996), 341–353.

Georges-Henri Cottet, Petros D Koumoutsakos, et al. 2000. *Vortex methods: theory and practice*. Vol. 8. Cambridge university press Cambridge.

Yitong Deng, Hong-Xing Yu, Diyang Zhang, Jiajun Wu, and Bo Zhu. 2023. Fluid Simulation on Neural Flow Maps. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–21.

Jean Délery. 2003. Robert Legendre and Henri Werlé: Toward the Elucidation of Three-Dimensional Separation. *Annual Review of Fluid Mechanics* 33 (11 2003), 129–154. doi:10.1146/annurev.fluid.33.1.129

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).

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.

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), 201.

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.

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.

Michael Lentine, Mridul Aanjaneya, and Ronald Fedkiw. 2011a. Mass and momentum conservation for fluid simulation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Vancouver, British Columbia, Canada) *(SCA '11)*. Association for Computing Machinery, New York, NY, USA, 91–100. doi:10.1145/2019406.2019419

Michael Lentine, Jón Tómas Grétarsson, and Ronald Fedkiw. 2011b. An unconditionally stable fully conservative semi-Lagrangian method. *Journal of computational physics* 230, 8 (2011), 2857–2879.

Xingqiao Li, Xingyu Ni, Bo Zhu, Bin Wang, and Baoquan Chen. 2023. GARM-LS: A Gradient-Augmented Reference-Map Method for Level-Set Fluid Simulation. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–20.

Zhiqi Li, Barnabás Börcsök, Duowen Chen, Yutong Sun, Bo Zhu, and Greg Turk. 2024a. Lagrangian Covector Fluid with Free Surface. In *ACM SIGGRAPH 2024 Conference Papers*. 1–10.

Zhiqi Li, Duowen Chen, Candong Lin, Jinyuan Liu, and Bo Zhu. 2024b. Particle-Laden Fluid on Flow Maps. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–12.

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

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

Geoffrey McGregor and Jean-Christophe Nave. 2019. Area-preserving geometric Hermite interpolation. *J. Comput. Appl. Math.* 361 (2019), 236–248.

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

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

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

Xingyu Ni, Bo Zhu, Bin Wang, and Baoquan Chen. 2020. A level-set method for magnetic substance simulation. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 29–1.

Valery Iustinovich Oseledets. 1989. On a new way of writing the Navier-Stokes equation. The Hamiltonian formalism. *Russ. Math. Surveys* 44 (1989), 210–211.

Marcel Padilla, Albert Chern, Felix Knöppel, Ulrich Pinkall, and Peter Schröder. 2019. On bubble rings and ink chandeliers. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Sang Il Park and Myoung Jun Kim. 2005. Vortex fluid for gaseous phenomena. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 261–270.

Tobias Pfaff, Nils Thuerey, and Markus Gross. 2012. Lagrangian vortex sheets for animating fluids. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–8.

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.

PH Roberts. 1972. A Hamiltonian theory for weakly interacting vortices. *Mathematika* 19, 2 (1972), 169–179.

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. In *SIGGRAPH Asia 2017 Technical Briefs*. 1–4.

Robert Saye. 2016. Interfacial gauge methods for incompressible fluid dynamics. *Science advances* 2, 6 (2016), e1501869.

Robert Saye. 2017. Implicit mesh discontinuous Galerkin methods and interfacial gauge methods for high-order accurate interface dynamics, with applications to surface tension dynamics, rigid body fluid–structure interaction, and free surface flow: Part I. *J. Comput. Phys.* 344 (2017), 647–682.

Benjamin Seibold, Jean-Christophe Nave, and Rodolfo Ruben Rosales. 2011. Jet schemes for advection problems. *arXiv preprint arXiv:1101.5374* (2011).

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.

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

DM Summers. 2000. A representation of bounded viscous flow based on Hodge decomposition of wall impulse. *J. Comput. Phys.* 158, 1 (2000), 28–50.

Seth Taylor and Jean-Christophe Nave. 2023. A characteristic mapping method for incompressible hydrodynamics on a rotating sphere. *arXiv preprint arXiv:2302.01205* (2023).

Jerry Tessendorf. 2015. Advection Solver Performance with Long Time Steps , and Strategies for Fast and Accurate Numerical Implementation. https://api.semanticscholar.org/CorpusID:35528536

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.*

Sinan Wang, Yitong Deng, Molin Deng, Hong-Xing Yu, Junwei Zhou, Duowen Chen, Taku Komura, Jiajun Wu, and Bo Zhu. 2024. An Eulerian Vortex Method on Flow Maps. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–14.

E Weinan and Jian-Guo Liu. 2003. Gauge method for viscous incompressible flows. *Communications in Mathematical Sciences* 1, 2 (2003), 317–332.

Steffen Weißmann and Ulrich Pinkall. 2010. Filament-based smoke with vortex shedding and variational reconnection. In *ACM SIGGRAPH 2010 papers*. 1–12.

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.

Shiying Xiong, Rui Tao, Yaorui Zhang, Fan Feng, and Bo Zhu. 2021. Incompressible flow simulation on vortex segment clouds. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–12.

Shiying Xiong, Zhecheng Wang, Mengdi Wang, and Bo Zhu. 2022. A clebsch method for free-surface vortical flow simulation. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–13.

Shuqi Yang, Shiying Xiong, Yaorui Zhang, Fan Feng, Jinyuan Liu, and Bo Zhu. 2021. Clebsch gauge fluid. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–11.

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.

Xinxin Zhang and Robert Bridson. 2014. A PPPM fast summation method for fluids and beyond. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–11.

Junwei Zhou, Duowen Chen, Molin Deng, Yitong Deng, Yuchen Sun, Sinan Wang, Shiying Xiong, and Bo Zhu. 2024. Eulerian-Lagrangian Fluid Simulation on Particle Flow Maps. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–20.