Cirrus: Adaptive Hybrid Particle-Grid Flow Maps on GPU

MENGDI WANG, Georgia Institute of Technology, USA
FAN FENG, Dartmouth College, USA
JUNLIN LI, Georgia Institute of Technology, USA
BO ZHU, Georgia Institute of Technology, USA

Fig. 1. *Left: smoke (large) passing a racing car and its vorticity field (small). Right: smoke (large) passing an aircraft with* 4 *rotating propellers at a 15-degree angle of attack and its vorticity field (small). The wingtip vortices are captured by our algorithm in the vorticity field. Effective resolutions are* $512 \times 512 \times 1024$ *on our adaptive grid implemented on GPU.*

We propose the *adaptive hybrid particle-grid flow map* method, a novel flow-map approach that leverages Lagrangian particles to simultaneously transport impulse and guide grid adaptation, introducing a fully adaptive flow map-based fluid simulation framework. The core idea of our method is to maintain flow-map trajectories separately on grid nodes and particles: the grid-based representation tracks long-range flow maps at a coarse spatial resolution, while the particle-based representation tracks both long and short-range flow maps, enhanced by their gradients, at a fine resolution. This hybrid Eulerian-Lagrangian flow-map representation naturally enables adaptivity for both advection and projection steps. We implement this method in *Cirrus*, a GPU-based fluid simulation framework designed for octree-like adaptive grids enhanced with particle trackers. The efficacy of our system is demonstrated through numerical tests and various simulation examples, achieving up to $512 \times 512 \times 2048$ effective resolution on an RTX 4090 GPU. We achieve a 1.5 to 2× speedup with our GPU optimization over the Particle Flow Map method on the same hardware, while the adaptive grid implementation offers efficiency gains of one to two orders of magnitude by reducing computational resource requirements.

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

Additional Key Words and Phrases: fluid simulation, adaptive grid, octree, flow map method, Eulerian-Lagrangian method, GPU simulation

Authors' addresses: Mengdi Wang, mengdi.wang@gatech.edu, Georgia Institute of Technology, Atlanta, GA, USA; Fan Feng, fan.feng.gr@dartmouth.edu, Dartmouth College, Hanover, NH, USA; Junlin Li, jli3518@gatech.edu, Georgia Institute of Technology, Atlanta, GA, USA; Bo Zhu, bo.zhu@gatech.edu, Georgia Institute of Technology, Atlanta, GA, USA.

ACM 0730-0301/2025/8-ART
https://doi.org/10.1145/3731190

ACM Reference Format:
Mengdi Wang, Fan Feng, Junlin Li, and Bo Zhu. 2025. *Cirrus*: Adaptive Hybrid Particle-Grid Flow Maps on GPU. *ACM Trans. Graph.* 44, 4 (August 2025), 17 pages. https://doi.org/10.1145/3731190

1 INTRODUCTION

Flow map methods have emerged as an effective numerical scheme for reducing numerical dissipation and preserving fine-scale vortex structures in recent years. Among these advancements, the Particle Flow Map (PFM) method [Zhou et al. 2024] achieves state-of-the-art performance by utilizing the Affine Particle-In-Cell (APIC) scheme [Jiang et al. 2015] to advect impulse with moving particles on a background grid. However, despite its notable vorticity preservation capability, PFM relies on maintaining a dense particle system across all grid cells to transport the impulse field and its gradients, resulting in high memory overhead and restricting resolution to a typical 256^3 uniform grid.

Adaptive grids, such as Octree or AMR [Aanjaneya et al. 2017; Ando and Batty 2020; Ando et al. 2013; Berger and Oliger 1984; Karnakov et al. 2020; Losasso et al. 2004; Popinet 2009; Setaluri et al. 2014], utilize hierarchical grid structures to allocate grid cells with different sizes in different regions, thereby significantly reducing the computational overhead. However, implementing adaptive grids—particularly on GPUs—remains challenging. For grid-based systems, the refinement decisions of an adaptive grid rely on heuristics derived from the fluid system, such as the distance to a solid/air boundary [Aanjaneya et al. 2017; Ando and Batty 2020], or on physical quantities stored on the grid, such as vorticity strength [Deng et al. 2023; Hejazialhosseini et al. 2010]. However, calculating these

quantities, in turn, is based on the grid, making it difficult to design grid adaptation algorithms. In hybrid particle-grid systems, handling particle-grid interpolation across different refinement levels can introduce numerical instability and add implementation complexity. Furthermore, adaptive grids have seen limited application in computer graphics, with few GPU-based adaptive grid fluid simulators available. We identify three primary challenges in this context: (1) designing a robust and dynamic grid refinement mechanism, (2) developing advection schemes for flow maps on adaptive grids, and (3) building an efficient, user-friendly GPU-based adaptive-grid fluid simulation framework.

In this paper, we propose the *adaptive hybrid particle-grid flow map* method, which integrates particles and adaptive grids for fluid simulation. In our approach, particles perform two tasks: advecting impulse in key regions and guiding grid adaptation. Specifically, Lagrangian particles are used to transport the impulse vector in areas requiring high resolution, ensuring detailed fluid structures are preserved. Simultaneously, these particles serve as indicators for refining the adaptive grid, focusing computational resources on regions with complex fluid dynamics. In less critical areas, grid-based advection is employed to maintain efficiency. By dynamically coupling the particle system with spatially adaptive grids, our method strikes a balance between preserving fine-scale details and optimizing computational cost.

For efficient implementation of this algorithm, we present *Cirrus*, a GPU-based fluid simulation framework designed to handle high-resolution simulations up to $512 \times 512 \times 2048$ on consumer-grade GPUs with hybrid particle-grid flow maps. It constructs a dynamic octree-like adaptive grid structure, where each grid level is stored as blocks in a hash table, therefore supporting arbitrary levels of grid resolution and allowing for dynamic topology modifications. Through our GPU optimizations, our *Cirrus* framework achieves a 2× speedup over PFM on a dense grid and a 1.5× speedup over PFM on an adaptive grid, both on the same hardware. Additionally, the adaptive grid strategy allows *Cirrus* to improve computational efficiency by one to two orders of magnitude based on effective resolution. We demonstrate the efficacy of *Cirrus* by simulating a range of high-resolution turbulent fluid examples, achieving state-of-the-art vortical flow simulation results by combining flow-map-based conservation and GPU parallelism.

2 RELATED WORK

Adaptive Grids. The term adaptive grids refers to a class of techniques that distribute computational grids non-uniformly in space, allowing for efficient allocation of computational resources. In Computational Fluid Dynamics (CFD), the use of adaptive data structures, including hierarchical Adaptive Mesh Refinement (AMR) grids [Berger and Oliger 1984] or octree [Popinet 2009] for fluid simulation, has been a long-established approach. These methods can be applied to both incompressible flow, [Almgren et al. 1996; Howell and Bell 1997], compressible flow [Coirier 1994; Hejazialhosseini et al. 2010; Khokhlov 1998], and two-phase flow [Karnakov et al. 2020; Popinet 2009] simulations. Subsequently, this approach has also been adopted [Losasso et al. 2004; Shi and Yu 2004] in Computer Graphics. Recent advancements have applied this method to simulate various physical phenomena, such as free-surface fluid dynamics [Ando and Batty 2020] and elastoplastic materials modeled with the MPM algorithm [Gao et al. 2017]. Adaptive grids can be implemented using various data structures, such as tree structures [Museth 2021; Museth et al. 2013], generalized octrees [Ferstl et al. 2014; Nielsen and Bridson 2016], single-level tiles [Narita and Ando 2022], far-field grid [Zhu et al. 2013], warping grid [Ibayashi et al. 2018], tetrahedral meshes [Ando et al. 2013; Batty et al. 2010; Chentanez et al. 2007], power diagrams [Zhai et al. 2018], and neural representations [Deng et al. 2023; Kim et al. 2024]. Some codimension data structures [Deng et al. 2022] can also be regarded as a form of adaptive grid. In this work, we use an adaptive grid divided into blocks, a straightforward and efficient data structure [Setaluri et al. 2014] for fluid simulation that has been widely adopted.

Adaptive Algorithms. Specially designed numerical algorithms are often required to implement fluid simulations on adaptive grids. First, dedicated algorithms for adaptive grid generation are necessary, such as refinement criteria based on locations [Klingner et al. 2006; Losasso et al. 2004; Setaluri et al. 2014], sizing functions calculated from physical quantities [Ando and Batty 2020; Shi and Yu 2002] or wavelet transforms [Hejazialhosseini et al. 2010]. Interpolation and advection schemes on adaptive grids also need to be carefully designed [Setaluri et al. 2014]. In particular, if particle systems are involved, different kernel functions are typically employed across grid levels to adapt to the varying resolutions, requiring custom numerical schemes to ensure stability [Gao et al. 2017].

Poisson Solver. A key component of incompressible fluid simulation is the projection step using a Poisson solver. Matrix-free multigrid solvers [McAdams et al. 2010] are commonly used to solve the Poisson equation efficiently. Algebraic multigrid (AMG) methods [Shao et al. 2022; Takahashi and Batty 2023, 2025] offer faster convergence, in exchange, the computational cost per iteration is increased. Implementing such solvers on adaptive grids is itself a specialized area of research. For solving Magnetohydrodynamics (MHD) [Teunissen and Ebert 2018; Tomida and Stone 2023] and two-phase flows [Karnakov et al. 2020; Popinet 2009], specific treatments [Losasso et al. 2006] or high-order interpolation schemes [Batty 2017; Guittet et al. 2015; Teunissen and Schiavello 2023] are often employed at T-junctions for higher accuracy. In this work, we follow the practice of Losasso et al. [2004] that uses constant interpolation.

Hybrid Particle-Grid Methods. The use of hybrid particle-grid methods in computer graphics can be traced back to PIC/FLIP [Boyd and Bridson 2012; Deng et al. 2022; Zhu and Bridson 2005], which subsequently became widely adopted. We adopt the Affine Particle-In-Cell (APIC) scheme proposed by Jiang et al. [2015] for hybrid particle-grid advection, which effectively reduces numerical dissipation. Particle systems, being inherently Lagrangian, are natural choices for advecting markers and are often used for interface tracking [Enright et al. 2002; Hieber and Koumoutsakos 2005]. Our method draws inspiration from NB-Flip [Ferstl et al. 2016; Sato et al. 2018b], which samples particles only near the fluid surface, enabling accurate and detailed interface tracking. Another important hybrid

particle-grid algorithm is the Material Point Method (MPM) [Jiang et al. 2016], initially developed for simulating soft bodies but later extended to a variety of other applications [Gao et al. 2018a; Han et al. 2019]. In this paper, our G2P (grid-to-particle) and P2G (particle-to-grid) transfers draw inspiration from modern MPM techniques, especially GPU-MPM [Gao et al. 2018b; Wang et al. 2020].

Flow Map Methods. The history of flow map methods can be traced back to the characteristic map method in CFD. It was first introduced to fluid simulation by Wiggert and Wylie [1976] to reduce numerical dissipation using a velocity field-advected long-range mapping. This method was later adapted to the graphics community by Hachisuka [2005] and Tessendorf and Pelfrey [2011]. The following research [Sato et al. 2018a, 2017; Tessendorf 2015] typically utilized a high computation demanding virtual particles method to track the flow map. Inspired by Kim et al. [2007], Qu* et al. [2019] later proposed a Semi-Lagrangian-like scheme to advect flow maps in a bidirectional manner to reduce time cost and improve the mapping accuracy. Recently, Nabizadeh et al. [2022] extended this concept to the impulse fluid model [Cortez 1996; Feng et al. 2022]. In addition to this, Neural Flow Maps (NFM) [Deng et al. 2023] introduced a new backward flow map advection scheme and employed a neural network to compress the required velocity buffers during flow map reconstruction efficiently. Zhou et al. [2024] introduced the concept of long-short flow maps and transported the impulse on Lagrangian particles with the APIC scheme [Jiang et al. 2015], achieving state-of-the-art results. Wang et al. [2024] combined the flow map concept with the vortex method to further improve its numerical stability and physical interpretability. Recent advances have also explored broader applications of flow maps. Li et al. [2024b] and Chen et al. [2024] studied particle-laden and solid-fluid interactions on flow maps, respectively. Furthermore, Li et al. [2024a] developed a Lagrangian covector fluid framework with free surface support, and Sun et al. [2024] presented an impulse-based ghost fluid method for handling two-phase flows, enriching the physical modeling capabilities of flow map methods. In this work, we mainly follow the PFM and NFM flow map schemes.

GPU-Based Simulation. The parallel computing power of GPUs enables the efficient implementation of fluid simulations. On dense grids, GPU-based fluid simulation has become a widely adopted approach [Hu et al. 2019]. Some fluid simulation algorithms, are more GPU-friendly, such as the Lattice Boltzmann Method (LBM) [Krüger et al. 2017] and wavelet-based techniques [Lichtl and Jones 2015]. LBM, in particular, solves the Boltzmann equations instead of the Navier-Stokes equations, offering a representation that is well-suited for large-scale computations on GPUs due to its algorithmic locality. It has been successfully implemented with adaptive grids on GPUs [Liu and Liu 2023; Lyu et al. 2018]. Recently, DCGrid [Raateland et al. 2022] introduces a GPU-efficient adaptive grid, but lacks a full multigrid solver, limiting visual fidelity. Despite these developments, most mainstream implementations of the projection method for solving incompressible fluids on adaptive grids in Computer Graphics remain CPU-based. In this paper, our proposed *Cirrus* system on adaptive grids achieves a 1.5× to 2× speedup compared to prior GPU-PFM implementations and significantly outperforms CPU-PFM.

3 PHYSICAL MODEL

3.1 Fluid Model

We consider the impulse-form incompressible flow equations without viscosity [Cortez 1996]:

$$\begin{aligned} \frac{\mathrm{D}\boldsymbol{m}}{\mathrm{D}t} &= -\left(\boldsymbol{\nabla u}\right)^T \boldsymbol{m}, \\ \nabla^2 \varphi &= \boldsymbol{\nabla} \cdot \boldsymbol{m}. \end{aligned} \tag{1}$$

We define $\boldsymbol{m}$ in (1) as the fluid impulse, a quantity related to the velocity field $\boldsymbol{u}$ through a scalar gauge variable φ:

$$\boldsymbol{m} = \boldsymbol{u} + \boldsymbol{\nabla}\varphi. \tag{2}$$

Note that $\boldsymbol{u}$ is divergence-free, allowing us to obtain $\boldsymbol{u}$ from $\boldsymbol{m}$ by removing its curl-free part based on Helmholtz decomposition.

3.2 Flow Map Model

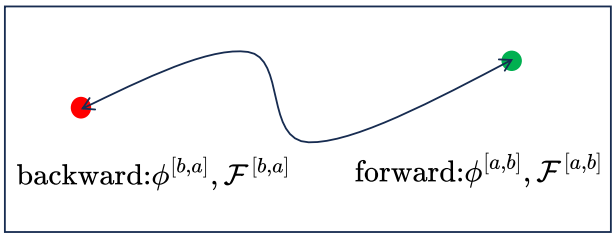

Fig. 2. *Forward flow map* $(\boldsymbol{\phi}^{[a,b]}, \mathcal{F}^{[a,b]})$ *(right) and backward flow map* $(\boldsymbol{\phi}^{[b,a]}, \mathcal{F}^{[b,a]})$ *(left). The physical particle path (left to right) is* $t_a \rightarrow t_b$.

Flow map is a mathematical model for a particle's trajectory in spacetime. Given two time instants a and b, we consider the forward flow map as the particle's path from t_a to t_b, while the backward flow map as the reverse path from t_b to t_a. In the remainder of this paper, we assume that t_a is the start time of the particle path, and t_b is the end time. Superscripts $\boldsymbol{x}^a, \boldsymbol{x}^b$ stand for particle positions $\boldsymbol{x}(t_a), \boldsymbol{x}(t_b)$ at different times.

We use $\boldsymbol{\phi}^{[a,b]}$ to represent the (forward) flow map that maps $\boldsymbol{x}^a$ at the beginning of the path to its position $\boldsymbol{x}^b$ at the end. It is formally defined as

$$\begin{cases} \dfrac{\partial \boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a)}{\partial t_b} = \boldsymbol{u}\left(\boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a), t_b\right), \\ \boldsymbol{\phi}^{[a,a]}(\boldsymbol{x}^a) = \boldsymbol{x}^a. \end{cases} \tag{3}$$

Here $\boldsymbol{\phi}^{[a,a]}$ = identity is the initial condition. In other words

$$\boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a) = \boldsymbol{x}^a + \int_{t_a}^{t_b} \boldsymbol{u}(\phi^{[a,\tau]}(\boldsymbol{x}^a), \tau)\mathrm{d}\tau. \tag{4}$$

Swapping the start and end points of the forward flow map yields the backward flow map $\boldsymbol{\phi}^{[b,a]}$, as shown in Fig. 2.

The flow map's Jacobian matrix $\mathcal{F}^{[a,b]}$ is defined as

$$\mathcal{F}^{[a,b]}(\boldsymbol{x}^a) = \frac{\partial \boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a)}{\partial \boldsymbol{x}^a}. \tag{5}$$

3.3 Flow Map Marching

Taking the material derivatives of Eq. 5 with t_a and t_b gives us

$$\frac{\mathrm{D}\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)}{\mathrm{D}t_b} = \nabla \boldsymbol{u}^b \mathcal{F}^{[a,b]}(\boldsymbol{x}^a), \tag{6a}$$

$$\frac{\mathrm{D}\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)}{\mathrm{D}t_a} = -\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)\nabla \boldsymbol{u}^a. \tag{6b}$$

The detailed derivations are provided in Appendix A.

Eq. 6a implies that starting from the initial conditions $\boldsymbol{\phi}^{[a,a]} = \boldsymbol{x}^a$ and $\mathcal{F}^{[a,a]}(\boldsymbol{x}^a) = \mathbf{I}$, we can march the **endpoint** $\boldsymbol{x}^b$ of the particle path by viewing t_b as a variable and performing time integration on the velocity field with RK4. This process finally yields the flow map $\boldsymbol{\phi}^{[a,b]}$ and its Jacobian matrix $\mathcal{F}^{[a,b]}$, when integrated to t_b.

Similarly, starting from the initial conditions $\boldsymbol{\phi}^{[b,b]} = \boldsymbol{x}^b$ and $\mathcal{F}^{[b,b]}(\boldsymbol{x}^b) = \mathbf{I}$, we can march the **starting point** $\boldsymbol{x}^a$ of the particle path by performing RK4 integration viewing t_b as a variable. This also yields $\boldsymbol{\phi}^{[a,b]}$ and $\mathcal{F}^{[a,b]}$ at the end.

3.4 Time Evolution

The impulse field $\boldsymbol{m}(\cdot, t_b)$ and its gradient are related to their values at t_a as [Zhou et al. 2024]

$$\boldsymbol{m}(\boldsymbol{x}^b, t_b) = \left(\mathcal{F}^{[b,a]}(\boldsymbol{x}^b)\right)^\top \boldsymbol{m}(\boldsymbol{\phi}^{[b,a]}(\boldsymbol{x}^b), t_a), \tag{7a}$$

$$\begin{aligned}\nabla \boldsymbol{m}(\boldsymbol{x}^b, t_b) = &\left(\mathcal{F}^{[b,a]}(\boldsymbol{x}^b)\right)^\top \nabla \boldsymbol{m}(\boldsymbol{\phi}^{[b,a]}(\boldsymbol{x}^b), t_a) \left(\mathcal{F}^{[b,a]}(\boldsymbol{x}^b)\right) \\ &+ \nabla \left(\mathcal{F}^{[b,a]}(\boldsymbol{x}^b)\right)^\top \boldsymbol{m}(\boldsymbol{\phi}^{[b,a]}(\boldsymbol{x}^b), t_a),\end{aligned} \tag{7b}$$

where the Hessian term $\nabla \mathcal{F}^T$ is often omitted without compromising the quality of the simulation results. Since the gauge variable φ does not appear in (3)-(7), we conveniently set $\varphi = 0$ at $t = 0$. Thus, a fluid system can be simulated with flow map advection in Alg. 1, which is essentially a variant of the classical projection method [Bridson 2015], however, the $p^\star$ solved in projection step is the sum of actual pressure and the gauge variable: $p^\star = p + \varphi$.

Algorithm 1 Flow Map Fluid Simulation

Input: intermediate velocity fields $\boldsymbol{u}^0, \ldots, \boldsymbol{u}^n$
Output: velocity field $\boldsymbol{u}^{n+1}$

1: $\boldsymbol{m}^0 \leftarrow \boldsymbol{u}^0$
2: Calculate $\boldsymbol{\phi}^{[n+1,0]}$ and $\mathcal{F}^{[n+1,0]}$ ▷ (6)
3: $\boldsymbol{m}^{n+1} \leftarrow \left(\mathcal{F}^{[n+1,0]}\right)^\top \boldsymbol{m}^0(\boldsymbol{\phi}^{[n+1,0]})$ ▷ (7)
4: Solve $\nabla^2 p^\star = \nabla \cdot \boldsymbol{m}^{n+1}$
5: $\boldsymbol{u}^{n+1} \leftarrow \boldsymbol{m}^{n+1} - \nabla p^\star$ ▷ projection

4 ADAPTIVE HYBRID PARTICLE-GRID FLOW MAP

In this paper, we propose a novel implementation of the flow map method on an octree-like adaptive grid, employing a hybrid particle-grid scheme for flow map advection. Fig. 3 shows the two main components of our data structure: an adaptive grid and a particle system. The particles are sampled around solid objects and will persist for a duration of time.

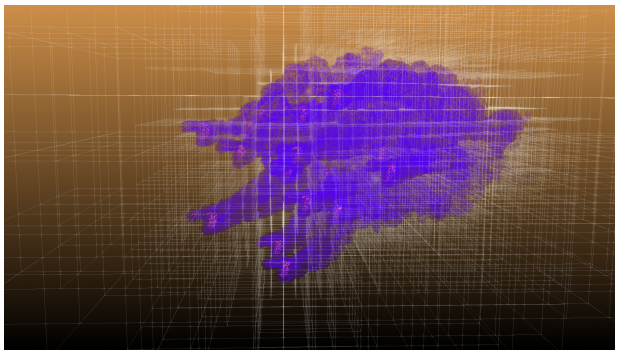

Fig. 3. *Our adaptive particle-grid structure. Particles (blue) are generated near solid objects (pink) and persist for a duration. The grid (white) refines to the highest resolution in regions with particles. Each cube corresponds to a tile of 8^3 cells.*

Inspired by the selective use of high-cost schemes in key regions [Narita and Ando 2022] and the observation by PFM that the particle system tracking $\nabla \boldsymbol{m}$ corresponds to shorter flow map lengths and smaller scales due to its responsiveness to distortions, our algorithm refines the grid to its maximum resolution in detailed fluid regions. On other areas which are less critical, we employ the memory-efficient grid advection.

Naturally, this method provides us a predicate for grid refinement, i.e., particles. In other words, our particle system performs two tasks: advecting the flow map and guiding grid adaptation. In regions with particles, we refine the grid to the highest resolution, as shown in Fig. 3. By employing particles as refinement indicators, we effectively circumvent both the dependence of sizing functions on the grid itself and the sensitivity of interpolation algorithms to grid structure.

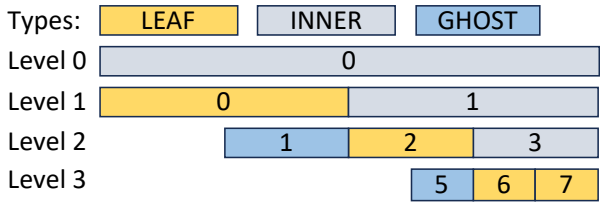

Fig. 4. *A simplified 1-D illustration of the grid structure. There are three types of tiles: yellow for leaf tiles, gray for inner tiles, and blue for ghost tiles. The maximum level difference of neighboring leaf tiles is* 1*, and a ghost tile is created in such T-junction cases.*

4.1 Adaptive Grid

We design our adaptive grid system as a tile-based octree-like structure. The grid is divided into multiple levels, denoted by an integer l. Level 0 is the coarsest level, and level L is the finest. The cell size h^l at level l is defined as $h^l = 1/2^{l+3}$.

Each grid level is divided into $8 \times 8 \times 8$ blocks of cells, referred to as a "tile". A tile at the coarsest level $l = 0$ spans a spatial domain of $1 \times 1 \times 1$. In our framework, a tile is the smallest unit for memory management, and the tiles form an octree.

A tile belongs to one of three types: **leaf**, **inner**, and **ghost**. **Leaf** tiles are the leaf nodes in the octree, holding active cells for computation. We use the MAC grid to store physical quantities. A leaf cell stores three velocity components at its $x-, y-, z-$ face centers and a pressure p at the cell center. **Inner** tiles, on the other hand, are the interior nodes in the octree structure. An inner tile is the parent of 8 leaf tiles or 8 inner tiles.

We apply a level constraint where the maximum level difference between two face-sharing neighbor leaf tiles (referred to as neighbors for the rest of this paper) is limited to one. We have found that this approach is sufficient to ensure a smooth transition between levels without introducing visible artifacts.

We refer to the case as a T-junction when two neighboring leaf tiles T_0, T_1 are at levels i and $i+1$ respectively (see Fig. 7). In this case, we create a **ghost** tile at level $i+1$ that is a child of T_0 and adjacent to T_1. Ghost tiles are useful to avoid cross-level access and reduce the computational cost at T-junctions in the Poisson solver, as we will explain in Sec. 5.4.

Fig. 4 illustrates a simplified 1-D example of a grid with 4 levels. Notice that the tile(s) at level 0 always span the entire computational domain. For example, if the computational domain is $1 \times 1 \times 2$, we will have two tiles $(0,0,0)$ and $(0,0,1)$ at level 0.

4.2 Grid Adaptation with Particles

A tile is the smallest unit for grid refinement and coarsening in our adaptive grid. It can only be refined as a whole, generating 8 child tiles, or coarsened with its 7 siblings together, making their common parent a new leaf tile.

Our grid adaptation algorithm relies on a level target function $F(T)$ for a tile T, that satisfies $F(T) = \max\{F(C) : C \text{ is a child of } T\}$. In our hybrid particle-grid method, it's defined as

$$F(T) = \begin{cases} L, & \text{if T contains a particle,} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

However, the grid adaptation algorithm itself can be generalized to other choices of F. It guarantees that $\text{level}(T) \geq F(T)$ and the 1-level constraint is not violated.

For refinement, A leaf tile T at level i must be refined if:

(1) $F(T) > i$, or
(2) There is a leaf neighbor of T at level $i+1$ that will be refined.

Coarsening follows a similar principle but is more complex because, in addition to the level constraint, 8 sibling leaf tiles must negotiate through their parent to decide whether they should be coarsened or not. A leaf tile T should be deleted if

(1) Its level $i > F(T)$, and
(2) All its siblings should be deleted, and
(3) There is no same-level neighbor S that is an inner tile and will not become a leaf.

A ghost tile G is deleted if its parent leaf tile is deleted, or all its leaf neighbors are deleted, making it no longer necessary.

The refinement and coarsening algorithms are given in detail in Appendix B. Note that both algorithms will add or remove only one level to the grid. Therefore, an iterative process is required to dynamically adjust the grid topology, as shown in Alg. 2. We demonstrate a simple example of grid adaptation in Fig. 5.

Algorithm 2 Dynamic Grid Adaptation

Input: adaptive grid $\mathcal{G}$, maximum level L, level target function F

```
while true do
    N ← RefineStep(G, L, F)                ▷ Algorithm 5
    if N = 0 then
        break
while true do
    N ← CoarsenStep(G, L, F)               ▷ Algorithm 6
    if N = 0 then
        break
```

4.3 Hybrid Flow Map Advection

For particle advection, we adopt the long-short flow map in PFM, that each particle carries $\boldsymbol{m}$ and $\nabla\boldsymbol{m}$. The impulse $\boldsymbol{m}$ is calculated with a k-step long flow map where $k = 5$ in our simulations, while $\nabla\boldsymbol{m}$ is calculated with a 1-step short flow map.

Suppose we're currently evolving the fluid system from time step i to time step $i+1$ and the flow map is initialized at time step i_0. We take $t_a = i_0\Delta t$, $t_e = i\Delta t$ and $t_b = (i+1)\Delta t$. The impulse $\boldsymbol{m}^b$ and its gradient $\nabla\boldsymbol{m}^b$ of a particle are forward-advected with Eq. 7:

$$\begin{aligned} \boldsymbol{m}^b &= \left(\mathcal{F}^{[b,a]}\right)^\top \boldsymbol{m}^a, \\ \nabla\boldsymbol{m}^b &= \left(\mathcal{F}^{[b,e]}\right)^\top \nabla\boldsymbol{m}^e \left(\mathcal{F}^{[b,e]}\right). \end{aligned} \tag{9}$$

Eq. 9 is calculated with Eq. 6b. The values $\boldsymbol{u}$, $\nabla\boldsymbol{u}$, $\boldsymbol{m}$, $\nabla\boldsymbol{m}$ required by the RK4 marching are calculated with the quadratic kernel interpolation [Jiang et al. 2016] on the grid. The kernel has a radius of $1.5h$, which means it uses a $3 \times 3 \times 3$ stencil. Since particles only reside on the finest level, this G2P transfer, or particle advection, is essentially the same as G2P on a uniform grid.

After G2P, the impulse field is transferred back to the grid with the APIC P2G scheme. In regions not covered by particles, we use Eq. 6a to calculate the impulse (7a) at grid face centers. Here, we must handle the cross-level cases when performing the quadratic kernel interpolation. First, we linearly interpolate [Setaluri et al. 2014] the velocity at the face centers of all inner cells. Then, starting from the finest level L, if the 3^3 stencil is fulfilled with leaf and interior cells of this level, we use the kernel interpolation results from this level. Otherwise, we fall back to the coarser level $L-1$, $L-2$, and so on, until the stencil is satisfied or the coarsest level is reached.

To compute Eq. 6a on the grid, we store k grids as the velocity buffer. Additionally, for each newly sampled particle, we backtrace it to t_a using Eq. 6b to align its flow map length with that of the other particles.

Fig. 6 illustrates the hybrid flow map advection scheme used by us. In regions with particles, we use Lagrangian particles to advect the flow map. In other regions, we use grid-based advection.

5 GPU IMPLEMENTATION

To efficiently implement the hybrid particle-grid flow map algorithm, we have developed *Cirrus*, a fluid simulation framework that utilizes adaptive grids and a multigrid solver on consumer-grade GPUs.

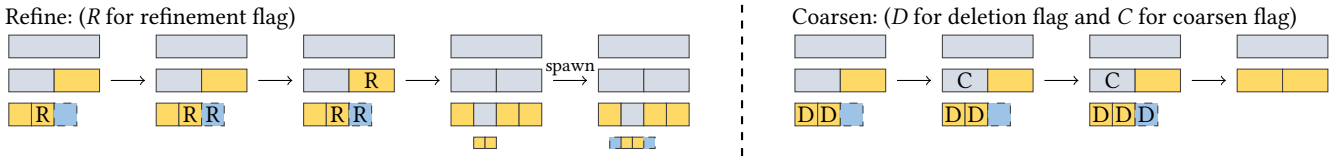

Fig. 5. *Top: an 1D illustration of grid refinement. The refinement flag is calculated and propagated to satisfy the neighbor level constraint. Ghost tiles are generated in the end. Bottom: a 1D illustration of grid coarsening. Both leaf tiles must be marked with a deletion flag to delete them simultaneously, which is guaranteed through their parents' coarsened flag. Ghost tiles that lose all neighboring tiles are also deleted.*

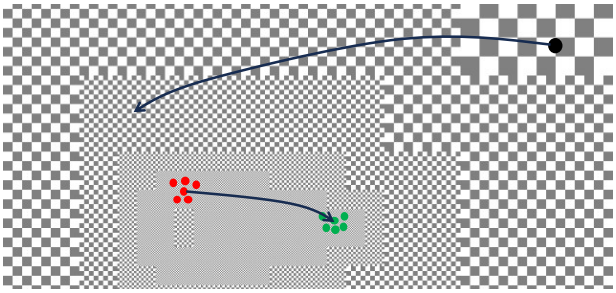

Fig. 6. *Our adaptive hybrid particle-grid flow map algorithm. Particles (bottom line) only reside on the finest level, while the grid flow map advection (top line) may cross multiple levels. They calculate the same flow map* $\boldsymbol{\phi}^{[b,a]}, \mathcal{F}^{[b,a]}$.

In addition to the hybrid particle-grid approach, it also supports traditional advection schemes like semi-Lagrangian.

5.1 Grid Storage

In our grid, tile entries, which store single precision floating point data, are maintained using a hash table-based approach inspired by Instant-NGP [Müller et al. 2022]. Each level has its own hash table, using tile indices as keys and the hash function:

$$\text{hash}(i,j,k) = \Big((i \cdot f_1 \oplus j \cdot f_2 \oplus k \cdot f_3) \mod 2^M\Big),$$

where $f_1 = 1$, $f_2 = 2654435761$, $f_3 = 805459861$, and $\oplus$ is bitwise XOR. Unlike Instant-NGP, we employ a linear probing technique to guarantee that each tile has a unique entry and allows tile deletion. The hash table size is set to M=18 in the finest level and $M = 16$ for all other levels.

To perform linear interpolation introduced by SPGrid [Setaluri et al. 2014], each tile is allocated space for $9^3 = 729$ elements to store data at the nodes (cell corners). Each tile contains m data channels, so its data structure is an $m \times 729$ array, similar to the AoSoA design [Wang et al. 2020]. Data within each channel is organized in lexicographical order.

CUDA kernels are typically launched with a block size of 128 to maximize SM utilization. Computationals are repeated four times to cover the tile, with thread i is processing cells $\{i, i{+}128, i{+}256, i{+}384\}$. To launch kernels on multiple tiles, we collect all the tile entries at each level into a single array whenever the tiles are added or removed. This gathering operation is performed after each time the grid structure is modified. We also cache the pointers to six neighboring tiles to improve the efficiency of local accesses.

For computation of the level target (maximum value) and dot product (summation), reduction operations are performed across the data within each tile. These reductions are implemented using the NVIDIA CUB library.

5.2 G2P Transfer

In *Cirrus*, particles are generated in the region where the distance to the nearest solid object is less than a threshold $\phi_g = 5\Delta x$. If a fluid cell inside the generation region doesn't contain particles, we randomly generate 8 particles inside. In implementation, we first count the number of particles to be generated for each cell, generate them on the CPU, and copy them back to GPU memory. Since the number of particles generated in each step is relatively small, this approach does not significantly impact the simulation's overall performance.

We have designed a particle extinction mechanism to allow for a trade-off between simulation performance and computational overhead. A particle is removed if any of the three conditions are met:

(1) G2P fails because some of the finest-level values in the kernel interpolation stencil are missing.
(2) It enters a non-fluid grid cell, e.g., wall boundaries.
(3) When a particle's lifetime exceeds a predefined threshold $\mathcal{L}$.

Larger values of the particle life $\mathcal{L}$ increase the number of cells and particles, which in turn allows vorticity to persist for a longer duration due to improved preservation in high-resolution, particle-advected critical regions. Given our hardware limitations, a typical maximum value for $\mathcal{L}$ is 0.5. Since the vortex structures in smoke originate at solid boundaries and are independent of the vorticity field further away, the length of $\mathcal{L}$ does not significantly affect smoke simulation results. As long as $\mathcal{L}$ is of sufficient length to generate the initial vorticity field, the visual outcome for smoke remains consistent. The impact of varying particle lifetimes is further explored in Sec. 8.1.

In order to optimize G2P transfer utilizing shared memory, we employ histogram-sorting [Gao et al. 2018b] to create sorted lists of particles within each tile. First, the number of particles within each cell is counted by atomic add, then the prefix sum of particle numbers is calculated within and across tiles.

We launch a CUDA block with 128 threads to process all particles within each tile. Suppose we have a tile with x index ranging $[0, 8)$, thus the spatial range of particles inside it may span $[0, 8h]$. We set the CFL number to 0.5 in our simulations, therefore, the particles will be contained in $[-0.5h, 8.5h]$ during the G2P step. Considering the kernel radius $r = 1.5h$ in Sec. 4.3, the access range of the

Algorithm 3 Full Approximation Scheme (FAS) V-Cycle

Input: grid $\mathcal{G}$, maximum level L, divergence b, initial guess $u = 0$
Output: approximate solution of $Au = b$

Downstroke:
for $l = L$ **to** 1 **do**
  $u^l \leftarrow \text{Smooth}(u^l, b^l)$ ▷ On leaf and inner tiles
  $r^l \leftarrow b^l - Au^l$ ▷ On leaf, inner and ghost tiles
  $b^{l-1} \leftarrow \text{Restrict}(r^l)$ ▷ Restrict to leaf and inner tiles
$u^0 \leftarrow \text{Smooth}(u^0, b^0)$
Upstroke:
for $l = 1$ **to** L **do**
  $u^l \leftarrow u^l + \text{Prolongate}(u^{l-1})$ ▷ Prolongate to all tiles
  $u^l \leftarrow \text{Smooth}(u^l, b^l)$ ▷ On leaf and inner tiles

kernel interpolation spans $(-2h, 10h)$. Since we're using the MAC grid, spatial coordinate x may correspond to face center index x/h (same-axis) or $x/h - 0.5$, therefore, the index access range becomes $(-2.5, 10)$, with 12 integers inside it. We therefore create a 3×12^3 shared buffer for the velocity data of 12^3 cells, which is loaded once from the global memory at the beginning of the CUDA block. Then the G2P function is performed entirely in shared memory, maximizing the memory access efficiency.

5.3 P2G Transfer

As the inverse operation of G2P, the APIC P2G scheme scatters the impulse $\boldsymbol{m}$ calculated with particles' $\boldsymbol{m}, \nabla\boldsymbol{m}$ back to the grid at the finest level L. At a face center node i, the P2G transfer will calculate a weighted impulse component s_i and a weight sum w_i. If $w_i < 1 - \varepsilon$ for a threshold $\varepsilon = 10^{-3}$, we consider that the P2G transfer has failed at this grid node, and the grid flow map is applied instead. Otherwise, the corresponding impulse component is calculated as $m_i = s_i / w_i$.

We also use particle-sorting and shared memory techniques to optimize the process. Without particle movement, the spatial range of the interpolation kernel becomes $(-1.5h, 9.5h)$, and the index access range becomes $(-2h, 9.5h)$, with 11 integers inside. We therefore create a 6×11^3 buffer in shared memory for 11^3 cells' s_i and w_i.

Gao et al. [2018b] also suggests that the majority of conflict cases occurs within warps. In order to reduce warp-level conflicts, we use their warp-level operations in the P2G transfer. Since our particles are sorted, the particles within each warp interacting with the same cells are contiguous. As suggested by them, we use a warp-level __shfl_down function with doubling strides to sum the values of particles targeting the same cell in the same warp, effectively replacing the expensive atomic operations.

5.4 Poisson Solver

We implemented a Poisson solver on GPU for the adaptive grid to perform projection. The projection step in fluid simulation requires solving a linear system $\nabla^2 p = \nabla \cdot \boldsymbol{u}$ on an adaptive grid. To ensure consistency across T-junctions in the adaptive grid, we employ the integral form of the Laplacian operator, which represents the fluxes through all faces:

$$\int_{C_I} -\nabla^2 p \mathrm{d}V = \oint_{\partial C_I} -\nabla p \cdot \boldsymbol{n} \mathrm{d}S = \sum_{J \in \mathcal{N}_I} \frac{p_I - p_J}{\Delta_{I,J}} S_{I,J}. \quad (10)$$

Here, C_I is the h^3 cube occupied by leaf cell I, and $\mathcal{N}_I$ represents its leaf neighbors. The distance between cell centers $\Delta_{I,J}$ equals h if both cells are at the same level. $S_{I,J}$ is the fluid area on the face between I, J, which is h^2 if both cells are full of fluid.

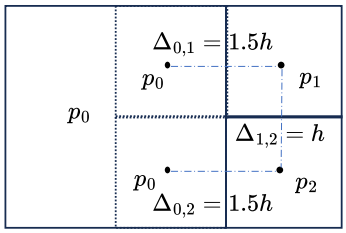

Fig. 7. *Laplacian operator at T-junction. Dotted lines draw two ghost cells in leaf cell* 0*, and their pressures are set to* p_0*. Cell distances are set to* $\Delta_{0,1} = \Delta_{0,2} = 1.5h$ *and* $\Delta_{1,2} = h$ *in computation. The fluxes from* $p_0 - p_1$ *and* $p_0 - p_2$ *are temporarily saved at ghost cells and later accumulated to their parent.*

At T-junctions, where a leaf cell I is at level l and its neighboring cell J is at level $l - 1$, we assume that the pressure p_J remains constant throughout the entire volume of C_J following Losasso et al. [2004]. And $\Delta_{I,J}$ is set to $1.5h$, the average distance between the two levels, following Setaluri et al. [2014].

We use ghost cells to eliminate the need for cross-level access at T-junctions by the Laplacian operator. The pressure values in ghost cells are set equal to their leaf parents. Suppose a leaf cell I at level i has a neighboring leaf J at level $i - 1$. Instead of directly accessing J, cell I interacts with the ghost children of J when calculating (10). Conversely, these ghost children temporarily store flux sums with I, which are later aggregated into J. Fig. 7 illustrates the Laplacian operator at a T-junction. Note that we have also provided the numerical scheme for calculating ∇p in this way.

To solve the Poisson equation on an adaptive grid, we employ the MGPCG algorithm [McAdams et al. 2010; Shao et al. 2022], where a multigrid solver is used for preconditioning. In our implementation, the relative tolerance threshold is set to 10^{-6}. On adaptive grids, the multigrid algorithms become the so-called Full Approximation Scheme (FAS) [Popinet 2003; Teunissen and Ebert 2018], as summarized in Alg. 3. It slightly differs from the standard multigrid method on uniform grids. For example, the commonly used prolongation coefficient $\alpha = 2$ does not align with the discretized Laplacian operator at T-junctions on an adaptive grid.

We use a red-black Gauss-Seidel algorithm for smoothing, with 1 red+black iteration before and after each level and 10 iterations at level 0. The smoothing operator considers only same-level neighbors, as T-junctions are handled using ghost cells. For a leaf cell I at level l with a ghost child J at level $l + 1$, the ghost cell J is excluded from smoothing during the downstroke pass, thus $u_J^{l+1} = 0$. However, the ghost cell J accounts for its leaf neighbors when computing the residual r^l. This mechanism propagates results from level $l + 1$ to level l at the T-junctions. During the upstroke pass, we effectively

make $u_J^{l+1} = u_I^l$, based on the principle that the ghost value equals its parent's value.

We use shared memory to optimize the Laplacian and smoothing operators in the Poisson solver, as our Laplacian operator only accesses same-level cells by utilizing ghost cells to store temporary values. We launch a block with 128 threads to process a tile. It reads 10^3 cell values to a shared buffer at the beginning for later access.

Due to the presence of the gauge variable φ, free-surface boundary conditions ($p = 0$) cannot be directly applied in the flow map fluid simulation. Therefore, we apply solid boundary conditions at level $l = 1$ on all six sides of the computational domain, ensuring planar solid boundaries.

Algorithm 4 Adaptive Particle-Grid Flow Map Time Evolution

Input: flow map length k, total time steps N

for $i \leftarrow 0$ to N **do**
  $t_e \leftarrow i\Delta t, t_b \leftarrow (i+1)\Delta t$ ▷ Sec. 5.2
  Generate new particles
  if $i \bmod n = 0$ **then**
    $t_a \leftarrow i\Delta t$
    $\boldsymbol{m}^a \leftarrow \boldsymbol{u}^a, \mathcal{F}^{[a,a]} \leftarrow \mathbf{I}$ for all particles
  else
    Compute $\boldsymbol{m}^a, \mathcal{F}^{[b,a]}$ for new particles
  Calculate $\nabla \boldsymbol{m}^e$ for all particles
  Advect particles and $\mathcal{F}^{[b,a]}$
  Remove invalid particles
  Dynamically adjust the grid ▷ Alg. 2
  Transfer $\boldsymbol{m}^b$ to the grid with APIC ▷ Sec. 5.3
  for all grid cells not processed by particle flow map **do**
    Calculate $\boldsymbol{m}^a, \mathcal{F}^{[b,a]}$ ▷ Eq. 6a
    $\boldsymbol{m}^b = \left(\mathcal{F}^{[b,a]}\right)^\top \boldsymbol{m}^a$
  Solve $\nabla^2 p^\star = \nabla \cdot \boldsymbol{m}^b$
  $\boldsymbol{u}^{i+1} \leftarrow \boldsymbol{m}^b - \nabla p^\star$ ▷ Sec. 5.4

6 TIME INTEGRATION

We summarize our time integration algorithm in Alg. 4. A time step begins with particle generation and G2P transfer, then the grid is dynamically adapted to fit the newly advected particle. P2G is performed consequently to calculate $\boldsymbol{m}$ on the grid. For the grid nodes not processed by P2G, we use grid-based flow map advection. Finally, we project $\boldsymbol{m}$ to make it divergence-free, obtaining the velocity field of the next time step.

7 VALIDATION

7.1 Numerical Accuracy

We validate the accuracy of our advection scheme and Poisson solver on an adaptive grid spanning $[0,1]^3$ that is refined at the center. Specifically, the level target function is defined as

$$F(T) = \begin{cases} l_0 + 2, & \text{if } (0.5, 0.5, 0.5) \in T, \\ l_0, & \text{otherwise,} \end{cases} \tag{11}$$

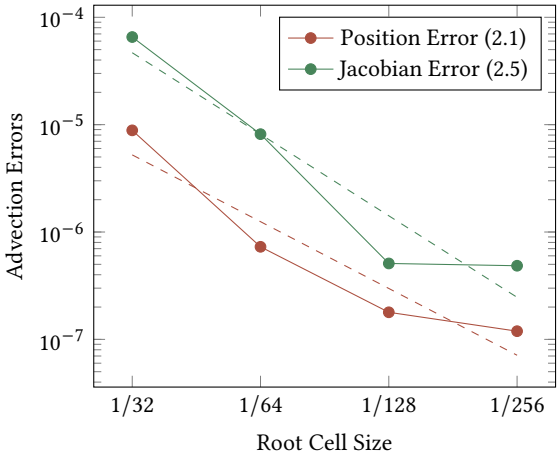

Fig. 8. *Accuracy of flow map advection on adaptive grids with different resolutions. Numbers are the convergence orders fitted by least squares.*

where l_0 is the root level. We tested with $l_0 = 2, \ldots, 5$, corresponding to finest cell sizes $1/128, \ldots, 1/1024$.

Advection. The equivalence of Eq. 6a and Eq. 6b is validated on a 3D deformation velocity field [Marić et al. 2020]

$$\begin{aligned} u(x,y,z,t) &= 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(\pi t/T_0), \\ v(x,y,z,t) &= -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos(\pi t/T_0), \\ w(x,y,z,t) &= -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos(\pi t/T_0), \end{aligned} \tag{12}$$

with $T_0 = 3$. We randomly sample 1M points in the grid and advect 5 time steps of $\Delta t = 1/1024$ each, therefore $t_a = 0, t_b = 5\Delta t$. We first use Eq. 6a to advect particle $\boldsymbol{x}_i$ to $\boldsymbol{x}_i^b$ along with $\mathcal{F}_i^{[a,b]}$, and then use Eq. 6b to advect it back to $\boldsymbol{x}_i$, calculating the same Jacobian matrix in the opposite directions. The position error is defined as the maximum error between the original $\boldsymbol{x}_i$ and the position that has been advected forward and back, and the Jacobian error is defined as the maximum Frobenius error between two Jacobian matrices.

Fig. 8 shows the advection errors on different grid resolutions. The results demonstrate that our advection scheme achieves second-order convergence and approaches single-precision floating-point machine precision at a finest cell size of $1/512$. On the other hand, it consistently approaches machine precision on a uniform grid, indicating that the primary source of the error originates from level switching near T-junctions.

Poisson Solver. To assess the accuracy of our MGPCG solver, we solve a Poisson system with all Neumann boundary conditions, where the solution is a known analytical function [Tomida and Stone 2023]

$$\begin{aligned} f(x,y,z) = &-4\pi GA\left(\left(\frac{2\pi}{L_x}\right)^2 + \left(\frac{2\pi}{L_y}\right)^2 + \left(\frac{2\pi}{L_z}\right)^2\right)^{-1} \\ &\times \sin\left(\frac{2\pi x}{L_x}\right)\sin\left(\frac{2\pi y}{L_y}\right)\sin\left(\frac{2\pi z}{L_z}\right) + \phi_0, \end{aligned}$$

where $L_x = L_y = L_z = G = A = 1$ and $\phi_0 = 0$.

We refer to the grid defined by (11) as Grid A. Additionally, we tested a more complex Grid B, defined similarly to (11) but only refines to level $l_0 + 2$ when tiles intersect a spherical shell centered at $(0.5, 0.5, 0.5)$ with a radius of 0.25. The results, shown in Fig. 9, indicate that our solver achieves second-order accuracy on the simpler Grid A, while exhibiting first-order accuracy on the more complex Grid B. To address the reduced accuracy on Grid B, we also evaluated the approach suggested by Losasso et al. [2006], which computes fluxes at T-junctions using coarser-level cells. This is equivalent to setting the flux across four smaller faces to their average value. The results of this averaging scheme are denoted as *Avg* in Figure 9, demonstrating second-order accuracy on both grids. For comparison, we also tested a uniform grid, under which both algorithms reduce to a standard second-order accurate multigrid. Table 1 compares the number of iterations required for both algorithms to converge to a relative error of 10^{-6}. As shown, the averaging scheme can lead to an increased number of iterations. Therefore, we retain our original method as its order of accuracy suffices to produce plausible simulation results

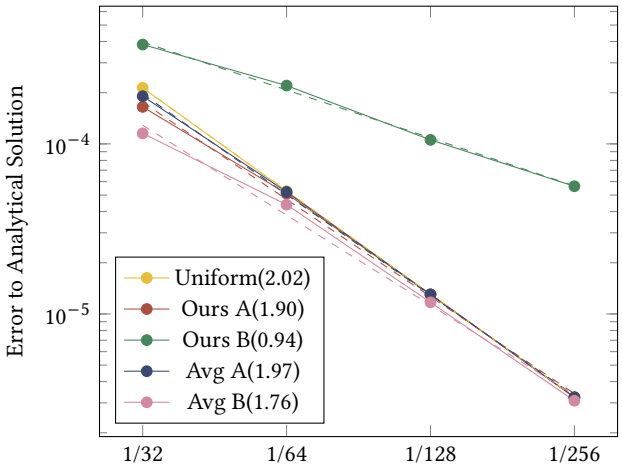

Fig. 9. *Errors of MGPCG solvers on different grids. Numbers are the convergence orders fitted by least squares. Avg represents the averaging scheme [Losasso et al. 2006] at T-junctions.*

Table 1. Iterations for MGPCG solvers to converge to 10^{-6} relative error.

	Root Cell Size			
	1/32	1/64	1/128	1/256
Uniform	11	14	18	23
Ours A	11	15	18	23
Ours B	11	15	19	24
Avg A	17	22	25	28
Avg B	31	36	49	577

Table 2. Comparison of our G2P and P2G operations with naively implemented versions (no use of shared memory). The G2P transfer time includes interpolating $\nabla \boldsymbol{m}$ and one flow map advection step. The P2G transfer time includes calculating Eq. 9 and scattering $\boldsymbol{m}$ to the grid with APIC.

	Naive	Ours	Speed-up
G2P time	115ms	14ms	
G2P throughput	139.13M/s	1142.86M/s	8.21×
P2G time	74ms	39ms	
P2G throughput	216.22M/s	410.26M/s	1.90×

7.2 Time Efficiency

To analyze algorithm performance across varying grid resolutions, we introduce the metrics of cell throughput and particle throughput as measures of efficiency. For adaptive grid methods, we calculate two types of voxel throughput: active and effective. Active voxel throughput measures the processing rate of active leaf cells, whereas effective voxel throughput is determined by the effective resolution. All our runtime statistics are obtained on a desktop equipped with an RTX 4090 GPU and an Intel i9-14900KF processor.

G2P and P2G Transfer. Table 2 compares our optimized G2P and P2G transfers with naively implemented methods on the grid (11) with root level 0 and finest level 6. The G2P runtime measures the efficiency of our monolithic CUDA kernel consisting of interpolating $\nabla \boldsymbol{m}$ and one flow map advection step. Similarly, the P2G runtime includes the calculation of Eq. 9 and impulse scattering with the APIC scheme in a monolithic CUDA kernel.

Table 2 shows that our optimized G2P transfer achieves 8.21× speed up against the naively implemented baseline. This is because the G2P process, especially RK4 advection, requires accessing a large amount of grid data. The use of shared memory significantly reduces the number of global memory accesses in this process. P2G also achieves a 1.90× speedup, attributed to the utilization of shared memory and warp-level operations.

Table 3. Comparison of efficiency with simulation throughput. The projection column refers to the active voxel throughput of the Poisson solver, while the active and effective columns refer to the voxel throughput per time step of the simulator. The Poisson solver runtime of Taichi [2019]is taken from Table 1.

Method	Cells	Projection	Active	Effective
SPGrid	135M	0.26M/s	0.23M/s	3.56M/s
Taichi[2019]	16M	14.16M/s	—	—
PFM (GPU)	2M	24.39M/s	7.04M/s	—
PFM (CPU)	2M	5.97M/s	0.36M/s	—
UAAMG	89.41M	41.59M/s	10.23M/s	—
Ours (PFM)	2M	90.90M/s	15.87M/s	—
Ours (sphere)	1.60M	133.33M/s	10.32M/s	825.81M/s
Ours (aircraft)	21.49M	36.42M/s	11.65M/s	168.78M/s

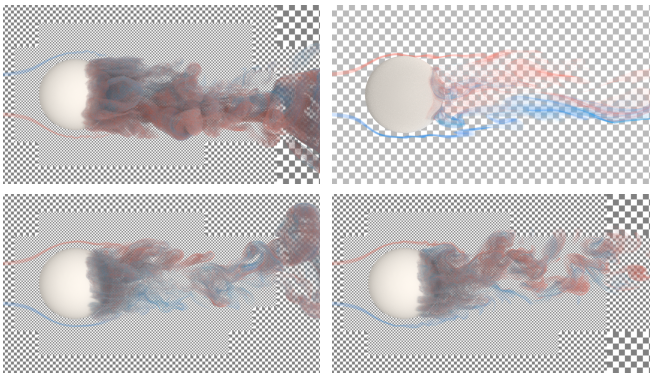

Fig. 10. *Sphere scene with different refinement strategies. Top left: ous (fully adaptive grid). Top right: dense PFM. Bottom left: 2-level adaptivity. Bottom right: 3-level adaptivity. The checkerboard pattern represents the grid structure, each square corresponds to an* 8^3 *tile.*

Projection and Simulation. Table 3 compares the efficiency of different algorithms. The runtime data is taken from the following sources: SPGrid [2014], reported in Table 3 (smoke flow past a sphere); GPU and CPU versions of PFM [2024], evaluated by running their 3D leapfrog case on a 128^3 grid on our machine; and UAAMG [2022], from Table 4 (river fall). We report the sphere case on dense and adaptive grids, and the aircraft case in Sec. 8 for our time statistics. The grid adaptivity and grid advection are disabled for the dense sphere case, following the original PFM setup.

On the same hardware configuration, our optimized implementation of the original PFM on a dense grid achieves a 2× speedup compared to GPU-PFM, and our full algorithm on the adaptive grid demonstrates a 1.5× speedup over GPU-PFM. Moreover, by employing our adaptive grid approach, we achieve another efficiency gain of one to two orders of magnitude in effective resolution throughput.

8 RESULTS

In this section, we present the simulation results. All vorticity fields are visualized directly using volume rendering. Smoke is visualized through a volume rendering of passively advected smoke particles. Detailed runtime statistics are reported in Table 4.

8.1 Ablation Tests

We employ a sphere scene to conduct ablation tests on various configurations of our algorithm. In this test, we placed a small sphere with a diameter of 0.1 in a computational domain $[0, 1]^3$, centered at $(0.5, 0.5, 0.3)$. The inflow and outflow boundary conditions are $\boldsymbol{u} = (0, 0, 0.4)$.

Comparison with PFM. The top row of Fig. 10 compares our method (left) with our implementation of PFM on a 128^3 dense grid. By utilizing adaptive grids, we achieved a 512^3 effective resolution with a time cost similar to PFM, capturing significantly richer fluid details.

Number of Levels. The bottom row of Fig. 10 shows the results of using only 2 or 3 levels for grid adaptivity. It can be observed that the results are similar to our method with full adaptivity, but the runtimes increased due to increased cell counts.

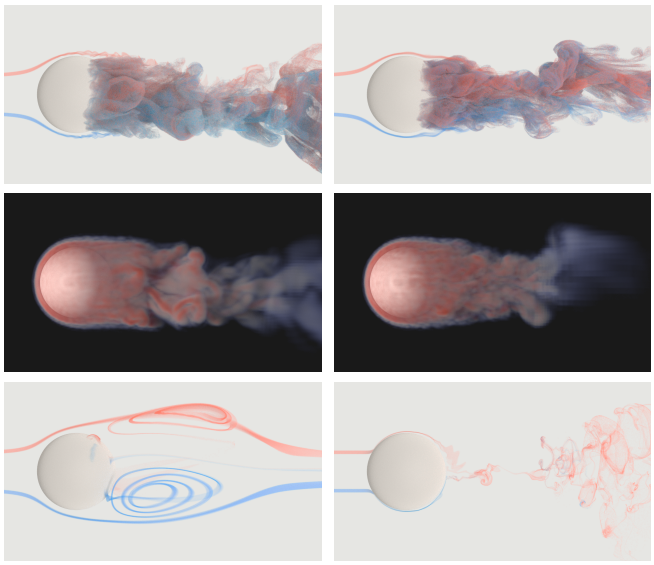

Fig. 11. *Sphere scene with different configurations. Top row: smoke visualizations of particle life* 0.5 *(left) and* 0.25 *(right). Middle row: vorticity visualizations of particle life* 0.5 *(left) and* 0.25 *(right). Bottom left: Semi-Lagrangian advection. Bottom right: 10 Jacobi iterations.*

Particle Life. The top two rows of Fig. 11 compares particle life $\mathcal{L} = 0.5$ (left) and 0.25 (right). We can observe that they produce similar smoke results, but vorticity fades sooner with shorter $\mathcal{L}$. The reason is that the smoke's vortex structure is generated at solid boundaries, and even after the velocity field dissipates, its shape still remains.

Semi-Lagrangian. The bottom-left image in Fig. 11 is the result using Semi-Lagrangian advection, showing smooth laminar flow.

Projection with Jacobi iterations. The bottom-right image in Fig. 11 shows the result using 10 Jacobi iterations instead of the full MGPCG solver, exhibiting visible artifacts due to the compressible velocity field.

Overall, both our flow map advection scheme and MGPCG solver on adaptive grids are necessary to produce a turbulent fluid with rich vortex details.

8.2 Solid Boundaries

We demonstrate our algorithm's ability to handle solid interactions and complex meshes using two classic test cases: the delta wing and the racing car.

Delta Wing. In Fig. 12, we position a delta wing at a 30-degree angle of attack in a flow boundary condition $(0, 0, 1)$. The computational domain spans $1 \times 1 \times 2$ with an effective resolution $512 \times 512 \times 1024$. The length of the delta wing along the z-axis is 0.6. As shown, rich vortex structures of multiple scales form along the edges of the delta wing, and roll upward, clearly exhibiting the

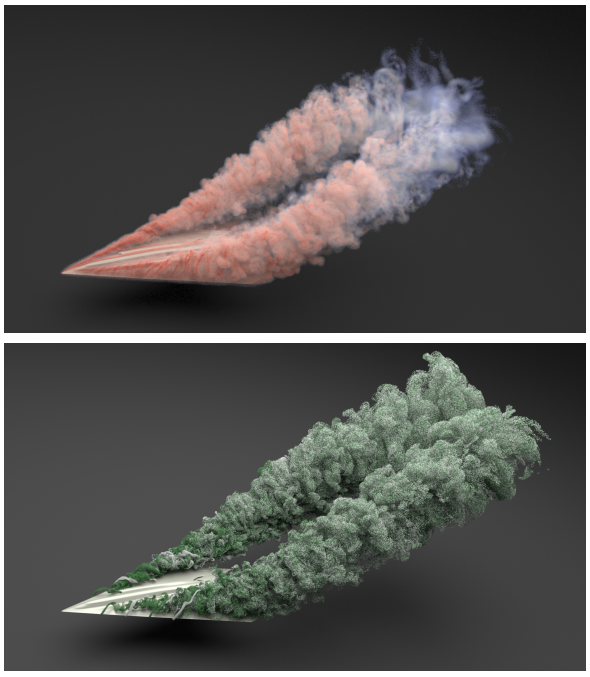

Fig. 12. *Vorticity (top) and smoke (bottom) of the delta wing simulation. The vorticity structure with multiple scales is clearly shown on top of the delta wing.*

Fig. 13. *Racing car, vorticity (top) and smoke (bottom).*

vortex lift effect [Délery 2001]. By employing an adaptive grid, we achieved a high resolution that successfully captured these intricate small-scale vortex structures.

Racing Car. We used a racing car model [Sketchfab 2023] to demonstrate our algorithm's capability to handle the interaction between fluid and complex meshes. The computational domain is also $1 \times 1 \times 2$ with effective resolution $512 \times 512 \times 1024$. The inflow and outflow are $\boldsymbol{u} = (0, 0, 1)$ and the length of the car is 0.9 in the z-axis. Volume rendering visualizations of vorticity and smoke demonstrate that our algorithm can handle complex meshes and produce rich flow details.

8.3 Moving Objects

Flamingo Flock. We used the Blender software [Soni et al. 2023] to create a particle system consisting of animated meshes of a flamingo [Contributors 2023] where the flamingo models flap their wings to simulate a flying flamingo flock. The flamingos flew a long distance as our effective resolution spans $512 \times 512 \times 2048$. We visualize the vorticity and the grid structures of the first and last frames in Fig. 14. This example demonstrates that by employing an adaptive grid, our algorithm is capable of covering a large computational domain, allowing objects to undergo long-range motion while dynamically updating the grid structure as needed. This ensures the continuous capture of the rich vortical structures generated by the solid objects.

Aircraft. Figure Fig. 15 shows an aircraft model [NASA Airborne Science 2025] in flow $(0, 0, 1)$ with 4 rotating propellers at a 15-degree angle of attack. The effective resolution is $512 \times 512 \times 1024$ in a computational domain $1 \times 1 \times 2$, and the length of the aircraft is 0.9. It demonstrates that our algorithm can efficiently simulate moving small objects and effectively reproduce physical phenomena such as wingtip vortices.

Bat. In Fig. 16, we further demonstrate the capability of our algorithm to handle the motion of complex meshes. We simulate a bat flapping its wings and visualize the resulting vorticity field. Even for such a complex mesh, our algorithm produces realistic simulation results with intricate vortex structures.

9 CONCLUSION

9.1 Discussion

In this work, the particle system serves a dual purpose: advecting the impulse field and guiding grid refinement, thereby enabling an innovative implementation of the flow map method on adaptive grids. Furthermore, we developed *Cirrus*, a novel GPU-based fluid simulator designed for adaptive grids, achieving large-scale simulation on consumer-grade GPUs. Through a series of numerical tests and simulations, we validate the accuracy, efficiency and quality of our method, demonstrating its ability to produce highly realistic results, capture fine-scale vortex structures, and simulate the free motion of small objects within large computational domains.

Connection to Previous Works. Several concepts related to adaptive flow maps have already been introduced by previous works. NFM [Deng et al. 2023] employed an adaptive octree grid for storing the velocity buffer but used dense grids for other parts of the algorithm. Narita and Ando [2022] allocate sparse tiles on a dense grid to reduce the computational cost of flow map tracking. PFM [Zhou et al. 2024] introduced the concept of long-short flow maps, where

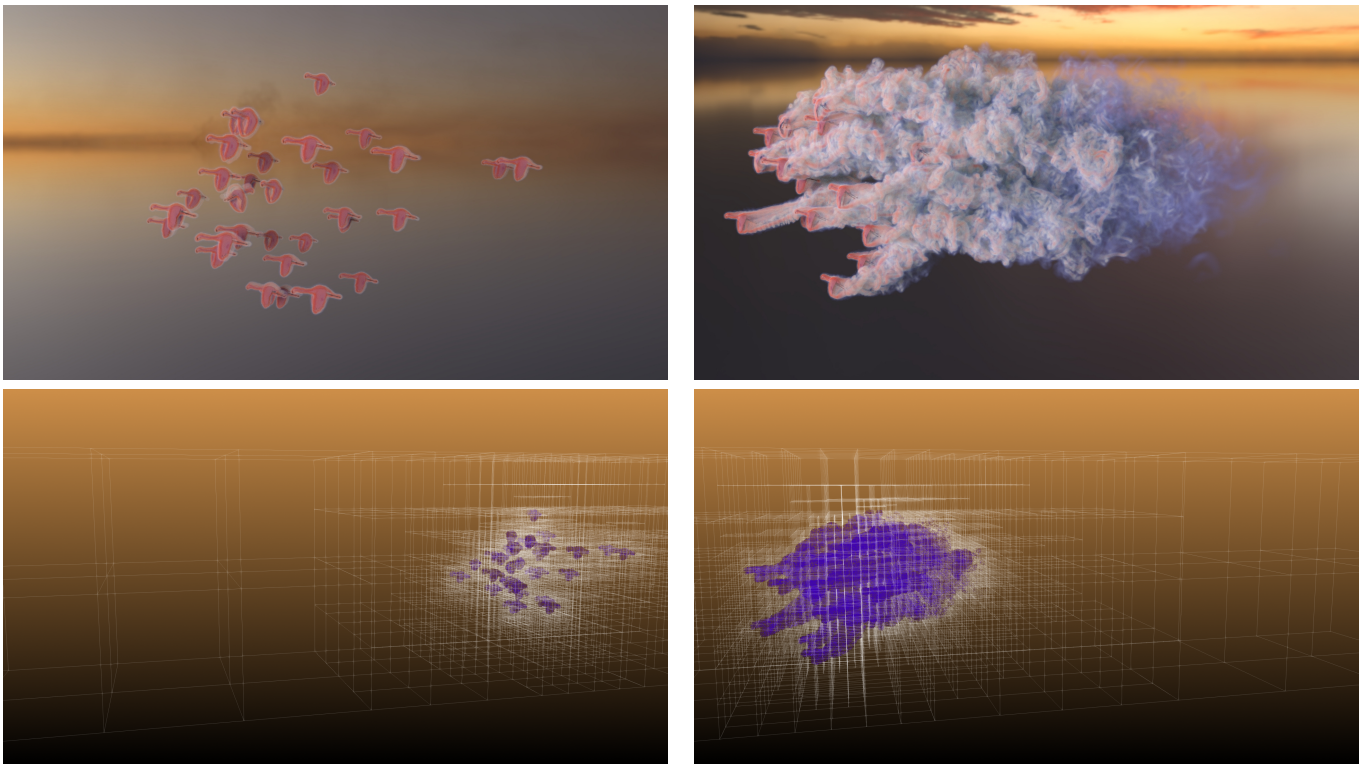

Fig. 14. *Top row: a flamingo flock at frame 1 (left) and 400 (right), visualizing the vorticity field. The camera moves with the object to better showcase the vortex structure in the wake of the bird flock. Bottom row: the leaf tiles of the octree grid, drawn with wireframes and particles (blue). To better illustrate the hierarchical structure of the grid, we fixed the camera position. The bottom row data is taken from a* $256 \times 256 \times 1024$ *simulation, as the tiles in the original data were visually too dense. Each cube corresponds to an* 8^3 *tile.*

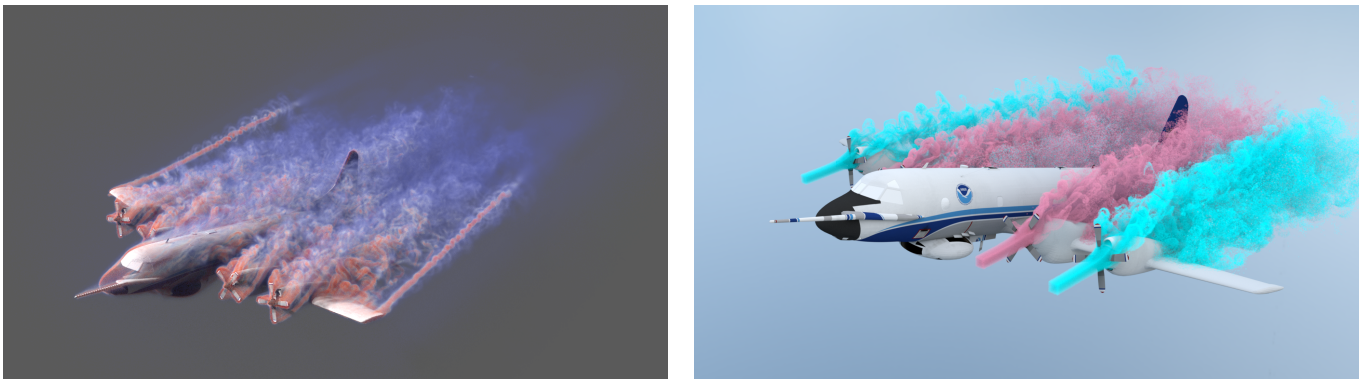

Fig. 15. *Aircraft with 4 rotating propellers, visualizing vorticity (left) and smoke (right). Note the clear wingtip vortex tubes behind the wings.*

a long-range flow map is used to compute $\boldsymbol{m}$ (7a), and a short-range flow map is used to compute $\nabla \boldsymbol{m}$ (7b), as the latter is more sensitive to flow distortions. This approach can be seen as a form of temporal adaptivity; however, it relies entirely on dense grids. Our adaptive hybrid particle-grid flow map is the first fully adaptive flow map-based fluid simulation framework.

Spatial Adaptivity. Spatial adaptivity is important in our algorithm. For a grid-based flow map, each cell needs to store 5 channels for projection [McAdams et al. 2010] and 5×3 channels for the velocity buffer (assuming a flow map length of 5 steps), totaling 20 floating-point numbers per cell. For PFM, each particle requires storage for position, impulse, and impulse gradient matrix, amounting to 15 floating-point numbers. The highest effective resolution of our simulations was $512 \times 512 \times 2048$. Using a dense grid at this resolution would result in a memory cost of 40 GB, while the memory cost for particles would reach 240 GB, far exceeding the capacity of our RTX 4090 GPU. As demonstrated in Sec. 8.1, our adaptive grid

Fig. 16. *A bat flapping its wings, with the vorticity field visualized. Frame 125 (left) and 500 (right).*

Table 4. Statistics of one time step for different simulation scenes. All data are recorded at the last time step. Particle sorting times are included in G2P and P2G. Values in parentheses represent the ratio of effective resolution to the actual active leaf cell count.

Scene	Effective Resolution	$\mathcal{L}$	Leaf Cells	Particles	G2P	P2G	Grid Adv	Projection	Total
Sphere (PFM)	$128 \times 128 \times 128(1\times)$	0.5	2.00M	14.00M	26ms	43ms	—	18ms	126ms
Sphere (2 levels)	$512 \times 512 \times 512(7.56\times)$	0.5	16.93M	2.27M	5ms	8ms	2898ms	181ms	3328ms
Sphere (3 levels)	$512 \times 512 \times 512(39.58\times)$	0.5	3.23M	2.30M	5ms	8ms	174ms	30ms	344ms
Sphere (short $\mathcal{L}$)	$512 \times 512 \times 512(39.58\times)$	0.25	1.21M	1.44M	3ms	5ms	28ms	11ms	105ms
Sphere	$512 \times 512 \times 512(80.17\times)$	0.5	1.60M	2.31M	6ms	10ms	30ms	12ms	155ms
Aircraft	$512 \times 512 \times 1024(11.92\times)$	0.5	21.49M	48.32M	115ms	234ms	560ms	590ms	1845ms
Delta Wing	$512 \times 512 \times 1024(8.68\times)$	1.0	29.50M	28.08M	71ms	151ms	1650ms	468ms	2743ms
Racing Car	$512 \times 512 \times 1024(12.66\times)$	0.5	20.22M	38.41M	98ms	202ms	1704ms	255ms	2533ms
Flamingo	$512 \times 512 \times 2048(20.89\times)$	0.4	24.51M	41.55M	97ms	185ms	2176ms	253ms	3012ms
Bat	$512 \times 512 \times 1024(17.80\times)$	0.5	14.38M	17.88M	42ms	75ms	964ms	160ms	1436ms

system exhibits clear efficiency advantages over both dense grids and simulations employing only 2 or 3 levels of adaptivity, without compromising simulation quality. This spatial adaptivity enables our proposed *Cirrus* system to handle large computational domains, as illustrated in our flamingo flock example.

Hybrid Particle-Grid Method. We use a hybrid particle-grid system to solve the fluid, although pure grid-based systems are also a common method for simulating fluids on adaptive grids [Aanjaneya et al. 2017; Setaluri et al. 2014]. Our algorithm also supports advection solely on grids; however, we still choose the hybrid particle-grid approach, mainly for three reasons. First, the adoption of the impulse gradient in PFM effectively reduces numerical dissipation, which requires a particle system using APIC interpolation. Second, our P2G and G2P operations are straightforward to optimize on the GPU because particles reside at the finest level, and the distance a particle can move in a time step is limited by the CFL number. However, the long-range flow maps computed on the grid inherently disrupt data locality, making them challenging to optimize and computationally expensive. Third, purely adaptive grids often struggle with defining a sizing function. Especially, the advection of time step $i+1$ requires a pre-allocated adaptive grid, yet defining the sizing function ideally depends on the physical quantities at step $i+1$. In our algorithm, we easily solve this by using particles as the grid's refinement indicator.

Multi-Level PFM. Theoretically, it is possible to further improve the simulation quality on coarser levels by placing particles at different refinement levels and using the Particle-Flow Map (PFM) advection. However, this approach would increase the memory overhead and require designing complex interpolation schemes at T-junctions to avoid numerical instability [Gao et al. 2017]. By restricting particles on the finest level, we successfully ensure the simplicity and scalability of our G2P and P2G algorithms.

9.2 Limitations and future works

Lack of full temporal adaptivity. Currently, we use a long-short flow map mechanism, where $\boldsymbol{m}$ and $\nabla \boldsymbol{m}$ are computed at different lengths. However, this is not full temporal adaptivity. Both small and large grid cells share the same flow map length and unified time step. Although the simulation is dominated by the smaller cells, theoretically, using different time steps for small and large cells could further optimize computational efficiency. Future work will explore spatiotemporal adaptivity.

Cross-level discontinuity. Our adaptive flow map advection scheme will fall back to coarser levels when some values in the stencil are missing, resulting in higher cross-level advection errors compared to same-level advection. This introduces numerical discontinuities

at T-junctions and may lead to significantly weaker vorticity preservation compared to the particle flow map, causing the length of solid wake vortices to depend on particle lifetime. We aim to develop new numerical schemes to enhance the accuracy of grid-based flow maps in the future.

T-junction treatment. Our current approach employs constant pressure interpolation at T-junctions and is presently limited to voxelized solid boundaries. While straightforward to implement and optimize on a GPU, this method exhibits reduced accuracy compared to higher-order interpolation techniques or body-fitting tetrahedral meshes, particularly on complex adaptive grids where it can degrade to first-order convergence. Although this accuracy limitation did not significantly impact our smoke simulations, it may introduce artifacts in scenarios involving free surfaces. Furthermore, it demonstrates slower convergence in comparison to MGPCG on uniform grids and notably slower convergence than AMG algorithms [Shao et al. 2022]. Future work will focus on implementing high-order interpolation schemes and fast-converging solvers to address these limitations, thereby enabling the simulation of cut-cell boundaries, free surfaces, and other complex geometric configurations with improved accuracy and efficiency.

More physical models. In the future, we plan to extend *Cirrus* to accommodate more physical models, such as free surface flows and multiphase fluids. We also plan to explore two-way coupling with solid objects on the adaptive grid to enable simulations of more fluid-solid interaction phenomena.

ACKNOWLEDGMENTS

We express our gratitude to the anonymous reviewers for their insightful feedback. Georgia Tech authors acknowledge NSF IIS #2433322, ECCS #2318814, CAREER #2433307, IIS #2106733, OISE #2433313, and CNS #1919647 for funding support. We credit the Houdini education license for video animations.

REFERENCES

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

Ann S Almgren, John B Bell, and William G Szymczak. 1996. A numerical method for the incompressible Navier-Stokes equations based on an approximate projection. *SIAM Journal on Scientific Computing* 17, 2 (1996), 358–369.

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.

Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

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://doi.org/10.1016/j.jcp.2016.11.035

Christopher Batty, Stefan Xenos, and Ben Houston. 2010. Tetrahedral Embedded Boundary Methods for Accurate and Flexible Adaptive Fluids. *Computer Graphics Forum* 29, 2 (2010), 695–704. https://doi.org/10.1111/j.1467-8659.2009.01639.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01639.x

Marsha J Berger and Joseph Oliger. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of computational Physics* 53, 3 (1984), 484–512.

Landon Boyd and Robert Bridson. 2012. MultiFLIP for energetic two-phase fluid simulation. *ACM Transactions on Graphics (TOG)* 31, 2 (2012), 1–12.

Robert Bridson. 2015. *Fluid simulation for computer graphics*. AK Peters/CRC Press.

Duowen Chen, Zhiqi Li, Junwei Zhou, Fan Feng, Tao Du, and Bo Zhu. 2024. Solid-Fluid Interaction on Particle Flow Maps. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–20.

Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O'Brien, and Jonathan R. Shewchuk. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. Eurographics Association, Goslar, DEU, 219–228.

William John Coirier. 1994. *An adaptively-refined, Cartesian, cell-based scheme for the Euler and Navier-Stokes equations*. University of Michigan.

Three.js Contributors. 2023. WebGL GPGPU Birds Example. https://threejs.org/examples/?q=bird#webgl_gpgpu_birds_gltf Accessed: 2025-01-21.

Ricardo Cortez. 1996. An Impulse-Based Approximation of Fluid Motion due to Boundary Forces. *J. Comput. Phys.* 123, 2 (Feb. 1996), 341–353. https://doi.org/10.1006/jcph.1996.0028

Jean M Délery. 2001. Robert Legendre and Henri Werlé: toward the elucidation of three-dimensional separation. *Annual review of fluid mechanics* 33, 1 (2001), 129–154.

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

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.

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

Fan Feng, Jinyuan Liu, Shiying Xiong, Shuqi Yang, Yaorui Zhang, and Bo Zhu. 2022. Impulse fluid simulation. *IEEE Transactions on Visualization and Computer Graphics* 29, 6 (2022), 3081–3092.

Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. 2016. Narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 225–232.

Florian Ferstl, Rüdiger Westermann, and Christian Dick. 2014. Large-Scale Liquid Simulation on Adaptive Hexahedral Grids. *IEEE Transactions on Visualization and Computer Graphics* 20, 10 (2014), 1405–1417. https://doi.org/10.1109/TVCG.2014.2307873

Ming Gao, Andre Pradhana, Xuchen Han, Qi Guo, Grant Kot, Eftychios Sifakis, and Chenfanfu Jiang. 2018a. Animating fluid sediment mixture in particle-laden flows. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11.

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

Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018b. GPU optimization of material point methods. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–12.

Arthur Guittet, Maxime Theillard, and Frédéric Gibou. 2015. A stable projection method for the incompressible Navier–Stokes equations on arbitrary geometries and adaptive Quad/Octrees. *J. Comput. Phys.* 292 (2015), 215–238. https://doi.org/10.1016/j.jcp.2015.03.024

Toshiya Hachisuka. 2005. Combined Lagrangian-Eulerian approach for accurate advection. In *ACM SIGGRAPH 2005 Posters* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 114–es. https://doi.org/10.1145/1186954.1187084

Xuchen Han, Theodore F Gast, Qi Guo, Stephanie Wang, Chenfanfu Jiang, and Joseph Teran. 2019. A hybrid material point method for frictional contact with diverse materials. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019), 1–24.

Babak Hejazialhosseini, Diego Rossinelli, Michael Bergdorf, and Petros Koumoutsakos. 2010. High order finite volume methods on wavelet-adapted grids with local time-stepping on multicore architectures for the simulation of shock-bubble interactions. *J. Comput. Phys.* 229, 22 (2010), 8364–8383.

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

Louis H Howell and John B Bell. 1997. An adaptive mesh projection method for viscous incompressible flow. *SIAM Journal on Scientific Computing* 18, 4 (1997), 996–1013.

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.

Hikaru Ibayashi, Chris Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. 2018. Simulating liquids on dynamically warping grids. *IEEE transactions on visualization and computer graphics* 26, 6 (2018), 2288–2302.

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.

Petr Karnakov, Fabian Wermelinger, Sergey Litvinov, and Petros Koumoutsakos. 2020. Aphros: High performance software for multiphase flows with large scale bubble and drop clusters. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–10.

Alexei M Khokhlov. 1998. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *J. Comput. Phys.* 143, 2 (1998), 519–543.

Byungmoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. 2007. Advections with Significantly Reduced Dissipation and Diffusion. *IEEE Transactions on Visualization and Computer Graphics* 13, 1 (2007), 135–144. https://doi.org/10.1109/TVCG.2007.3

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

Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. 2006. Fluid animation with dynamic meshes. *ACM Trans. Graph.* 25, 3 (July 2006), 820–825. https://doi.org/10.1145/1141911.1141961

Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. 2017. The lattice Boltzmann method. *Springer International Publishing* 10, 978-3 (2017), 4–15.

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. *arXiv preprint arXiv:2409.06246* (2024).

A Lichtl and S Jones. 2015. GPUs to Mars: Full scale simulation of SpaceX's Mars rocket engine. *GTC2015.[accessed 3 Mar. 2019]. Available from Internet: http://on-demand. gputechconf. com/gtc/2015/presentation/S5398-Stephen-Jones-Adam-Lichtl. pdf* (2015).

Mengyun Liu and Xiaopei Liu. 2023. A Parametric Kinetic Solver for Simulating Boundary-Dominated Turbulent Flow Phenomena. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–20.

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.

Frank Losasso, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating water and smoke with an octree data structure. In *Acm siggraph 2004 papers*. 457–462.

Luan Lyu, Xiaohua Ren, Wei Cao, Jian Zhu, and Enhua Wu. 2018. Adaptive narrow band MultiFLIP for efficient two-phase liquid simulation. *Science China. Information Sciences* 61, 11 (2018), 114101.

Tomislav Marić, Douglas B Kothe, and Dieter Bothe. 2020. Unstructured un-split geometrical volume-of-fluid methods–a review. *J. Comput. Phys.* 420 (2020), 109695.

Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids.. In *Symposium on Computer Animation*, Vol. 65. 74.

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.

Ken Museth. 2021. NanoVDB: A GPU-friendly and portable VDB data structure for real-time rendering and simulation. In *ACM SIGGRAPH 2021 Talks*. 1–2.

Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *Acm siggraph 2013 courses*. 1–1.

Mohammad Sina Nabizadeh, Stephanie Wang, Ravi Ramamoorthi, and Albert Chern. 2022. Covector fluids. *ACM Trans. Graph.* 41, 4, Article 113 (July 2022), 16 pages. https://doi.org/10.1145/3528223.3530120

Fumiya Narita and Ryoichi Ando. 2022. Tiled Characteristic Maps for Tracking Detailed Liquid Surfaces. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 231–242.

NASA Airborne Science. 2025. 3D Models Gallery. https://airbornescience.nasa.gov/content/3D_Models_Gallery Accessed: 2025-01-21.

Michael B. Nielsen and Robert Bridson. 2016. Spatially adaptive FLIP fluid simulations in bifrost. In *ACM SIGGRAPH 2016 Talks* (Anaheim, California) *(SIGGRAPH '16)*. Association for Computing Machinery, New York, NY, USA, Article 41, 2 pages. https://doi.org/10.1145/2897839.2927399

Stéphane Popinet. 2003. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of computational physics* 190, 2 (2003), 572–600.

Stéphane Popinet. 2009. An accurate adaptive solver for surface-tension-driven interfacial flows. *J. Comput. Phys.* 228, 16 (2009), 5838–5866.

Ziyin Qu*, Xinxin Zhang*, Ming Gao, Chenfanfu Jiang, and Baoquan Chen. 2019. Efficient and Conservative Fluids with Bidirectional Mapping. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2019)* 38, 4 (2019). (*Joint First Authors).

Wouter Raateland, Torsten Hädrich, Jorge Alejandro Amador Herrera, Daniel T. Banuti, Wojciech Pałubicki, Sören Pirk, Klaus Hildebrandt, and Dominik L. Michels. 2022. DCGrid: An Adaptive Grid Structure for Memory-Constrained Fluid Simulation on the GPU. *Proc. ACM Comput. Graph. Interact. Tech.* 5, 1, Article 3 (May 2022), 14 pages. https://doi.org/10.1145/3522608

Takahiro Sato, Christopher Batty, Takeo Igarashi, and Ryoichi Ando. 2018a. Spatially adaptive long-term semi-Lagrangian method for accurate velocity advection. *Computational Visual Media* 4 (2018), 223–230. https://api.semanticscholar.org/CorpusID:51917726

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* (Bangkok, Thailand) *(SA '17)*. Association for Computing Machinery, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/3145749.3149443

Takahiro Sato, Christopher Wojtan, Nils Thuerey, Takeo Igarashi, and Ryoichi Ando. 2018b. Extended narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 169–177.

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.

Han Shao, Libo Huang, and Dominik L Michels. 2022. A fast unsmoothed aggregation algebraic multigrid framework for the large-scale simulation of incompressible flow. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–18.

Lin Shi and Yizhou Yu. 2002. *Visual Smoke Simulation with Adaptive Octree Refinement*. Technical Report. USA.

Lin Shi and Yizhou Yu. 2004. Visual smoke simulation with adaptive octree refinement. In *Computer Graphics and Imaging*. 13–19.

Sketchfab. 2023. *Oracle Red Bull F1 Car RB19 2023*. https://sketchfab.com/3d-models/oracle-red-bull-f1-car-rb19-2023-e4afe46f3aab4b23a418da06fc163821 Accessed: 2025-01-21.

Lav Soni, Amanpreet Kaur, and Avinash Sharma. 2023. A Review on Different Versions and Interfaces of Blender Software. In *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 882–887.

Yuchen Sun, Linglai Chen, Weiyuan Zeng, Tao Du, Shiying Xiong, and Bo Zhu. 2024. An Impulse Ghost Fluid Method for Simulating Two-Phase Flows. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–12.

Tetsuya Takahashi and Christopher Batty. 2023. A Multilevel Active-Set Preconditioner for Box-Constrained Pressure Poisson Solvers. *Proc. ACM Comput. Graph. Interact. Tech.* 6, 3, Article 50 (Aug. 2023), 22 pages. https://doi.org/10.1145/3606939

Tetsuya Takahashi and Christopher Batty. 2025. A Primal-Dual Box-Constrained QP Pressure Poisson Solver With Topology-Aware Geometry-Inspired Aggregation AMG. *IEEE Transactions on Visualization and Computer Graphics* 31, 4 (2025), 2058–2072. https://doi.org/10.1109/TVCG.2024.3378725

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. https://api.semanticscholar.org/CorpusID:16674280

Jannis Teunissen and Ute Ebert. 2018. Afivo: A framework for quadtree/octree AMR with shared-memory parallelization and geometric multigrid methods. *Computer Physics Communications* 233 (2018), 156–166.

Jannis Teunissen and Francesca Schiavello. 2023. Geometric multigrid method for solving Poisson's equation on octree grids with irregular boundaries. *Computer Physics Communications* 286 (2023), 108665.

Kengo Tomida and James M Stone. 2023. The Athena++ Adaptive Mesh Refinement Framework: Multigrid Solvers for Self-gravity. *The Astrophysical Journal Supplement Series* 266, 1 (2023), 7.

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 Trans. Graph.* 43, 6, Article 268 (Nov. 2024), 14 pages. https://doi.org/10.1145/3687996

Xinlei Wang, Yuxing Qiu, Stuart R Slattery, Yu Fang, Minchen Li, Song-Chun Zhu, Yixin Zhu, Min Tang, Dinesh Manocha, and Chenfanfu Jiang. 2020. A massively parallel and scalable multi-GPU material point method. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 30–1.

David C. Wiggert and Evan Benjamin Wylie. 1976. Numerical predictions of two-dimensional transient groundwater flow by the method of characteristics. *Water Resources Research* 12 (1976), 971–977. https://api.semanticscholar.org/CorpusID:121058330

Xiao Zhai, Fei Hou, Hong Qin, and Aimin Hao. 2018. Fluid simulation with adaptive staggered power particles on GPUs. *IEEE Transactions on Visualization and Computer Graphics* 26, 6 (2018), 2234–2246.

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.

Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. 2013. A new grid structure for domain extension. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

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

A DERIVATION OF THE GENERALIZED FLOW MAP

In this paper, we will adopt the Einstein summation convention by default. We take the Jacobian matrix of flow map $\boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a)$ in the form that components are denoted by subscripts:

$$\mathcal{F}_{ij}^{[a,b]}(\boldsymbol{x}^a) = \frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial x_j^a} \tag{13}$$

Note that $\boldsymbol{x}^a = \boldsymbol{x}(t_a)$ is not a function of t_b, which means

$$\frac{\partial x_i^a}{\partial t_b} = \frac{\partial x_i^a}{\partial t_a}\frac{\partial t_a}{\partial t_b} = 0. \tag{14}$$

The material derivative (full derivative) of $\mathcal{F}$ with respect to t_b thus equals to the partial derivative

$$\frac{\mathrm{D}\mathcal{F}_{ij}^{[a,b]}}{\mathrm{D}t^b} = \frac{\partial \mathcal{F}_{ij}^{[a,b]}}{\partial t_b} = \frac{\partial^2 \phi_i^{[a,b]}(x^a)}{\partial t^b \partial x_j^a} = \frac{\partial}{\partial x_j^a}\frac{\partial \phi_i^{[a,b]}(x^a)}{\partial t_b}. \tag{15}$$

By the definition (3)

$$(\ldots) = \frac{\partial u_i(\boldsymbol{\phi}, t_b)}{\partial x_j^a} = \frac{\partial u_i(\boldsymbol{\phi}, t_b)}{\partial \phi_k}\frac{\partial \phi^k}{\partial x_j^a} = (\nabla \boldsymbol{u}^b \mathcal{F}^{[a,b]})_{ij}. \tag{16}$$

Therefore we have

$$\boxed{\frac{\mathrm{D}\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)}{\mathrm{D}t_b} = \nabla \boldsymbol{u}^b \mathcal{F}^{[a,b]}(\boldsymbol{x}^a).} \tag{17}$$

Then we calculate the material derivatives with respect to t_a. It's easy to see from Eq. 4 that

$$\boldsymbol{\phi}^{[a,b]}(\boldsymbol{\phi}^{[b,a]}(\boldsymbol{x}^1)) = \boldsymbol{x}^b. \tag{18}$$

Similar to Eq. 14 we have $\frac{\partial \boldsymbol{x}^b}{\partial t_a} = 0$. Taking partial derivatives with respect to t_a on both sides of Eq. 18 produces

$$\begin{aligned}
&\frac{\partial \phi_i^{[a,b]}(\boldsymbol{\phi}^{[b,a]}(\boldsymbol{x}^b))}{\partial t_a} \\
&+ \frac{\partial \phi_i^{[a,b]}(\boldsymbol{\phi}^{[b,a]}(\boldsymbol{x}^b))}{\partial \phi_j^{[b,a]}(\boldsymbol{x}^b)}\frac{\partial \phi_j^{[b,a]}(\boldsymbol{x}^b)}{\partial t^0} \\
=&\frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial t_a} + \frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial x_j^a}\frac{\partial x_j^a}{\partial t_a} \\
=&0.
\end{aligned} \tag{19}$$

Note that $\boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a)$ is a function of t_a, t_b and $\boldsymbol{x}^a$, and the terms related to $\partial t_b/\partial t_a$ disappears since it equals to zero. Therefore

$$\begin{aligned}
\frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial t_a} &= -\frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial x_j^a}\frac{\partial x_j^a}{\partial t_a} \\
&= -\frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial x_j^a} u_j^a(\boldsymbol{x}^a).
\end{aligned} \tag{20}$$

By moving the right-hand side of Eq. 20 to the left-hand side, it can be observed that $\frac{\mathrm{D}\boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a)}{\mathrm{D}t_a} = \frac{\mathrm{D}\boldsymbol{x}^b}{\mathrm{D}t_a} = 0$. Intuitively, $\boldsymbol{\phi}^{[a,b]}(\boldsymbol{x}^a)$ always maps the particle back to its position $\boldsymbol{x}^b$ at time t_b. This position $\boldsymbol{x}^b$ can be viewed as an ID carried by the particle, and hence its material derivative is always zero.

Now we take the time derivative of $\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)$ with respect to t_a.

$$\frac{\mathrm{D}\mathcal{F}_{ij}}{\mathrm{D}t_a} = \frac{\partial}{\partial x_j^a}\frac{\partial \phi_i^{[a,b]}(x^a)}{\partial t_a} + u_k(\boldsymbol{x}^a, t_a)\frac{\partial^2 \phi_i^{[a,b]}(x^a)}{\partial x_k^a \partial x_j^a}. \tag{21}$$

Substituting (20) into (21):

$$\begin{aligned}
(\ldots) &= \frac{\partial}{\partial x_j^a}\left(-\frac{\partial \phi_i}{\partial x_k^a}u_k(\boldsymbol{x}^a, t_a)\right) + u_k(\boldsymbol{x}^a, t_a)\frac{\partial^2 \phi_i}{\partial x_k^a \partial x_j^a} \\
&= -\frac{\partial \phi_i^{[a,b]}(\boldsymbol{x}^a)}{\partial x_k^a}\frac{\partial u_k(\boldsymbol{x}^a, t_a)}{\partial x_j^a} \text{ (product rule)} \\
&= -\mathcal{F}_{ik}\nabla \boldsymbol{u}_{kj}^a.
\end{aligned} \tag{22}$$

Thus

$$\boxed{\frac{\mathrm{D}\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)}{\mathrm{D}t_a} = -\mathcal{F}^{[a,b]}(\boldsymbol{x}^a)\nabla \boldsymbol{u}^a.} \tag{23}$$

B GRID REFINEMENT AND COARSENING ALGORITHMS

B.1 Refinement Algorithm

For grid refinement, we calculate and propagate a refinement flag on tiles from the finest level L to the coarsest level 0 based on the rules proposed in Sec. 4.2. In particular, we use ghost tiles as temporary variables to propagate the flag through T-junctions. If a ghost tile G at level i has a neighbor leaf tile T at the same level, and the refinement flag of T is set, G's leaf parent P must also be refined regardless of whether it has reached the level target $F(P)$. Otherwise, the level difference between P and its new leaf neighbor, the refined child of T, will become 2, violating the 1-level constraint. In this case, we set G's refinement flag and propagate to P when processing level $i-1$.

The grid refinement algorithm is given in Alg. 5. Note that the ghost tiles are re-generated at the end because the refinement process only guarantees that the leaf and inner tiles are correctly placed.

Algorithm 5 RefineStep

Input: Adaptive grid $\mathcal{G}$, maximum level L, target level function F
Output: Number of newly added tiles

$N \leftarrow 0$
for each tile T in $\mathcal{G}$ **do**
    T.refine_flag $\leftarrow$ false
for $i = L$ **to** 0 **do**
    for each leaf tile T at level i **do**
        T.refine_flag $\leftarrow (F(T) > i)$
        for each ghost child C of T **do**
            if C.refine_flag **then**
                T.refine_flag $\leftarrow$ true
        if T.refine_flag **then**
            Add 8 leaf children of T at level $i+1$ to $\mathcal{G}$
            $N \leftarrow N + 8$
            Mark T as inner
    for each ghost tile G at level i **do**
        for each leaf tile $H \in \mathcal{N}_i(G)$ **do**
                ▷ $\mathcal{N}_i$: neighbors at level i
            if H.refine_flag **then**
                G.refine_flag $\leftarrow$ true
Generate ghost tiles as in Sec. 4.1
Return N

Algorithm 6 CoarsenStep

Input: adaptive grid $\mathcal{G}$, maximum level L, target level function F
Output: Number of deleted tiles

$N \leftarrow 0$
for each tile T in $\mathcal{G}$ **do**
    T.coarsen_flag $\leftarrow$ false
    T.delete_flag $\leftarrow$ false
for $i = L$ **to** 0 **do**
    for each inner tile S at level i **do**
        S.coarsen_flag $\leftarrow$ true
        for each child C of S **do**
            if C.delete_flag = false **then**
                ▷ The delete flag of an inner tile is always false.
                S.coarsen_flag $\leftarrow$ false
    for each leaf tile T at level i **do**
        T.delete_flag $\leftarrow (F(T) < i)$
        for each inner tile $Q \in \mathcal{N}_i(T)$ **do**
                ▷ $\mathcal{N}_i$: neighbors at level i
            if Q.coarsen_flag = false **then**
                T.delete_flag $\leftarrow$ false
for $i = 0$ **to** L **do**
    for each leaf tile T at level i **do**
        $P \leftarrow$ the parent of T
        T.delete_flag $\leftarrow P$.coarsen_flag
    for each ghost tile G at level i **do**
        $P \leftarrow$ the parent of G
        G.delete_flag $\leftarrow P$.delete_flag
        nb_all_deleted $\leftarrow$ true
        for each leaf neighbor $H \in \mathcal{N}_i(G)$ **do**
            if H.delete_flag = false **then**
                nb_all_deleted $\leftarrow$ false
        if nb_all_deleted **then**
            G.delete_flag $\leftarrow$ true
for $i = 0$ **to** L **do**
    for each tile T at level i **do**
        if T.delete_flag **then**
            Delete T from $\mathcal{G}$
            $N \leftarrow N + 1$
        else if T.coarsen_flag **then**
            Change T's type to leaf
Generate ghost tiles as in Sec. 4.1
Return N

B.2 Grid Coarsening

Similar to the grid refinement, we also perform the grid coarsening algorithm based on two flags in tiles: a coarsening flag and a deletion flag. The deletion flag means that a tile should be deleted, while the coarsening flag means that the children of the tile should be deleted. We calculate and propagate these two flags with a two-pass approach:

Bottom-up Pass. In the first pass, we iterate from the finest level L to the coarsest level 0. At level i, each leaf tile T requests deletion if $F(T) < i$, and all its inner neighbors will become a leaf by setting the coarsening flag, to comply with the 1-level constraint. For each inner tile S, its coarsening flag is set to true only if all 8 of its children requested deletion.

Top-down Pass. In the second pass, we iterate from level 0 to L. At level i, the coarsening flags of inner tiles are propagated to their leaf tile children's deletion flags, and the leaf tiles' deletion flags are propagated to their ghost children. Additionally, if a ghost tile loses all its leaf neighbors, its deletion flag will also be set because it no longer resides at a T-junction.

Finally, all tiles with deletion flags are deleted, and all tiles with coarsening flags are set to the leaf type. We summarize the whole process in Alg. 6.