EULERIAN SMOKE SIMULATION WITH MULTIPLE FIELDS

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of

Master of Science

in

Computer Sciences

by Diyang Zhang

Guarini School of Graduate and Advanced Studies
Dartmouth College
Hanover, New Hampshire

January 2025

Examining Committee:

Bo Zhu, Chair

Lorie Loeb

Soroush Vosoughi

F. Jon Kull, Ph.D.
Dean of the Guarini School of Graduate and Advanced Studies

Abstract

Fluid simulation is a cornerstone of computer graphics, enabling the realistic depiction of dynamic phenomena such as smoke, fire, and other gaseous behaviors. This thesis focuses on advancing Eulerian smoke simulation techniques, with a particular emphasis on grid-based simulations that capture intricate vortical structures and fine visual details.

We propose several detail-preserving frameworks that incorporate various scalar and vector fields within the simulation pipeline, including velocity, impulse, and Lamb vectors, along with their decompositions and alternative representations. By mathematically analyzing the properties of impulse, we derive its scalar fields decomposition (ImpSFD), which introduces an alternative numerical explanation, and Vortex-Particles in Impulse (VPImp) which provides enhanced control over fluid turbulence. Additionally, we integrate error correction post-processing techniques and explore advanced numerical schemes to mitigate dissipation and preserve ultra-fine details.

Furthermore, we initiate discussions on the potential of Lamb vectors, supported by preliminary experiments and results. This vector field, both intuitively and empirically, appears to have a deep connection to the essence of turbulence, warranting further investigation.

This thesis underscores the value of these fields in improving the accuracy, control, and visual quality of fluid simulations, offering directions for future research.

Acknowledgments

Not Available.

Contents

Chapter 1

Introduction

Fluid simulation is an essential component of modern computer graphics, enabling the realistic portrayal of natural phenomena such as water, smoke, fire, snow and explosion. The visual realism of fluid behavior has become increasingly important in fields including visual effects, video games, virtual reality, and scientific visualization. Simulating fluids accurately poses significant computational challenges due to their complex and dynamic nature. As fluid systems involve intricate physical behaviors such as turbulence, diffusion, coupling and boundary interaction, capturing the resulting details in a visually plausible yet computationally feasible manner is an active area of research.

Over the years, various methods have been developed to simulate fluid dynamics, most of which are rooted in the Navier-Stokes equations, which describe the motion of fluid substances [46]. These equations account for essential factors such as velocity, pressure, density, and viscosity, offering a mathematical framework to model the complex interactions that characterize fluid motion.

However, solving the Navier-Stokes equations directly, particularly for turbulent or high-resolution fluid animations, presents significant challenges. For complicated real-world fluid scenarios, the search for a closed-form general analytic solution to

these nonlinear partial differential equations remains an unsolved problem for mathematicians. As a result, in the context of computer graphics, where visual realism must be balanced with computational efficiency, various numerical techniques and optimizations [43] have been employed to approximate fluid dynamics while maintaining visually convincing results.

Numerical methods for fluid simulations can be broadly categorized into Eulerian and Lagrangian approaches. Eulerian methods [14] rely on fixed grids, calculating fluid properties like velocity and pressure around each grid cell discretized in space. On the other hand, Lagrangian methods [29] track the fluid motion as if the observer follows an individual fluid particle as it moves through space and time. Hybrid methods [58] have been additionally introduced to computer graphics with a core representation as a cloud of particles for solving dynamics, but an auxiliary background grid to efficiently enforce boundary conditions and incompressibility.

Alongside core simulation techniques, visual realism can often be enhanced by integrating turbulence models [22], adaptive resolutions and grids [57], numerical scheme optimizations [53], post-processing techniques [21], and advanced rendering methods [18]. Generally, these approaches either strictly adhere to physical principles to improve the physical accuracy of the numerical solution, or they involve non-physical but mathematically inspiring approximations or artificial and aesthetic control.

This thesis explores the techniques and algorithms used in detailed-preserved physics-based Eulerian fluid simulation for computer graphics, focusing on smoke and gaseous behaviors with sophisticated vortical structures, which are among the most challenging to simulate realistically due to their turbulent and sometimes chaotic nature. By building upon various existing interdisciplinary methods, this work aims to investigate, develop and optimize offline simulation frameworks so that while maintaining computational efficiency they are capable of better achieving physical fidelity.

Chapter 2

Preliminaries

Fluids play a vital role in our world, from the air currents that drive weather patterns to the flowing rivers and seas that shape our environment. In computer graphics, simulating fluid behavior has become essential for creating realistic but eye-catching visual effects. Capturing these fluid motions requires a deep understanding of its underlying physics, where central to these simulations are the incompressible Navier-Stokes equations, which govern the motion of fluids. These equations are usually expressed as a set of partial differential equations that describe how the fluid velocity, pressure, and other properties change over time and space. For incompressible fluids, the equations take the form:

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho} \nabla p &= \boldsymbol{g} + \nu \nabla \cdot \nabla \boldsymbol{u} \\ \nabla \cdot \boldsymbol{u} &= 0 \end{aligned} \tag{2.1}$$

These equations are essential to accurately representing the flow and movement of fluids. They seem complex at first glance. We will break them down step by step, explaining the meaning behind each term, to provide a clear foundation for understanding how they are applied in simulations for computer graphics in this chapter.

Section 2.1

The Equations of Fluids

Here we will break down the Navier-Stokes equations 2.1 in detail and explain the physical meaning of each term and symbol. We'll start by focusing on each part of the two equations.

2.1.1. The Momentum Equation

The first differential equation in 2.1, which is a vector equation encompassing three components, is known as the **momentum equation**. At its core, it represents Newton's law, $\boldsymbol{F} = m\boldsymbol{a}$, in a different form. This equation describes how the fluid's acceleration results from the various forces applied to it.

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho}\nabla p = \boldsymbol{g} + \nu \nabla \cdot \nabla \boldsymbol{u} \tag{2.2}$$

In other words, the equation 2.2 describes the momentum conservation for incompressible fluids, while each symbol represents the following:

- $\boldsymbol{u}$: This is the **velocity field** of the fluid, usually written as a vector (u_x, u_y, u_z) representing the velocity of the fluid at every point in space. It describes how fast and in which direction the fluid is moving.

- $\frac{\partial \boldsymbol{u}}{\partial t}$: This term is the **local acceleration**, or rate of change of velocity over time at a fixed point in space. It captures how the fluid velocity is evolving over time.

- $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$: This term represents the **convective acceleration**, or the change in velocity due to the motion of fluid particles themselves. Mathematically, it describes how the velocity changes as you move along the flow direction.

Physically, this accounts for the effects of inertia as fluid elements move through the velocity field.

- ∇p: This is the **pressure gradient**, where p is the **pressure** in the fluid. The gradient operator ∇ applied to pressure indicates how the pressure changes in space. This term drives the fluid from regions of high pressure to low pressure.

- ρ: This represents the **density** of the fluid. $\frac{1}{\rho}\nabla p$ thus relates the pressure forces to fluid density, showing how a change in pressure affects fluid acceleration.

- $\boldsymbol{g}$: This is the external force field, typically **gravity**. It acts as a source of acceleration, causing the fluid to move in response to gravitational forces.

- $\nu\nabla \cdot \nabla \boldsymbol{u}$: This term represents the **viscous forces** in the fluid, where ν is the kinematic viscosity of the fluid (a measure of the fluid's internal resistance to flow). The entire term is the Laplacian of the velocity field, which describes how velocity diffuses or smooths out due to internal friction. This term accounts for the "**stickiness**" or "**thickness**" of the fluid, causing it to resist rapid changes in velocity.

Consider a blob of fluids with mass m, volume V and velocity $\boldsymbol{u}$, by applying Newton's law $\boldsymbol{F} = m\boldsymbol{a}$ we can get its motions so integrating the system forward. On its left hand side, fluid motion is primarily driven by two forces: pressure and viscosity. Pressure arises from imbalances between regions of high and low pressure, exerting a net force on the fluid that pushes it toward lower-pressure areas, mathematically represented as the negative gradient of pressure $-\nabla p$. Viscosity, on the other hand, resists the deformation of fluid by promoting uniformity in velocity between neighboring particles. This force is modeled using the Laplacian operator, $\nabla \cdot \nabla$, which measures the deviation of a fluid particle's velocity from the average in its surroundings.

To compute the force, we have to integrate them over the volume of the fluid blob, say multiply by V on approximation. Adding gravity and gathering them together, we have:

$$\boldsymbol{F} = m\boldsymbol{g} - V\nabla p + V\mu\nabla\cdot\nabla\boldsymbol{u} = m\boldsymbol{a} = m\frac{D\boldsymbol{u}}{Dt} \tag{2.3}$$

which can be clearly rewritten as the original momentum equation 2.2 after rearrangement. It remains to understand what is exactly the acceleration term $\boldsymbol{a} = \frac{D\boldsymbol{u}}{Dt}$.

Material Derivative. Considering fluids as a system of particles each with its own position $\boldsymbol{x}$ (*Lagrangian Viewpoint*). Then, generic quantity q such as density, velocity and temperature can be defined on each particle as well, but as a function of space and time, rather than as of particles themselves. More precisely, the function $q(t, \boldsymbol{x}(t))$ denotes the value of q at time t for the specific particle placed at position $\boldsymbol{x}$ in some domain at that time t (*Eulerian Viewpoint*). It follows by Chain Rule that the total derivatives can be written as the form:

$$\begin{aligned}\frac{d}{dt}(t, \boldsymbol{x}(t)) &= \frac{\partial q}{\partial t} + \nabla q \cdot \frac{d\boldsymbol{x}}{dt} \\ &= \frac{\partial q}{\partial t} + \nabla q \cdot \boldsymbol{u} \\ &= \frac{Dq}{Dt}\end{aligned} \tag{2.4}$$

which is named **material derivative** for quantity q, describing the rate of change of q for the particle placed at position $\boldsymbol{x}(t)$ of time t. For quantity $q = \boldsymbol{u}$ representing the velocity, its material derivative is therefore the acceleration term in Equation 2.3.

2.1.2. Incompressibility

Incompressibility refers to the property of fluids, particularly liquids like water, where their volume remains nearly constant under normal conditions. While real fluids, including both liquids and gases, can change their volume to some extent,

these changes are typically minimal, especially for liquids. Even in gases like air, volume changes are relatively small unless subjected to extreme conditions such as high pressures or shockwaves which result in phenomena like explosions. This minimal change in volume allows us to treat most fluids as incompressible in the context of computer animation, particularly when simulating macroscopic phenomena like splashing water or billowing smoke.

Mathematically, incompressibility is captured by the condition that the divergence of the velocity field must be zero, which means that the **fluid's volume remains unchanged over time**. Pick arbitrarily any volume of fluids named Ω with its boundary $\partial\Omega$, we can apply Fundamental Theorem of Calculus and write:

$$\begin{aligned} \frac{d}{dt}\text{Vol}(\Omega) &= \iint_{\partial\Omega} \boldsymbol{u} \cdot \hat{\boldsymbol{n}} \\ &= \iiint_{\Omega} \nabla \cdot \boldsymbol{u} \\ &= 0 \end{aligned} \tag{2.5}$$

Note that the equation 2.5 holds true for any choice of Ω, any region of fluid. And the only continuous function that integrates to zero in this case is zero itself. Hence we obtain:

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2.6}$$

This is called incompressiblility condition, or with respect to velocity field, the **divergence-free** condition, which is the second part and which can be viewed as the constraint of the incompressible Navier-Stokes equations.

Putting Them Together. The first equation 2.2 describes how fluid velocity changes due to various forces—such as pressure, viscous forces, and external influences like gravity, while the second equation 2.6 enforces the incompressibility condition, ensur-

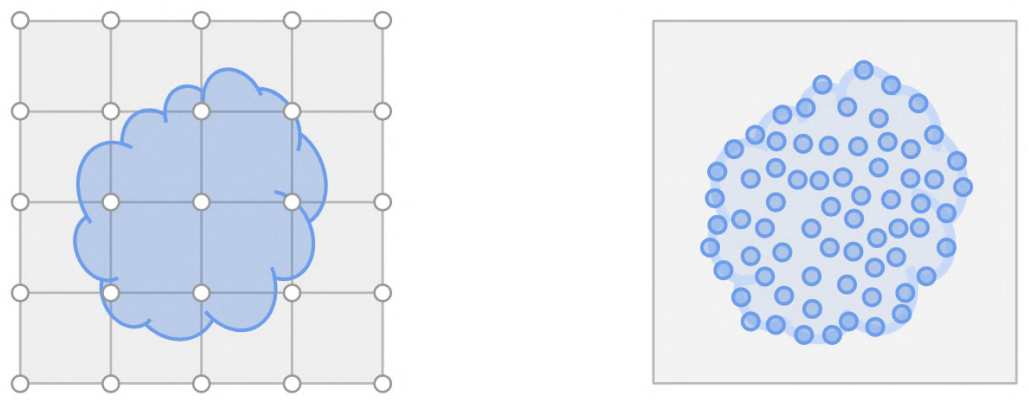
ing that the fluid maintains a constant density and volume.

Together, these two equations 2.1 provide a comprehensive description of fluid behavior, capturing both the motion and the physical constraints that govern fluid flow.

Section 2.2

Lagrangian and Eulerian Viewpoints

In general, there are two ways thinking about the dynamics of a continuum like a fluid and tracking its motion: the Lagrangian viewpoint and the Eulerian viewpoint. See the Figure 2.1 below.



(a) Eulerian Viewpoint (b) Lagrangian Viewpoint

Figure 2.1: Concepts of Eulerian and Lagrangian viewpoint

2.2.1. Lagrangian Viewpoint

The Lagrangian viewpoint treats the continuum as a particle system (Figure 2.1b). Each point in the fluid or solid is labeled as a separate particle, or one molecule, with its own position $\boldsymbol{x}$, velocity $\boldsymbol{u}$, and possibly other properties like mass m, temperature T, and viscosity η. The motion of each particle is described by Newton's second law of motion, which relates the particle's acceleration to the sum of the forces acting on

it. The forces acting on a particle could include pressure gradients, viscosity, gravity, and other external forces from wind sources or obstacles. Over time, the particles move in response to these forces, which altogether results in the simulation of fluid behavior.

There are schemes and methods such as vortex methods [52] [39] [33], smoothed particle hydrodynamics (SPH) [12] [30] [3] [42], Particle-In-Cell (PIC) [58] [19] [34] which are built upon Lagrangian viewpoint. Based on the key idea to treat fluid as a collection of discrete particles, these methods can in general excel at capturing fluid interfaces and handling free-surface flows without complex numerical techniques like level-sets of volume-of-fluids [49], flawlessly track fluid's motion trajectories based on its inherent quantity preservation characteristics [23], and effortlessly manage the boundary and solid-fluid coupling issues through conformal discretization [51].

2.2.2. Eulerian Viewpoint

The Eulerian approach focuses on fixed points in space and measures the changes of fluid quantities as time evolves. For example, for a fixed point in a flow region, the density ρ at a point will decrease as a dense cloud of smoke gradually disperses and no additional gas then flows through; similarly, the temperature T at a point will decrease when an old hot flow passes by and a new cold flow arrives. These quantities are discretely positioned and tracked, approximating nearly each point in the flow field (depending on the resolution of spatial discretization) and describing the contour and changes of fluids in the flow region by capturing the evolution of these quantities over time (Figure 2.1a); rather than being tied to any single fluid particle, as in the Lagrangian viewpoint (Figure 2.1b).

Relying on Eulerian approach, it is easier to numerically calculate the spatial derivatives such as $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$ and $\frac{1}{\rho}\nabla p$ in Equation 2.1 for fluid dynamics within a fixed Eulerian grid rather than on a cluster of arbitrarily moving Lagrangian molecules.

Chapter 3

Eulerian Fluid Simulation

The Eulerian framework is widely used in fluid simulation for computer graphics due to its ability to efficiently model large-scale fluid flows and complex interactions within fixed environments. As we presented in the previous chapter, instead of tracking individual fluid particles, the Eulerian method focuses on the flow of fluid by discretizing the continuous fluid domain into predefined grid cells. Fluid properties such as velocity, pressure, and density are calculated on a fixed grid, naturally allowing for the straightforward application of numerical solvers to the Navier-Stokes equations. Techniques like adaptive grids [40] [35] [50] [57], data structure optimizations [27] [2], higher-order finite differences and interpolations [26] [25] [41] have been further developed to improve the numerical accuracy and computational efficiency, making Eulerian methods a staple in modern fluid simulation techniques in scientific area, delivering realistic visual effects in not only films, video games, other computer-generated imagery but also in rigorous scientific and industrial domains such as robotics, automotive manufacturing, aerospace and astronomy.

In Computer Graphics, building upon the foundational work of Foster and Metaxas [15] and Stam [43], Eulerian fluid simulation pipeline has been established in general through the use of uniform Marker-and-Cell (MAC) grids [16], Chorin's projection

method [7], and the semi-Lagrangian advection scheme [36]. After the construction of basic solvers, researchers began expanding and exploring the Eulerian fluid simulation pipeline in various directions, such as improving computational efficiency [17] [28] [6] [48], reducing numerical errors [38] [21] [53], enhancing simulation details [22] [4], handling complex media coupling and collisions [47] [32] [20], equipping with human interaction and control [8], and combining it with Lagrangian viewpoint to create hybrid approaches that leverage the advantages from both perspectives [19] [55] [56].

Section 3.1

Spatial Discretization

As we have discussed earlier, Eulerian fluid simulation is based on a predefined fixed grid. Pioneering work [16] introduced the marker-and-cell (MAC) method for solving incompressible flow problems, which is particularly effective for enforcing incompressibility, even though it may seem somewhat counterintuitive in most of other aspects. In fact, even today, many state-of-the-art methods still rely on this seemingly redundant spatial discretization approach. We will first begin by introducing how spatial derivatives are calculated under the Eulerian viewpoint, followed by an explanation in detail of the essential advantages of the MAC-Grid setup that have allowed it to remain commonly in use to this day.

3.1.1. Spatial Derivatives Under Eulerian Viewpoint

For any quantity defined in 3-dimension, we have its spatial derivative in a form decoupled in three directions:

$$\nabla q_{i,j,k} = \begin{bmatrix} \partial q_{i,j,k}/\partial x \\ \partial q_{i,j,k}/\partial y \\ \partial q_{i,j,k}/\partial z \end{bmatrix} \tag{3.1}$$

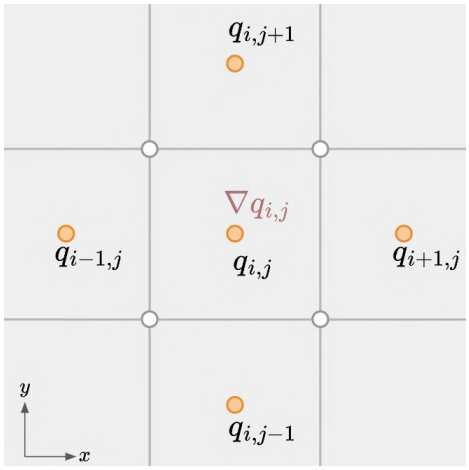


Figure 3.1: The 2-dimensional collocated grid

Now let us take its simplified 2-dimensional case for understanding. When all the quantities (including the spatial derivatives) are stored in the **collocated grid** at the center of grid cell (see Figure 3.7), generally we have two options to compute $\partial q/\partial x$ at cell center, first using forward or backward finite difference, such as

$$(\frac{\partial q}{\partial x})_i = \frac{q_{i+1} - q_i}{\Delta x} \tag{3.2}$$

which is biased to the right and only accurate to $O(\Delta x)$. Otherwise, we can apply more accurate central difference:

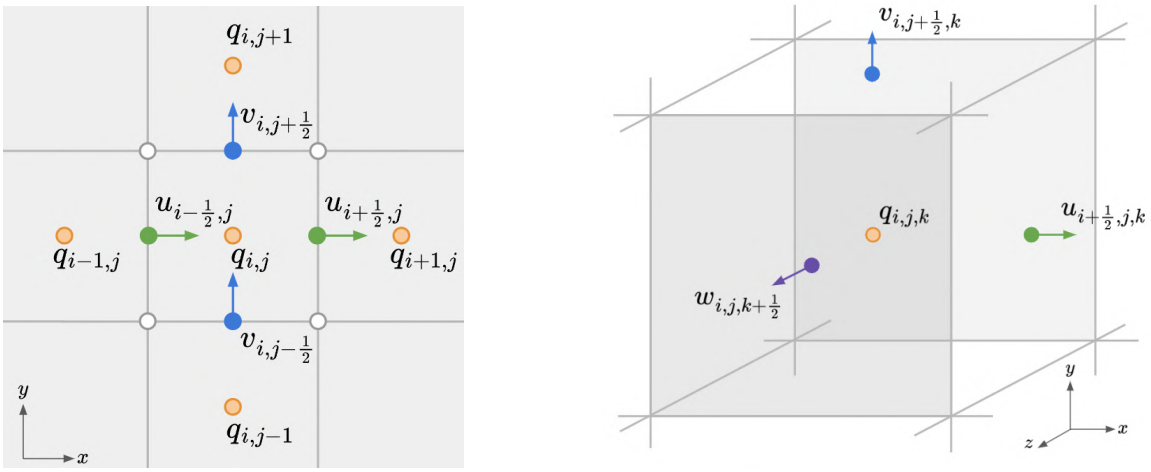
$$(\frac{\partial q}{\partial x})_i = \frac{q_{i+1} - q_{i-1}}{2\Delta x} \tag{3.3}$$

This is indeed an unbiased approach and numerically accurate to $O(\Delta x^2)$. However, on collocated grid, it still has a major problem in that the spatial derivative estimated at cell center (i, j) completely ignores the value $q_{i,j}$ which is originally sampled here. Therefore, as long as the values stored at adjacent grid points are equal, it registers a derivative of zero at (i, j), even for non-constant functions with huge local variations. In other words, this leads to misleading results, creating a so-called non-trivial null-

space, such that oscillating functions for example can still satisfy the central difference formula and be incorrectly classified as continuously having zero derivative based on this collocated spatial discretization.

3.1.2. Marker-And-Cell Grid

The solution to utilize unbiased second-order central finite difference without null-space problem is to dicretize the space via **staggered MAC grid**.



(a) The 2D MAC Grid (b) The 3D MAC Grid

Figure 3.2: Example of MAC Grid

In general, we store the scalar quantities, such as temperature, density and pressure still at **cell centers**, while registering the vector fields such as velocity using a staggered fashion. In 2-dimension, shown as above (Figure 3.2a), the x-component of velocity $\boldsymbol{u}$ is stored along y-directions at **face centers**, oriented in the same direction as the y-direction face normals. Similarly, the y-component of velocity follows the reverse.

Now, the derivative at (i, j) along x-direction can be estimated by central difference as in a general form:

$$(\frac{\partial q}{\partial x})_i = \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x} \tag{3.4}$$

It is unbiased, still accurate to $O(\Delta x^2)$, and not skipping over any values of q as in previous setup with formula 3.3.

While the staggered MAC grid is therefore perfectly suitable wherever we need to compute a derivative in the pressure solve for handling the incompressibility condition 2.6, it is painful at the face centers as we only have half information of the velocity field tracked at that location (i.e., just one directional component explicitly stored on each face). In order to reconstruct the complete vector information at any points across the grid, numerical interpolation has to be performed. Taking averages as the simplest approach, we have

$$\begin{aligned} \boldsymbol{u}_{i,j} &= (\frac{u_{i-1/2,j} + u_{i+1/2,j}}{2}, \quad \frac{v_{i-1/2,j} + v_{i+1/2,j}}{2}) \\ \boldsymbol{u}_{i+1/2,j} &= (u_{i+1/2,j}, \quad \frac{v_{i,j-1/2} + v_{i,j+1/2} + v_{i+1,j-1/2} + v_{i+1,j+1/2}}{4}) \\ \boldsymbol{u}_{i,j+1/2} &= (\frac{u_{i-1/2,j} + u_{i+1/2,j} + u_{i-1/2,j+1} + u_{i+1/2,j+1}}{4}, \quad v_{i,j+1/2}) \end{aligned} \tag{3.5}$$

In 3-dimension, the MAC grid also applies the same staggered pattern as shown in Figure 3.2b. We similarly store scalar quantities at cell center and velocity components on face center respectively.

Section 3.2

Advection-Projection Scheme

After spatial discretization, we obtain the basic data structure prerequisite for numerically solving the Navier-Stokes Equations 2.1. The upcoming challenge is to compute the quantities that we concern about with time evolution. For such partial differential equations, the general strategy is to apply **time-splitting**. In our case, for inviscous

incompressible fluids, we can rewrite 2.1 as:

$$\begin{aligned} \frac{Dq}{Dt} &= 0 \qquad \text{(Advection)} \\ \frac{\partial \boldsymbol{u}}{\partial t} &= \boldsymbol{f}_{\text{ext}} \qquad \text{(Apply Force)} \\ \frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0 \quad &\text{such that } \nabla \cdot \boldsymbol{u} = 0 \qquad \text{(Projection)} \end{aligned} \tag{3.6}$$

For advection term, not only the velocity u itself but also other generic quantities that evolve over time with velocity updates can be advected based on their own material derivatives. We will look at the numerical method solving this advection equation in the following sections.

For forcing term, simplest forward Euler $\boldsymbol{u}^* = \boldsymbol{u} + \Delta t \cdot \boldsymbol{f}_{ext}$ or other higher-order explicit time integration schemes can be directly applied.

For projection term, we will introduce the algorithm commonly used which calculate the right pressure p under the divergence-free constraint of $\boldsymbol{u}$ with boundary conditions.

Altogether, the fundamental advection-projection algorithm can be written as

Algorithm 1 Advection-Projection

Input: Divergence-free velocity field $\boldsymbol{u}^n$ at $t = n$
Output: Divergence-free velocity field $\boldsymbol{u}^{n+1}$ at $t = n + 1$
1: Compute a good time step Δt
2: $\tilde{\boldsymbol{u}}^{n+1} = \mathit{Advect}(\boldsymbol{u}^n, \Delta t; \quad \boldsymbol{u}^n)$
3: $\tilde{\boldsymbol{u}}^{n+1} = \mathit{ApplyForce}(\boldsymbol{f}_{\text{ext}}; \quad \tilde{\boldsymbol{u}}^{n+1})$
4: $\boldsymbol{u}^{n+1} = \mathit{Project}(\tilde{\boldsymbol{u}}^{n+1})$

For timestep size Δt, it is undesirable for it to be too small, as this would result in inefficient computation, nor should it be too large, as this could introduce significant numerical errors. A widely used model relies on the **CFL condition**, which requires

that for a sufficiently small constant parameter C,

$$\Delta t \leq \frac{C\Delta x}{|\boldsymbol{u}|} \tag{3.7}$$

where Δx denotes the sidelength of each grid cell.

3.2.1. Advection

As simplification, write out the advection equation $Dq/Dt = 0$ in 3.6 in 1-dimension:

$$\frac{\partial q}{\partial t} + \boldsymbol{u}\frac{\partial q}{\partial u} = 0 \tag{3.8}$$

A direct way to solve this equation is to separate it into two parts explicitly, as time discretization and spatial distretization. If we use forward Euler,

$$q_i^{n+1} = q_i^n + \Delta t \cdot \boldsymbol{u}_i \frac{\partial q_i}{\partial x} \tag{3.9}$$

From the numerical perspective, it's unconditionally unstable to only first-order accuracy, leading to both significant computational inaccuracies and eventual systematic blow-up. And even thought it's possible to employ some much more stable and exact time integration scheme, the spatial discretization also induces troubles, even leveraging the MAC grid.

For scalar quantities stored at cell center, as we discussed about Equation 3.3, the central difference may directly result in null-space problems. For vector fields registered at face centers, although we can assume that their spatial derivatives can be correctly calculated at the cell centers, this value ultimately needs to be re-applied back to the face centers. This requires at least two interpolations for each velocity sampling point across the entire grid. Moreover, for each time step the vector fields also depend on their predecessor from the previous time step. Such a great amount

of numerical operations that repeatedly grow linearly with time may potentially accumulate unpredictable effects on the entire simulation system.

Semi-Lagrangian Advection Scheme. The fundamental approach in Computer Graphics introduced by [43] to stably solve the advection equation is so-called **Semi-Lagrangian** method. Essentially it's physically motivated and based on the *method of characteristics* for partial differential equations from mathematics, which can both practically and analytically result in an unconditionally stable solver no matter how large the timestep size is, and which can be understood intuitively.

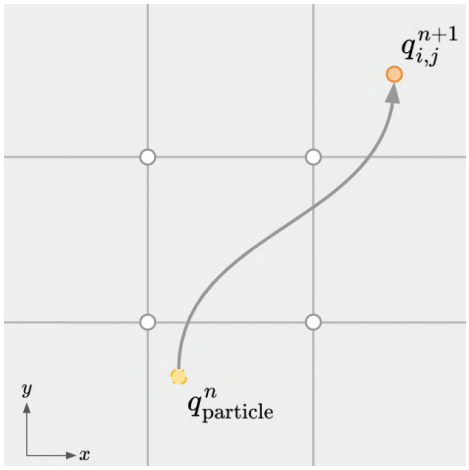


Figure 3.3: Concept of Semi-Lagrangian advection scheme

To solve the value of quantity q at grid cell indexed (i, j) at time $t = n + 1$, see the Figure 3.3 above, we imagine that there exists a Lagrangian particle at that point, and using velocity field information to trace back the position of this imaginary particle at previous timestep $t = n$, and finally look up this value by interpolating the old values on the grid. Mathematically, we can write

$$\begin{aligned} q_{i,j}^{n+1} &= q^{n+1}(x^{n+1}) \qquad \text{(Imagining Lagrangian Particle)} \\ &= q^{n}(x^{n}) \qquad \text{(Traceback)} \\ &= \mathit{Interpolate}(q^{n}, x^{n+1} - \Delta t \cdot u^{n}) \qquad \text{(Interpolation)} \end{aligned} \tag{3.10}$$

If we rewrite the above in 1-dimesnion as in 3.9, assuming $\Delta t \cdot \boldsymbol{u}^n$, we have

$$\begin{aligned} q_i^{n+1} &= \frac{\Delta t \cdot \boldsymbol{u}^n}{\Delta x} q_{i-1}^n + (1 - \frac{\Delta t \cdot \boldsymbol{u}^n}{\Delta x}) q_i^n \\ \Rightarrow q_i^{n+1} &= q_i^n - \Delta t \cdot \boldsymbol{u}^n \frac{q_i^n - q_i^{n-1}}{\Delta x} \end{aligned} \tag{3.11}$$

It remains highly similar to Equation 3.9 and is, in essence, still a forward Euler scheme but with a "velocity-aware" one-sided finite difference method. In computational fluid dynamics, it is called **upwind scheme**, which estimates the derivatives using a set of data points biased to be more "upwind" of the query point, with respect to the direction of the flow. This method can dramatically reduce numerical oscillations, smooth the entire solved vector field, eventually alleviate numerical noise in the visualization and ensure the stability of the entire system. To second-order accuracy, applying the **Rugge-Kutta 2 (RK2) method**, we write the Semi-Lagrangian advection algorithm:

Algorithm 2 RK2 Semi-Lagrangian Advection

Input: $\boldsymbol{q}^n$, $\boldsymbol{x}^n$
Output: $\boldsymbol{q}^*$

1: $\boldsymbol{x}_{\text{mid}} = x^n - \frac{1}{2}\Delta t \cdot u^n(x^n)$
2: $\boldsymbol{x}_{\text{prev}} = x^n - \Delta t \cdot u^n(x_{\text{mid}})$
3: $\boldsymbol{q}^* = Interpolate(\boldsymbol{q}^n, \boldsymbol{x}_{\text{prev}})$

where $\boldsymbol{q}^n$ is the field of quantity we aim to advect at time $t = n$, $\boldsymbol{x}^n$ represents the field of locations where this quantity is stored on the grid, and $\boldsymbol{q}^*$ is the advected quantity field prepared for the next timestep. The trajectory tracing the imaginary particle's position is spitted into substeps at its halfway, and the velocity value interpolated at this intermediate position is used to calculate the final velocity for better accuracy.

Admittedly, the Semi-Lagrangian advection scheme is unconditionally stable. However, modified wavenumber analysis shows that even utilizing high-order time inte-

gration schemes, with decent timestep size for computational efficiency, it introduces severe numerical diffusion and dissipation in regions with large gradients. This occurs intrinsically because sharp gradients require high wavenumbers for accurate representation, which the scheme inherently struggles to preserve. We will introduce some methods to mitigate this issue in the following sections.

3.2.2. Projection

It remains to solve the incompressiblity on the advected velocity field u^*. It guarantees the conservation law of mass, enforces the velocity field to be divergence-free at the end of each step, so that the numerical simulation can smoothly capture the natural flow dynamics, without producing unrealistic artifacts such as "expanding" or "shrinking" flows which may eventually result in instability of the entire system. Rewrite the last equation in 3.6, we obtain

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \Delta t \cdot \frac{1}{\rho} \nabla p \qquad \text{such that} \quad \nabla \cdot \boldsymbol{u}^{n+1} = 0 \tag{3.12}$$

which has to additionally satisfy some **boundary conditions** of PDEs (for fluids, free-surface boundary condition and the solid wall boundary condition) in order to obtain explicit solutions provided with any arbitrary scenes.

On free-surface, where the interface between fluids and air happens, the motion of fluids behaves "freely", allowing it to naturally deform, flow or splash. We **directly specify the value** of pressure at the free-surface boundary to be zero. In the study of PDEs, this is called **Dirichlet boundary condition**.

$$p = 0 \qquad \text{on free surfaces} \tag{3.13}$$

On the solid wall, where collisions between fluids and solids occur, the motion of

the fluids is obstructed by the solid boundary and cannot penetrate naturally along its original flow trajectory. In such cases, we require the velocity of the fluid to match the velocity of the solid.

$$\boldsymbol{u}^{n+1} \cdot \hat{\boldsymbol{n}} = \boldsymbol{u}_{\text{solid}} \cdot \hat{\boldsymbol{n}} \qquad \text{at solid boundaries} \tag{3.14}$$

Together with 3.12, this can be rewritten as $\frac{\Delta t}{\rho}\nabla p \cdot \hat{\boldsymbol{n}} = (\boldsymbol{u} - \boldsymbol{u}_{\text{solid}}) \cdot \hat{\boldsymbol{n}}$, which in essence **specify the value of derivatives** of pressure on the boundary. In the study of PDEs, this is called **Neumann Boundary Condition**.

Now, back to 3.12, applying divergence operator on both sides, we have

$$-\frac{\Delta t}{\rho}\nabla \cdot \nabla p = -\nabla \cdot \boldsymbol{u}^* \tag{3.15}$$

Therefore, for every grid cell, there is one unknown $p_{i,j}$ and one equation in discrete form

$$\frac{\Delta t}{\rho \Delta x^2}(4p_{i,j} - p_{i+1,j} - p_{i-1,j} - p_{i,j+1} - p_{i,j-1}) = -\frac{u^*_{i+1/2,j} - u^*_{i-1/2,j} + v^*_{i,j+1/2} - v^*_{i,j-1/2}}{\Delta x} \tag{3.16}$$

Along with the boundary conditions we just explained above, this induces a linear system in the form of

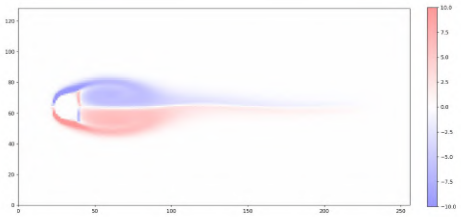
$$Ap = -d \tag{3.17}$$

Here, the right hand side is the negative divergence of velocity which can be numerically approximated at cell centers using central differences across the MAC Grid. And on the left hand side, we can write $A = \frac{\Delta t}{\rho \Delta x^2} M$, where M is a sparse matrix of the same dimension as grid resolution, encoding both the Dirichlet and Neumann boundary information. Multiple well-established linear solvers can be applied to this linear system with sparse matrix, among which preconditioned conjugate gradient

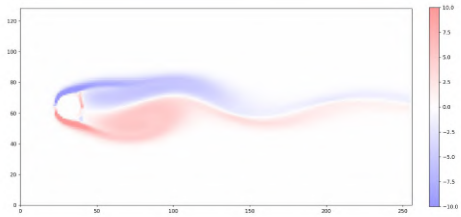
(PCG) [28] is currently in common use for fluids.

3.2.3. Stable Fluids

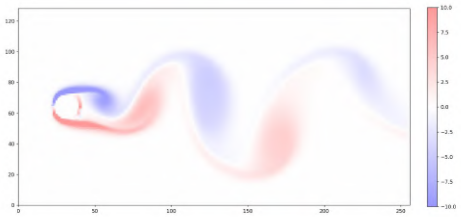
Putting them together, this illustrates how we develop the Advection-Projection Algorithm 1, the core of the Stable Fluids (SF) method [43]. Here we show some of the results based on our implementation.



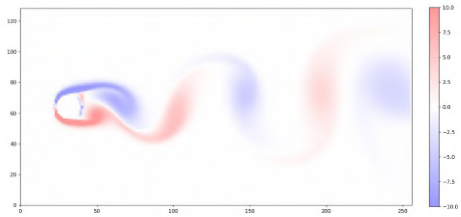
(a) 50th frame



(b) 80th frame



(c) 110th frame

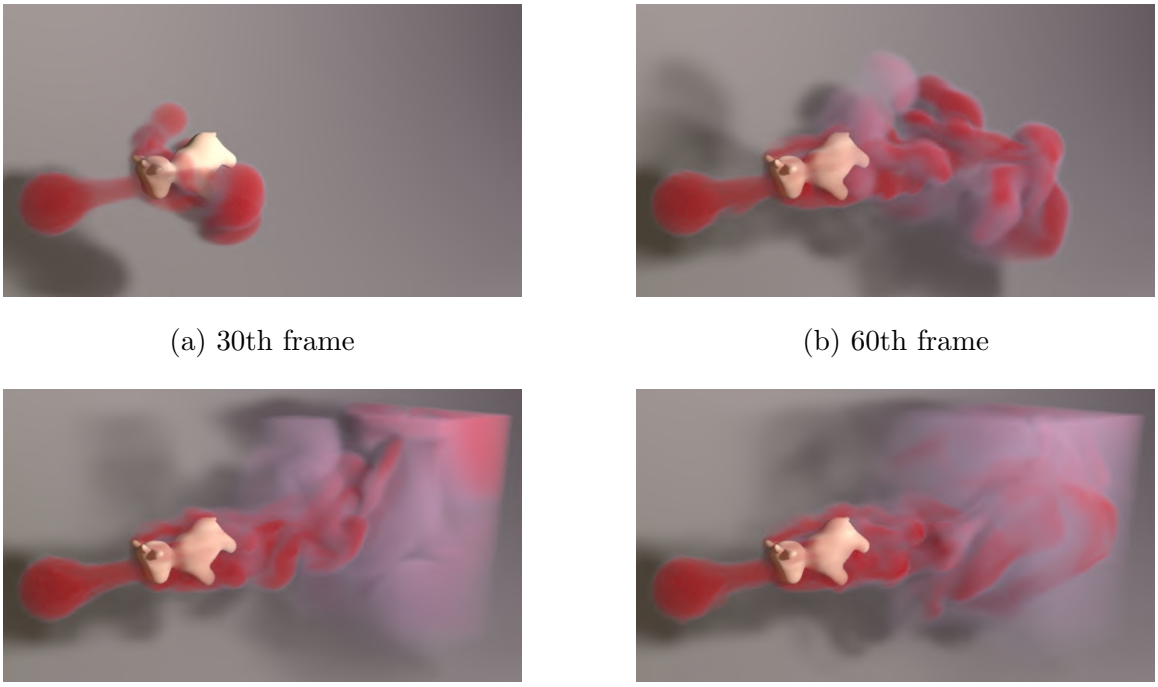


(d) 140th frame

Figure 3.4: 2D Karman Vortex Street [Vorticity]: Stable Fluids

Karman Vortex Street. Figure 3.4 shows the simulation results of Karman Vortex Street [1] in 2-Dimension, a classical example in computational fluid dynamics, demonstrating a repeating pattern of swirling vortices caused by a process known as vortex shedding, which is responsible for the unsteady separation of flow of a fluid around blunt bodies. Specifically, the flow enters the domain at a certain velocity from the left boundary and interacts with a fixed circular obstacle within the domain, gradually forming oscillating vortices over time. The simulation runs in 128×256 resolution, and the magnitude of vorticity values is visualized.

Smoke Plume and Solid. Stable fluids also works smoothly in 3-Dimension. Here we show a smoke plume interacting with a solid cartoon cow object [10] fixed at some world positions during the simulation process. A slight perturbation is added to a spherical smoke source, and it is enforced to continuously emit smoke in some fixed directions within each timestep. After colliding with the solid cow and the fluid domain boundaries, it produces the visual effects shown in Figure 3.5. The simulation runs in $128 \times 256 \times 128$ resolution, and the density field advected along with velocity is stored for visualization and for rendering.



(a) 30th frame (b) 60th frame

(c) 90th frame (d) 120th frame

Figure 3.5: 3D Smoke Plume and Solid [Density]: Sable Fluids

Section 3.3

Detail Preservation

As we previously discussed, Advection-Projection (SF) methods for fluid animation are widely valued for their stability and efficiency. However, it is not only the numer-

ical inaccuracies of the Semi-Lagrangian advection step that contribute to the issue; the Advection-Projection scheme's inherent approach of simply separating the original Navier-Stokes Equations into only two separate parts through time-splitting also introduces energy dissipation into the system, resulting in artificial viscosity and the suppression of small-scale details. Therefore, researchers began focusing on finding concise and efficient alternative approaches around Advection-Projection method to reduce its numerical errors and preserve fluid simulation details without significantly increasing computational cost or memory overhead.

3.3.1. Error Correction

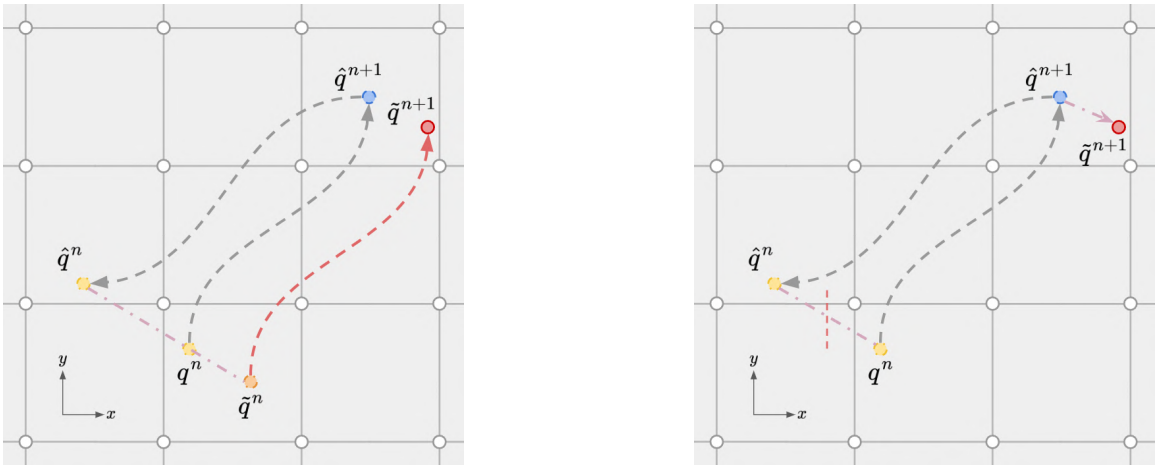
Back-and-Forth Error Correction (BFECC) [21] and MacCormack method (MC) [38] are two well-established, early-proposed, and currently widely accepted post-processing error correction techniques that are appreciated by graphics researchers for their significant improvements in fluid simulation. Both methods encode a similar idea: to find a reliable way to approximate advection errors and then, based on this error estimate, compensate for the inaccuracies in the original numerical solution.

Algorithm 3 Back-and-Forth Error Correction (BFECC)

Input: $\boldsymbol{u}^n$
Output: $\boldsymbol{u}^{n+1}$
1: $\hat{\boldsymbol{u}}^{n+1} = Advect(\boldsymbol{u}^n, \Delta t; \quad \boldsymbol{u}^n)$
2: $\hat{\boldsymbol{u}}^{n} = Advect(-u^n, \Delta t; \quad \hat{\boldsymbol{u}}^{n+1})$
3: $\tilde{\boldsymbol{u}}^n = \boldsymbol{u}^n + (\boldsymbol{u}^n - \hat{\boldsymbol{u}}^n)/2$
4: $\tilde{\boldsymbol{u}}^{n+1} = Advect(\boldsymbol{u}^n, \Delta t; \quad \tilde{\boldsymbol{u}}^n)$
5: $\boldsymbol{u}^{n+1} = Project(\tilde{\boldsymbol{u}}^{n+1})$

Intuitively, an imaginary fluid particle on the grid following the Semi-Lagrangian advection scheme should return to its original position after moving back and forth from its starting point. Therefore, naturally in practice, the numerical error between this intermediate position (after being pushed forward and then dragged back) and its starting point represents the value that needs to be compensated. Based on this

error estimate, we correct the original starting position and perform the advection again using the original velocity. This forms the idea of BFECC, see Figure 3.6a.



(a) BFECC (b) MacCormack Method

Figure 3.6: Concepts of error correction techniques

Algorithm 4 MacCormack Method

Input: $\boldsymbol{u}^n$
Output: $\boldsymbol{u}^{n+1}$
1: $\hat{\boldsymbol{u}}^{n+1} = Advect(\boldsymbol{u}^n, \Delta t; \quad \boldsymbol{u}^n)$
2: $\hat{\boldsymbol{u}}^n = Advect(-\boldsymbol{u}^n, \Delta t; \quad \hat{\boldsymbol{u}}^{n+1})$
3: $\tilde{\boldsymbol{u}}^{n+1} = \hat{\boldsymbol{u}}^{n+1} + (\boldsymbol{u}^n - \hat{\boldsymbol{u}}^n)/2$
4: $\boldsymbol{u}^{n+1} = Project(\tilde{\boldsymbol{u}}^{n+1})$

MacCormack method (MC, Algorithm 4), see Figure 3.6b, similarly, attempts to move the imaginary fluid particle back and forth to account for errors, but it differs from BFECC in how it utilizes this error estimate. Essentially, MacCormack can be seen as a cheaper version of BFECC. Instead of correcting the starting point and re-performing the advection, MC directly penalizes the error value on the intermediate result from the first advection step, avoiding an additional numerical time evolution. Both methods can dramatically improve the original advection-projection scheme, and the underlying ideas are broadly applicable to optimizing other numerical discretization schemes for PDEs.

3.3.2. Advection-Reflection

The Advection-Reflection (R) Method [53] offers another approach to addressing the problem of numerical dissipation. Instead of acknowledging and estimating advection errors to make error-based adjustments, it takes a more macroscopic perspective on the inherent issues of time splitting in the projection scheme. Although, apparently, or even from the point of view of numeric, the algorithm written may seem somewhat similar to the aforementioned two methods, it attempts to radically rearrange and modify the time-splitting scheme itself rather than rely on post-processing. This concise approach has also inspired amount of new ideas among graphics researchers.

Algorithm 5 Advection-Reflection

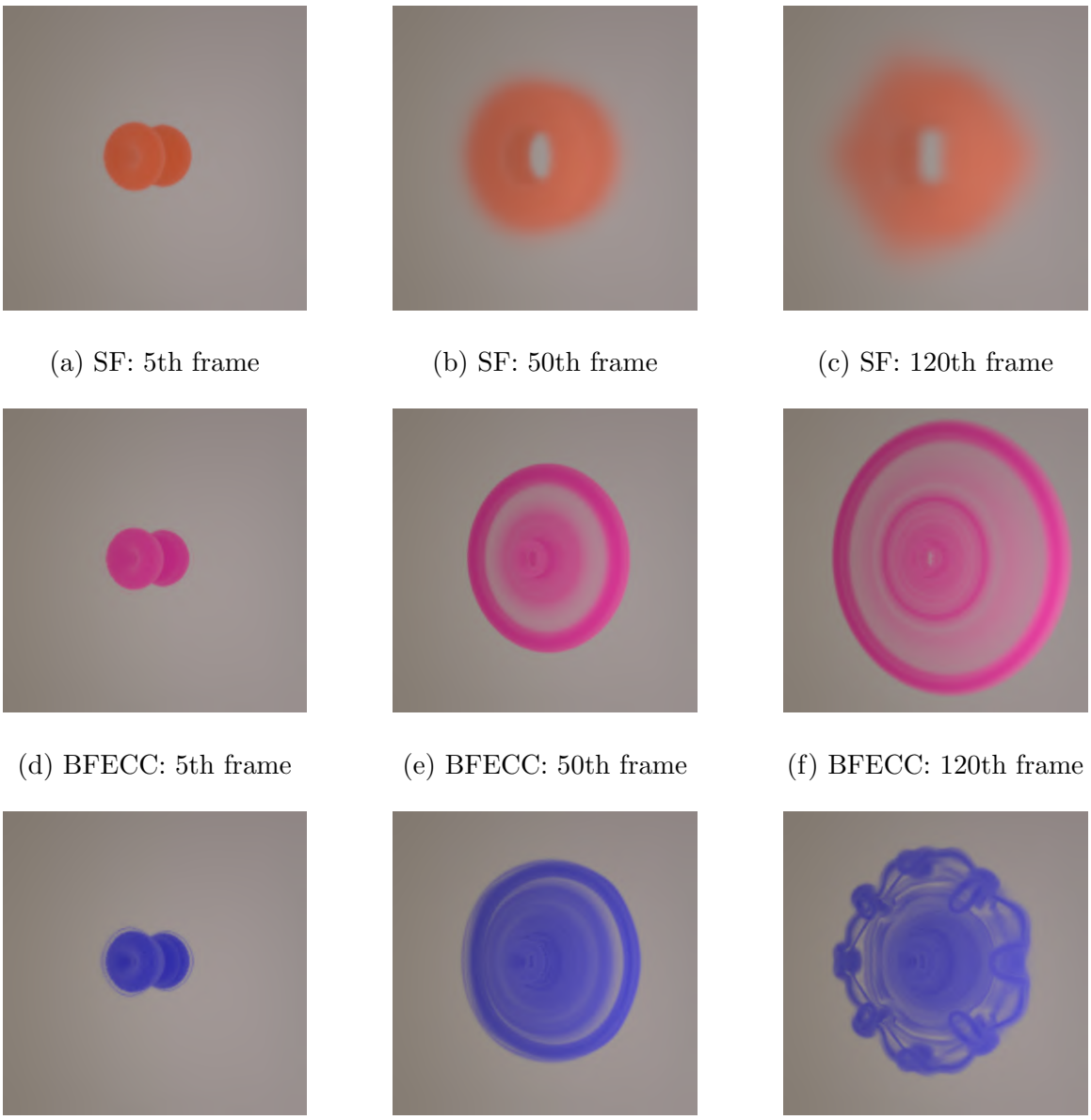
Input: $\boldsymbol{u}^n$
Output: $\boldsymbol{u}^{n+1}$
1: $\tilde{\boldsymbol{u}}^{n+1/2} = \mathit{Advect}(\boldsymbol{u}^n, \frac{1}{2}\Delta t; \quad \boldsymbol{u}^n)$
2: $\boldsymbol{u}^{n+1/2} = \mathit{Project}(\tilde{\boldsymbol{u}}^{n+1/2})$
3: $\hat{\boldsymbol{u}}^{n+1/2} = 2\boldsymbol{u}^{n+1/2} - \tilde{\boldsymbol{u}}^{n+1/2}$
4: $\tilde{\boldsymbol{u}}^{n+1} = \mathit{Advect}(\boldsymbol{u}^{n+1/2}, \frac{1}{2}\Delta t; \quad \hat{\boldsymbol{u}}^{n+1/2})$
5: $\boldsymbol{u}^{n+1} = \mathit{Project}(\tilde{\boldsymbol{u}}^{n+1})$

Compared to existing advection-projection methods, it replaces the energy-dissipating projection typically applied at the end of a simulation loop with an energy-preserving reflection intermediate step performed mid-interval. This reflection solver demonstrates provably superior energy-conservation properties and significantly reduces vorticity diffusion, preserving more detail without importing too much computational cost. Additionally, this method is straightforward to implement, and while it can integrate seamlessly with existing grid-based solvers, it can moreover be combined with the previously introduced error correction post-processing techniques to further reduce numerical viscosity, making the simulation decently closed to physical realism.

Vortex Rings Headon. We compare and showcase the simulation results of using the error correction techniques to improve advection-projection scheme, based on our implementation. Here we modify the classical vortex rings collision case from the experimental physics by [24]. When two vortex rings of opposite circulation collide, they stretch in the y-z plane, thin along the x-axis, and become unstable. This instability leads to the breaking and reconnection of the vortex filaments, forming smaller secondary rings that face radially outward. These processes significantly alter the flow topology, creating complex and fine-scale structures.

Figure 3.7: A frame captured from a *vortex rings headon* physics experiment [13]

In our experiment, we employ slight different settings. Two vortex rings with the identical vorticity strength of 0.02 and the same major radius of 0.045, behaves the headon collision starting from an initial separation of distance 0.32. A uniform difference less than 1% of their minor radius uniquely contributes to the observed vortex filament separation and reconnection phenomenon. The simulation runs in $128 \times 256 \times 256$ with a relatively large CFL$=$ 2.0. While less turbulent sources are introduced to the system compared to the experiment setup above, we obtain the results shown in Figure 3.8 below.



(a) SF: 5th frame (b) SF: 50th frame (c) SF: 120th frame

(d) BFECC: 5th frame (e) BFECC: 50th frame (f) BFECC: 120th frame

(g) MC+R: 5th frame (h) MC+R: 50th frame (i) MC+R: 120th frame

Figure 3.8: Vortex Rings Headon Collision [Density]

These results clearly illustrates how the numerical dissipation in different numerical schemes eventually affect the visualization, with identical initial condition setup. Quantitative differences triggers qualitative distinctions. MC+R outperforms the others with vortex filaments reconnections and separations, while smoke almost depletes with SF, and only the overall vortical structures can be maintained by BFECC.

Chapter 4

Impulse-Based Simulation

In the previous sections, we concentrated on the temporal evolution of the velocity field during fluid simulations. However, if we adopt a more macroscopic perspective on fluid dynamics, to discover equivalent mathematical transformations of the Navier-Stokes equations, or to examine additional physical quantities that govern fluid behavior and that are closely corresponding to the vortex formation, we may gain deeper insights and uncover new directions for analysis.

In computer graphics, the well-established **vortex method** is one of these approaches. The method investigates fluid behavior by analyzing the evolution of the vorticity field and subsequently reconstructing the velocities based on the time-evolving vorticity. From an intuitive standpoint, vorticity, defined as the curl of velocity, is fundamentally connected to vortex dynamics, making it particularly relevant to both physical phenomena and human perception of fluid motion. Moreover, in certain scenarios, vorticity is directly required for the rendering and visualization of simulation data. Numerous methods have been developed around this vector field [39] [54] [45], with the aim of further exploring its potential in fluid dynamics and computer graphics.

This series of methods has provided us with much inspiration: Are there any

other vector fields worth exploring, beyond velocity and vorticity, that could capture certain unique properties of fluids and flow fields? Perhaps, physically, such a field might carry more significance and information about fluid dynamics than the velocity field itself; Or numerically, it might be more amenable to accurate spatial and temporal discretization; Or mathematically, it might also allow for free one-way or multi-directional transformations between itself and foundational quantities like velocity and vorticity. A pioneering work [9] introduces the concept of impulse, which appears to perfectly align with these expectations. In the following sections, we will design and elaborate several algorithms centered on this.

Section 4.1

Impulse

Given an arbitrary bounded domain Ω, then, any vector-valued functions $\boldsymbol{q}$ defined on Ω can be uniquely written as the sum

$$\boldsymbol{q} = \boldsymbol{u} + \nabla\zeta \tag{4.1}$$

where $\nabla \cdot \boldsymbol{u} = 0$. In other words, any vector fields in the domain can be uniquely partitioned into the sum of a divergence-free vector field and a curl-free gradient field. This is called **Hodge Decomposition**.

If we apply this theorem to the fluids in our situation, we are able to denote $\boldsymbol{m}$ to be a vector field equivalent to the divergence-free velocity vector field up to any arbitrary gradient, i.e.

$$\boldsymbol{m} = \boldsymbol{u} + \nabla\zeta \tag{4.2}$$

Such a vector field $\boldsymbol{m}$ is defined as **impulse**. It shares the same units as velocity but differ from it with certain gradients. And we also noticed that although for a

fixed impulse, its Hodge decomposition must be unique (meaning we can uniquely derive one velocity field from a given impulse), the reverse is not true. Freely we have transformations $\boldsymbol{m} \leftarrow \boldsymbol{m} + \nabla\tilde{\zeta}$ for any arbitrary scalar function $\tilde{\zeta}$, resulting in indefinite choice of $\boldsymbol{m}$.

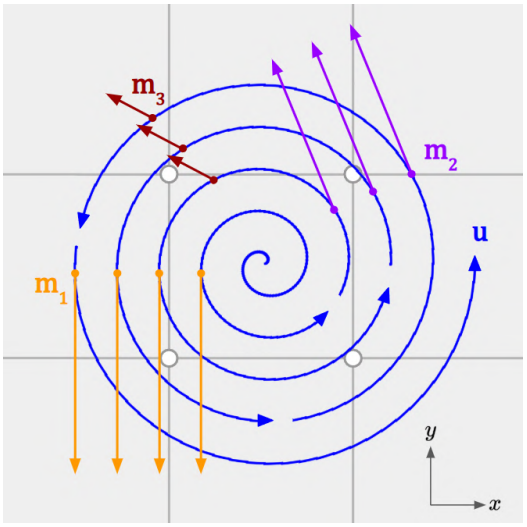


Figure 4.1: Conceptual Diagram of impulse and velocity fields

Conceptually, see the Figure 4.1 above, for the blue-marked velocity field $\boldsymbol{u}$ at a specific moment in the flow field, all the colored vector fields $\boldsymbol{m}_i$ shown in the figure can serve as its impulse.

Meanwhile, both the velocity field and the vorticity field can be seamlessly reconstructed from any of such given impulse. The *Projection* operator extracts the divergence-free component from the impulse, which, based on the uniqueness of the Hodge decomposition, is indeed the velocity. The vorticity can be found through $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$.

4.1.1. Equation of Motions for Impulse

When we aim to track the impulse in the flow field, we have to know its equation of motion over time.

$$\begin{aligned}\frac{D\boldsymbol{m}}{Dt} &= \frac{D(\boldsymbol{u}+\nabla\zeta)}{Dt} \\ &= \frac{D\boldsymbol{u}}{Dt} + \frac{D}{Dt}\nabla\zeta\end{aligned} \tag{4.3}$$

Using the product rule of material Derivatives

$$\frac{D}{Dt}\nabla\zeta = \nabla(\frac{D\zeta}{Dt}) - (\nabla\boldsymbol{u})^T\nabla\zeta \tag{4.4}$$

It follows that

$$\frac{D\boldsymbol{m}}{Dt} = \frac{D\boldsymbol{u}}{Dt} + \nabla(\frac{D\zeta}{Dt}) - (\nabla\boldsymbol{u})^T\nabla\zeta \tag{4.5}$$

Note that

$$\nabla(\frac{1}{2}|\boldsymbol{u}|^2) = (\nabla\boldsymbol{u})^T\boldsymbol{u} \tag{4.6}$$

We can write

$$\begin{aligned}\frac{D\boldsymbol{m}}{Dt} &= \frac{D\boldsymbol{u}}{Dt} + \nabla(\frac{D\zeta}{Dt} + \frac{1}{2}|\boldsymbol{u}|^2 - \frac{1}{2}|\boldsymbol{u}|^2) - (\nabla\boldsymbol{u})^T\nabla\zeta \\ &= \frac{D\boldsymbol{u}}{Dt} + \nabla(\frac{D\zeta}{Dt} + \frac{1}{2}|\boldsymbol{u}|^2) - \nabla(\frac{1}{2}|\boldsymbol{u}|^2) - (\nabla\boldsymbol{u})^T\nabla\zeta \\ &= \frac{D\boldsymbol{u}}{Dt} + \nabla(\frac{D\zeta}{Dt} + \frac{1}{2}|\boldsymbol{u}|^2) - (\nabla\boldsymbol{u})^T\boldsymbol{u} - (\nabla\boldsymbol{u})^T\nabla\zeta \\ &= \frac{D\boldsymbol{u}}{Dt} + \nabla(\frac{D\zeta}{Dt} + \frac{1}{2}|\boldsymbol{u}|^2) - (\nabla\boldsymbol{u})^T\boldsymbol{m}\end{aligned} \tag{4.7}$$

Rearrange the original Navier-Stokes Equaiton (2.1), withou external force nor viscosity, the material derivative of velocity can be written as

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{1}{\rho}\nabla p \tag{4.8}$$

Thus,

$$\frac{D\boldsymbol{m}}{Dt} = \nabla(\frac{D\zeta}{Dt} + \frac{1}{2}|\boldsymbol{u}|^2 - \frac{1}{\rho}p) - (\nabla\boldsymbol{u})^T\boldsymbol{m} \tag{4.9}$$

Recall that $\boldsymbol{m}$ can be defined with any choice of scalar field ζ, therefore, we safely choose ζ such that

$$\nabla(\frac{D\zeta}{Dt} + \frac{1}{2}|\boldsymbol{u}|^2 - \frac{1}{\rho}p) = \nabla\boldsymbol{Z} = 0 \tag{4.10}$$

we eventually obtain the equation of motion for impulse $\boldsymbol{m}$ in free space with no external force to be

$$\frac{D\boldsymbol{m}}{Dt} = -(\nabla\boldsymbol{u})^T\boldsymbol{m} \tag{4.11}$$

4.1.2. Flow Map

From a numerical perspective, the purpose of evolving the impulse is to encode time information of higher order. Unlike previous SF-based approaches, where the information at the current timestep $t = n$ could only be directly derived from the previous timestep $t = n - 1$, we can now span multiple timesteps with impulse. Using the $\boldsymbol{m}$ information "rewounded" to some earlier time $t = t_0$, we can reconstruct the new $\boldsymbol{m}$ for the current timestep $t = n$. This relies on the method called **flow map**.

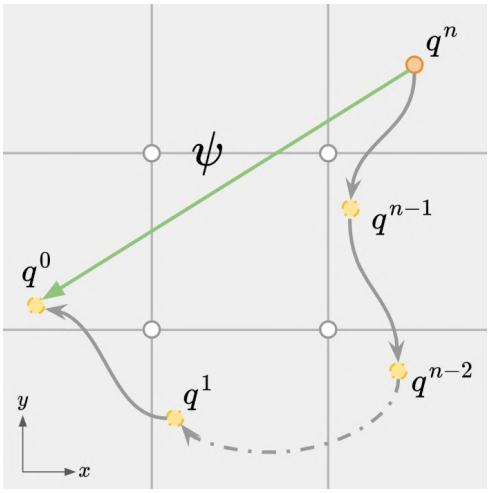


Figure 4.2: Conceptual diagram of backward flow map ψ

For a spatiotemporal velocity field $\boldsymbol{u}(\boldsymbol{p}, \tau)$ representing the velocity at location p and time τ in the bounded flow domain Ω. Given a material point $\boldsymbol{x} \in \Omega_t$ at time $t = n$, we define the backward flow map $\psi : \Omega_t \to \Omega_0$ as

$$\begin{aligned} \frac{\partial \psi(\boldsymbol{x}, \tau)}{\partial \tau} &= \boldsymbol{u}(\psi(\boldsymbol{x}, \tau), \tau) \\ \psi(\boldsymbol{x}, n) &= x \\ \psi(\boldsymbol{x}, 0) &= X \end{aligned} \tag{4.12}$$

Conceptually, see Figure 4.2, the backward map ψ trace the trajectory of material point's position $\boldsymbol{x}$ at current time $t = n$ back to its original position bmX at $t = 0$. Abstractly speaking, within the n-th timestep, it achieves a composition of n single-step Semi-Lagrangian advection functions, retrieving values that fill across the temporal history back to its initial state.

If we further denote the spatial gradients, or Jacobians, of ψ with $\mathcal{T}$ as

$$\mathcal{T} = \frac{\partial \psi}{\partial \boldsymbol{x}} \tag{4.13}$$

which is also known as *deformation gradient*, with its formula of temporal evolution as

$$\frac{D\mathcal{T}}{Dt} = -\mathcal{T} \nabla \boldsymbol{u} \tag{4.14}$$

along with 4.11 and 4.12, we are able to compute the impulse at any time t within certain time intervals as

$$\boldsymbol{m}(\boldsymbol{x}, t) = \mathcal{T}^T \boldsymbol{m}(\psi(\boldsymbol{x}), 0) \tag{4.15}$$

Intuitively, we start by mapping the current material point $\boldsymbol{x}$ back to its initial location $\boldsymbol{X} = \psi(\boldsymbol{x})$ and then applying the extra deformation by $\mathcal{T}^T$ on the original impulse $\boldsymbol{m}^0$ we just backtracked to obtain the current impulse.

Note that we can also thoroughly follow a similar derivation for the forward map ϕ and its Jacobians $\mathcal{F}$ with respect to $\boldsymbol{X}$ at time $t = 0$.

Conclusively, directly retrieving the required values from the temporal history avoids the numerical error accumulation caused by frequent interpolation of the tracked physical quantities, effectively and implicitly improving the accuracy of time integration. Moreover, as long as the backward map remains well-aligned with physical reality and its Jacobian deformation does not deviate significantly, which ensures the stability of the simulation system, we can, within a certain time range, reduce the loss of the physical quantity of interest, in this case, the impulse $\boldsymbol{m}$, during the temporal evolution to a satisfiable low level.

Some recent methods [31] [11] [44] [5] in computer graphics built upon impulse and flow maps have achieved great results. These techniques managed to simulate challenging scenarios that were previously impossible to achieve at relatively low resolutions with classical numerical methods and are applicable to a wide range of initial conditions. This further validates the value of using impulse in fluid simulations for computer graphics.

Section 4.2

Scalar Fields Decomposition

Here, we propose and explain in detail some more algorithms with respect to the impulse vector field in the following sections, along with more experiments and results shown meanwhile.

4.2.1. Proposition

Let $P(\boldsymbol{x}, t)$ and $H(\boldsymbol{x}, t)$ to be two scalar fields such that

$$\begin{aligned} P(\boldsymbol{x}, t) &= P(\psi, 0) \\ H(\boldsymbol{x}, t) &= H(\psi, 0) \end{aligned} \tag{4.16}$$

Assume

$$\begin{aligned} \boldsymbol{J}(\boldsymbol{x}, t) &= \frac{\partial P(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} \\ \boldsymbol{m}(\boldsymbol{x}, 0) &= H(\boldsymbol{x}, 0)\boldsymbol{J}(\boldsymbol{x}, 0) \end{aligned} \tag{4.17}$$

then,

$$\boldsymbol{m}(\boldsymbol{x}, t) = H(\boldsymbol{x}, t)\boldsymbol{J}(\boldsymbol{x}, t) \tag{4.18}$$

4.2.2. Proof

Assume $\boldsymbol{K}(\boldsymbol{x}, t) = H(\boldsymbol{x}, t)\boldsymbol{J}(\boldsymbol{x}, t)$, then,

$$\begin{aligned} \frac{\partial \boldsymbol{K}(\boldsymbol{x}, t)}{\partial t} &= \frac{\partial H(\boldsymbol{x}, t)}{\partial t}\boldsymbol{J}(\boldsymbol{x}, t) + H(\boldsymbol{x}, t)\frac{\partial \boldsymbol{J}(\boldsymbol{x}, t)}{\partial t} \\ &= \frac{\partial H(\boldsymbol{x}, t)}{\partial t}\boldsymbol{J}(\boldsymbol{x}, t) + H(\boldsymbol{x}, t)\left[-u_i(\boldsymbol{x}, t)\frac{\partial \boldsymbol{J}(\boldsymbol{x}, t)}{\partial x_i} - J_i(\boldsymbol{x}, t)\frac{\partial u_i(\boldsymbol{x}, t)}{\partial \boldsymbol{x}}\right] \\ &= \frac{\partial H(\boldsymbol{x}, t)}{\partial t}\boldsymbol{J}(\boldsymbol{x}, t) - H(\boldsymbol{x}, t)u_i(\boldsymbol{x}, t)\frac{\partial \boldsymbol{J}(\boldsymbol{x}, t)}{\partial x_i} - K_i(\boldsymbol{x}, t)\frac{\partial u_i(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} \\ &= -\frac{\partial H(\boldsymbol{x}, t)}{\partial x_i}u_i(\boldsymbol{x}, t)\boldsymbol{J}(\boldsymbol{x}, t) - H(\boldsymbol{x}, t)u_i(\boldsymbol{x}, t)\frac{\partial \boldsymbol{J}(\boldsymbol{x}, t)}{\partial x_i} - K_i(\boldsymbol{x}, t)\frac{\partial u_i(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} \\ &= -u_i(\boldsymbol{x}, t)\frac{\partial \boldsymbol{K}(\boldsymbol{x}, t)}{\partial x_i} - K_i(\boldsymbol{x}, t)\frac{\partial u_i(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} \end{aligned} \tag{4.19}$$

Rearrange this, we have

$$\begin{aligned} \frac{\partial \boldsymbol{K}(\boldsymbol{x}, t)}{\partial t} + u_i(\boldsymbol{x}, t)\frac{\partial \boldsymbol{K}(\boldsymbol{x}, t)}{\partial x_i} &= -\frac{\partial u_i(\boldsymbol{x}, t)}{\partial \boldsymbol{x}}K_i(\boldsymbol{x}, t) \\ \Rightarrow \qquad \frac{D\boldsymbol{K}}{Dt} &= -(\nabla \boldsymbol{u})^T\boldsymbol{K} \end{aligned} \tag{4.20}$$

Given (4.11), $\boldsymbol{K}$ is equivalent to impulse $\boldsymbol{m}$ by definition, hence proved.

4.2.3. Remark

This proposition allows us to define two scalar fields during the initialization phase based on the initial impulse field $\boldsymbol{m}$. Additionally, instead of explicitly evolving the impulse vector field $\boldsymbol{m}$ (which requires maintaining its Jacobian $\mathcal{T}$ of the backward map through advection at each frame and applying additional deformation), we can backtrack the scalar fields using the backward map itself and reconstruct the velocity field at any given time using Proposition (4.18). Intuitively, the temporal advection of a mapping function is replaced by a spatial derivatives computation of a scalar field within a single time step, with respect to quantity values all stored at the cell centers.

4.2.4. Algorithm

At the very beginning, the initial step is to obtain an appropriate impulse field $\boldsymbol{m}$ based on the initial velocity field. Here, we select the simplest example such that $\boldsymbol{m} = \boldsymbol{u} + \nabla \mathbf{0} = \boldsymbol{u}$. It follows that we need to numerically perform scalar field decomposition on $\boldsymbol{m} = \boldsymbol{u}$.

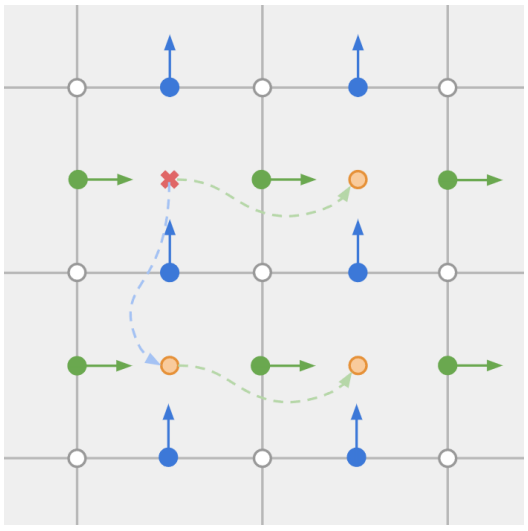


Figure 4.3: Conceptual diagram of reconstructing scalar field P

Recall that we store vector fields, such as velocity, on the face centers of the MAC Grid. If we wish to construct a scalar field such that its gradient field matches the vector field stored at the face-centers, we first need to fix one scalar value at an arbitrary point stored at the cell center. Consequently, by performing pointwise integration along the coordinate axes, as shown in Figure 4.3, the influence of this value can propagate globally through the gradient values stored at the face-centers. We have some degree of freedom when assigning this value and its location since it serves as an intermediate value in the simulation process. What we truly care about is the gradient of the scalar field radiated from it.

Algorithm 6 Impulse Scalar Fields Decomposition (Impulse-SFD) Initialization

Input: $\boldsymbol{u}^0$
Output: P^0, H^0
1: Set $\boldsymbol{m}^0 = \boldsymbol{u}^0$
2: Choose H^0
3: Compute $\nabla P^0 = \boldsymbol{m}/H^0$
4: Fix P at $(i_{min}, j_{min}, k_{min})$
5: Construct $\hat{P}^0$ using ∇P^0
6: Repeat 4, 5 For $\hat{P}_i^0$
7: Smooth P^0 as average of $\hat{P}_i^0$

For instance, we can choose H stored at cell centers such that it equals the *velocity magnitude*, and it follow that ∇P describes the *velocity direction*. We are then able to construct P^0 directly using discrete spatial integration of velocities based on its gradient field and world position information encoded in the MAC Grid. Note that, from our observations and experiments, arbitrarily selecting a single point and performing only once such a global reconstruction from it often leads to singularities or uniform velocity shifts in a particular direction. To mitigate this issue, some artificial smoothing is need, for instance, building the same P field based on the fixed value of P at different positions, or performing the same numerical integration along different axes with different order, and eventually, we take their average as the final values.

Algorithm 7 Impulse Scalar Fields Decomposition (Impulse-SFD) Advance

Input: $\psi^n, \boldsymbol{u}^n; P^0, H^0$
Output: $\psi^{n+1}, \boldsymbol{u}^{n+1}$
1: $\psi^{n+1} = Backtrack(\boldsymbol{u}^n, \Delta t; \quad \psi^n)$
2: $H^{n+1} = Pullback(\psi^{n+1}; H^0)$
3: $P^{n+1} = Pullback(\psi^{n+1}; P^0)$
4: $\boldsymbol{m}^{n+1} = H^{n+1}\nabla P^{n+1}$
5: $\boldsymbol{u}^{n+1} = Project(\boldsymbol{m}^{n+1})$

Now let us focus on the specific time-stepping process. The first step is to update the flow map information across the time step. Subsequently, the new values of H and P can be directly obtained from this backward flow map without performing any deformation operations, as given by Impulse-SFD assumption (4.16). In this way, we can compute the current impulse field m by (4.18) and derive the corresponding divergance-free velocity field. Note that numerical spatial derivatives has to be performed when computing the impulse on face centers.

Algorithm 8 Impulse Scalar Fields Decomposition (Impulse-SFD) Re-initialization

Input: $\psi^n, \boldsymbol{u}^n$
Output: ψ^n, P^0, H^0
1: **if** the time range covered by ψ^n is too long **then**
2: Set $\psi^n = I$
3: $P^0, H^0 \leftarrow$ Impulse-SFD Init($\boldsymbol{u}^n$) [Algorithm 6]
4: **end if**

As previously mentioned, due to the chaotic nature of the flow field and the resulting spatial distortion of such mapping functions, it becomes increasingly difficult to maintain the integrity of a long-time-span flow map only through numerical timestepping. Admittedly, the magnitudes of the scalar values obtained from H_0 and P_0 are naturally preserved, nevertheless the values are possibly actually being sampled from incorrect positions in that case. This phenomenon significantly heightens the risk of the simulation system's collapse. Therefore, at regular time intervals, performing the re-initialization process is obligatory.

Here we complete the **Impulse-SFD** algorithm.

4.2.5. Scene

We set up a scenario called **Inkdrop Butting-Heads**, where two fluid-constructed cows, positioned face-to-face at a certain distance, carry equal and opposite initial velocities of small but non-uniform distributions, attached by random yet symmetric subtle perturbations. The collision begins subsequently.

In the background grids, both velocity and density are uniformly set to zero; whereas inside each inkdrop, the velocity has been set as we described earlier, and the density is uniformly set to one. Additionally, the two cows carry distinct color fields at cell centers respectively, which are similarly advected during time integration and used for the final rendering and visualization. This initial setup is achieved using our implemented GPU-based parallel ray-casting [37] method as a simplified version, which will also be used for the further examples in the next sections. In other words, after embedding the object file with corresponding rigid-body transformations onto the MAC Grid, we perform the following algorithm.

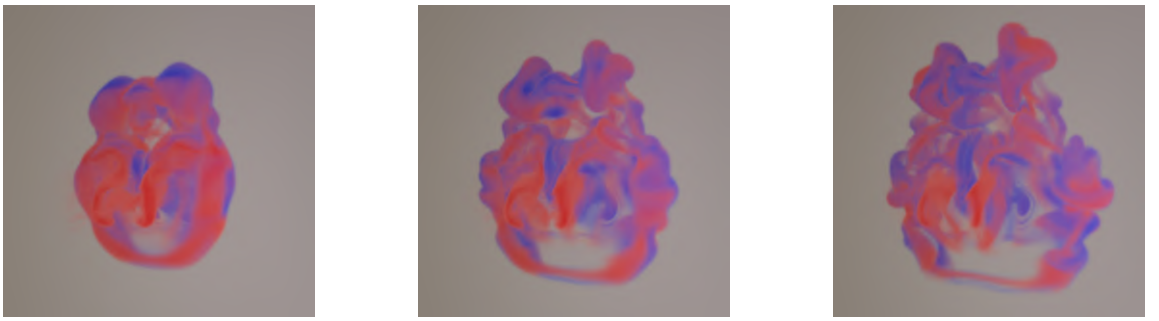
Algorithm 9 Compute INSIDE field on grid through Ray-Casting

```
1: for in parallel each grid point do
2:     Fix ray direction to [1.0, 0.0, 0.0]
3:     for each face on the object mesh do
4:         if ray intersect with face then
5:             Count intersection once
6:         end if
7:     end for
8:     Grid point is INSIDE if intersection count is odd
9: end for
```

4.2.6. Results

We have done the scene initialization. We are ready to choose appropriate methods to run our simulation.

(a) 0th frame (b) 20th frame (c) 40th frame



(d) 60th frame (e) 80th frame (f) 100th frame

Figure 4.4: Inkdrop Butting-Heads [Density]: BFECC

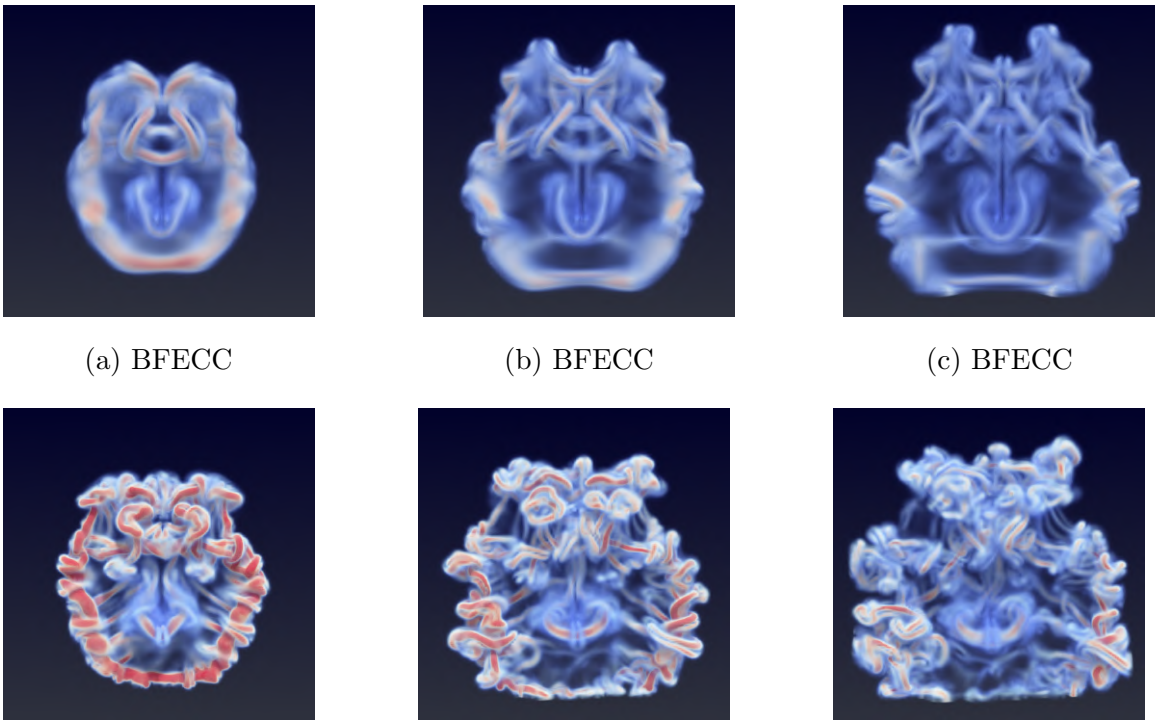
(a) 0th frame (b) 20th frame (c) 40th frame

(d) 60th frame (e) 80th frame (f) 100th frame

Figure 4.5: Inkdrop Butting-Heads [Density]: ImpSFD+BFECC

Here we show selected frames based on our implementation. The simulation runs with CFL=0.5 in low resolution as $128 \times 128 \times 128$ but still shows visually amazing results. Starting from an irregular yet smooth initial manifold pattern, we observe that these two smoke cows evolve into turbulent, sophisticated, and visually stunning vortical structures, based on their interactions within the flow field.

We compare our results using regular BFECC correction technique and our Impulse-SFD algorithm with BFECC. For the impulse based method, we reinitialize the flow maps and scalar fields H and P for every 10 substeps.

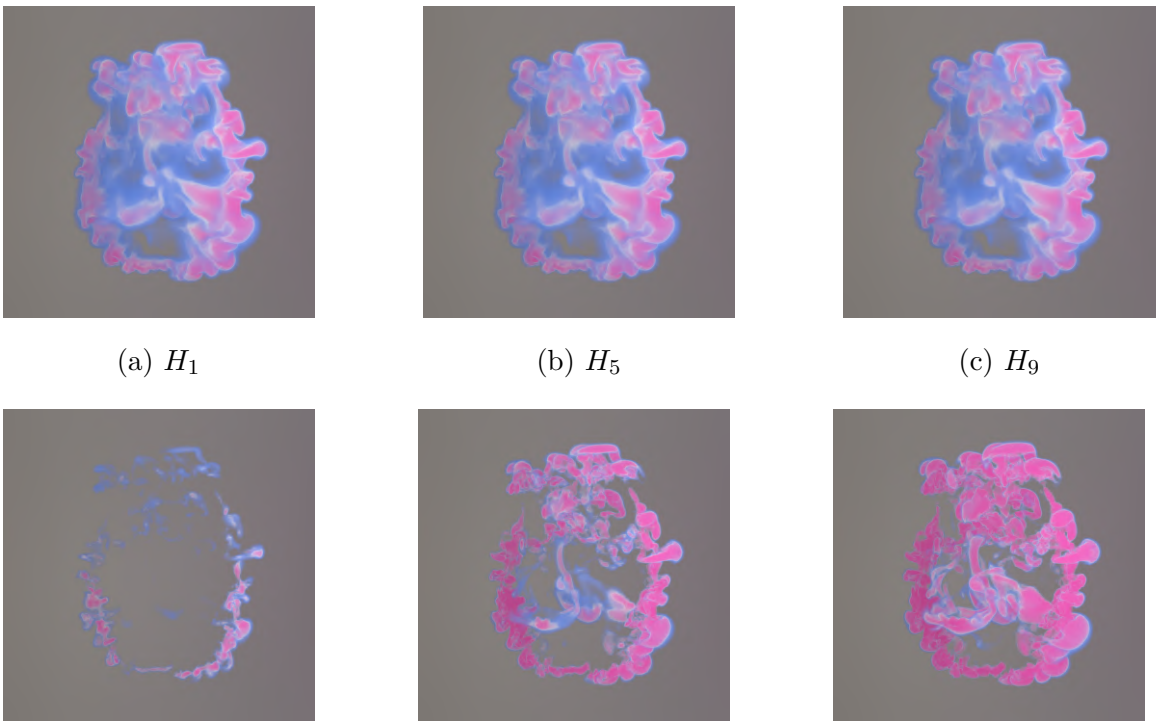


(a) BFECC (b) BFECC (c) BFECC

(d) ImpSFD+BFECC (e) ImpSFD+BFECC (f) ImpSFD+BFECC

Figure 4.6: Inkdrop Butting-Heads [Vorticity]: 60/80/100-th frames

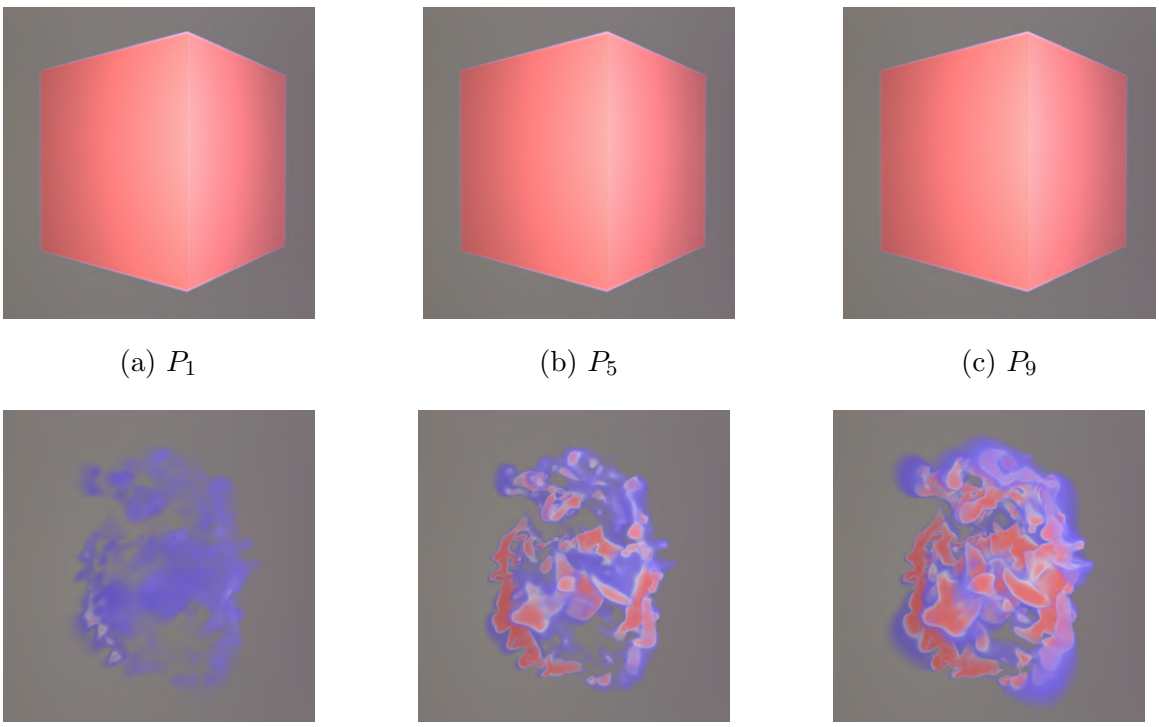
It is clearly observable that starting from frame 60 in the density plot, our impulse-based method maintains a greater volume and finer detail of the smoke, continuing until the simulation ends. See Figure 4.6, we further reassure this observation based on the visualization of vorticity magnitude of the same side view, where the colors from blue to red map the vorticity norms clamped into interval from 1 to 10.



(a) H_1 (b) H_5 (c) H_9

(d) $(15\times)$ $H_1 - H_0$ (e) $(15\times)$ $H_5 - H_0$ (f) $(15\times)$ $H_9 - H_0$

Figure 4.7: **H** Plot: 55-th Reinitialization



(a) P_1 (b) P_5 (c) P_9

(d) $P_1 - P_0$ (e) $P_5 - P_0$ (f) $P_9 - P_0$

Figure 4.8: **P** Plot: 55-th Reinitialization

We also show the H and P plot within certain re-initialization loop. H holds constantly a similar pattern throughout such time interval, and P appears always like a "completely solid cube". For better understanding, we plot the difference images between the scalar fields we just reinitialized and the ones at current substep. Such differences in H describe the variances of velocity magnitudes, while the differences in P, hidden inside the box, symbolize the topological changes of fluids in space, which provides some abstract inspirations into how the flow map enforces the fluid to follow the correct kinematics.

4.2.7. Analysis

Quantitatively, as shown in the plots below, our Impulse-SFD method outperforms the traditional one in terms of energy conservation and provides greater circulation over the long term.

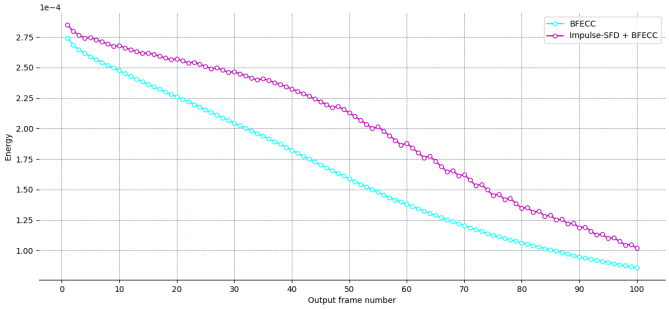


Figure 4.9: Energy Over Time: Inkdrop Butting-Heads

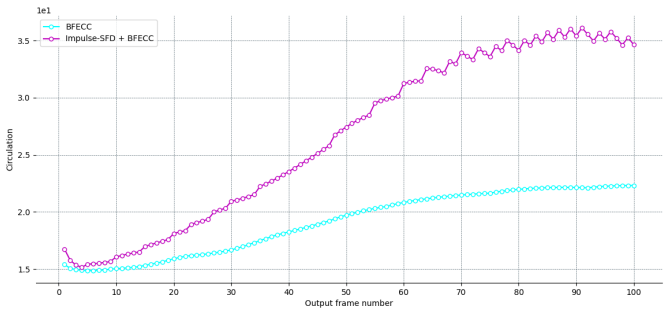


Figure 4.10: Circulation Over Time: Inkdrop Butting-Heads

Section 4.3

Vortex Particles in Impulse

While the Impulse-SFD method brings improved energy conservation and enhanced visual details to our fluid solver, we are not satisfied with just this; we also aim for greater control or even enhancement of the vortical structures. This leads to the idea of integrating **vortex particles** with the impulse field.

4.3.1. Intuition

The concept of vortex particles was first introduced by [39] into computer graphics, achieving notable advancements in the field of film special effects (VFX) at that time. This approach has since been widely adopted and continues to influence researchers to the present day. It is indeed a grid-based method with additional forces induced by particles, added to the system for enhancing details. Specifically, the vortex particles being spread to the system serve as indication of locations of vortical structures, driving the vorticity toward their centers, aiming at preserving vorticity "concentration" based on the confinement approach.

Technically, the algorithm can be rewritten as

Algorithm 10 Vortex Particles Confinement

1: **for** each vortex particle p **do**
2: Compute analytically $\tilde{\boldsymbol{\omega}}_p(\boldsymbol{x}) = \varepsilon_p(\boldsymbol{x} - \boldsymbol{x}_p)\boldsymbol{\omega}_p$
3: Compute vorticity locations fields $\boldsymbol{N}_p(\boldsymbol{x}) = (\boldsymbol{x}_p - \boldsymbol{x})/||\boldsymbol{x}_p - \boldsymbol{x}||$
4: Compute confinement force $\boldsymbol{F}_p(\boldsymbol{x}) = \epsilon_p(\boldsymbol{N}_p \times \tilde{\boldsymbol{\omega}}_p)$
5: **end for**
6: Apply $\boldsymbol{F}_{\text{total}} = \sum \boldsymbol{F}_p$ into the system

4.3.2. Algorithm

Our algorithm also aims to highlight vortex structures in chaotic turbulence. However, we do not wish to explicitly and manually apply external forces to the entire

flow field at each frame, as this could easily lead to instability in the simulation system, introduce visualization noise due to numerical oscillations, and make the overall simulation process appear less aligned with physical fidelity from a macroscopic perspective.

In previous section, we explained in detail the Equation 4.18. Indeed, it can be further decomposed linearly, or say partitioned, into multiple parts, as

$$\boldsymbol{m} = \sum_i H_i \nabla P_i \tag{4.21}$$

We omit the proof here. Leveraging this theorem, we are able to split the impulse into two parts: the one induced by vortex particles, while the remaining to be residual. We therefore start from our initialization algorithm:

Algorithm 11 Vortex Particles in Impulse (VPImp) Initialization

```
Init smoke and velocity u
Set m_residual = u
Set m_vp = 0
Splash vortex particles where you desire on the grid
for each face centers do
    Assign position vector x
    for each vortex particle p do
        Denote ω_p to p
        Induce velocity u_vp by (ω_p, x_p; x)
        Add to velocity u(x) = u(x) + u_vp
        Record in m_vp(x) = u_vp
    end for
end for
```

We have done our initialization of two impulse field $\boldsymbol{m}_{residual}$ and $\boldsymbol{m}_{vp}$. For the time integration, we treat these two fields separately, using different advection techniques (must be compatible with motion of equation of impulse (4.11)). They are then recombined using formula 4.21 to serve as the total impulse for next step.

Note that reinitialization steps similarly have to be performed because of the intrinsic issues of using flow maps to advect impulse-related values.

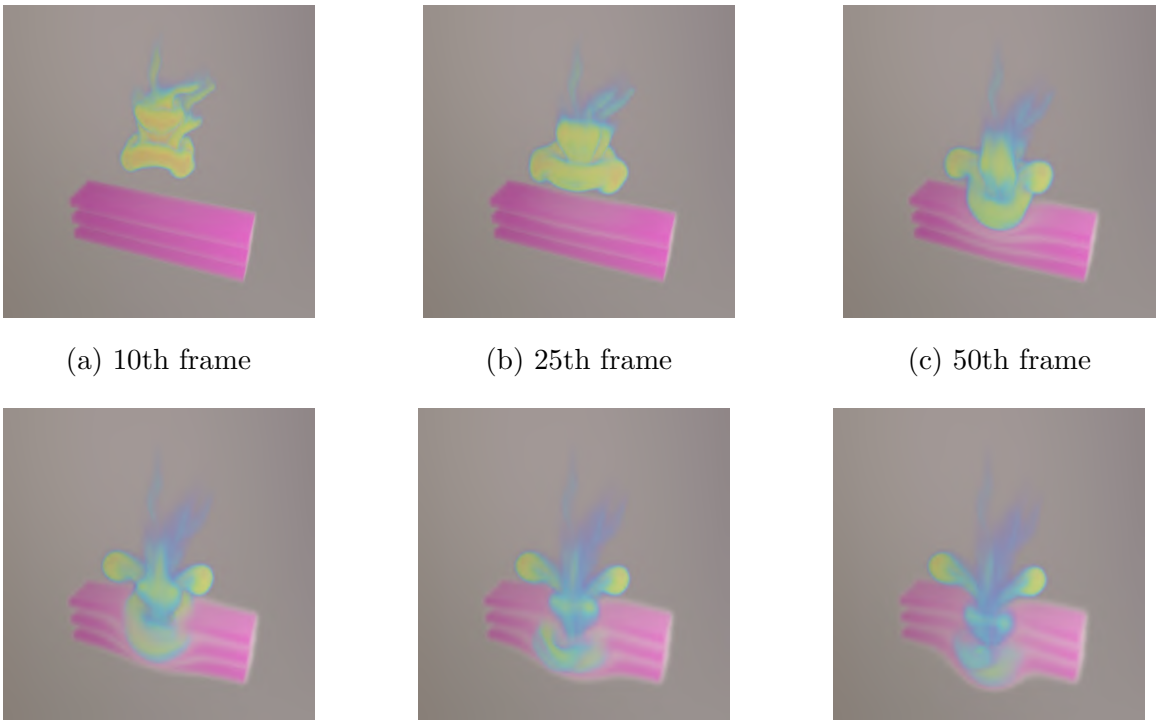
Algorithm 12 Vortex Particles in Impulse (VPImp) Advance

1: $\boldsymbol{m}_{vp}^{n+1} = Advect_{\alpha}(\boldsymbol{m}_{vp})$
2: $\boldsymbol{m}_{residual}^{n+1} = Advect_{\beta}(\boldsymbol{m}_{residual})$
3: $\boldsymbol{m}^{n+1} = \boldsymbol{m}_{vp}^{n+1} + \boldsymbol{m}_{residual}^{n+1}$
4: $\boldsymbol{u}^{n+1} = Project(\boldsymbol{m}^{n+1})$

4.3.3. Scene

Here, we set up a traditional inkdrop scenario, referred to as **Pikachu Inkdrop**. What makes it unique is that the velocity describing the inkdrop's descent is stored in $\boldsymbol{u}_{residual}$, while the disturbances responsible for generating vortices and turbulence are initialized based on our method as $\boldsymbol{u}_{vp}$, induced by strength of vortex particles. During its descent, the inkdrop also interacts with the underlying fluid thin blocks that pre-exist in the scene.

4.3.4. Results



(a) 10th frame (b) 25th frame (c) 50th frame

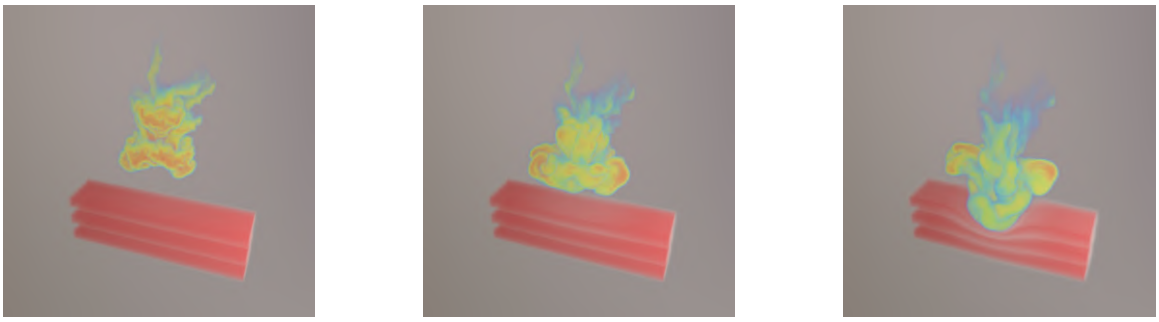
(d) 75th frame (e) 100th frame (f) 125th frame

Figure 4.11: Pikachu Inkdrop [Density]: Regular Impulse Method

(a) 10th frame (b) 25th frame (c) 50th frame

(d) 75th frame (e) 100th frame (f) 125th frame

Figure 4.12: Pikachu Inkdrop [Density]: ImpSFD+MC+R

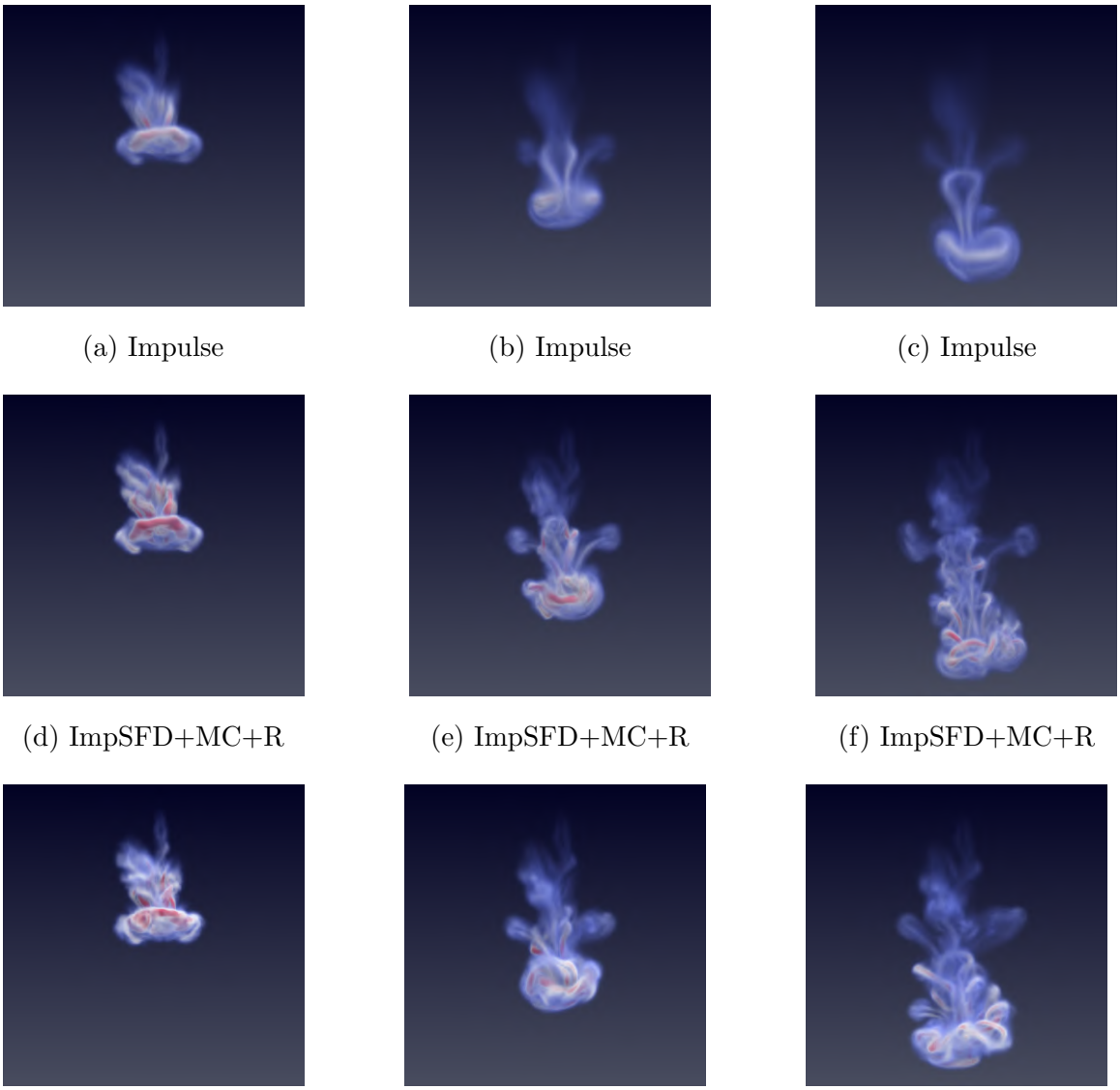


(a) 10th frame (b) 25th frame (c) 50th frame

(d) 75th frame (e) 100th frame (f) 125th frame

Figure 4.13: Pikachu Inkdrop [Density]: VPImp

The simulation runs with CFL=0.5 in low resolution as $128 \times 128 \times 128$. We first compare the density plot. From an aesthetic perspective, it is hard to determine which method ranks the best. Nevertheless, it is evident that from a physical viewpoint, particularly in terms of smoke volume conservation, the capabilities of these three techniques improve progressively in sequence. Upon closer observation, especially during the initial phase, see 10-th and 25-th frames, VPImp demonstrates a significantly superior ability to preserve details compared to the other two methods.



(a) Impulse (b) Impulse (c) Impulse

(d) ImpSFD+MC+R (e) ImpSFD+MC+R (f) ImpSFD+MC+R

(g) VPImp (h) VPImp (i) VPImp

Figure 4.14: Pikachu Inkdrop [Vorticity]: 20/70/125-th frames

Similarly, we capture some of the vorticity fields to reinforce our statement, refering to Figure 4.14 above.

In fact, achieving results with such visual complexity at a resolution of $128 \times 128 \times 128$ is extremely challenging, even when considering other state-of-the-art methods recently developed in computer graphics.

4.3.5. Remark

It is important to acknowledge that, we do not add external forces explicitly to the system, thereby avoiding artifacts introduced by such means; however, the turbulence field generated by $\boldsymbol{u}_{vp}$ is usually inherently difficult to maintain, with frequent abrupt transitions of both velocity magnitude and directions in many sub-regions. This may not only trigger the noise during its own numerical advection but, similar to the original version [39] of *vortex particle* methods, can easily lead to the collapse of the entire simulation system if not handled and controlled cautiously. To mitigate these issues, we need to ensure the following:

- Select the most accurate advection scheme available.
- Ensure that the theoretical discrepancies in the algorithms used for advecting the two fields do not have differences in order of accuracy.
- Keep the re-initialization intervals sufficiently short.
- Opt to shut down VPImp when necessary.

Regarding the final point, as described in (4.21), VPImp can be directly converted into regular impulse or ImpSFD method, both of which are comparably stable. Transitioning to these methods after some iterations with VPImp allows the simulation to restart under relatively optimal intermediate conditions, still ultimately leading to improved visual results.

Section 4.4

Discussions

We introduced two algorithms related to the impulse field: **ImpSFD** and **VPImp**, each with its own focus, strengths, and weaknesses, as well as application scenarios where they are most suitable.

4.4.1. ImpSFD vs. VPImp

ImpSFD is more general-purpose, as it can be abstracted as another way to explain the current impulse field. This method can be applied to solve scenarios regardless of their specific definitions. Additionally, it is highly compatible with any error correction methods we discussed earlier. When combined with flowmap-based advection methods specifically, applying error correction often yields surprisingly better results. However, it is worth noting that when used alone, ImpSFD may not stand out. Its performance might be average compared to traditional impulse methods or other conventional techniques without any further post-processing.

VPImp, on the other hand, is more of an application layer on top of the impulse method. It is naturally suited for scenarios defined by vortex particles or highly chaotic flow fields initialized with a great amount of random inputs. Observations show that VPImp excels in maintaining turbulence structures during the early stages and overall in preserving the magnitude of fields involved in the time evolution. However, for situations without random processes involved in their definition, and when our primary focus is on ensuring that large-scale vortex tube structures remain prominent and neat throughout the simulation, it becomes challenging to define how to pre-allocate vortex particles at this outset. In other words, the partition of impulse fields into $\boldsymbol{m}_{residual}$ and $\boldsymbol{m}_{vp}$ in some reasonable ways is relatively difficult, or sometimes unnecessary, at the very beginning. VPImp, like its predecessor, vortic-

ity confinement, fails to demonstrate their advantages in these cases. Furthermore, using VPImp requires highly accurate advection schemes to ensure stability during the long-term simulations.

4.4.2. Flow Map Range and Reinitialization

Both of these two methods are build upon the impulse-based vector fields of fluids, and they highly rely on flowmap-based advection scheme for their outperformances with respect to numeric and physical fidelity.

We recall that the flow map enables the direct duplication of initial values to their new positions. Therefore, an "ideal" flow map would perfectly adhere to the law of mass conservation, regardless of the length of the time span. As long as the mapping function could be somehow preserves exactly, this value transfer can result in zero dissipation. However, whether talking to scalar fields H and P in impSFD, or vector fields m_{res} and m_{vp} of in VPImp, there are always some fields for which it is difficult to maintain a satisfactory, or even least acceptable, discretized flow map within certain time intervals. A compromise approach is to slightly increase memory overhead for maintaining different flow maps of different fields. Precisely, we can use a long-range flow map to preserve fields that are less distorted in the flow field, maximizing the energy-conserving properties of the flow map for these, without being constrained by the reinitialization necessaries of other fields (we may additionally maintain a short-range flow map specifically for them).

This also reminds us that reinitialization can sometimes become the source of energy loss. Observations similarly reveal that, especially if we directly set the current fields as the new identity for reinitialization, several quantities are arbitrarily lost. However, other methods might on the other hand violate the theoretical foundations (4.18) and (4.21), or cause numerical disturbances due to re-discretization, leading to systematic instability. Better reinitialization strategies remain to be explored.

Chapter 5

Lamb Vector and Simulation

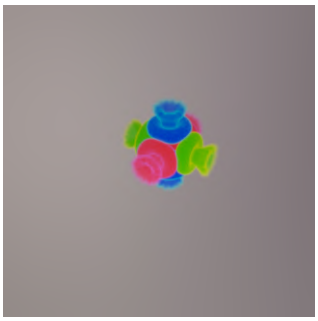
Not Available.

Appendix A

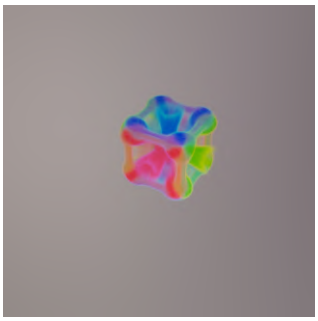
Supplementary Results

Section A.1

Multiple Vortex Rings Collision



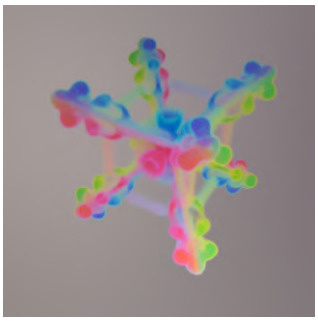
(a) 20th frame



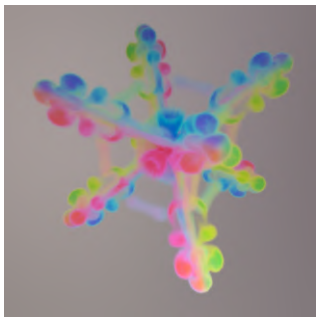
(b) 40th frame

(c) 60th frame

(d) 80th frame



(e) 100th frame

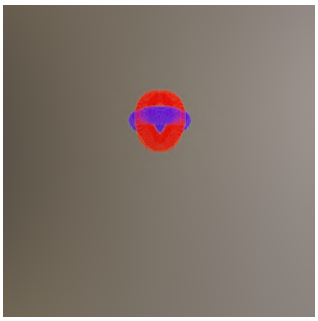


(f) 120th frame

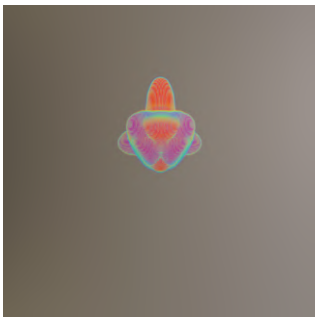
Figure A.1: Six Vortex Rings Collision ($128 \times 128 \times 128$): **ImpSFD+BFECC**

Section A.2

Intersected Uniform Vortex Rings



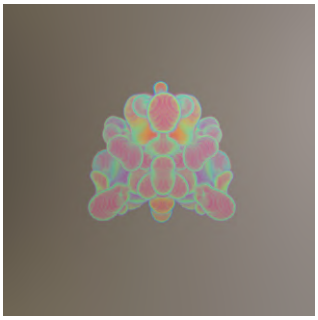
(a) 0th frame



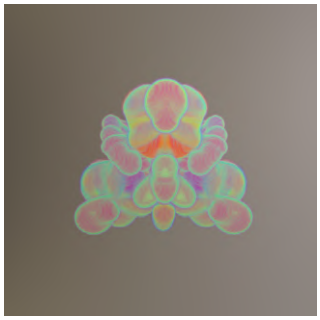
(b) 15th frame

(c) 30th frame

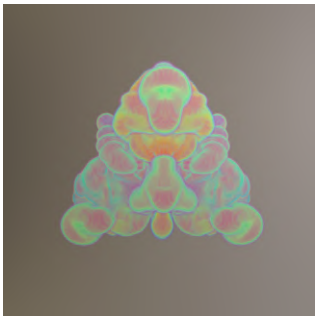
(d) 45th frame



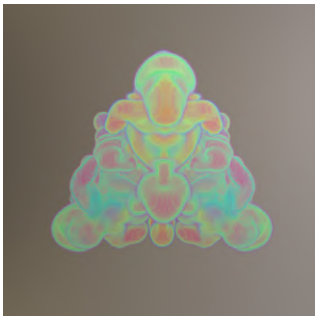
(e) 60th frame



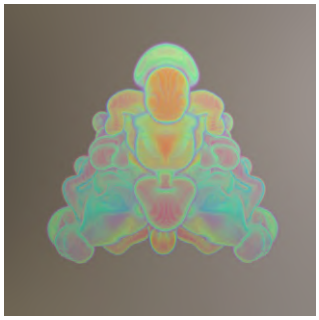
(f) 75th frame



(g) 90th frame



(h) 105th frame



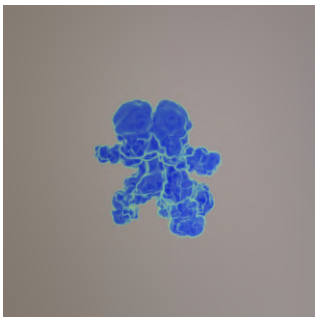
(i) 120th frame

Figure A.2: Intersected Uniform Vortex Rings ($128 \times 128 \times 128$): **ImpSFD+MC+R**

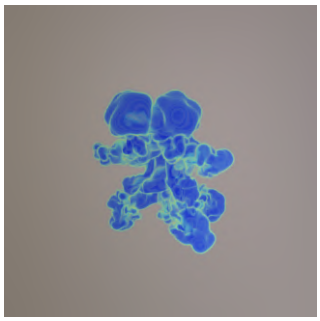
Section A.3

Irregular Inkdrop Random Fission

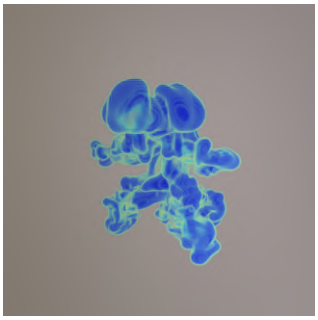
(a) 0th frame



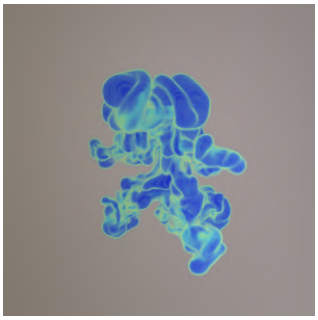
(b) 15th frame



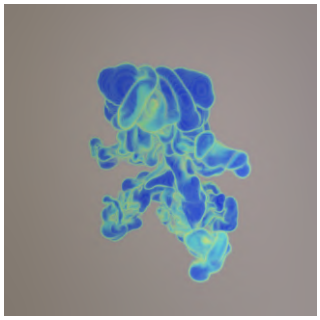
(c) 30th frame



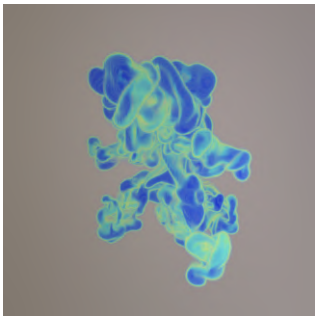
(d) 45th frame



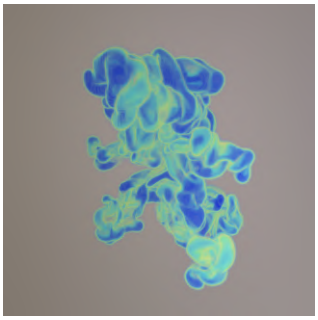
(e) 60th frame



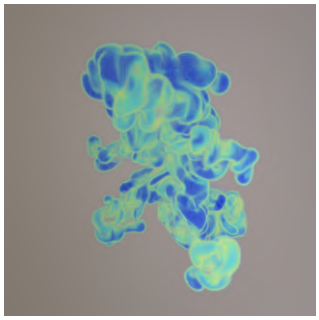
(f) 75th frame



(g) 90th frame



(h) 105th frame



(i) 120th frame

Figure A.3: Squirtle Inkdrop Fission ($128 \times 128 \times 128$): **VPimp**

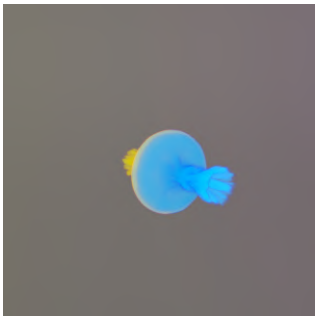
Section A.4

Vortex Rings Headon Revisit

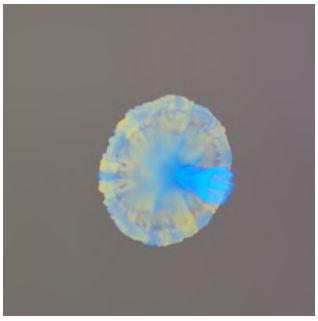
(a) 5th frame

(b) 20th frame

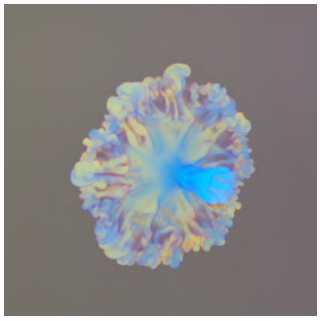
(c) 35th frame



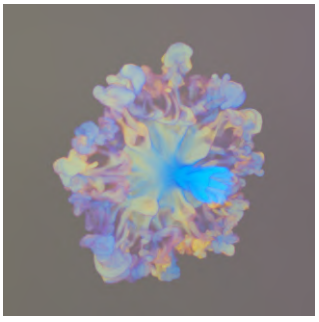
(d) 50th frame



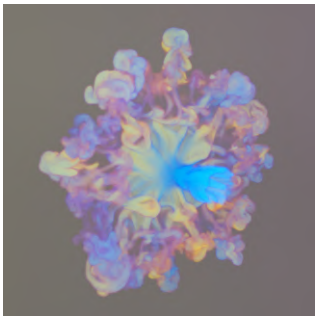
(e) 65th frame



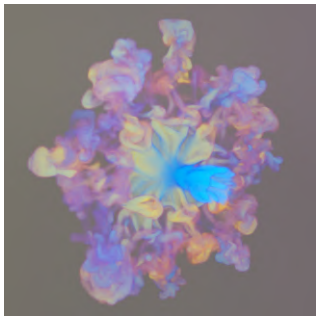
(f) 80th frame



(g) 95th frame



(h) 110th frame



(i) 125th frame

Figure A.4: Vortex Rings Headon Revisit ($256 \times 256 \times 256$): **VPimp**

Bibliography

[1] *Ueber den mechanismus des widerstandes, den ein bewegter körper in einer flüssigkeit erfährt*, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse **1911** (1911), 509–517.

[2] Ryoichi Ando, Nils Thürey, and Chris Wojtan, *Highly adaptive liquid simulations on tetrahedral meshes*, ACM Trans. Graph. **32** (2013), no. 4.

[3] Markus Becker and Matthias Teschner, *Weakly compressible sph for free surface flows*, Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '07, Eurographics Association, 2007, p. 209–217.

[4] Robert Bridson, Jim Houriham, and Marcus Nordenstam, *Curl-noise for procedural fluid flow*, ACM Trans. Graph. **26** (2007), no. 3, 46–es.

[5] Duowen Chen, Zhiqi Li, Junwei Zhou, Fan Feng, Tao Du, and Bo Zhu, *Solid-fluid interaction on particle flow maps*, ACM Trans. Graph. **43** (2024), no. 6.

[6] Nuttapong Chentanez and Matthias Müller, *Real-time eulerian water simulation using a restricted tall cell grid*, ACM SIGGRAPH 2011 Papers (New York, NY, USA), SIGGRAPH '11, Association for Computing Machinery, 2011.

[7] Alexandre Joel Chorin, *Numerical solution of the navier-stokes equations*, Mathematics of computation **22** (1968), no. 104, 745–762.

[8] Jonathan M. Cohen, Sarah Tariq, and Simon Green, *Interactive fluid-particle simulation using translating eulerian grids*, Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (New York, NY, USA), I3D '10, Association for Computing Machinery, 2010, p. 15–22.

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

[10] Keenan Crane, Ulrich Pinkall, and Peter Schröder, *Robust fairing via conformal curvature flow*, ACM Transactions on Graphics (TOG) **32** (2013), no. 4, 1–10.

[11] Yitong Deng, Hong-Xing Yu, Diyang Zhang, Jiajun Wu, and Bo Zhu, *Fluid simulation on neural flow maps*, ACM Trans. Graph. **42** (2023), no. 6.

[12] Mathieu Desbrun and Marie-Paule Gascuel, *Smoothed particles: a new paradigm for animating highly deformable bodies*, Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96 (Berlin, Heidelberg), Springer-Verlag, 1996, p. 61–76.

[13] Adam Fabio, *When vortex rings collide*, 2018.

[14] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen, *Visual simulation of smoke*, Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA), SIGGRAPH '01, Association for Computing Machinery, 2001, p. 15–22.

[15] Nick Foster and Dimitris Metaxas, *Controlling fluid animation*, Proceedings of the 1997 Conference on Computer Graphics International (USA), CGI '97, IEEE Computer Society, 1997, p. 178.

[16] Francis H Harlow and J Eddie Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, The physics of fluids **8** (1965), no. 12, 2182–2189.

[17] Mark Harris, *Fast fluid dynamics simulation on the gpu*, ACM SIGGRAPH 2005 Courses (New York, NY, USA), SIGGRAPH '05, Association for Computing Machinery, 2005, p. 220–es.

[18] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen, *The beam radiance estimate for volumetric photon mapping*, ACM SIGGRAPH 2008 Classes (New York, NY, USA), SIGGRAPH '08, Association for Computing Machinery, 2008.

[19] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin, *The affine particle-in-cell method*, ACM Trans. Graph. **34** (2015), no. 4.

[20] Byungmoon Kim, *Multi-phase fluid simulations using regional level sets*, ACM Trans. Graph. **29** (2010), no. 6.

[21] ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac, *Flowfixer: using bfecc for fluid simulation*, Proceedings of the First Eurographics Conference on Natural Phenomena (Goslar, DEU), NPH'05, Eurographics Association, 2005, p. 51–56.

[22] Theodore Kim, Nils Thürey, Doug James, and Markus Gross, *Wavelet turbulence for fluid simulation*, ACM Trans. Graph. **27** (2008), no. 3, 1–6.

[23] Zhiqi Li, Barnabás Börcsök, Duowen Chen, Yutong Sun, Bo Zhu, and Greg Turk, *Lagrangian covector fluid with free surface*, ACM SIGGRAPH 2024 Conference Papers (New York, NY, USA), SIGGRAPH '24, Association for Computing Machinery, 2024.

[24] TT Lim and TB Nickels, *Instability and reconnection in the head-on collision of two vortex rings*, Nature **357** (1992), no. 6375, 225–227.

[25] Xu-Dong Liu and Stanley Osher, *Convex eno high order multi-dimensional schemes without field by field decomposition or staggered grids*, J. Comput. Phys. **142** (1998), no. 2, 304–330.

[26] Xu-Dong Liu, Stanley Osher, and Tony Chan, *Weighted essentially non-oscillatory schemes*, J. Comput. Phys. **115** (1994), no. 1, 200–212.

[27] Frank Losasso, Frédéric Gibou, and Ron Fedkiw, *Simulating water and smoke with an octree data structure*, ACM Trans. Graph. **23** (2004), no. 3, 457–462.

[28] A. McAdams, E. Sifakis, and J. Teran, *A parallel multigrid poisson solver for fluids simulation on large grids*, Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '10, Eurographics Association, 2010, p. 65–74.

[29] Joe J Monaghan, *Smoothed particle hydrodynamics*, In: Annual review of astronomy and astrophysics. Vol. 30 (A93-25826 09-90), p. 543-574. **30** (1992), 543–574.

[30] Matthias Muller, David Charypar, and Markus Gross, *Particle-based fluid simulation for interactive applications*, Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (Goslar, DEU), SCA '03, Eurographics Association, 2003, p. 154–159.

[31] Mohammad Sina Nabizadeh, Stephanie Wang, Ravi Ramamoorthi, and Albert Chern, *Covector fluids*, ACM Trans. Graph. **41** (2022), no. 4.

[32] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold, *A gradient-augmented level set method with an optimally local, coherent advection scheme*, Journal of Computational Physics **229** (2010), no. 10, 3802–3827.

[33] Sang Il Park and Myoung Jun Kim, *Vortex fluid for gaseous phenomena*, Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (New York, NY, USA), SCA '05, Association for Computing Machinery, 2005, p. 261–270.

[34] Ziyin Qu, Minchen Li, Fernando De Goes, and Chenfanfu Jiang, *The power particle-in-cell method*, ACM Trans. Graph. **41** (2022), no. 4.

[35] Wouter Raateland, Torsten Hädrich, Jorge Alejandro Amador Herrera, Daniel T. Banuti, Wojciech Pałubicki, Sören Pirk, Klaus Hildebrandt, and Dominik L. Michels, *Dcgrid: An adaptive grid structure for memory-constrained fluid simulation on the gpu*, Proc. ACM Comput. Graph. Interact. Tech. **5** (2022), no. 1.

[36] André Robert, *A stable numerical integration scheme for the primitive meteorological equations*, Atmosphere-ocean **19** (1981), 35–46.

[37] Scott D Roth, *Ray casting for modeling solids*, Computer Graphics and Image Processing **18** (1982), no. 2, 109–144.

[38] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac, *An unconditionally stable maccormack method*, J. Sci. Comput. **35** (2008), no. 2–3, 350–371.

[39] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw, *A vortex particle method for smoke, water and explosions*, ACM Trans. Graph. **24** (2005), no. 3, 910–914.

[40] Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis, *Spgrid: a sparse paged grid structure applied to adaptive smoke simulation*, ACM Trans. Graph. **33** (2014), no. 6.

[41] Chi-Wang Shu, *High order eno and weno schemes for computational fluid dynamics*, High-order methods for computational physics, Springer, 1999, pp. 439–582.

[42] B. Solenthaler and R. Pajarola, *Predictive-corrective incompressible sph*, ACM SIGGRAPH 2009 Papers (New York, NY, USA), SIGGRAPH '09, Association for Computing Machinery, 2009.

[43] Jos Stam, *Stable fluids*, Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (USA), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., 1999, p. 121–128.

[44] Yuchen Sun, Linglai Chen, Weiyuan Zeng, Tao Du, Shiying Xiong, and Bo Zhu, *An impulse ghost fluid method for simulating two-phase flows*, ACM Trans. Graph. **43** (2024), no. 6.

[45] Ningxiao Tao, Liangwang Ruan, Yitong Deng, Bo Zhu, Bin Wang, and Baoquan Chen, *A vortex particle-on-mesh method for soap film simulation*, ACM Trans. Graph. **43** (2024), no. 4.

[46] Roger Temam and A Chorin, *Navier stokes equations: Theory and numerical analysis*, (1978).

[47] Yun Teng, David I. W. Levin, and Theodore Kim, *Eulerian solid-fluid coupling*, ACM Trans. Graph. **35** (2016), no. 6.

[48] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin, *Accelerating eulerian fluid simulation with convolutional networks*, Proceedings

of the 34th International Conference on Machine Learning - Volume 70, ICML'17, JMLR.org, 2017, p. 3424–3433.

[49] Mengdi Wang, Yitong Deng, Xiangxin Kong, Aditya H. Prasad, Shiying Xiong, and Bo Zhu, *Thin-film smoothed particle hydrodynamics fluid*, ACM Trans. Graph. **40** (2021), no. 4.

[50] Yuwei Xiao, Szeyu Chan, Siqi Wang, Bo Zhu, and Xubo Yang, *An adaptive staggered-tilted grid for incompressible flow simulation*, ACM Trans. Graph. **39** (2020), no. 6.

[51] Tianyi Xie, Minchen Li, Yin Yang, and Chenfanfu Jiang, *A contact proxy splitting method for lagrangian solid-fluid coupling*, ACM Trans. Graph. **42** (2023), no. 4.

[52] Larry Yaeger, Craig Upson, and Robert Myers, *Combining physical and visual simulation—creation of the planet jupiter for the film "2010"*, SIGGRAPH Comput. Graph. **20** (1986), no. 4, 85–93.

[53] Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski, *An advection-reflection solver for detail-preserving fluid simulation*, ACM Trans. Graph. **37** (2018), no. 4.

[54] Xinxin Zhang, Robert Bridson, and Chen Greif, *Restoring the missing vorticity in advection-projection fluid solvers*, ACM Trans. Graph. **34** (2015), no. 4.

[55] Xinxin Zhang, Minchen Li, and Robert Bridson, *Resolving fluid boundary layers with particle strength exchange and weak adaptivity*, ACM Trans. Graph. **35** (2016), no. 4.

[56] Junwei Zhou, Duowen Chen, Molin Deng, Yitong Deng, Yuchen Sun, Sinan Wang, Shiying Xiong, and Bo Zhu, *Eulerian-lagrangian fluid simulation on particle flow maps*, ACM Trans. Graph. **43** (2024), no. 4.

[57] Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw, *A new grid structure for domain extension*, ACM Trans. Graph. **32** (2013), no. 4.

[58] Yongning Zhu and Robert Bridson, *Animating sand as a fluid*, ACM Trans. Graph. **24** (2005), no. 3, 965–972.