Editing Fluid Flows with Divergence-Free Biharmonic Vector Field Interpolation

by

Tümay Özdemir

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Abstract

Achieving satisfying fluid animation through numerical simulation can be time-consuming because such simulations are computationally expensive to perform and there are few practical post-processing tools for editing of completed simulations – often, the user must modify their scene setup and launch it again from scratch. To address this challenge, we present a divergence-free biharmonic vector field interpolation and extrapolation method for reusing and/or stitching together spatial regions of existing flows. Given velocities and velocity gradients on the boundary of a domain at each timestep, which may be either user-defined or drawn from existing simulations, we fill in the given domain by constructing an optimally smooth, divergence-free, boundary-satisfying vector field. We measure smoothness using the Laplacian energy to allow smooth boundary behavior and enforce divergence constraints through explicit Lagrange multipliers. The prior methods for this problem suffer from non-zero divergence and associated visible compression artifacts, or cannot smoothly match the desired slopes at the domain boundaries. Moreover, we introduce a new extrapolation scheme that can handle unprescribed boundaries by smoothly extending the vector field through the unspecified boundary. In this case, we measure the smoothness using the Hessian energy which provides well-behaved solutions for "free" or natural boundary conditions. We demonstrate that our new interpolation and extrapolation procedures always produce smooth and incompressible flows, as well as enabling a range of natural simulation editing capabilities including hole-filling, copy-pasting, extrapolation, and scene stretching.

Acknowledgements

I would like to express my deepest gratitude and thanks to Professor Christopher Batty for supervising me through this degree. I learned so much from him. His incredible knowledge on physics-based animation opened my eyes to new directions in computer graphics research.

I also would like to extend my sincere thanks to my committee members, Professor Craig S. Kaplan and Professor Toshiya Hachisuka, for reviewing this thesis and making it better with their insightful comments.

Finally, I would like to extend my heartfelt thanks to my fiancé, Vincent, my parents, Sibel and Hakkı, and my sister, Simay, for their words of support and encouragement inspiring me to overcome my obstacles during this degree.

Dedication

To my parents, Sibel & Hakkı Özdemir.

Table of Contents

List of Figures

Chapter 1

Introduction

Physically-based simulation of fluids has become an important tool for many visual effects applications, such as movies and computer games. However, high quality fluid simulations remain computationally and financially expensive. They are computationally expensive because standard simulation methods scale roughly quartically with grid resolution. Moreover, there is not an easy way to to guess the scene parameters that must be used to obtain the desired animation before simulating. As a result, artists are bound to resimulate repeatedly with different parameters until they obtain the desired animation.

Due to the high computation time of standard fluid simulation methods, re-simulating from scratch becomes a bottleneck for visual effects production. For that reason, providing tools that enable reuse of existing simulations is an important task. While the main goal is to synthesize a new flow reusing existing simulation data, it is important and also challenging to provide a tool that respects the governing physics, which give the fluid its natural appearance. In particular, it is essential to preserve incompressibility to avoid volume change. Incompressibility implies that the divergence of the vector field representing the fluid's velocity should always be zero.

Cut-and-paste editing has been ubiquitously applied to seamlessly combine images and videos; however, it has also been recently introduced to physically-based fluid animation by Sato et al. [2018]. Sato et al. applied this concept to fluid animation to combine pieces of existing simulations using an interpolation scheme. The central task is to fill a volumetric region with a smooth incompressible vector field, given only the boundary data, which comes from the existing simulations. This approach can be used for several meaningful scenarios such as hole-filling, copy-paste, and scene stretching.

Unfortunately, existing methods for this vector field interpolation problem possess key limitations. The method of Sato et al. [2018] introduces significant divergence inside the interpolation region depending on the choice of input flows. As a result, the interpolated vector fields fail to satisfy the divergence-free property, violating the underlying incompressibility assumption and leading to undesired expansion and contraction effects. Instead, to strictly enforce incompressibility, one might use a potential-flow-based interpolation [Nielsen and Bridson, 2011], or solve a Stokes problem [Bhattacharya et al., 2012], which better preserves rotational motion. The shortcoming of these methods is that, while the generated vector field matches the prescribed boundary velocity values, it does not match their gradients, resulting in undesirable derivative discontinuities along the border of the generated vector field. This effect can highlight the presence of the computational boundary and break the illusion of a natural flow.

To address this challenge, we develop a novel divergence-constrained biharmonic interpolation technique for fluid animation editing. We pose this task as an optimization problem discretized on a grid over the region of interest, where we minimize an appropriately chosen smoothness energy subject to the incompressibility constraint. The result is a maximally smooth field that is able to simultaneously match prescribed velocity values and derivatives on the boundary; meanwhile the use of explicit Lagrange multipliers for the constraints tightly couples the three velocity components together to ensure a divergence-free field. Furthermore, unlike prior work we generalize our approach to handle regions with open or partially specified boundaries, leading to a new method for incompressible vector field extrapolation. We demonstrate our new vector field interpolation and extrapolation tools on various flow field editing scenarios.

Chapter 2

Related Work

The control and editing of fluid flows has been an interesting problem in computer graphics and a wide range of approaches have been proposed. We will focus our review primarily on control of smoke, although a significant amount of work, going back to Foster and Metaxas [1997], has also explored controlling liquid surfaces.

A keyframe-based approach is used to make the fluid flow evolve in a way that the fluid density or shape matches a sparse set of user-provided target keyframes while the resulting flow still respects the governing physics. Fattal and Lischinski [2004], McNamara et al. [2004], Pan et al. [2013], Shi and Yu [2005], Thuerey et al. [2006], and Treuille et al. [2003] formulate this task as an optimization problem. For example, McNamara et al. [2004] introduce an adjoint method to efficiently compute the gradients for their gradient-based nonlinear optimization.

A boundary conditions-based approach relies on applying velocity boundary conditions to the underlying simulation [Rasmussen et al., 2004; Raveendran et al., 2012; Stomakhin and Selle, 2017; Wiebe and Houston, 2004]. Often these methods have been used to cause liquid to follow a target motion or character, with varying degrees of "looseness" allowed in order to retain a fluid-like effect. For example, Raveendran et al. [2012] introduce a novel volume-preserving morphing method for liquid shapes and motions based on input meshes (can be poses for a character or deformations of a 3D model) with an additional divergence-free constraint.

A guiding-based approach is employed to combine the low-frequency features that provide overall flow structure at the coarse scale with the high-frequency features that give the fluid its high resolution turbulent details [Forootaninia and Narain, 2020; Inglis et al., 2017; Nielsen and Bridson, 2011; Nielsen and Christensen, 2010; Nielsen et al., 2009; Pan et al., 2013; Sato et al., 2021; Thuerey et al., 2006]. Typically, the low-frequency guiding vector field is either designed manually by an artist or generated using a fast low-resolution simulation. Given a low-resolution precomputed or procedural flow lacking small-scale turbulent details, guiding-based methods aim to obtain a new simulation based on the low-resolution input simulation that has the additional high-resolution details. For example, Sato et al. [2021] propose an optimization-based approach in the stream function space. They first convert an input low-resolution guide simulation into a so-called stream function representation that guarantees incompressibility by construction. Then, at each timestep, they solve for a new high resolution velocity field that balances between matching the coarse guide data and the high resolution physics of the unconstrained simulation, thereby adding physically plausible fine details while preserving the overall coarse flow structure.

There are other approaches that directly modify the fluid geometry (surface shape or density field), rather than the velocity field, such as space-time fluid sculpting [Manteaux et al., 2016] and fluid-carving [Flynn et al., 2019]. Another geometric approach seeks to interpolate the global fluid shape and motion from multiple input simulations [Raveendran et al., 2014; Thuerey, 2016]. Since these methods focus on geometry rather than the underlying velocities, they don't necessarily enforce the incompressibility condition and can therefore struggle to preserve volume.

A related task in liquid animation is to insert a localized 3D simulation, such as the region around a ship or swimming character, into a much larger surrounding procedural ocean or shallow water model. This effect has typically been achieved through the use of non-reflecting boundary conditions [Bojsen-Hansen and Wojtan, 2016; Söderström et al., 2010], which smoothly damp out the surface flow to match the prescribed exterior model.

Closest to our method are PDE-based velocity interpolation approaches that solve for a discrete vector field over a bounded region. Inspired by Poisson image editing [Pérez et al., 2003], recent work by Sato et al. [2018] demonstrates the potential of a cut-and-paste metaphor for fluids. Given two velocity fields, referred to as the source and target fields, the user defines a copy region on the source velocity field to be pasted on the target velocity field. The method of Sato et al. [2018] consists of two processes: "frame matching" and "flow interpolation". We focus our review on their "flow interpolation" process since it is more relevant to our work. To ease or smooth out the transition of the velocity field from the exterior target region into the pasted interior source region, a narrow buffer or blend region is introduced at their border. To fill in the blend region Ω between the pasted region of the source flow and its new surrounding target flow, their "flow interpolation" step consists of solving a vector Laplacian problem. Unfortunately, they show that their approach suffers from compression artifacts even when the inputs are divergence-free. We provide more insight on their method in Section 5.1.

Nielsen and Bridson propose using a potential flow approximation for incompressible velocity interpolation in the context of a guiding method for liquid animation [Nielsen and Bridson, 2011]. Potential flow has the advantage that it strictly ensures incompressibility by construction. Subsequently, Bhattacharya et al. [2012] show that interpolation based on the Stokes equations offers better preservation of rotational modes than the potential-flow-based interpolation. However, the Stokes formulation offers only C^0 continuity on the velocity field at the boundaries, leading to a lack of visual smoothness. These formulations also do not offer convenient support for open/unprescribed boundaries (i.e., extrapolation rather than interpolation). Since the derivations for the potential flow and Stokes flow approaches provide essential context for our method, we review them in Sections 5.2 and 5.3.

Chapter 3

Mathematical Background

We propose a method to synthesize a new fluid simulation using pieces of existing (previously simulated) fluid simulations that we combine via an interpolation scheme. In our implementation, the input simulations are drawn from standard grid-based fluid simulations. Thus, for a better understanding of our method, in this chapter, we give a brief background on the governing physical equations and the advection/pressure projection steps behind these fluid simulations. For a complete introduction to these topics, we refer the reader to the textbook by Bridson [2015].

We also describe our spatial discretization details in Section 3.2 and the divergence operator that we make use of in our formulations in Section 3.3.

3.1 Governing Equations

Most fluid flows of interest to animation are primarily determined by the *incompressible Navier-Stokes equations*, a system of partial differential equations, formulated as:

$$\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}\right) + \frac{1}{\rho}\nabla p = g + \nu \nabla \cdot \nabla \mathbf{u} \tag{3.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{3.2}$$

where $\mathbf{u}$ is fluid velocity, t is time, ρ is fluid density, p is pressure, g is the body forces such as gravity (and possibly other control forces applied to guide the fluid towards a desired shape), and ν is kinematic viscosity [Bridson, 2015].

Equation (3.1) is essentially the well-known Newton's law of conservation of momentum, $\vec{F} = m\vec{a}$, adapted to the case of a continuous medium. In equation (3.1), the terms surrounded by parentheses constitute the material derivative, denoted as $\frac{D\mathbf{u}}{Dt} = (\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u})$. The material derivative describes the rate of change of velocity with respect to the moving velocity field. Incompressibility is enforced with a divergence-free condition in equation (3.2).

For the kinds of smoke animations that will be the focus of this thesis, we might prefer to omit the viscosity since the effect of viscosity is negligibly small. Moreover, most numerical methods for simulating fluids inevitably introduce errors that have an apparent effect similar to viscosity, i.e., applying damping effects on the velocity. Navier-Stokes equations without the viscosity term are called the *Euler equations* and have the following form [Bridson, 2015]:

$$\frac{D\mathbf{u}}{Dt} + \frac{1}{\rho}\nabla p = g \tag{3.3}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{3.4}$$

Operator splitting is applied to the incompressible Euler equations to divide the solution procedure into three separate parts: the advection part, the body forces part, and the pressure/incompressibility part. Then, we obtain the following three equations:

$$\frac{D\mathbf{u}}{Dt} = 0 \text{ (advection)} \tag{3.5}$$

$$\frac{\partial \mathbf{u}}{\partial t} = g \text{ (body forces)} \tag{3.6}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0$$

$$\text{such that } \nabla \cdot \mathbf{u} = 0 \text{ (pressure projection).} \tag{3.7}$$

Loosely speaking, advection (3.5) transports the fluid velocities along the velocity vector field itself, body forces (3.6) apply acceleration due to gravity, and pressure projection (3.7) ensures that the velocity field at the end of a timestep is volume preserving.

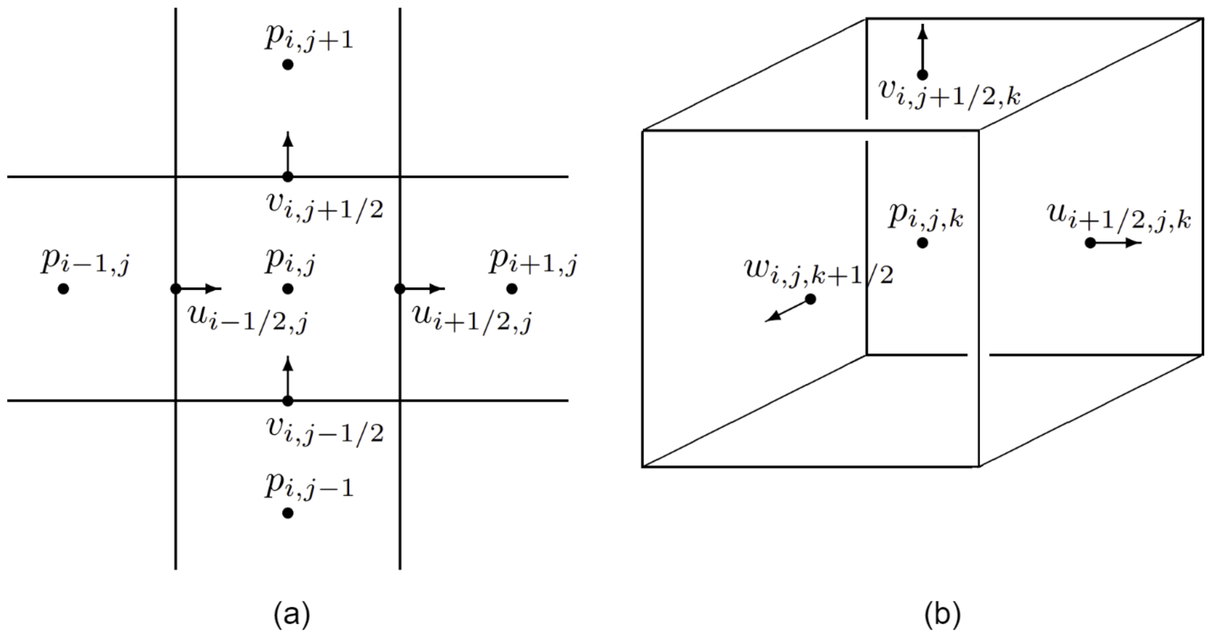


Figure 3.1: **A MAC grid cell in 2D (a) and in 3D (b).** (a) For the cell (i, j) in 2D, pressure, indicated by $p_{i,j}$, is sampled at the cell center. The horizontal u-components of the velocity, indicated by $u_{i-1/2,j}$, $u_{i+1/2,j}$, are sampled at the vertical cell faces and the vertical v-components of the velocity, indicated by $v_{i,j-1/2}$, $u_{i,j+1/2}$, are sampled at the horizontal cell faces. (b) Similarly, for the cell (i, j, k) in 3D, pressure is sampled at the cell center. The incoming and outgoing velocity normal components are sampled at the corresponding cell faces [Bridson, 2015].

3.2 Spatial Discretization

3.2.1 Staggered Grid

We use the staggered "marker-and-cell" (MAC) spatial discretization introduced by Harlow and Welch [1965] in our implementation. With this discretization, after laying a uniform Cartesian grid over the fluid domain of interest, pressure values are sampled at the cell centers and the normal velocity components are sampled at the cell faces as illustrated in Figure 3.1.

This type of discretization allows us to formulate second-order accurate central differences, which are used in pressure projection, without causing the undesirable non-trivial null-space [Bridson, 2015]. We provide more explanation on this in Section 3.3.

3.3 Discrete Divergence

The desired incompressibility condition is satisfied with a divergence-free constraint stating that the divergence of the field is zero: $\nabla \cdot \mathbf{u} = 0$. The divergence operator is:

$$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}, \tag{3.8}$$

where u, v, and w correspond to the three components of the velocity in x, y, and z direction. This operator is discretized using finite differences on the staggered grid and in 3D, for the grid cell *(i,j,k)*, it has the form:

$$(\nabla \cdot \mathbf{u})_{i,j,k} \approx \frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} + \frac{v_{i,j+1/2,k} - v_{i,j-1/2,k}}{\Delta x} + \frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta x}. \tag{3.9}$$

It is this discrete measure of incompressibility that will be enforced by pressure projection (Section 3.5), and which we will later seek to preserve in our new interpolation scheme.

Unlike a staggered grid, a regular collocated grid, where all velocity components are sampled at the same grid location, would make the divergence computation less robust. Let us consider an alternative, collocated central differences-based approximation formulated in 2D as $(\nabla \cdot \mathbf{u})_{i,j,k} \approx \frac{u_{i+1,j} - u_{i-1,j}}{\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{\Delta x}$. If a highly divergent velocity field, such as $\mathbf{u}_{i,j} = ((-1)^i, (-1)^j)$, is used, the divergence operator can incorrectly measure zero divergence at some grid locations, resulting in high-frequency oscillations and local compression errors [Bridson, 2015]. In contrast, on the staggered grid, even for high frequency data, the corresponding divergence operator does not overlook any divergence present in the velocity field. Therefore, a staggered grid tends to be a better fit for discrete divergence computation.

3.4 FLIP Advection

The advection step in equation (3.5) can be performed by using various methods. In our implementation, we use fluid-implicit-particle (FLIP) advection [Brackbill et al., 1988] while generating the input simulations; therefore, we focus on this advection scheme in this section.

The FLIP method is a hybrid particle-grid method that makes use of both Lagrangian (moving particle) and Eulerian (stationary grid) viewpoints. In this method, particles with assigned velocities are responsible for advecting the velocity field. Velocities are trivially advected with particles and $\frac{D\mathbf{u}}{Dt} = 0$ is automatically satisfied when particles are advected in a Lagrangian manner through an interpolated grid-based velocity field. After advection, velocity data is transferred to the grid and all other non-advection steps such as addition of body forces and pressure projection are performed on the grid in an Eulerian fashion. The full simulation loop for a simulator based on FLIP advection can be summarized by the following ordered steps [Bridson, 2015]:

- Transfer particle velocities $\mathbf{u}_p$ to grid velocities $\mathbf{u}^n$. Transferring particle velocities to the grid is performed by simply taking a weighted sum of particle velocities at each grid location *(i,j,k)*:

$$\mathbf{u}^n_{i,j,k} = \sum_p \mathbf{u}_p \frac{(\vec{x}_p - \vec{x}_{i,j,k})}{W}, \tag{3.10}$$

  where $\vec{x}$ represents the position and W is the weight value which depends on the particular interpolation kernel one chooses to use.

- Perform addition of body forces, such as buoyancy forces, and pressure projection on the grid. After we apply body forces and pressure, we obtain a new grid velocity field $\mathbf{u}^{n+1}$.

- Update the particle velocities by combining the interpolated grid velocities with particles' velocities. Classic particle-in-cell (PIC) interpolation sets the new particle velocity $\mathbf{u}_p$ as:

$$\mathbf{u}_p^{\mathrm{PIC}} = \mathit{bilerp}(\mathbf{u}^{n+1}, \mathbf{x}_p), \tag{3.11}$$

  where *bilerp* indicates a bilinear interpolation operation given grid velocities $\mathbf{u}$ and particle's position $\mathbf{x}_p$. On the other hand, "pure" FLIP interpolation sets the new particle velocity $\mathbf{u}_p$ as:

$$\mathbf{u}_p^{\mathrm{FLIP}} = \mathbf{u}_p + \mathit{bilerp}(\mathbf{u}^{n+1} - \mathbf{u}^n, \mathbf{x}_p), \tag{3.12}$$

  i.e., it determines the difference/change from the grid $\mathbf{u}^{n+1} - \mathbf{u}^n$, and increments $\mathbf{u}_p$ with the difference information, rather than overwriting the particle data completely like PIC does. This incremental approach avoids the severe smoothing suffered by PIC due to repeated interpolation and overwriting of the velocities.

Finally, Bridson [2015] recommends updating $\mathbf{u}_p$ with a combination of PIC and FLIP, to smooth out some of FLIP's inherent noisiness as:

$$\mathbf{u}_p = 0.99\mathbf{u}_p^{\text{FLIP}} + 0.01\mathbf{u}_p^{\text{PIC}}. \tag{3.13}$$

- Advect the particles using the grid velocities with a standard integrator (forward Euler, Runge-Kutta, etc.).

3.5 Pressure Projection

Pressure projection is a critical step in incompressible fluid simulations because it makes the fluid incompressible and simultaneously imposes the boundary conditions. It is formulated as [Bridson, 2015]:

$$\mathbf{u}^{n+1} = \mathbf{u} - \frac{\Delta t}{\rho}\nabla p \tag{3.14}$$

$$\text{such that } \nabla \cdot \mathbf{u}^{n+1} = 0 \tag{3.15}$$

$$\text{and } \mathbf{u}^{n+1} \cdot \mathbf{n} = \mathbf{u}_{\text{solid}} \cdot \mathbf{n} \text{ at solid boundaries.} \tag{3.16}$$

It consists of subtracting the pressure gradient from the fluid velocity $\mathbf{u}$ such that the resulting velocity $\mathbf{u}^{n+1}$ will be divergence-free and satisfy the free-slip boundary conditions.

Equation (3.14) has the following discretized form:

$$\begin{aligned} u^{n+1}_{i+1/2,j,k} &= u_{i+1/2,j,k} - \frac{\Delta t}{\rho}\frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x} \\ v^{n+1}_{i,j+1/2,k} &= v_{i,j+1/2,k} - \frac{\Delta t}{\rho}\frac{p_{i,j+1,k} - p_{i,j,k}}{\Delta x} \\ w^{n+1}_{i,j,k+1/2} &= w_{i,j,k+1/2} - \frac{\Delta t}{\rho}\frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta x} \end{aligned} \tag{3.17}$$

for each u-, v-, and w-component of the velocity in 3D.

We can now combine equations (3.14) and (3.15) by simply inserting $\mathbf{u} - \Delta t\frac{1}{\rho}\nabla p$ from (3.14) into (3.15) in the place of $\mathbf{u}^{n+1}$. After rearranging, the resulting pressure equation has the following form in 3D:

$$\frac{\Delta t}{\rho}\left(\frac{4p_{i,j,k}-p_{i+1,j,k}-p_{i,j+1,k}-p_{i,j,k+1}-p_{i-1,j,k}-p_{i,j-1,k}-p_{i,j,k-1}}{\Delta x^2}\right)= \tag{3.18}$$
$$-\left(\frac{u_{i+1/2,j,k}-u_{i-1/2,j,k}}{\Delta x}+\frac{v_{i,j+1/2,k}-v_{i,j-1/2,k}}{\Delta x}+\frac{w_{i,j,k+1/2}-w_{i,j,k-1/2}}{\Delta x}\right).$$

Note that the resulting equation (3.18) is the numerical approximation of the *Poisson equation*, $-\frac{\Delta t}{\rho}\nabla\cdot\nabla p=-\nabla\cdot\mathbf{u}$. These discrete equations can be straightforwardly modified to enforce static interior and exterior solid boundaries by requiring the appropriate normal velocity components to be zero.

Chapter 4

Problem Statement

The central task that we seek to address in this thesis is to fill a volumetric domain Ω with a divergence-free fluid vector field $\mathbf{u}$, given prescribed vector field data on its boundary $\partial\Omega = \Gamma$. That is, we seek to perform *incompressible vector field interpolation*.

Furthermore, one can also perform *incompressible vector field extrapolation* in addition to interpolation by introducing support for open rather than prescribed boundaries. In the presence of one or more unprescribed boundaries, we aim to find a vector field that smoothly extends into the unspecified boundaries.

For there to exist a feasible incompressible interpolated velocity field, an additional compatibility condition must be satisfied: the net flux of fluid entering and leaving the domain across the prescribed boundaries must be zero, meaning that the input boundary velocities must integrate to zero. (This stipulation can be straightforwardly upgraded to a prescribed net in/out flux.) It can be formulated as:

$$\iiint_{\Omega} \nabla \cdot \mathbf{u}\, dV = 0 = \iint_{\Gamma} \mathbf{u} \cdot \mathbf{n}\, dA \tag{4.1}$$

Let us now consider a few use cases.

4.1 Hole-Filling

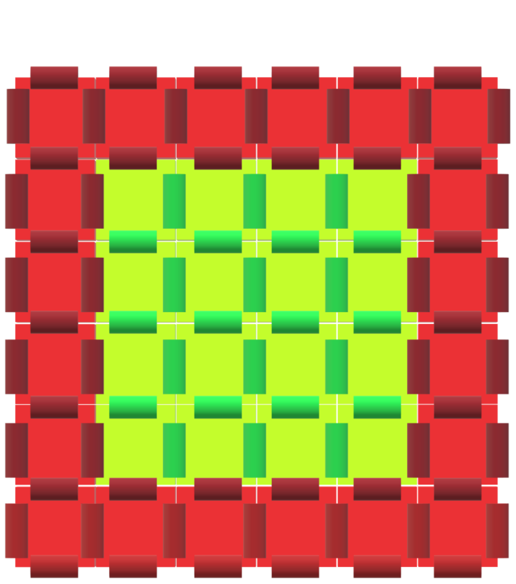
The simplest meaningful use case is a vector field hole-filling scenario, where Ω is a region of an existing simulation that has been deleted and must now be smoothly replaced. For example, if a region of a simulation yields a local motion that turns out to be undesirable (e.g., due to an object, some external force, etc.), one can simply delete that region and, on every frame, interpolate a new vector field to replace it. For the typical case of time-dependent simulations, the process should be performed for every timestep, using the corresponding boundary data from the existing simulation. Figure 4.1 illustrates the discrete grid layout for a low resolution example hole-filling scenario. In practice both the hole region and the surrounding simulation region will consist of many more voxels than shown here.

4.2 Copy-Paste

Another use case, explored by Sato et al. [2018], is to combine two (or more) previously computed simulations in a copy-and-paste fashion, as follows. Select the region to be copied in the source simulation. Place it into the desired location in the target simulation. Add a buffer region of a user-defined width around the source region, which will provide the flexibility to blend between the inputs. The buffer region should be at least one layer and the thicker it is the more flexibility the optimization has to find a smoother blend field. On the other hand, increasing the thickness of the buffer region makes our system bigger and as a result more costly to solve. Thus, we/the user choose the width of the buffer region such that the result is satisfyingly smooth and also has acceptable performance. Finally, using the boundary data from the source and target regions, interpolate a new vector field for the buffer region (Figure 4.2). This copy-and-paste methodology also naturally generalizes to N source regions being pasted into the target.

4.3 Open Boundary Scenarios

The scenarios above were based on vector field interpolation, i.e., when boundaries are fully prescribed. However, many other interesting use cases involving one or more open/ unprescribed boundaries can also be considered. In this thesis, we use the term "extrapolation" for scenarios with one or more unprescribed boundaries and the term "interpolation" for scenarios with fully prescribed boundaries. For example, by slicing a simulation into a series of layers along one axis, spacing them apart, and filling the spaces in by extrapolation, a *scene stretching* mechanism can be supported (Figure 4.3c). This scenario contains unprescribed boundaries where the region to be filled coincides with part of the exterior boundary along the sides, thus it requires an extrapolation scheme to mimic open


(a) 2D

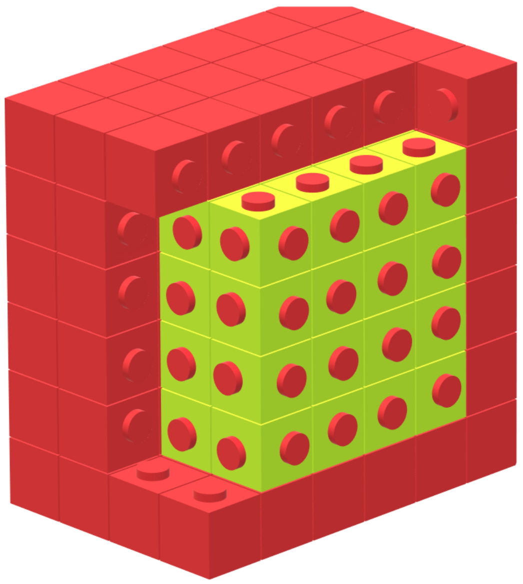
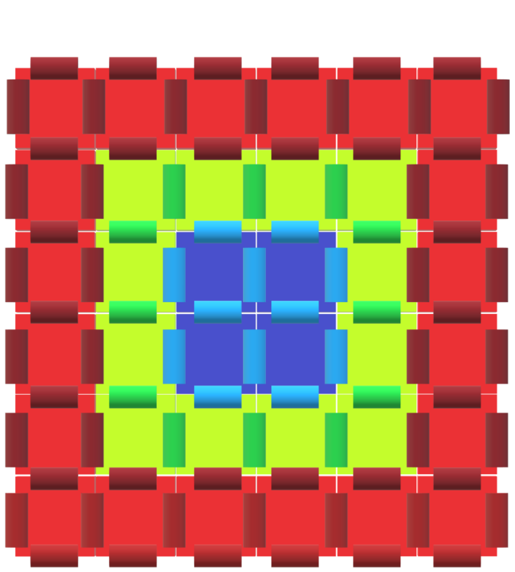

(b) 3D

Figure 4.1: **Hole-Filling structure in 2D (a) and in 3D (b).** Basic setup for filling a hole in a simulation on a MAC grid. (a) In 2D, each square (red or yellow) represents a grid cell and the velocity components (dark red or green) are sampled at the edges. Yellow squares represent the hole region to be filled using the boundary data on the edges (dark red) between the hole (yellow) and the input simulation (red). The interpolated velocity field is represented by the velocity components inside the hole (green). (b) Similarly, in 3D, voxels represent the grid cells and cylinders represent the velocity components which are sampled at cell faces. Yellow voxels represent the hole to be filled using the simulation data (red). Boundary velocity components (red cylinders) around the hole come from the input simulation (red).


(a) 2D

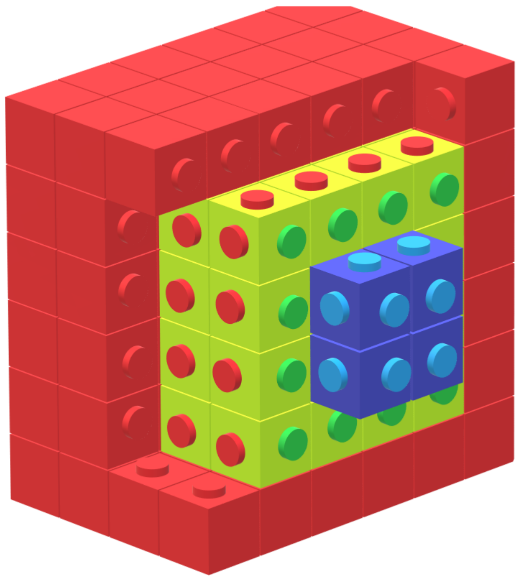
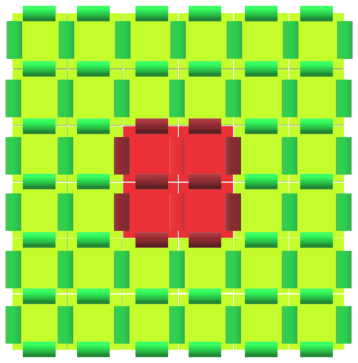

(b) 3D

Figure 4.2: **Copy-paste structure in 2D (a) and in 3D (b).** Basic setup for combining pieces of existing one source and one target simulation on a MAC grid. (a) Voxels represent the grid cells and cylinders represent the velocity components which are sampled at cell faces. Simulation domain is divided into three distinct regions: target (red), blend (yellow), and source (blue). Yellow voxels represent the blend region between source (blue) and target (red) regions. Outer boundary velocity components (red cylinders) of the blend region come from the target simulation (red) and inner boundary velocity components (blue cylinders) come from the source simulation (blue). Green cylinders represent the interpolated velocity components in the blend region. Note: in this low-resolution visualization, blend regions have one-cell width for simplicity; but, in practice we use wider blend regions in real applications so there is more flexibility in fitting a smooth blending field.

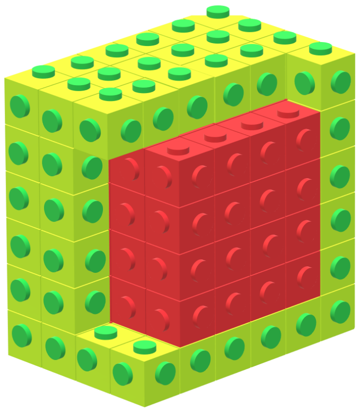
boundaries. Similarly, multiple distinct previously-computed simulations can afterwards be patched together side-by-side to synthesize a new larger combined scene. This idea can be viewed as a very basic (but efficient) domain decomposition strategy.

In another scenario, extrapolation could be used to change the behavior of the domain boundaries. For example, having a previously-computed simulation of a vector field with closed boundaries, one might wish to generate an animation that has the same inner vector field but with apparently open outer boundaries. This can be achieved by removing a few layers of cells on the boundary, and filling them back in with our extrapolation scheme.

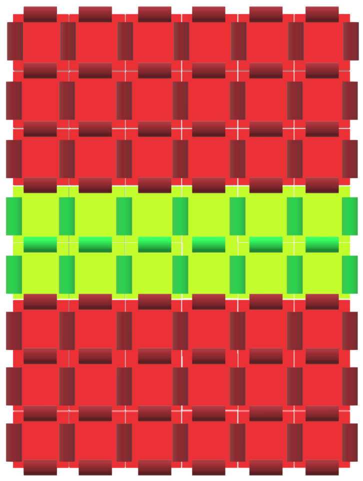
In addition to changing the outer boundary type from closed to open, another plausible use case for this extrapolation method can be domain extension. For example, you may have already computed a smoke simulation over a small region, but you want to expand the apparent size of the simulation in a way that is not immediately obvious to the viewer without re-simulation. Then, our extrapolation method can be used to fill in a large outer padding area surrounding the input simulation.



(a) 2D



(b) 3D



(c) Scene Stretching

Figure 4.3: **Extrapolation structure in 2D (a), in 3D (b), and Scene Stretching in 2D (c).** Basic setup for scenarios with open boundaries on a MAC grid. (a) In 2D, each square (red or yellow) represents a grid cell and the velocity components (dark red or green) are sampled at the edges. Yellow squares represent the extrapolation region to be filled using the boundary data on the edges (dark red) between the extrapolation region (yellow) and the input simulation (red). Exterior boundary of the extrapolation region is unprescribed (green velocity components). For the unprescribed boundaries, open or natural boundary conditions are used to allow the vector field to flow in and out through the unprescribed boundary. (b) Similarly, in 3D, yellow voxels represent the extrapolation region to be filled using the simulation data (red).

Chapter 5

Limitations of the Existing Methods

Having laid out the core problem, we now proceed to consider methods for solving it. In Chapter 2, we referred to several approaches that use PDEs for vector field interpolation as closest to our method. To motivate our approach, in this chapter, we investigate these methods and identify their limitations.

5.1 Vector Harmonic Interpolation

Let us think about the characteristics that we want our output field $\mathbf{u}$ to possess. First, $\mathbf{u}$ should match the boundary data $\mathbf{u}_b$ on the boundary Γ. Second, it should be smooth inside the interpolation domain Ω. We will make these requirements mathematically precise below.

Having only these two characteristics in mind, *harmonic interpolation* [Joshi et al., 2007] (adapted to vector data) might be a good choice and can be expressed as minimizing the vector Dirichlet energy ([Stein et al., 2020]):

$$\begin{aligned} &\underset{\mathbf{u}}{\operatorname{argmin}} \iiint_\Omega \frac{1}{2}\|\nabla \mathbf{u}\|_F^2 \, dV \\ &\text{subject to } \mathbf{u} = \mathbf{u}_b \text{ on } \Gamma. \end{aligned} \tag{5.1}$$

The minimizer satisfies $\nabla \cdot \nabla \mathbf{u} = 0$, i.e., a vector Laplacian on the velocity, with Dirichlet boundary condition (i.e., prescribed boundary values). In the context of a fluid problem, this Dirichlet boundary condition is also a no-slip condition.

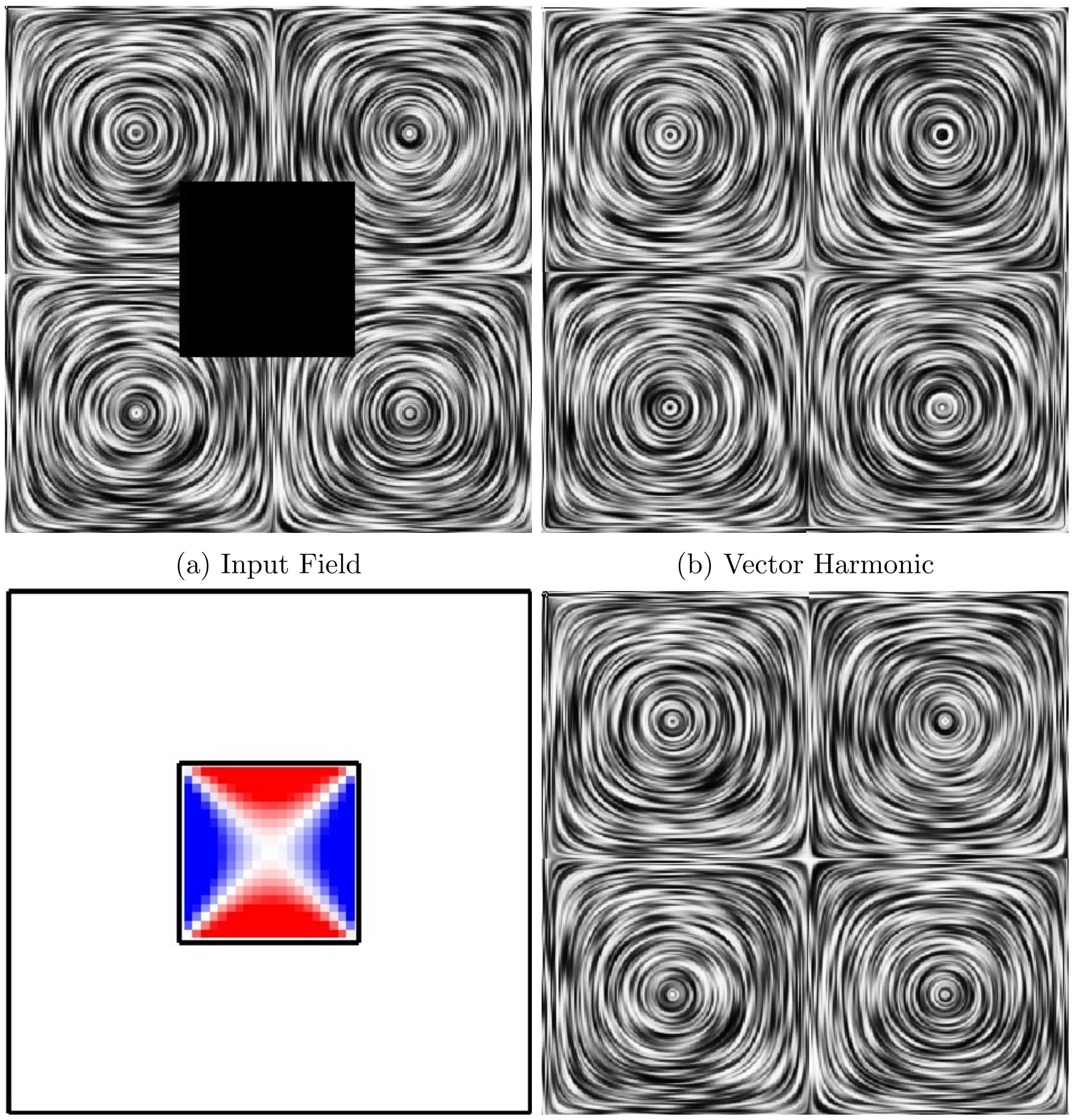
The vector Dirichlet energy, $\mathrm{E}(\mathbf{u}) = \iiint_\Omega \frac{1}{2}\|\nabla \mathbf{u}\|_F^2 \, dV$, measures how much a vector field $\mathbf{u}$ varies in Ω [Stein et al., 2020]. For example, the vector Dirichlet energy is zero for a constant vector field, whereas it will be high for rapidly changing vector fields. In equation (5.1), by minimizing the magnitude of the gradient of $\mathbf{u}$, this formulation fits the data while avoiding steep slopes in the output field.

In addition to smoothness and boundary agreement, we want our field to possess a third characteristic: incompressibility. However, Dirichlet energy is not guaranteed to produce a divergence-free field (Figure 5.1). One might enforce incompressibility by projecting $\mathbf{u}$ to be incompressible as a post-process. However, in doing so, the resulting harmonic flow's smoothness might be distorted but more importantly, tangential velocity discontinuities might occur at the boundaries because pressure projection only applies a free-slip condition (matching boundary normal components but not tangential components).

We use line integral convolution (LIC) [Cabral and Leedom, 1993] to visualize resulting two dimensional vector fields in our comparisons (such as Figure 5.1) between existing methods and our method since it provides a convenient way of visualizing static vector fields. While it is sufficient to use LIC to visualize the local behavior of vector fields, it does not capture the orientation and strength of these fields and is thus limited in some ways. Lastly since the LIC method starts from random noise and deforms it according to the vector field, we obtain different results from each run of the method.

5.2 Potential Flow

Inspired by the work of Hong and Kim [2004], Shi and Yu make use of potential functions to enforce the shape of rapidly changing target shapes for liquids [Shi and Yu, 2005]. The authors aim to generate forces that cause a liquid surface to move in such a way that it better coincides with a prescribed target surface and the liquid remains incompressible, at each frame. Using boundary forces, which are obtained by the difference between target shapes' boundary velocities and liquid boundary velocities, one can compute forces inside the liquid domain. The authors argue that it is important to obtain a divergence-free force field inside the liquid domain; otherwise, the pressure projection step can change the original force field that should be applied to match the target shape best in order to make the resulting field incompressible. To obtain a divergence-free force field, Shi and Yu [2005] formulate their force field as the gradient of a potential function H, $f = \nabla H$, and enforce the divergence-free constraint by requiring that $\nabla \cdot \nabla H = 0$. Their method has



(a) Input Field (b) Vector Harmonic

(c) Divergence Plot of (b) (d) Constrained Biharmonic (Ours)

Figure 5.1: **Vector Harmonic Interpolation** vs. our method on a four-vortex hole-filling test. (a) The input simulation with black indicating the hole region to be filled in. (b) The result of vector harmonic interpolation. (c) A visualization of the divergence of the field in (b), where red indicates positive and blue indicates negative. (d) The result of our method looks similar in the static image, but has uniformly zero divergence. (Note: The LIC visualization does not indicate velocity magnitudes.)

the following form:

$$\nabla^2 H = 0,\ \nabla H|_{\partial\Omega} = f|_{\partial\Omega} \tag{5.2}$$

where $\nabla^2 = \nabla \cdot \nabla$ and f is the boundary force.

While Shi and Yu [2005] use this idea to construct a divergence-free force field, Nielsen and Bridson [2011] use potential flow for velocity field interpolation letting the interpolated velocity $\mathbf{u}$ be the gradient of a scalar potential ϕ, $\mathbf{u} = \nabla\phi$. Nielsen and Bridson [2011] similarly define the potential flow using the Laplace equation, $\nabla \cdot \nabla\phi = 0$, with Neumann boundary condition, $\partial\phi/\partial n = \mathbf{u} \cdot \mathbf{n}$ to match the boundary velocity normal components. Since this approach offers a divergence-free interpolated field, it represents another possible solution to our task.

As we did for the harmonic interpolation approach, we can express the potential flow problem in optimization form to provide additional insight. We simply minimize the kinetic energy of the flow, subject to matching boundary (normal) velocities, giving

$$\begin{gathered}\underset{\phi}{\operatorname{argmin}} \iiint_\Omega \frac{1}{2}\|\nabla\phi\|^2\, dV \\ \text{subject to } \nabla\phi \cdot \mathbf{n} = \mathbf{u}_b \cdot \mathbf{n} \text{ on } \Gamma.\end{gathered} \tag{5.3}$$

This form yields a scalar Laplacian problem, $\nabla \cdot \nabla\phi = 0$, with the Neumann boundary condition, i.e., exactly the same PDE as discussed above. Since $\nabla \cdot \nabla\phi = \nabla \cdot \mathbf{u} = 0$, the resulting flow is incompressible. Unfortunately, Bhattacharya et al. [2012] demonstrated that the irrotationality property sacrifices important rotational motions, as we also demonstrate in Figure 5.2.

5.3 Stokes Flow

We can combine the strengths of harmonic interpolation (smoothness, no-slip boundaries) and potential flow (incompressibility) by adding an explicit divergence-free constraint to (5.1):

$$\begin{gathered}\underset{\mathbf{u}}{\operatorname{argmin}} \iiint_\Omega \frac{1}{2}\|\nabla\mathbf{u}\|_F^2\, dV \\ \text{subject to } \nabla \cdot \mathbf{u} = 0 \text{ on } \Omega, \\ \mathbf{u} = \mathbf{u}_b \text{ on } \Gamma.\end{gathered} \tag{5.4}$$

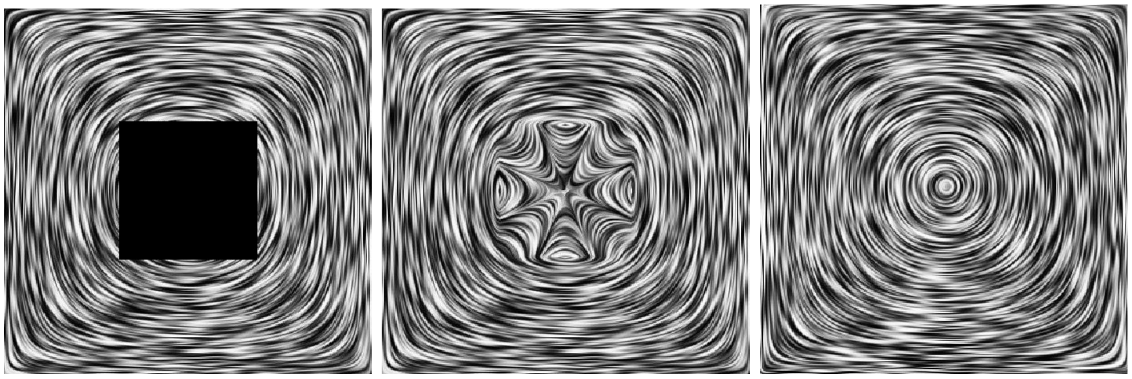

(a) Input Field (b) Potential Flow (c) Ours

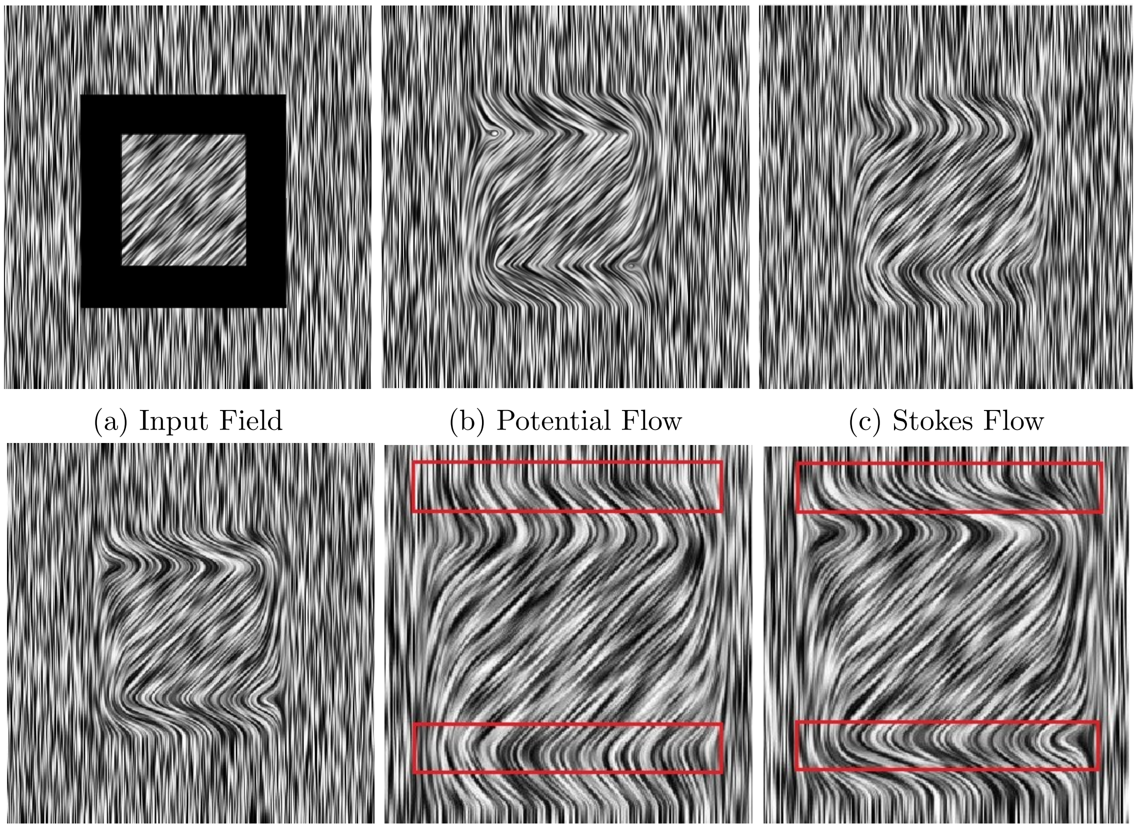
Figure 5.2: **Potential Flow** vs. our method on a rotational hole-filling test. (a) A square hole in the middle of the image is filled in using (b) potential flow and (c) our constrained biharmonic interpolation. Although potential flow is divergence free, it cannot recover the expected rotation and yields an unnatural result.

Applying constraints with Lagrange multiplier p, the optimality conditions yield the *steady Stokes equations* with unit viscosity,

$$\Delta \mathbf{u} - \nabla p = 0, \tag{5.5}$$
$$\nabla \cdot \mathbf{u} = 0, \tag{5.6}$$

where the operator $\Delta = \nabla \cdot \nabla$ indicates the Laplacian (in this case, the vector Laplacian). Bhattacharya et al. [2012] were the first to suggest using Stokes flow for fluid interpolation and demonstrate its superior rotational behavior compared to potential flow.

As we discuss more in Chapter 6, this approach is relatively effective but still provides insufficient smoothness along the boundary and does not support open boundaries. It produces sharp transitions at the boundaries between regions (Figure 5.3c and 5.3e). This is because Stokes flow is unable to match the gradients of velocities at the boundaries resulting in velocity gradient discontinuities at the boundaries. For this reason, the combined flow lacks visual smoothness.


(a) Input Field (b) Potential Flow (c) Stokes Flow

(d) Ours (e) Stokes – Zoomed (f) Ours – Zoomed

Figure 5.3: **Stokes Flow** vs. our method on a fluid cut-and-paste test where the outer flow is vertical and the (pasted) inner flow is diagonal. While both are continuous and divergence-free, Stokes flow interpolation (a) only matches velocity values at the boundaries. Constrained biharmonic interpolation (b) additionally matches gradients at boundaries. As a result, Stokes flow interpolation exhibits nonsmooth velocity transitions between regions, especially along the topmost and bottommost borders.

Chapter 6

Proposed Methods, Discretization and Solution Procedure

In Chapter 5, we provided our insights on existing methods and their limitations. We concluded that the resulting Stokes flow field is smooth, incompressible, and satisfies no-slip boundaries. However, this remains insufficient. To understand why, let us now consider our copy-paste application. Given previously simulated source and target flow data, the newly interpolated flow indeed satisfies smoothness, incompressibility, and the no-slip boundary condition. However, the no-slip boundary condition $\mathbf{u} = \mathbf{u_b}$ provides only C^0 smoothness across the boundary; the vector field's derivative is nonsmooth at the boundaries where the flow crosses from the previously simulated region into the newly interpolated region. Visually, this effect can produce sharp transitions at the boundaries that highlight the borders of the interpolation domain. This becomes a significant limitation because the newly interpolated flow combined with the previously simulated flow does not give the desired illusion of a single natural flow, which is our main task. Moreover, Stokes flow and other existing methods do not provide support for open/unprescribed boundaries (i.e., extrapolation rather than interpolation). We now introduce our new formulations that address these limitations.

6.1 Divergence-Constrained Biharmonic Interpolation

Our approach is to upgrade our objective function in (5.4) from the vector Dirichlet energy to the vector Laplacian energy. Intuitively, this change implies that, rather than minimizing the steepness (gradient magnitude) of the vector field, we will be minimizing its curvature. A useful analogy to consider is cubic spline interpolation, which yields the minimum curvature curve that passes through a given set of points. In classic clamped boundary conditions for splines, we specify not just the values/positions of the curve endpoints but also their slopes. In much the same way, since it is a higher order energy, the vector Laplacian energy necessitates providing an additional boundary condition, for which various options exist (see e.g., Stein et al. [2018]). The advantage of this choice is that, by specifying the vector field values and (first) derivatives across the boundary, we recover C^1 continuity and achieve the desired visual smoothness:

$$\begin{aligned} \underset{\mathbf{u}}{\operatorname{argmin}} \iiint_{\Omega} \frac{1}{2}\|\Delta \mathbf{u}\|_F^2 \, dV \qquad (6.1)\\ \text{subject to } \nabla \cdot \mathbf{u} = 0 \text{ on } \Omega, \\ \mathbf{u} = \mathbf{u}_b \text{ on } \Gamma, \\ \nabla \mathbf{u} \cdot \mathbf{n} = \nabla \mathbf{u}_b \cdot \mathbf{n} \text{ on } \Gamma. \end{aligned}$$

Again using a Lagrange multiplier p to enforce incompressibility, the optimality conditions yield the partial differential equations:

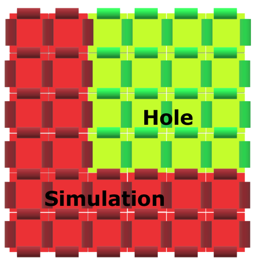
$$\Delta^2 \mathbf{u} - \nabla p = 0, \qquad (6.2)$$

$$\nabla \cdot \mathbf{u} = 0, \qquad (6.3)$$

where Δ^2 represents the bilaplacian operator. The solution to these equations is our *divergence-constrained biharmonic interpolant.*

6.2 Divergence-Constrained Biharmonic Extrapolation

Equation (6.1) is well-suited to interpolation, i.e., where the entire boundary is prescribed. However, one might alternatively wish to fill in a region that has one or more boundaries that lack specified values and derivatives (as suggested in Section 4.3). Consider the hole-filling task in a case where the region to be filled happens to overlap part of the exterior boundary of the original simulation domain (see inset). While the boundary data (values and derivatives) between the hole and the original simulation is drawn from the original simulation, the



exterior part of the boundary of the hole is not specified at all. In this case, we would rather use *"free" or natural boundary conditions* without specifying values and derivatives in order to find an optimally smooth extension of the vector field into the unspecified boundary. In this way, we can mimic the effect of an open boundary, where the vector field is free to flow in or out of the exterior boundary to optimize smoothness.

Smoothness energies involving second derivatives, such as the Laplacian energy, $\mathrm{E}_{\Delta^2}(\mathbf{u}) = \iiint_\Omega \frac{1}{2}\|\Delta\mathbf{u}\|_F^2 \ dV$, in (6.1), are by definition more expressive compared to first-order smoothness energies, such as the Dirichlet energy in (5.1); however, they require careful treatment of boundary conditions. As illustrated by Stein et al. [2018], they suffer from unnatural and unfavorable artifacts along boundaries when boundary conditions are not chosen carefully. In Section 6.1, we specified values and derivatives for the Laplacian energy, which was sufficient for prescribed boundary behavior. However, Stein et al. showed (in the scalar case) that the Laplacian energy fails to produce the desired behavior and leads to boundary-dependent bias in the presence of natural boundary conditions, where there are no explicit boundary conditions. As a result, Stein et al. propose using the *Hessian energy* which is also second-order and provides well-behaved solutions for natural boundary conditions.

Inspired by the work of Stein et al. [2018], we replace the Laplacian energy used in (6.1) with the Hessian energy. The Hessian energy is extended from the scalar to the vector case by applying it in a componentwise fashion. Our formulation becomes:

$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{u}} \iiint_\Omega \frac{1}{2}(\|\mathbf{H}_u\|_F^2 + \|\mathbf{H}_v\|_F^2)\ dV \\ \text{subject to } &\nabla\cdot\mathbf{u} = 0 \text{ on } \Omega, \\ &\mathbf{u} = \mathbf{u}_b \text{ on } \Gamma_\mathrm{P}, \\ &\nabla\mathbf{u}\cdot\mathbf{n} = \nabla\mathbf{u}_b\cdot\mathbf{n} \text{ on } \Gamma_\mathrm{P}, \end{aligned} \tag{6.4}$$

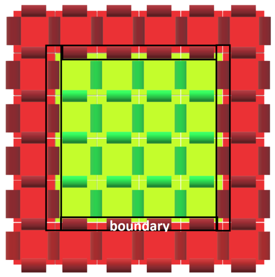
where $\mathbf{H}_u$ and $\mathbf{H}_v$ represent the Hessian operator applied to each velocity component u and v in two dimensions and Γ_P represents the prescribed part of the boundary. In our problem, the whole boundary consists of two parts: the prescribed and the natural part, where the latter is denoted as Γ_N, i.e., $\Gamma = \Gamma_\mathrm{P} \cup \Gamma_\mathrm{N}$. Our formulation above does not explicitly mention Γ_N because the natural boundary conditions are handled "naturally" (i.e., implicitly, as a result of the minimization).

Stein et al. [2018] show that the optimality conditions of the Hessian energy yield the same PDE operator (i.e., the bilaplacian: Δ^2) as the Laplacian energy in the interior of the domain. Furthermore, when both Dirichlet and Neumann boundary conditions are defined (i.e., $\mathbf{u} = f$ and $\nabla\mathbf{u}\cdot\mathbf{n} = g$ on Γ), minimizing the Hessian and the Laplacian energies also yield the same PDE operator (i.e., the bilaplacian: Δ^2) [Stein et al., 2018] since boundary values and normal derivatives uniquely define biharmonic functions [Evans, Lawrence C., 2010]. However, in the presence of natural boundary conditions, i.e., where there are no explicit boundary conditions, the solution of the Hessian energy differs from the solution of the Laplacian energy; the Hessian energy's solution is the desired unbiased free boundary conditions. To reiterate, the fundamental difference between our formulation in (6.4) and that of Stein et al. [2018] is that we consider the case of a vector field rather than a scalar field, and we additionally enforce incompressibility.

6.3 Discretization

In our implementation, the simulation domain is discretized using the standard staggered MAC grid and similarly, the input boundary data is drawn from standard grid-based fluid simulations described in Chapter 3. Recall that in this discretization, velocity components are placed on the cell faces and Lagrange multiplier (pressure) degrees of freedom are placed at grid cell centers (Figure 3.1).

In our discretization, cell faces, which contain the velocity data, separate distinct regions, such as the simulation (red) and the hole (yellow), and become the boundary between different regions (see inset). The boundary data is taken from the input simulation(s).



The divergence-free constraint on each cell is formulated using finite differences (Section 3.3). The discrete bilaplacian operator is formed by applying the classic Laplacian stencil twice, which leads to a 13-point stencil in 2D and a 25-point stencil in 3D. Below, we describe how the bilaplacian stencil is derived in 2D and in 3D.

6.3.1 Discrete Bilaplacian

The discrete Laplacian operator has the following formula in 2D:

$$(\Delta_h u)_{i,j} = (D^2_{xx}u)_{i,j} + (D^2_{yy}u)_{i,j} \tag{6.5}$$

where $(D^2_{xx}u)_{i,j} = \frac{u_{i-1,j}+u_{i+1,j}-2u_{i,j}}{h^2}$, $(D^2_{yy}u)_{i,j} = \frac{u_{i,j-1}+u_{i,j+1}-2u_{i,j}}{h^2}$, and the subscript h refers to the grid cell size. Inserting $(D^2_{xx}u)_{i,j}$ and $(D^2_{yy}u)_{i,j}$ into (6.5), the Laplacian operator is:

$$(\Delta_h u)_{i,j} = \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2}. \tag{6.6}$$

As previously mentioned, the discrete bilaplacian can be derived by applying the Laplacian operator twice:

$$(\Delta_h^2 u)_{i,j} = (\Delta_h(\Delta_h u))_{i,j} = (D_{xx}^2(\Delta_h u))_{i,j} + (D_{yy}^2(\Delta_h u))_{i,j}. \tag{6.7}$$

Inserting the right-hand side of (6.6) in place of $(\Delta_h u)_{i,j}$ into (6.7), and then applying the second-order partial derivative operators, i.e., D_{xx}^2 and D_{yy}^2, the bilaplacian becomes:

$$\begin{aligned}(\Delta_h^2 u)_{i,j} = \frac{1}{h^4}(&u_{i-2,j} + u_{i,j} - 2u_{i-1,j} + u_{i,j} + u_{i+2,j} - 2u_{i+1,j}\\ &+ u_{i-1,j-1} + u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j+1} + u_{i+1,j+1} - 2u_{i,j+1}\\ &- 4(u_{i-1,j} + u_{i+1,j} - 2u_{i,j})\\ &+ u_{i-1,j-1} + u_{i-1,j+1} - 2u_{i-1,j} + u_{i+1,j-1} + u_{i+1,j+1} - 2u_{i+1,j}\\ &+ u_{i,j-2} + u_{i,j} - 2u_{i,j-1} + u_{i,j} + u_{i,j+2} - 2u_{i,j+1}\\ &- 4(u_{i,j-1} + u_{i,j+1} - 2u_{i,j})).\end{aligned} \tag{6.8}$$

Rearranging the terms in (6.8), we obtain the 13-point bilaplacian stencil in 2D:

$$\begin{aligned}(\Delta_h^2 u)_{i,j} = \frac{1}{h^4}(&20u_{i,j} - 8(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})\\ &+ 2(u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j+1} + u_{i+1,j-1})\\ &+ (u_{i-2,j} + u_{i+2,j} + u_{i,j-2} + u_{i,j+2})).\end{aligned} \tag{6.9}$$

The discrete Laplacian operator is extended to 3D with an additional second-order partial derivative with respect to the additional variable, i.e., z:

$$(\Delta_h u)_{i,j,k} = (D_{xx}^2 u)_{i,j,k} + (D_{yy}^2 u)_{i,j,k} + (D_{zz}^2 u)_{i,j,k}, \tag{6.10}$$

where $(D_{xx}^2 u)_{i,j,k} = \frac{u_{i-1,j,k}+u_{i+1,j,k}-2u_{i,j,k}}{h^2}$, $(D_{yy}^2 u)_{i,j,k} = \frac{u_{i,j-1,k}+u_{i,j+1,k}-2u_{i,j,k}}{h^2}$, and $(D_{zz}^2 u)_{i,j,k} = \frac{u_{i,j,k-1}+u_{i,j,k+1}-2u_{i,j,k}}{h^2}$. Then:

$$(\Delta_h u)_{i,j,k} = \frac{u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + +u_{i,j,k+1} - 6u_{i,j,k}}{h^2}. \tag{6.11}$$

Similarly, in 3D, the discrete bilaplacian is:

$$(\Delta_h^2 u)_{i,j,k} = (\Delta_h(\Delta_h u))_{i,j,k} = (D_{xx}^2(\nabla_h u))_{i,j,k} + (D_{yy}^2(\nabla_h u))_{i,j,k} + (D_{zz}^2(\nabla_h u))_{i,j,k}. \tag{6.12}$$

The right-hand side of (6.11) is inserted into (6.12) in place of $(\Delta_h u)_{i,j,k}$, and then the second-order derivatives, i.e., D^2_{xx}, D^2_{yy}, and D^2_{zz}, are applied. The resulting 25-point bilaplacian stencil in 3D is:

$$
\begin{aligned}
(\Delta_h^2 u)_{i,j,k} = \frac{1}{h^4}(&42u_{i,j,k} - 12(u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1}) \\
&+ u_{i+2,j,k} + u_{i-2,j,k} + u_{i,j+2,k} + u_{i,j-2,k} + u_{i,j,k+2} + u_{i,j,k-2} \\
&+ 2(u_{i+1,j+1,k} + u_{i-1,j+1,k} + u_{i-1,j-1,k} + u_{i+1,j-1,k} + u_{i+1,j,k+1} + u_{i-1,j,k+1} \\
&+ u_{i-1,j,k-1} + u_{i+1,j,k-1} + u_{i,j+1,k+1} + u_{i,j-1,k+1} + u_{i,j-1,k-1} + u_{i,j+1,k-1})).
\end{aligned} \tag{6.13}
$$

Recall that we will use this bilaplacian operator for pure interpolation problems. For problems with open or natural boundaries we must instead discretize our Hessian-based energy.

6.3.2 Discrete Hessian-based Stencil Derivation

The Hessian energy is discretized using a sparse matrix $\mathbf{H}$ [Stein et al., 2018]:

$$
\mathbf{H} = \begin{bmatrix} \mathbf{H}^T_{xx} \\ \mathbf{H}^T_{yy} \\ \sqrt{2}\mathbf{H}^T_{xy} \end{bmatrix}, \tag{6.14}
$$

where $\mathbf{H}_{xx}(u) \approx \partial^2 u/\partial x^2$, $\mathbf{H}_{yy}(u) \approx \partial^2 u/\partial y^2$, and $\mathbf{H}_{xy}(u) \approx \partial^2 u/\partial x \partial y$. Using the standard, second-order approximation of the second-order derivatives, the Hessian of a scalar velocity component u has the following discrete form:

$$
\mathbf{H_u} = \begin{bmatrix} (u_{i-1,j} - 2u_{i,j} + u_{i+1,j})/h^2 \\ (u_{i,j-1} - 2u_{i,j} +_{i,j+1})/h^2 \\ \frac{\sqrt{2}}{4}(u_{i-1,j-1} + u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1})/h^2 \end{bmatrix}. \tag{6.15}
$$

By plugging $\mathbf{H_u}$ from (6.15) into $\mathrm{E}_{\mathbf{H}^2} = \iiint_\Omega \frac{1}{2}\|\mathbf{H_u}\|_F^2 \, dV$, we obtain the u-contribution to the discrete Hessian energy:

$$
\begin{aligned}
\mathrm{E}_{\mathbf{H}^2}(u) = \frac{1}{2h^4}\sum_{i,j} vol_{i,j} \; ((u_{i-1,j} - 2u_{i,j} + u_{i+1,j})^2 + (u_{i,j-1} - 2u_{i,j} + u_{i,j+1})^2 \\
+ \frac{1}{8}(u_{i-1,j-1} + u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1})^2),
\end{aligned} \tag{6.16}
$$

where $vol_{i,j}$ are volumetric integration weights (i.e., the discrete approximation of dV from the continuous integral) for the corresponding $u_{i,j}$ sample points. Since the energy integral is over the interior of the interpolation/extrapolation region, these weights implicitly delineate the boundary between the interior being solved for and the exterior region that is unprescribed (i.e., where natural boundary conditions should be applied). While partial volume fractions or other quadrature schemes could be used for approximating the integral, we employ simple binary (inside vs. outside) weights. (The actual volume of a cell introduces a rescaling that does not change the solution, thus we ignore it and treat the weights as binary.) We set the weights as follows. First, we call cells inside the extrapolation domain active and cells outside inactive. Since velocity samples lie on the faces between cells, we set the $vol_{i,j}$ weight for a velocity sample to 1 if it is a face of any active cell, and otherwise set it to 0.

To derive the stencil for the minimizer u in (6.4), we take the derivative of the discrete Hessian energy, $\mathrm{E}_{\mathbf{H}^2}$, from (6.16) with respect to one particular velocity sample $u_{i,j}$ and equate it to 0:

$$\begin{aligned}
\frac{\partial}{\partial u_{i,j}}\mathrm{E}_{\mathbf{H}^2}(u) =& \frac{1}{h^4} vol_{i,j}\left(-2(u_{i-1,j}-2u_{i,j}+u_{i+1,j})-2(u_{i,j-1}-2u_{i,j}+u_{i,j+1})\right) \\
&+ vol_{i-1,j}\left(u_{i-2,j}-2u_{i-1,j}+u_{i,j}\right) \\
&+ vol_{i+1,j}\left(u_{i,j}-2u_{i+1,j}+u_{i+2,j}\right) \\
&+ vol_{i,j-1}\left(u_{i,j-2}-2u_{i,j-1}+u_{i,j}\right) \\
&+ vol_{i,j+1}\left(u_{i,j}-2u_{i,j+1}+u_{i,j+2}\right) \\
&+ (vol_{i-1,j-1}\left(u_{i-2,j-2}+u_{i,j}-u_{i-2,j}-u_{i,j-2}\right) \\
&+ \ vol_{i+1,j+1}\left(u_{i,j}+u_{i+2,j+2}-u_{i,j+2}-u_{i+2,j}\right) \\
&- \ vol_{i-1,j+1}\left(u_{i-2,j}+u_{i,j+2}-u_{i-2,j+2}-u_{i,j}\right) \\
&- \ vol_{i+1,j-1}\left(u_{i,j-2}+u_{i+2,j}-u_{i,j}-u_{i+2,j-2}\right)) \ / \ 8 = 0.
\end{aligned} \tag{6.17}$$

Rearranging the terms in (6.17), we obtain the 13-point Hessian-based stencil in 2D:

$$
\begin{aligned}
(\mathbf{H}_h^2 u)_{i,j} =& \frac{1}{h^4}(u_{i,j}\,(8vol_{i,j} + vol_{i-1,j} + vol_{i+1,j} + vol_{i,j-1} + vol_{i,j+1} \\
&+ \frac{1}{8}\,(vol_{i-1,j-1} + vol_{i+1,j+1} - vol_{i-1,j+1} - vol_{i+1,j-1})) \\
&+ u_{i-1,j}\,(-2vol_{i,j} - 2vol_{i-1,j}) + u_{i+1,j}\,(-2vol_{i,j} - 2vol_{i+1,j}) \\
&+ u_{i,j-1}\,(-2vol_{i,j} - 2vol_{i,j-1}) + u_{i,j+1}\,(-2vol_{i,j} - 2vol_{i,j+1}) \\
&+ u_{i-2,j}\left(vol_{i-1,j} - \frac{1}{8}\,(vol_{i-1,j-1} + vol_{i-1,j+1})\right) \\
&+ u_{i+2,j}\left(vol_{i+1,j} - \frac{1}{8}\,(vol_{i+1,j+1} + vol_{i+1,j-1})\right) \\
&+ u_{i,j-2}\left(vol_{i,j-1} - \frac{1}{8}\,(vol_{i-1,j-1} + vol_{i+1,j-1})\right) \\
&+ u_{i,j+2}\left(vol_{i,j+1} - \frac{1}{8}\,(vol_{i+1,j+1} + vol_{i-1,j+1})\right) \\
&+ u_{i-2,j-2}\frac{1}{8}vol_{i-1,j-1} + u_{i+2,j+2}\frac{1}{8}vol_{i+1,j+1} \\
&+ u_{i-2,j+2}\frac{1}{8}vol_{i-1,j+1} + u_{i+2,j-2}\frac{1}{8}vol_{i+1,j-1}).
\end{aligned} \tag{6.18}
$$

6.3.3 Resulting Linear System

In the pure interpolation problem, (6.2) and (6.3) lead to the following linear system:

$$
\begin{bmatrix} B_x & 0 & 0 & D_x^T \\ 0 & B_y & 0 & D_y^T \\ 0 & 0 & B_z & D_z^T \\ D_x & D_y & D_z & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \end{bmatrix} = \mathbf{b}, \tag{6.19}
$$

where B_x, B_y, B_z are the scalar bilaplacian operator on each velocity component and D_x, D_y, D_z are the components of the discrete divergence operator for each cell. The solution is the set of interior cell face velocities u, v, w and interior pressures p. The right-hand-side vector $\mathbf{b}$ is initially set to zero. Then, when setting up the stencils for each unknown using the bilaplacian stencil in (6.13) and the stencil for the discrete divergence in (3.9), whenever a stencil entry "touches" a sample that lies outside the interpolation region, we transfer its contribution to the right-hand-side using the known value at that location (drawn from the input simulations). This simple boundary condition treatment straightforwardly enforces continuous value and derivative conditions on velocity, as well as Neumann conditions on pressure.

For the Hessian-based extrapolation case, we replace the B terms with the Hessian-based stencil in (6.18). Binary per-face weights are used to approximate the integral in (6.4), to distinguish active and inactive cells and thereby enforce the natural boundary conditions.

6.3.4 Getting rid of the Pressure Nullspace

The linear system in (6.19) leads to a one-dimensional pressure null space. It arises because our equations only constrain the gradient of p. We can eliminate this null space by simply pinning one pressure degree of freedom to zero. Doing this would not change the solution for the velocity field. Let us consider a viable pressure field solution p_s; then, for any constant c, $p_s + c$ is also a valid solution because adding a constant does not change the value of the field's gradient. For this, we perform the following steps:

- Select a row in the matrix $\begin{bmatrix} D_x & D_y & D_z \end{bmatrix}$ and delete all the entries in the corresponding row (set them to zero).
- Place a "1" on the diagonal in the selected row of the matrix $\begin{bmatrix} D_x & D_y & D_z \end{bmatrix}$.
- Lastly, place a "0" on the other entries in the same column as the newly placed 1, to maintain the symmetry of the whole system by deleting the appropriate entries.

6.4 Solution Procedure

The linear system in (6.19) is symmetric indefinite (i.e., a saddle point system) for which a wide array of solution techniques have been proposed [Benzi et al., 2005]. We adopt a simple conjugate-gradient (CG) scheme as follows. Our system has the general form:

$$\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \tag{6.20}$$

where $A = \begin{bmatrix} B_x & 0 & 0 \\ 0 & B_y & 0 \\ 0 & 0 & B_z \end{bmatrix}$, $C = \begin{bmatrix} D_x & D_y & D_z \end{bmatrix}$, and $\mathbf{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$. The system in (6.20) corresponds to the following equations:

$$A\mathbf{u} + C^T p = f \tag{6.21}$$

$$C\mathbf{u} = g. \tag{6.22}$$

Then, we apply a Schur complement transformation to our system to eliminate the velocity variables $\mathbf{u}$, by plugging $\mathbf{u} = A^{-1}(-C^T p + f)$ from (6.21) into (6.22) in place of $\mathbf{u}$. By doing so, we obtain the following system:

$$CA^{-1}C^T p = CA^{-1} f - g. \tag{6.23}$$

The resulting system in (6.23) is a symmetric positive definite (SPD) system allowing us to compute a solution using a standard CG solver. After solving for p, we can plug it back in (6.21) to recover $\mathbf{u}$. One problem would be that A^{-1} is potentially a much denser matrix due to fill-in, so we would prefer not to explicitly form it. Fortunately, Krylov solvers like CG do not require us to provide an explicit matrix. Instead, it is sufficient to provide the operation of that matrix on a vector, i.e., given x, we only need to be able to determine the result $y = CA^{-1}C^T x$. In our case, we can compute the matrix-vector product $CA^{-1}C^T x$ in three steps:

- Multiply $C^T x$ to obtain a vector c, where $c = C^T x$, i.e., the matrix-vector product.
- Then, we need to compute $A^{-1}c$ to obtain a vector d, where $d = A^{-1}c$. To compute d, we can apply a forward/backward substitution to solve the system $Ad = c$, after performing LU factorization on A.
- Finally, multiply Cd to obtain the final result of the product.

To compute the right-hand-side matrix-vector product $CA^{-1}f$ in (6.23) avoiding explicitly forming a possibly dense matrix A^{-1}, we adapt the same solution procedure described above in the second step as follows. To compute the resulting vector d of the matrix-vector product $A^{-1}f$, where $d = A^{-1}f$, we apply a forward-backward substitution to solve the system $Ad = f$ without explicitly forming A^{-1}. Lastly, after solving for p, we perform one last forward/backward substitution to recover $\mathbf{u}$ in (6.21), i.e., $A\mathbf{u} = f - C^T p$. In the common case where the shape and location of the interpolation region does not change over time, we can perform the factorization on A once and reuse it on every time step.

For example, let us consider a hole-filling scenario with a 20-cell thick hole/interpolation region. In this scenario, in 3D, there are 8000 cells in the interpolation region which leads to A being of size 22800×22800. As illustrated in this example, A can be a large matrix, therefore; it is important not to form a possibly dense matrix A^{-1} explicitly and to compute the matrix-vector products with A^{-1} efficiently.

For completeness, we list the details of the CG algorithm (Algorithm 1) we use to solve (6.23) for p. The inputs are $b = CA^{-1}f - g$ and $M = CA^{-1}C^T$. However, we do not need to explicitly form the matrix $M = CA^{-1}C^T$. Instead, we can just compute the matrix-vector product Mx_0 and Mp_k (see Algorithm 1) using the 3-step process described above involving two matrix-vector multiplications and one forward/backward substitution with the factors of A.

Algorithm 1 Conjugate-Gradient (CG)

```
1: $r_0 := b - \mathbf{M}x_0$
2: If $r_0$ is sufficiently small, then **return** $x_0$ as the result
3: $p_0 := r_0$
4: $k := 0$
5: **while** *true* **do**
6:     $prod = \mathbf{M}p_k$                    ▷ Compute using the 3-step procedure
7:     $\alpha_k := \frac{r_k^T r_k}{p_k^T prod}$
8:     $x_{k+1} := x_k + \alpha_k p_k$
9:     $r_{k+1} := r_k - \alpha_k prod$
10:    If $r_{k+1}$ is sufficiently small, then exit the loop
11:    $\beta_k := \frac{{r_k+1}^T r_{k+1}}{r_k^T r_k}$
12:    $p_{k+1} := r_{k+1} - \beta_k p_k$
13:    $k := k + 1$
14: **end while**
15: **return** $x_{k+1}$ as the result
```

As alluded to above, there are many other approaches one could take to further accelerate the solution of this system, such as incorporating multigrid techniques or preconditioning, but the approach above was sufficient for our purposes.

As in the work of Sato et al. [2018], the full trajectories of any material to be visualized/rendered (e.g., smoke densities and/or tracer particles) must be recomputed from scratch because the new combined vector field differs significantly from its input field(s). The data must be advected through the new velocity field to yield a consistent final result.

Fortunately, this can often be done efficiently and in parallel, since each (passive) particle's motion does not affect other particles. The velocity field on the grid is interpolated to particles using bilinear interpolation and we trace their trajectories with second-order Runge-Kutta.

6.5 One Fluid-Editing Timestep

Putting everything together, we obtain Algorithm 2 for a single fluid-editing timestep. The inputs are input simulation(s) per-frame velocity field data and the output is the edited velocity field. After obtaining the resulting velocity field over the whole domain, tracer particles and/or densities are advected via the output velocity field using standard Lagrangian or semi-Lagrangian advection procedures.

Algorithm 2 One Fluid-Editing Timestep

```
1: Read input simulation(s) current timestep velocity field data into appropriate grid
   locations (the part of the domain with known values).
2: Perform divergence-constrained biharmonic interpolation (Eq 6.1) or extrapolation (Eq
   6.4) on the grid locations without known values, i.e., interpolation/extrapolation do-
   main, as follows:
3: if interpolation then
                              ▷ In case of hole-filling & copy-paste scenarios (Sec 4.1, 4.2)
4:     Use the scalar bilaplacian operator (Eq 6.9 in 2D & 6.13 in 3D) to fill in $B_x$, $B_y$,
       and $B_z$ in Eq (6.19).
5: else if extrapolation then
                                        ▷ In case of open boundary scenarios (Sec 4.3)
6:     Use the Hessian-based stencil (Eq 6.18 in 2D) to fill in $B_x$ and $B_y$ in Eq (6.19).
7: end if
8: Use the discrete divergence operator (Eq 3.9) to fill in $D_x$, $D_y$, and $D_z$.
9: Eliminate the one-dimensional pressure nullspace (Sec 6.3.4).
10: Apply Schur complement transformation to the saddle point system in (6.20) to obtain
    the SPD system in (6.23) (Sec 6.4).
11: Solve (6.23) for $p$ using the CG algorithm (Alg 1).
12: Recover $\mathbf{u}$ from (6.21) using the computed values of $p$ (Sec 6.4).
13: Output: A combined velocity field over the whole domain, i.e., input velocities (drawn
    from the input simulation(s)) $\cup$ interpolated/extrapolated velocities $\mathbf{u}$.
```

Line 12 and **Algorithm** 1 require a forward/backward substitution with A (Sec 6.4). In our implementation, we use Intel Math Kernel Library (MKL)'s PARDISOLU solver through the Eigen library [Guennebaud et al., 2010] to factorize A and then perform the necessary forward/backward substitutions. Note: factorization of A is performed only once in the beginning and reused in the forward/backward substitutions.

Chapter 7

Results

The static illustrations in Chapter 5 demonstrated some of the technical aspects of our numerical method, in terms of its improvement over prior alternatives. We now consider a collection of illustrative dynamic scenarios to further demonstrate the behaviour of our method. Several of our figures make use of passive marker particles with alternating colors in initially horizontal rows to better highlight the developing flow structure. Other figures use the LIC visualization seen earlier to emphasize the structure of the vector field at a particular time instant. We primarily evaluate the capabilities of our method in two dimensions, but include one 3D example to demonstrate that the approach can extend naturally to higher dimensions.

7.1 Hole-Filling

Our first scenario (Figure 4.1) is a hole-filling scenario of a rotational vector field. Given a rotational vector field with a square region in the middle being erased, our goal is to fill in the missing region with a smooth incompressible vector field using the boundary data (drawn from the input simulation). Our constrained biharmonic interpolant seamlessly fills in the region lacking vector field data and the resulting simulation of the combined vector field (interpolated vector field plus the input vector field) gives the illusion of being a simulation of a single natural flow. The result is qualitatively indistinguishable from the input.

Another scenario (Figure 7.2) consists of filling "over" a hole with a solid obstacle in an input animation to produce a new animation without the obstacle. Let us consider a case

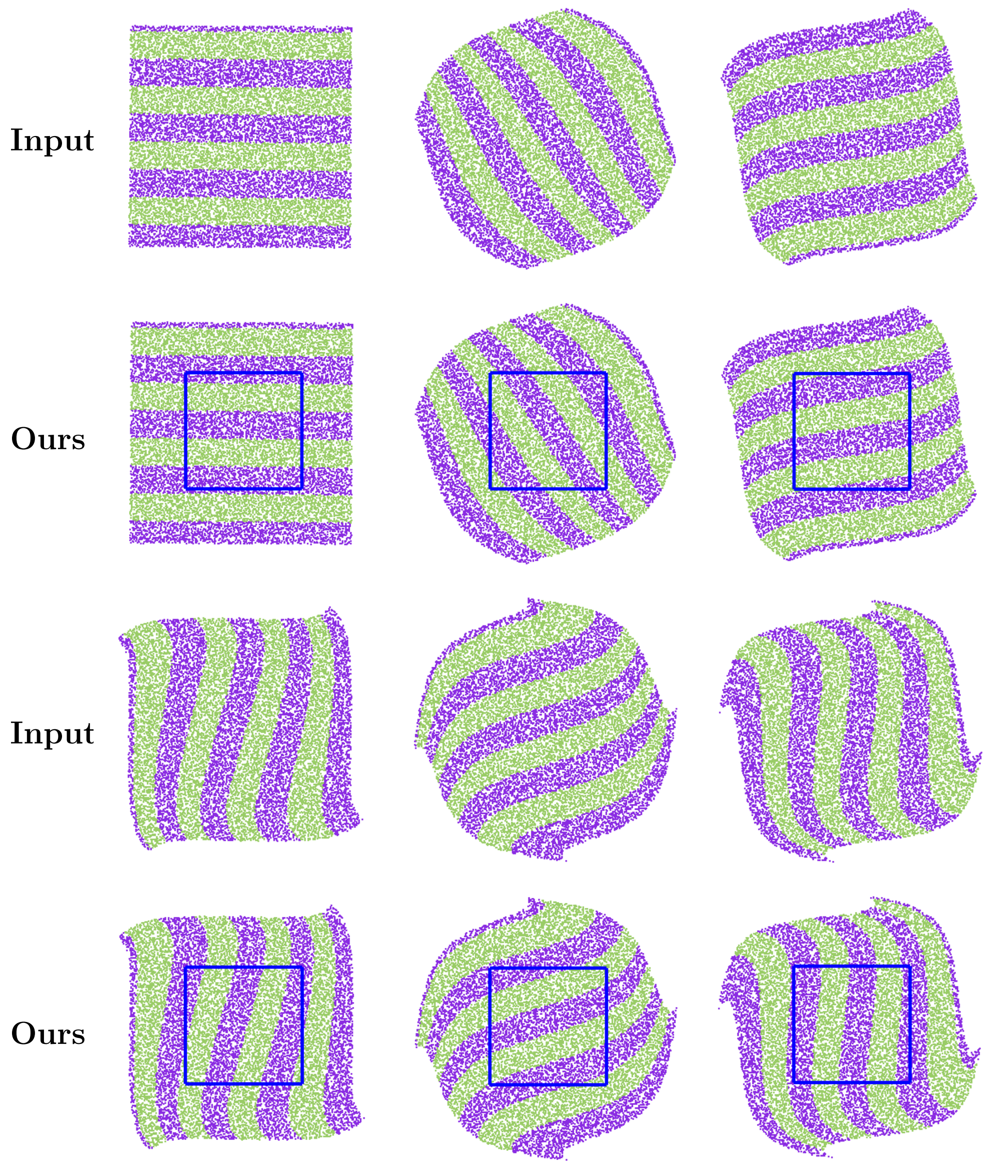


Figure 7.1: **Hole-filling Scenario:** Our simplest scenario involves performing a hole-filling task (with region to be filled surrounded in blue). A rotational vector field with a square region in the middle being erased (surrounded in blue) is used as input to our method. Our constrained biharmonic interpolant interpolates a smooth incompressible vector field inside the hole (surrounded in blue) using the boundary data (outside of the blue square). Six frames of the input animation are shown on the first and third rows. The same six frames of the edited animation result of our hole-filling approach are shown on the second and fourth rows.

where we already have a previously-computed simulation over a domain with an obstacle in the center. However, we now wish to remove the obstacle from the scene and obtain the same animation without the obstacle without re-simulating from scratch. In this case, we perform our divergence-constrained biharmonic interpolation method for a hole-filling scenario. Figure 7.2 shows that our method successfully removes the obstacle from the input animation and smoothly interpolates a vector field inside the hole. However, even though our approach produces a new plausible animation, it fails to recover a perfectly vertically-flowing animation. This happens because the presence of the obstacle *globally* disturbed the motion of the input animation such that after particles move past the obstacle, an empty region (with no inflow/outflow of particles) behind the obstacle occurs (Figure 7.2c, 7.2d). Since our method strictly enforces the divergence-free constraint, to achieve a divergence-free interpolation inside the hole, it necessarily gives up on generating a perfectly vertical flow.

7.2 Copy-Paste

Next, we consider a copy-paste scenario (Figure 7.3) consisting of a static solid disk in a vertical wind-tunnel with inflow at the bottom and outflow at the top. We wish to paste the disk and its surroundings from the source simulation into an even simpler empty vertically translating wind-tunnel target simulation. This yields a smooth divergence-free combination of the two flows, where the flow outside of the blend region is completely undisturbed. Our result necessarily differs from the source animation, since in the source animation the presence of the disk *globally* disturbed the flow; our biharmonic interpolation approach must therefore deform the flow more strongly in the blend region to compensate. Yet, we still achieve a visually plausible flow. Figure 7.4 demonstrates a few frames of the edited animation for the scenario described above to show how the combined flow is evolving.

Similarly, our next example (Figure 7.5) involves copying the flow around two disks from a source simulation into an obstacle-free target simulation. However, in this example, we use a more complex flow structure for the source and target simulations. The source simulation features a rising smoke plume due to buoyancy forces in a domain containing two disk obstacles. The target simulation is of a rising smoke plume in an obstacle-free domain. The resulting flow is a smooth divergence-free combination of the source and target flows. In Figure 7.5, the copied flow around two disks is pasted on the same location on the target animation as the source animation. Then we compare our combined result with the simulated source animation. Doing so, we demonstrate that our method produces

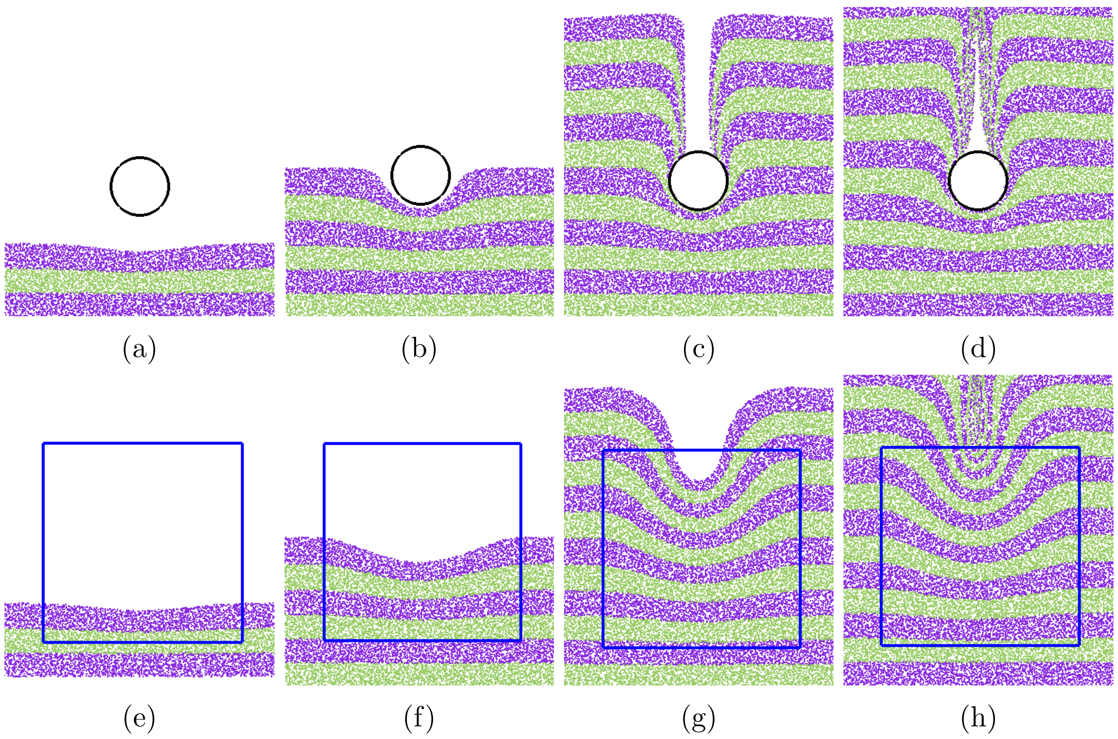


Figure 7.2: **Hole-Filling Scenario To Remove an Obstacle:** A vertical flow animation in a domain with a circular obstacle in the center is used as input to our method (a,b,c,d). To remove the obstacle, we apply our divergence-constrained biharmonic interpolant to fill over the region with the obstacle. Our method successfully removes the obstacle from the input animation (with region filled over surrounded in blue) (e,f,g,h). The same four frames of the input and edited animations are shown for comparison.

comparable results as the original source simulation for the same scenario (having the obstacle on the same location). In practical use cases, one can copy the desired flow from its source animation and paste it onto any desired location on the target animation to obtain a rising smoke simulation with two disk obstacles in any desired location.

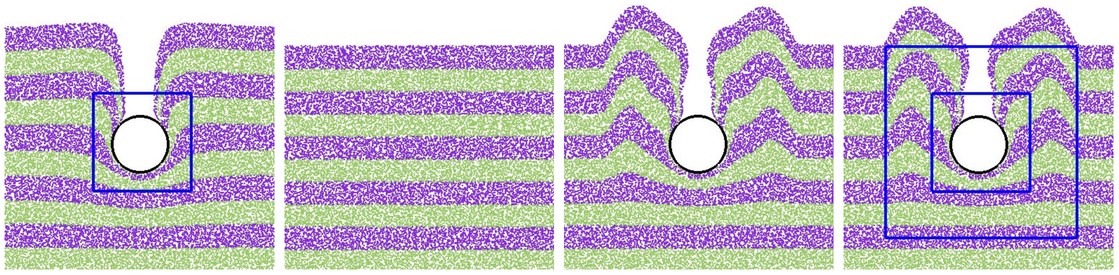


Figure 7.3: **Basic Setup for Fluid Copy-Paste:** Our simplest copy-paste scenario involves copying the flow around a disk from its source simulation (a), (with region to be copied surrounded in blue) into an obstacle-free target simulation (b). The result of our method is a new smoothly combined flow (c). Our combined flow with the blue lines that denote the inner and outer borders of the blend region over which we apply our biharmonic interpolation is shown in (d). (The same frame of animation is shown in all four images.)

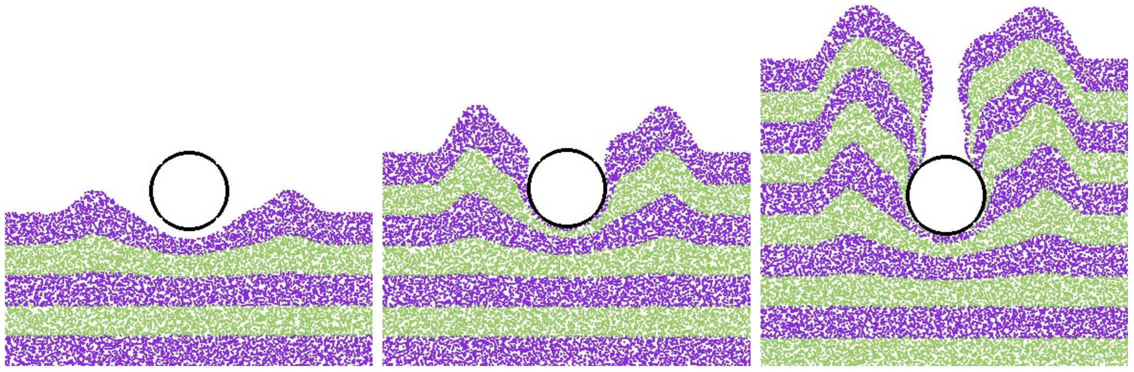


Figure 7.4: **Merged Flow Over Time:** Three frames of the edited animation result of our approach based on the scenario described in Figure 7.3.

Our 3D copy-paste scenario (Figure 7.6) demonstrates an example of how our method can be used in real applications. Let us consider a case where we have a previously-computed rising smoke simulation in an obstacle-free environment. However, we now wish to place a solid spherical obstacle in any desired location in the domain and obtain approximately the same rising smoke simulation in the presence of the added obstacle without re-simulating. In this case, our method can be used to paste the obstacle and its

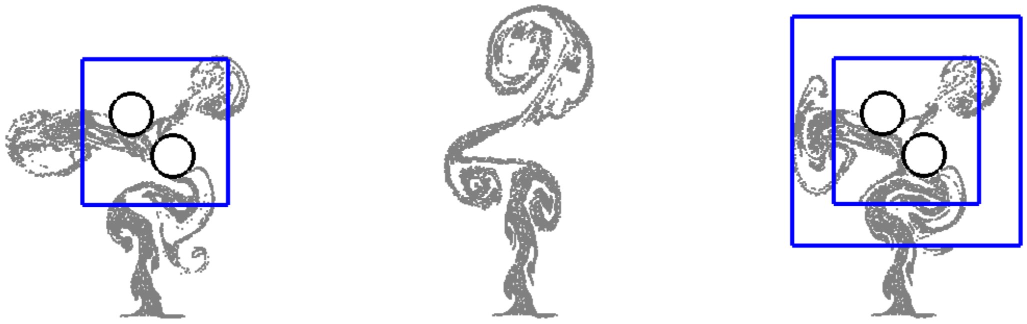

(a) Source (b) Target (c) Result

Figure 7.5: **2D Copy-Paste Scenario:** The flow around two disks from its rising smoke source simulation (a) is copied (with region copied surrounded in blue) into a 2D obstacle-free rising smoke target simulation (b). The result of our method is a new smoothly combined flow (c). The blue lines in (c) denote the inner and outer borders of the blend region over which we apply our biharmonic interpolation. (The same frame of animation is shown in all three images.)

surroundings from a previously-computed source simulation into the desired location in the target simulation. The resulting flow is a smooth divergence-free combination of the source and target flows.

To further stress-test our method, we consider some challenging scenarios analogous to those suggested by Sato et al. [2018]. We combine flows in which the source and target differ in direction or speed. The method of Sato et al. [2018] leads to severe failure of the divergence-free condition for larger speed and angle deviations between the source and target. We refer the reader to the supplemental video of their paper to see the visualizations of this issue. On the other hand, our method produces a divergence-free output field regardless of the size of the speed and angle deviations. Let us consider the same scenario performed in Figure 5.3, where a source simulation is pasted on a vertically-flowing target simulation, except that in this case, we change the ambient flow direction of the source simulation to have steadily increasing angles between source and target flows (Figure 7.7). While this leads to an increasingly unnatural look, the resulting flow field is still smooth in the blend region interior and along the boundaries (between blend region and source/target regions) and divergence-free independent of this artistic decision.

Similarly, we perform a test (Figure 7.8) in which the speed of the source (inner) simulation is slower or faster than the target (outer) simulation. Once again more severe

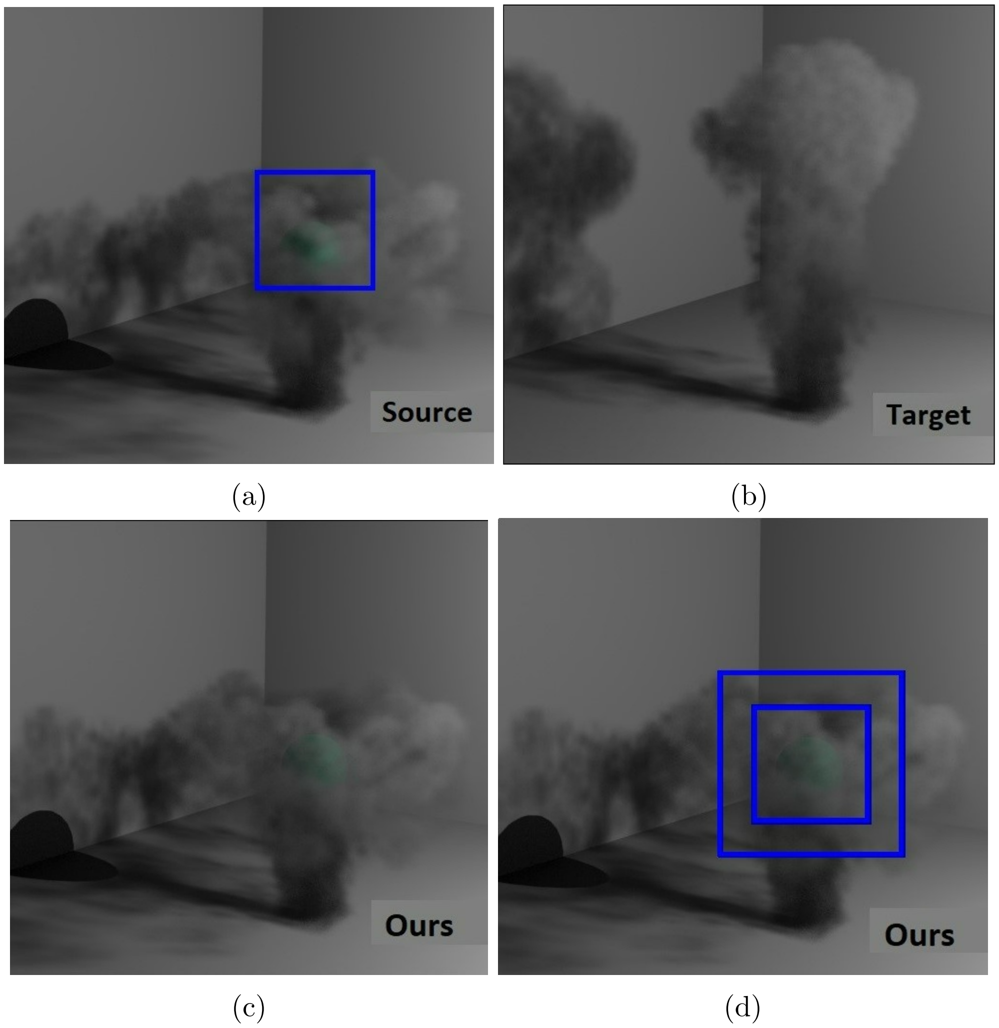
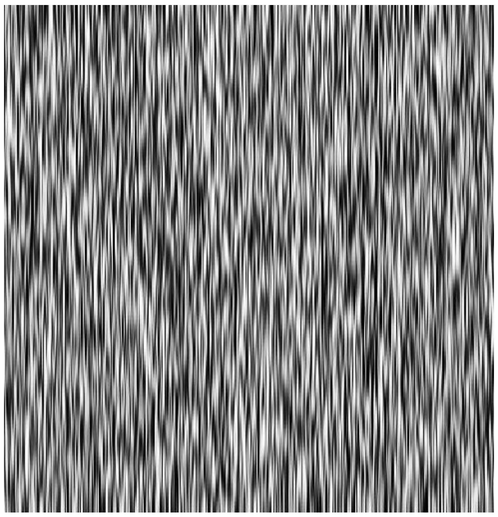
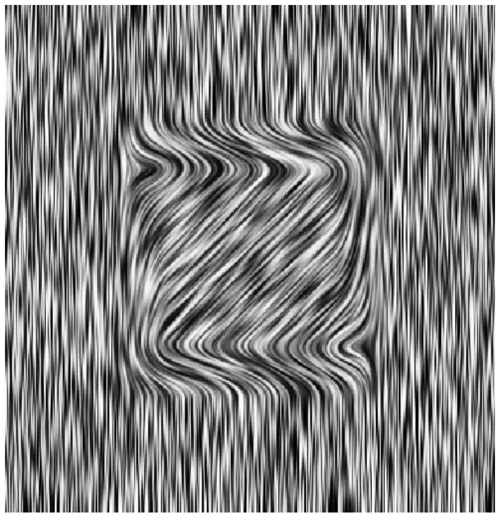
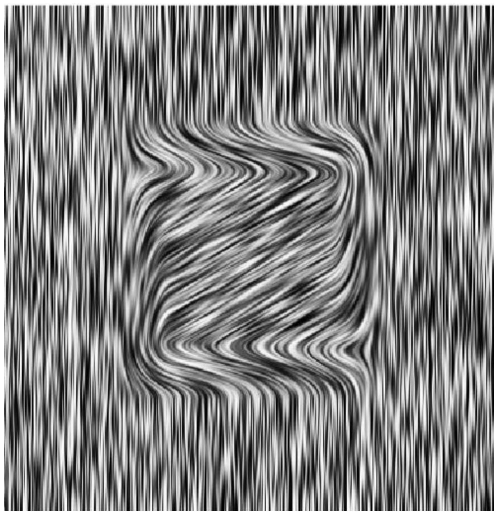


Figure 7.6: **3D Copy-Paste Scenario:** Our 3D copy-paste scenario involves copying the flow around a spherical obstacle from its source simulation (a), (with region to be copied surrounded in blue) into an obstacle-free rising smoke target simulation (b). The result of our method is a new smoothly combined flow (c). The blue lines in (d) denote the inner and outer borders of the blend region over which we apply our biharmonic interpolation. (The same frame of animation is shown in all four images.) Note: In this figure, the blue boxes show approximately where source and blend regions are (might not be on exact positions used while simulating). They are placed later on the rendered images to help the reader understand our method.

speed differences lead to more unusual motions in the blend region in order to compensate for the difference. For example, when the speed ratio between the source and target is 3.0 (Figure 7.8d), a more elaborate interior circulation of the flow in the blend region becomes necessary to satisfy the incompressibility condition. However, because the source and target are divergence-free and therefore provide compatible boundary conditions, the result is still divergence-free.


(a)


(b)


(c)

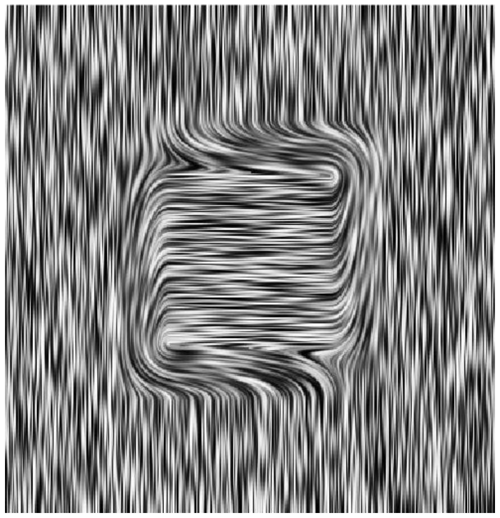

(d)

Figure 7.7: **Varying Angles Test:** Experimental results combining two velocity fields with different angles. (a) A vertical target velocity field. (b) The target field combined with a a 45-degree angled source flow. (c) The target field combined with a 60-degree angled source flow. (d) The target field combined with a 90 degree-angled source flow. Despite the extreme difference in inputs, we always obtain a divergence-free velocity field.

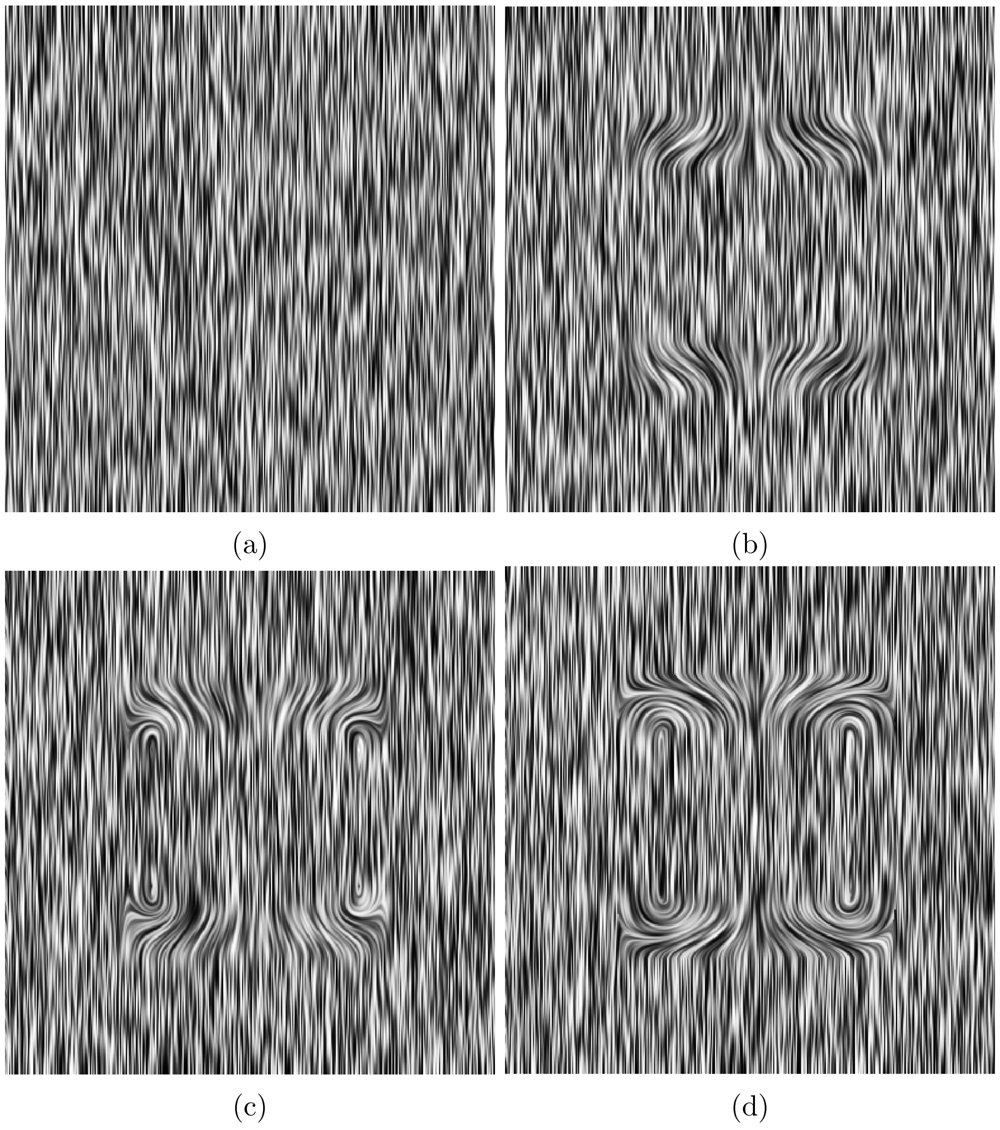

(a) (b) (c) (d)

Figure 7.8: **Varying Speeds Test:** In this test, we combine two vertical velocity fields with different speeds. The outer velocity field drawn from the target simulation has a fixed speed while the pasted inner velocity field from the source simulation has a speed ratio of: 1.0 (a), 0.5 (b), 1.5 (c), and 3.0 (d). In all cases, the resulting velocity field is divergence-free.

7.3 Open Boundary Scenarios

Lastly, we consider other scenarios (described in Section 4.3) which make use of our divergence-constrained biharmonic extrapolation scheme. Our first example (Figure 7.9) is a vector-field extrapolation scenario. In this scenario, given a previously-computed simulation of a four-vortex vector field with closed boundaries, our goal is to replace closed boundaries with open or natural boundaries without re-simulating. In this case, we achieve the desired simulation by smoothly extending a selected interior region with our extrapolation scheme. Unlike previous scenarios, this scenario consists of an unprescribed exterior boundary. In this case, we use natural boundary conditions where the vector field is free to flow in or out through the exterior boundary as in Figure 7.9c.

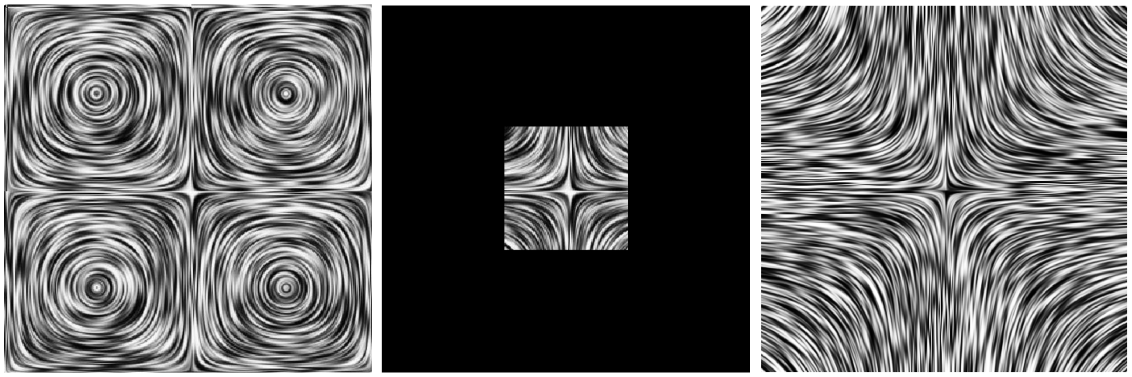

(a) Input Flow (b) Trimmed Input (c) Div-Constr Extrapolation

Figure 7.9: **Vector-Field Extrapolation Test:** Input flow (a) is a four-vortex flow generated using closed boundary conditions. (b) shows a square region in the middle of the same flow in (a) which we use as the input for our extrapolation task. Our divergence-constrained biharmonic extrapolator fills in the outer black region in (b) by smoothly extending the flow inside the square region in the middle in (b). Lastly, (c) shows the extrapolated flow result of our divergence-constrained biharmonic extrapolation scheme.

Our next example (Figure 7.10) is a scene stretching scenario of a simple vertically translating simulation. Given an input simulation of a vector field, our divergence-constrained biharmonic extrapolation synthesizes a simulation of the same vector field in an extended domain as follows. First, the input simulation is divided into two regions and a padding region is added in between. Then, our method fills in the padding region with a vector field by smoothly interpolating the boundary data on the top and bottom. Natural boundary conditions are applied on the unprescribed side boundaries of the padding region.

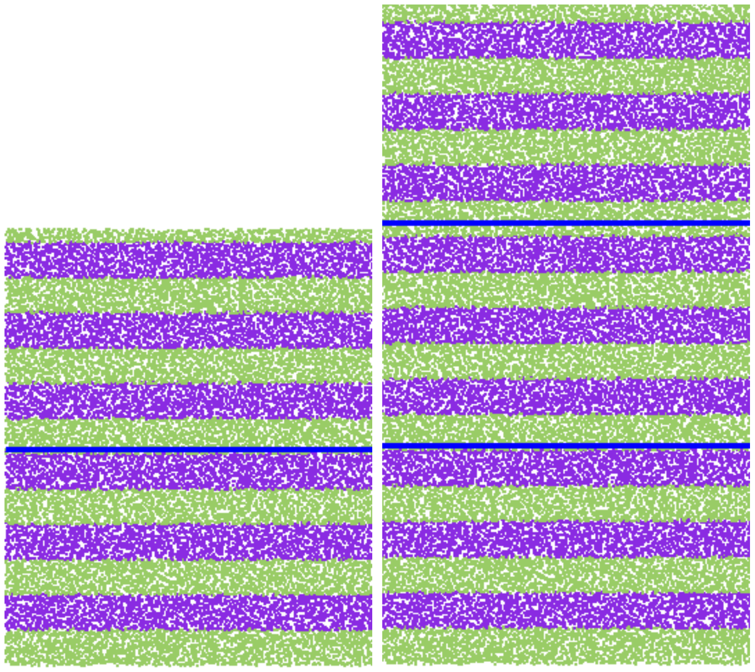
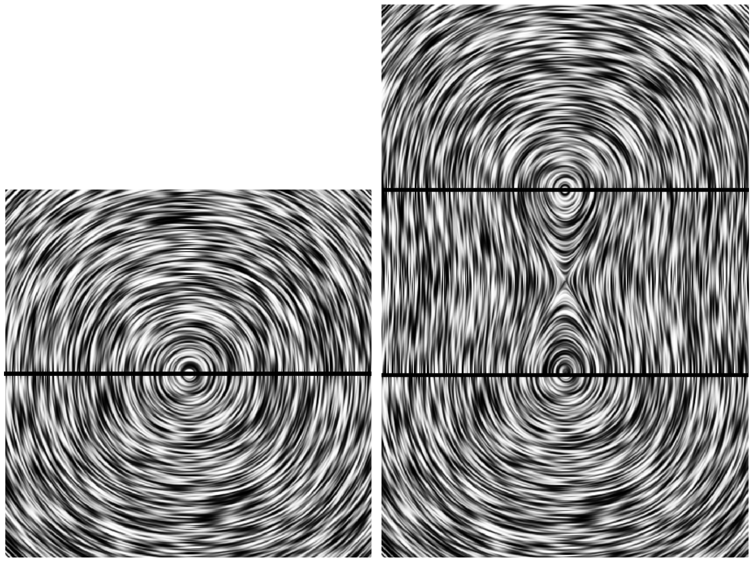

(a) Input Flow (b) Scene Stretching Output

Figure 7.10: **A Simple Scene Stretching Test:** The input (a) is a simple vertically flowing wind-tunnel simulation in a domain with size 1×1. The output (b) is a stretched version of the input simulation in a domain with size 1×1.5. We apply our divergence-constrained biharmonic extrapolation within the region in between the blue lines in (b).

We perform another scene stretching test (Figure 7.11) on a more complicated rotational input flow. In this case, our divergence-constrained biharmonic extrapolation scheme stretches the input vector field as in Figure 7.11b. Our method predictably produces inner circulations in the padding region because it matches both boundary values and gradients while extrapolating a vector field inside the padding region. Similarly, natural boundary conditions are applied along the unprescribed side boundaries of the padding region. Interestingly, one might initially anticipate that the extrapolated field would contain only vertical velocities, rather than produce the interesting rotational pattern that is observed. However, the result is indeed correct: it occurs because the extrapolant matches the nonzero gradients of the velocity field at the boundaries, so the rotations present in the input field are smoothly extended into the extrapolation region.

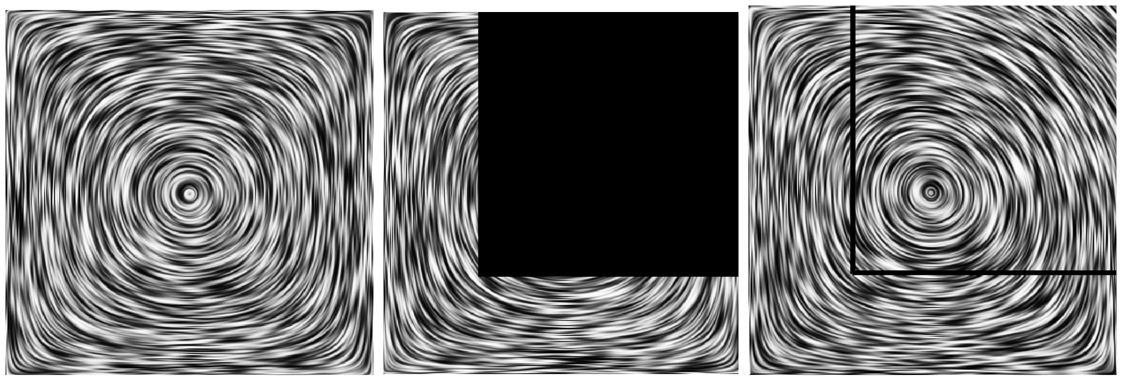
Our last example (Figure 7.12) is a vector field extrapolation scenario where the region to be filled overlaps part of the exterior boundary of the domain (as described in Section



(a) Input Flow (b) Scene Stretching Output

Figure 7.11: **Scene Stretching Test on a Rotational Vector Field:** The input (a) is a rotational flow simulation. The output (b) is a stretched version of the input simulation. We apply our divergence-constrained biharmonic extrapolation within the padding region in between the black lines in (b).

6.2). Given a simulation of a rotational vector field generated using closed boundaries, our divergence-constrained biharmonic extrapolant produces a smooth divergence-free extension of the input flow with open exterior boundaries. We observe that in the presence of open or natural boundaries as on the top-right corner of Figure 7.12c, the result flows freely in or out through the exterior boundary unlike the input simulation as expected (Figure 7.12a).



(a) Input Flow (b) Input (c) Div-Constr Extrapolation

Figure 7.12: **Specific Extrapolation Test:** Input flow (a) is a rotational flow generated using closed boundary conditions. A black square region on the top-right corner of the same flow in (a) is used as the extrapolation domain (b). Our divergence-constrained biharmonic extrapolator fills in the black region in (b) by smoothly extending the flow outside the black region in (b). Lastly, (c) shows the extrapolated flow result of our divergence-constrained biharmonic extrapolation scheme.

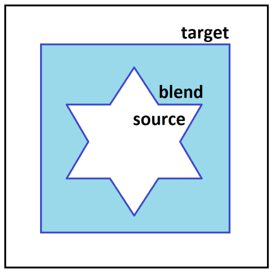
Chapter 8

Conclusion and Future Work

In this thesis, we have presented a new vector field interpolation and extrapolation scheme for simulation editing based on a vector biharmonic problem with an added incompressibility constraint. In chapter 7, we demonstrated the success of the proposed methods in achieving desired behavior in a wide range of scenarios. Our work suggests several directions to explore in future work.

First, we assumed that the interpolation region borders are voxelized, and saw no artifacts from doing so. However, it would be interesting to explore a cut-cell approach [Batty et al., 2007; Larionov et al., 2017] for regions with irregular shapes, based on a more precise discretization of the relevant energies. This would add greater flexibility and potentially make the region borders even less apparent.

Moreover, by introducing a cut-cell approach allowing interpolation region borders to be nonvoxelized, one might simply use our method to add an arbitrarily shaped obstacle in an obstacle-free target simulation by initializing the source region with a zero velocity field. Here the source region with zero velocity acts as an obstacle. For example, to put a star-shaped obstacle in an obstacle-free target simulation, one might initialize a star-shaped source region with a zero velocity field and let our divergence-constrained biharmonic interpolant fill in the blend region between the source and target regions (see inset).



Next, the hole-filling scenario when used to remove an obstacle from an input simulation (as shown in Figure 7.2) cannot perfectly recover the input simulation without the obstacle. In the general case, this is likely to be impossible. Since the presence of the obstacle changes net in/out flux along the blend region borders, to achieve a divergence-free interpolation inside the hole, our method gives up on exactly recovering the input flow without the obstacle. Even though our method successfully removes the obstacle and fills over the region with a plausible smooth divergence-free vector field, as future work, one might want to implement the divergence condition as a penalty-like energy term to control how much divergence is allowed instead of strictly enforcing the divergence-free constraint. In that case, the user would have a parameter controlling the balance between the smoothness energy and incompressibility, thereby providing greater flexibility in the resulting behavior.

Lastly, the bilaplacian operator does not offer the type of maximum principle exhibited by the Laplacian, and thus it is possible for the interpolated velocities to overshoot the boundary data, i.e., the combined or interpolated field may contain faster velocities than the input velocity field(s) alone. However, since we used the resulting velocity only for advecting passive data, rather than advecting velocity, such overshoots cannot feed back into an underlying simulation and cause instabilities. If all overshoots were deemed undesirable, enforcing bounds using inequality constraints is an option [Jacobson et al., 2011]; however, doing so may also harm smoothness of the boundaries.

References

Batty, C., Bertails, F., and Bridson, R. (2007). A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (TOG)*, 26(3):100–es.

Benzi, M., Golub, G. H., and Liesen, J. (2005). Numerical solution of saddle point problems. *Acta numerica*, 14:1–137.

Bhattacharya, H., Nielsen, M. B., and Bridson, R. (2012). Steady state stokes flow interpolation for fluid control. In *Eurographics (Short Papers)*, pages 57–60. Citeseer.

Bojsen-Hansen, M. and Wojtan, C. (2016). Generalized non-reflecting boundaries for fluid re-simulation. *ACM Transactions on Graphics (TOG)*, 35(4):1–7.

Brackbill, J. U., Kothe, D. B., and Ruppel, H. M. (1988). Flip: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38.

Bridson, R. (2015). *Fluid simulation for computer graphics.* CRC Press, Boca Raton, second edition. edition.

Cabral, B. and Leedom, L. C. (1993). Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, page 263–270, New York, NY, USA. Association for Computing Machinery.

Evans, Lawrence C. (2010). Partial Differential Equations: Second Edition.

Fattal, R. and Lischinski, D. (2004). Target-driven smoke animation. *ACM Trans. Graph. (SIGGRAPH)*, 23(3):441–448.

Flynn, S., Egbert, P., Holladay, S., and Morse, B. (2019). Fluid carving: intelligent resizing for fluid simulation data. *ACM Transactions on Graphics (TOG)*, 38(6):1–14.

Forootaninia, Z. and Narain, R. (2020). Frequency-domain smoke guiding. *ACM Transactions on Graphics (TOG)*, 39(6):1–10.

Foster, N. and Metaxas, D. (1997). Controlling fluid animation. In *Computer Graphics International*, page 178.

Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

Harlow, F. H. and Welch, J. E. (1965). Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189.

Hong, J. M. and Kim, C. H. (2004). Controlling fluid animation with geometric potential. *Computer Animation and Virtual Worlds*, 15(3-4):147–157.

Inglis, T., Eckert, M.-L., Gregson, J., and Thuerey, N. (2017). Primal-dual optimization for fluids. In *Computer Graphics Forum*, volume 36, pages 354–368. Wiley Online Library.

Jacobson, A., Baran, I., Popovic, J., and Sorkine, O. (2011). Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78.

Joshi, P., Meyer, M., DeRose, T., Green, B., and Sanocki, T. (2007). Harmonic coordinates for character articulation. *ACM Transactions on Graphics (TOG)*, 26(3):71–es.

Larionov, E., Batty, C., and Bridson, R. (2017). Variational Stokes: a unified pressure-viscosity solver for accurate viscous liquids. *ACM Transactions on Graphics (TOG)*, 36(4):1–11.

Manteaux, P.-L., Vimont, U., Wojtan, C., Rohmer, D., and Cani, M.-P. (2016). Space-time sculpting of liquid animation. In *Proceedings of the 9th International Conference on Motion in Games*, pages 61–71.

McNamara, A., Treuille, A., Popović, Z., and Stam, J. (2004). Fluid control using the adjoint method. *ACM Transactions on Graphics*, 23(3):449.

Nielsen, M. B. and Bridson, R. (2011). Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph. (SIGGRAPH)*, 30(4):1.

Nielsen, M. B. and Christensen, B. B. (2010). Improved variational guiding of smoke animations. In *Computer Graphics Forum*, volume 29, pages 705–712. Wiley Online Library.

Nielsen, M. B., Christensen, B. B., Zafar, N. B., Roble, D., and Museth, K. (2009). Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 217–226.

Pan, Z., Huang, J., Tong, Y., Zheng, C., and Bao, H. (2013). Interactive localized liquid motion editing. *ACM Transactions on Graphics (TOG)*, 32(6):1–10.

Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. *ACM Trans. Graph. (SIGGRAPH)*, 22(3):313–318.

Rasmussen, N., Enright, D., Nguyen, D., Marino, S., Sumner, N., Geiger, W., Hoon, S., and Fedkiw, R. (2004). Directable photorealistic liquids. In *Symposium on Computer Animation*, pages 193–202.

Raveendran, K., Thuerey, N., Wojtan, C., and Turk, G. (2012). Controlling liquids using meshes. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*, pages 255–264.

Raveendran, K., Wojtan, C., Thuerey, N., and Turk, G. (2014). Blending liquids. *ACM Transactions on Graphics*, 33(4):1–10.

Sato, S., Dobashi, Y., and Kim, T. (2021). Stream-guided smoke simulations. *ACM Transactions on Graphics (TOG)*, 40(4):1–7.

Sato, S., Dobashi, Y., and Nishita, T. (2018). Editing fluid animation using flow interpolation. *ACM Trans. Graph.*, 37(5):1–12.

Shi, L. and Yu, Y. (2005). Taming liquids for rapidly changing targets. In *Symposium on Computer Animation*, pages 229–236.

Söderström, A., Karlsson, M., and Museth, K. (2010). A pml-based nonreflective boundary for free surface fluid animation. *ACM Transactions on Graphics (TOG)*, 29(5):1–17.

Stein, O., Grinspun, E., Wardetzky, M., and Jacobson, A. (2018). Natural boundary conditions for smoothing in geometry processing. *ACM Transactions on Graphics (TOG)*, 37(2):1–13.

Stein, O., Wardetzky, M., Jacobson, A., and Grinspun, E. (2020). A simple discretization of the vector dirichlet energy. In *Computer Graphics Forum*, volume 39, pages 81–92. Wiley Online Library.

Stomakhin, A. and Selle, A. (2017). Fluxed animated boundary method. *ACM Transactions on Graphics (TOG)*, 36(4):1–8.

Thuerey, N. (2016). Interpolations of smoke and liquid simulations. *ACM Trans. Graph.*, 36(1):1–16.

Thuerey, N., Keiser, R., Pauly, M., and Rüde, U. (2006). Detail-preserving fluid control. In *Symposium on Computer Animation*, pages 7–12.

Treuille, A., McNamara, A., Popović, Z., Stam, J., Treuille, A., McNamara, A., Popović, Z., and Stam, J. (2003). Keyframe control of smoke simulations. *ACM Transactions on Graphics*, 22(3):716.

Wiebe, M. and Houston, B. (2004). The Tar Monster: Creating a character with fluid simulation. In *SIGGRAPH Sketches.*