Novel Paradigms in Physics-Based Animation: Pointwise Divergence-Free Fluid Advection and Mixed-Dimensional Elastic Object Simulation

by

Jumyung Chang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Tamar Shinar Associate Professor, Dept. of Computer Science and Engineering, University of California, Riverside
Supervisor(s):	Christopher Batty Associate Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Stephen Mann Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Justin Wan Professor, Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Sander Rhebergen Associate Professor, Dept. of Applied Mathematics, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis consists of research material written for publication where Jumyung Chang was the lead author under the supervision of Dr. Christopher Batty and in collaboration with Dr. Eitan Grinspun, Dr. Fang Da, and Dr. Vinicius Azevedo.

Research presented in Chapter 5–6, and 10–13 largely consists of contribution and results sections adapted from (Chang et al., 2019) and (Chang et al., 2021), respectively. Chapter 1–4, 7–9, and 14 consist of a combination of sole-authored content by Jumyung Chang and paraphrased material from (Chang et al., 2021, 2019), adapted to the structure of a PhD thesis.

For the first part of the thesis (Chapter 2–6), Dr. Fang Da, Dr. Christopher Batty, and Dr. Eitan Grinspun proposed the idea of frame-based energies using a conformal mesh, and I devised and completed the connection energies under the supervision of Dr. Eitan Grinspun and Dr. Christopher Batty. For the second part of the thesis (Chapter 7–13), Dr. Vinicius Azevedo and Dr. Christopher Batty proposed the idea of pointwise divergence-free interpolation in uniform grids in 2D. I extended the coverage to include cut-cells, 3D, and exact boundary enforcement in collaboration with Dr. Vinicius Azevedo under the supervision of Dr. Christopher Batty.

Abstract

This thesis explores important but so far less studied aspects of physics-based animation: a simulation method for *mixed-dimensional* and/or *non-manifold* elastic objects, and a *pointwise divergence-free* velocity interpolation method applied to fluid simulation. Considering the popularity of single-type models e.g., hair, cloths, soft bodies, etc., in deformable body simulations, more complicated coupled models have gained less attention in graphics research, despite their relative ubiquity in daily life. This thesis presents a unified method to simulate such models: elastic bodies consisting of mixed-dimensional components represented with potentially non-manifold simplicial meshes. Building on well-known simplicial rod, shell, and solid models, this thesis categorizes and defines a comprehensive palette expressing all possible constraints and elastic energies for stiff and flexible connections between the 1D, 2D, and 3D components of a single conforming simplicial mesh. For fluid animation, this thesis proposes a novel methodology to enhance grid-based fluid animation with pointwise divergence-free velocity interpolation. Unlike previous methods which interpolate discrete velocity values directly for advection, this thesis proposes using intermediate steps involving vector potentials: first build a discrete vector potential field, interpolate these values to form a pointwise potential, and apply the continuous curl to recover a pointwise divergence-free flow field. Particles under these pointwise divergence-free flows exhibit significantly better particle distributions than divergent flows over time. To accelerate the use of vector potentials, this thesis proposes an efficient method that provides boundary-satisfying and smooth discrete potential fields on uniform and cut-cell grids. This thesis also introduces an improved ramping strategy for the "Curl-Noise" method of Bridson et al. (2007), which enforces exact no-normal-flow on the exterior domain boundaries and solid surfaces. The ramping method in the thesis effectively reduces the incidence of particles colliding with obstacles or creating erroneous gaps around the obstacles, while significantly alleviating the artifacts the original ramping strategy produces.

Acknowledgements

First of all, I would like to thank my supervisor, Christopher Batty. I sincerely appreciate all your help: your engagement and interest in my research helped me stay motivated throughout my studies. Your brilliant ideas and academic advice kept my research going whenever I floundered in a sea of unresolved issues. You have been a very friendly, approachable, and trustworthy mentor; I respect your opinions not just in the field of academia, and I remember knocking on your office door numerous times before I made big decisions.

I also would like to thank Eitan Grinspun and Vinicius Azevedo. Your insightful advice has been always inspiring, and reminded me how deep and wide the field is.

I would like to thank my committee members, Stephen Mann, Justin Wan, Sander Rhebergen, and Tamar Shinar for taking the time to read my thesis and providing insightful feedback.

I would like to thank my colleagues at the Computational Motion Group: Ryan Goldade, Yu Gu, Michael Honke, Nathan King, Jade Marcoux-Ouellet, Tümay Özdemir, Jonathan Panuelos, Yipeng Wang, and Omar Zarifi (alphabetical order). I miss the old days when we would hang out in the office, and have chats about anything including research ideas and life.

I will also miss hanging out with my comrades in the Scientific Computing lab on a daily basis. I still vividly remember, if any of us said "C&D", we formed a platoon and marched down to coffee places for a break.

I would like to thank the Computer Graphics lab. The presentations by great people in weekly meetings have exposed me to diverse and up-to-date research topics and details in the general field of computer graphics, which I would have missed otherwise.

I would also like thank the Simulation team at NVIDIA and Weta Digital. I learned very distinctive industrial skills from the teams, and met and worked with great people.

Finally, special regards to my family; I can never express how grateful I am to my family for their constant support.

Dedication

For my wife, Stacey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The importance of artistic skills and engineering support are well balanced in visual effects. Engineers and researchers provide increasingly advanced and diverse tools over time; artists utilize these tools to produce results closest to their goals, whether they desire photorealism or to achieve a certain aesthetic. However, not all features that artists would want are available, so laborious manual manipulation by artists is often required to obtain results due to unsupported features. Taking a physics-based animation as an example, simulating highly deformable materials with a large number of elements (e.g., vertices, triangles, etc.) would be challenging if not supported by a tool. Artists may not want to prescribe the motion of smoke via a myriad of particles or other complex representations, nor would they likely want to control each vertex of a garment mesh to simulate a wrinkled motion every frame. This preference is not only because the work is time-consuming, but also because it is often difficult to know a plausible motion in the absence of valid real-world reference footage of the specific scenario (which may not even be possible to capture). This highlights the importance of simulation research in visual effects. With the progress in simulation research and tool development over roughly the past two decades, complicated smoke or cloth motions are now made possible via computational tools incorporated into standard visual effects software such as Houdini, Maya, Blender, etc. Nevertheless, there are still many areas of unsupported features and limitations of existing techniques, which complicate or impede their use for certain tasks. This thesis therefore addresses two such challenging simulation problems, where manual manipulation is almost infeasible: one in the context of deformable elastic object animation and one in the context of fluid animation. By developing new, more powerful numerical simulation capabilities for visual effects, this thesis contributes to the creation of animation software that empowers artists to more easily achieve their artistic visions.

In Part I of the thesis, elastic objects with mixed-dimensional components and/or non-manifold connection types are presented. Elastic objects have been popular subjects in graphics, and we often categorize them based on their dimensionality (i.e., 1D, 2D, 3D) as objects with different dimensionality are subject to different specialized types of forces or energies. For example, 1D models are subject to stretching, twisting, and bending, 2D models can have planar deformation and bending, and 3D models only exhibit volumetric deformation. Hair strands, cloths or clothes, and rubber or flesh would be representative examples in each dimension, respectively, which we often see in movies or games. The importance of each single-type model is obvious, but each can only represent a subset of elastic materials: purely 1D, 2D, or 3D materials. Many real-world objects are comprised of an interconnected blend of different types, and cannot naturally or efficiently be expressed with just one type alone. Leveraging the single-type models, this thesis aims to provide the missing connection strategies between different single-type models. Using similar ideas, this thesis further considers non-manifold connections such as two triangles connected by a joint vertex. The resulting computational framework offers a flexible and unified approach to physics-based simulation of 3D elastic objects of any shape, topology, and dimensional type.

Part II of the thesis addresses incompressible fluid flows. In fluid animation, incompressible flows are typically assumed; since most fluids we see in everyday life undergo a negligible amount of compression, and the computational cost and complexity to simulate compressible flows are viewed as significant drawbacks in light of often negligible visual improvements the compression provides. Incompressibility is also directly related to volume conservation of the fluid, as it implies the volume should be strictly unchanging. However, most existing fluid animation techniques enforce incompressibility only at a coarse scale dictated by the resolution of the underlying grid used to discretize the fluid equations. This thesis instead aims to enforce incompressibility at every scale, through a novel pointwise divergence-free interpolation procedure. The drawbacks of divergent interpolation can be demonstrated by inspecting the behaviors of passive particles moving with an interpolated velocity field: i.e., seeding particles uniformly in the domain, moving them with the pointwise velocity obtained by the divergent interpolation, and checking the particle distribution over time. Divergent interpolation will lead to divergent velocity fields, and consequently, the divergent fields would generate or accelerate particle clustering or vacancy. In the presence of solid obstacles, the drawbacks of previous interpolation approaches often stand out even more conspicuously. These drawbacks result from treating particle collisions against solids through simple post-processing: e.g., particles are pushed back to the surface of the solid upon collision. This simple strategy of "putting out fires" removes the most immediate visual artifacts of fluid particles piercing solid objects, but

cannot recover a natural flow near the solid (e.g., flows wrapping tightly around a solid) and the artificial position correction can cause additional undesirable particle clumping. This thesis aims to ensure that the resulting incompressible, interpolated fluid velocity fields are also boundary-respecting, thereby dramatically reducing the number of spurious collisions between flowing particles and solid obstacles.

Part I

A Unified Simplicial Model for Mixed-Dimensional and Non-Manifold Deformable Elastic Objects

Chapter 2

Introduction

Most numerical methods for elastic deformable bodies in computer graphics have focused on objects that exhibit a single uniform type or effective dimensionality: 1D rods, 2D shells, or 3D volumetric solids. Here, the term shell refers to a flexible thin structure where the ratio of width to thickness is large. Simulating such objects as 3D solids is an inefficient option due to their structural degeneracy in one dimension, thus a specialized model is typically used for such thin structures. Also, shells in the simulation context are not limited to hard surfaces but they contain general thin and elastic materials, thus often referred as elastic shells. A popular example in graphics that uses shell models for simulation would be cloths or clothes. Likewise, elastic rods refer to deformable curves embedded in 3D space, which can stretch, twist, and bend. This term does not imply the stiffness of the material but rather the (effective) dimensionality. Hair strands and threads are often simulated using rod models.

Although such single-type models have consistently been popular topics in simulation, many common objects possess either *non-manifold* connections between parts of possibly differing dimensions, or *embedded* features where a lower dimensional object passes through a higher-dimensional one. Examples include wires or rods threaded through tents, kites, or umbrellas; molded rubber and plastic objects composed of smoothly connected components of varying shape and thickness; tendons embedded in flesh or skin wrapped around muscle; and sandwich-structured composites, in which volumes of a soft low-density material are sandwiched between sheets of a stiffer material like aluminum.

Our aim is to characterize and develop a comprehensive palette of mixed-dimensional and/or non-manifold connection types for such elastic objects represented by a single conforming simplicial mesh. We use the term *simplicial mesh* to mean a mesh comprised

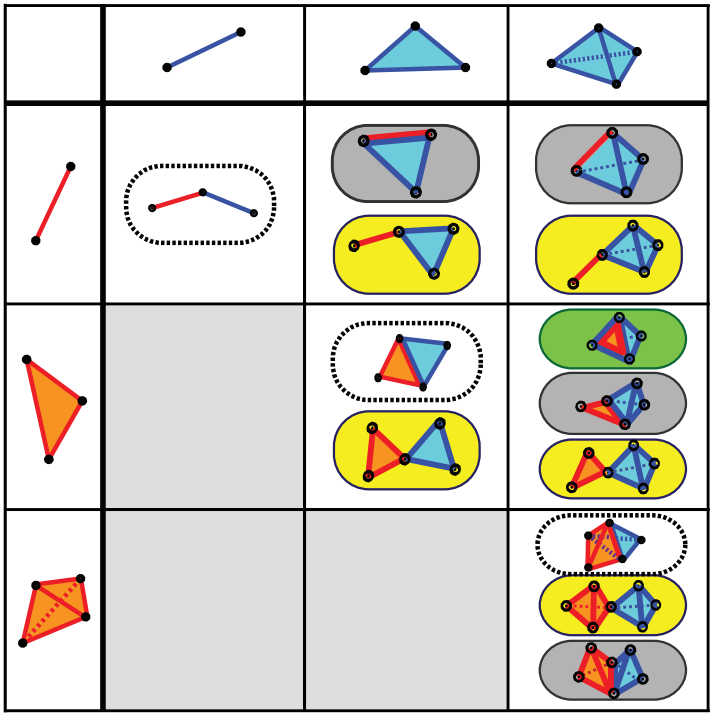


Figure 2.1: **The set of possible connections among elastic models in a general 3D simplicial mesh.** Dashed: single-type. Yellow: shared vertex. Grey: shared edge. Green: shared face. (The connections in the grayed-out cells have been left blank as they are identical to their symmetric counterparts in the top-right.)

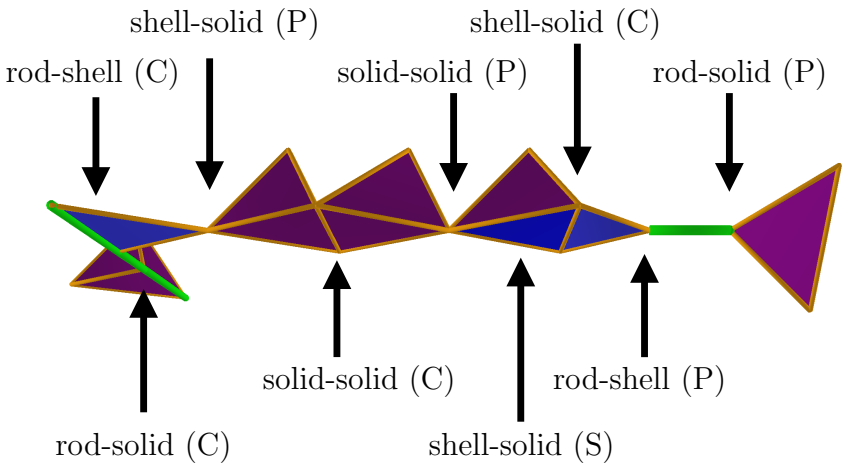


Figure 2.2: **A simplicial object decorated with many connection types.** Purple indicates solid tetrahedra, blue indicates shell triangles, and green indicates rod edges. (P), (C), and (S) mean Point, Curve, and Surface connection, respectively.

of 1-,2-, and 3-simplices, which are line segments, triangles, and tetrahedra, respectively. Also, we interpret *conforming* to mean that two components are connected exactly along shared simplices (i.e., shared vertices, edges, and/or faces). Figure 2.1 catalogs all the basic pairwise manifold and non-manifold connections that can arise on such a mesh; these can also be "stacked" to handle higher valence connections.

To express the deformation of components of a particular dimensionality, we leverage existing models. Specifically, we adopt Bergou's (time-parallel) discrete elastic rods (Bergou et al., 2010), Grinspun's discrete shells with hinge-based bending (Grinspun et al., 2003) and linear elastic stretching (Gingold et al., 2004), and St. Venant-Kirchhoff tetrahedral elastic solids (O'Brien and Hodgins, 1999). Other choices are possible, but these are well-studied, widely adopted, and provide a balance of accuracy, simplicity, and efficiency.

While the space of possible connections is large (Figure 2.1) we distinguish three families: *point (vertex) connections*, *curve (edge) connections*, including rods embedded into shells and solids, and *surface (face) connections*, of which a shell embedded in a solid is the only example. This categorization reduces the conceptual and implementation complexity of methodically developing a unified model. For each case, we describe both hard constraints and soft elastic joints by constructing appropriate reference coordinate frames and measuring deformations in the available modes. Our implementation and evaluations focus on the elastic case for its greater expressivity.

Our primary contribution is the design of a unified framework for arbitrary simplicial

deformable objects. The design of appropriate forces hinges on the ability to perform accurate comparisons between neighboring simplices' orientations, and two enabling technical contributions are particularly critical. First, we argue that joints in elastic rods provide a natural analog for non-manifold point connections. We therefore expand the application of parallel transport from rods to the task of measuring relative deviation in general simplices' reference frames. Second, we observe that large accumulated rotations can lead to sudden spurious jumps in potential energy at point or curve connections when relative angle deviations wrap around in the space of angles. We propose a new incremental angle-update strategy that resolves this issue.

By augmenting standard single-dimensional models with these new capabilities and making judicious design choices, we assemble a comprehensive system for animating simplicial deformable objects. An end-user only needs to model an object's non-manifold and/or mixed-dimensional geometric mesh and specify its behavior by decorating simplices with model (rod, shell, solid) and connection types. This provides a simple, flexible, and powerful paradigm for modeling diverse deformable objects, which we demonstrate in a variety of application scenarios.

Chapter 3

Related Work

Our elastic coupling strategy leverages existing single-type solvers and provides a complete set of connections between single-type models. In this chapter, we review popular approaches for each single-type model in graphics, followed by previous coupling strategies that mainly focus on specific types of connections.

3.1 (Manifold) Single-Type Elastic Models

3.1.1 Elastic Rods

Stretching of rods is either disallowed (inextensible) or treated with a spring-like approach. However, handling twisting and bending of rods requires a representation of the orientation of the rod along its length, in addition to a representation of spatial position. It is typical to assign a local (explicit or implicit) coordinate frame to each point on the rod's curve called the material frame, and track how it changes. We briefly review popular rod models in graphics with their coordinate frames.

Cosserat Rods

Since Pai (2002) introduced Cosserat theory of elastic rods, which is a special case of more general Cosserat continua, Cosserat rods have been heavily used in graphics to simulate rods or hair strands. Bertails et al. (Bertails, 2009; Bertails et al., 2006) extended the model to simulate curly hair by using piecewise helical rods or *Super-Helices*. These methods rely

on implicit representations of the centerline which needs to be recovered every frame by integrating the rod's curvature from one end. Instead, Spillmann and Teschner (2007) used *explicit* representations of the centerline. Their tangential axis of the rod's coordinate frame is not aligned with the rod's centerline; thus, they impose additional constraints to match the tangential axis to the rod's centerline.

Bishop Frame

The Bishop frame is physically the most relaxed frame, in the sense of having minimal twist along the rod's centerline. Bergou et al. (2008) used a material frame that follows the centerline by using the Bishop frame as a reference frame and using the angular deviation from the Bishop frame to represent the material frame. Consequently, this approach resolves the issue of following centerlines by construction. The Bishop frame evolves in space along the rod using parallel transport. For better efficiency, this space-parallel reference frame was replaced by a time-parallel reference frame using a time-space analogy; the reference frame on each edge evolves in time (Bergou et al., 2010; Kaldor et al., 2010).

Others

Rods can be simulated in other various frameworks inheriting the advantages and disadvantages of the framework with their own energy formulations. For example, mass spring models (McAdams et al., 2009; Plante et al., 2002; Rosenblum et al., 1991; Selle et al., 2008b) and position-based dynamics (Angles et al., 2019; Umetani et al., 2014) are valid and popular alternatives. They are in general relatively simple and efficient, but they often have limitations in the fidelity of the simulation results.

3.1.2 Elastic Shells and Solids

The elastic behaviors of a deformable object both in 2D and 3D are often specified by a hyperelastic energy density which is a function of deformation gradient (Kim and Eberle, 2020; Sifakis and Barbic, 2012). In general, this approach reflects the behavior of real materials more accurately as the volumetric body can change its shape in more complex ways in comparison with simpler models such as mass-spring methods (Choi and Ko, 2002; Provot, 1995). Also, it overall provides improved mesh-independence and convergence to analytical solutions under refinement. Thus, we use FEM-based energy models, and especially choose St.Venant-Kirchhoff model for 3D solids (O'Brien and Hodgins, 1999),

and a linear elastic constitutive model for 2D cloths (Gingold et al., 2004). As our work primarily focuses on the coupling between models, different energy models can also be adapted. Other popular models in graphics are briefly introduced in the paragraphs below. Also, for 2D shells only, additional bending effects must be considered since the energies above account only for in-plane deformations like stretching and shearing. A simple and popular choice for bending is a hinge-based energy model using dihedral angles (Bridson et al., 2003; Grinspun et al., 2003), and we use the Discrete Shells bending energy (Grinspun et al., 2003).

Baraff's model

Since Baraff and Witkin (1998) introduced a constraint-based method with implicit time-stepping in cloth simulation, their method has been particularly popular in graphics. Pixar and Walt Disney Animation Studios also use their model in many films (Eberle, 2018; Tamstorf et al., 2015). The energy function of Baraff and Witkin (1998) is similar to the stretching term, $||G||_F^2$, of the St.Venant-Kirchhoff energy but they use a square root of the strain for the stretch forces. Here, G refers to the Green's strain tensor. The same logic can be applied to position-based dynamics models in both 2D and 3D (Müller et al., 2014). Although Baraff and Witkin (1998) did not explicitly use hyperelastic energies, it was later proven that their model is equivalent to the use of an anisotropic hyperelastic strain energy (Kim, 2020).

Corotated Model

A corotated model is a popular linear model that has been consistently adapted in graphics (Bender and Deul, 2013; Chao et al., 2010; McAdams et al., 2011). What makes a corotated model different from a typical linear elastic model is its rotational invariance. Corotational models often came with different descriptions or even different names (Etzmuß et al., 2003; Irving et al., 2004; Müller et al., 2002), but they all aimed to remove rotational artifacts in a linear model. Therefore, they have an additional cost for factoring out rotation in the deformation gradient. They can be further modified in different ways for better computational cost (Kugelstadt et al., 2018), better robustness (Stomakhin et al., 2012), etc.

Neo-Hookean

The Neo-Hookean model is a popular *nonlinear* hyperelastic model which is in general more suitable for large deformation than linear models. The most common version was introduced by Bonet and Wood (2008). This model is especially immune to excessive compression: the energy density contains a logarithmic term, $(log(J))^2$, where J is the determinant of the deformation gradient, representing volume changes of the model. Under excessive compression where J is small, the logarithmic term exhibits a large energy to cancel out the compression. There exist many other versions (Bower, 2009; Wang and Yang, 2016), and relatively recently, Smith et al. (2018) proposed a stable Neo-Hookean model that is more robust to extreme inversions and rotations and better preserves the volume of the material than previous methods.

3.2 Non-Manifold Single-Type Elastic Models

Unlike the (manifold) models in the preceding section (Section 3.1), single-type models can also have a non-manifold shape, i.e., more than two rod segments connected by a joint vertex, more than two shell faces connected by a joint edge, two shells or two solids connected by a joint vertex, etc. Spillmann and Teschner (2009) generalized their CoRDe elastic rod model to non-manifold (T-junction) rod configurations to represent Cosserat nets, and Bertails et al. (2006) used non-manifold joints within the super-helices model for branching tree structures. Pérez et al. (2015) used the notion of a connection edge, which acts as a counterpart of an edge representing all the other edges at a point connection. Similarly, Cirak and Long (2011) considered non-manifold shells in which multiple surfaces share an edge or sequence of edges. These higher-valence connections can be handled by our framework, but are not our primary focus.

3.3 Unified Non-Manifold Elastic Models

Autodesk's Nucleus platform (Stam, 2009) and position-based dynamics (Macklin et al., 2014; Müller and Chentanez, 2011; Müller et al., 2007) are the approaches most conceptually similar to ours. These approaches use non-manifold mesh structures augmented with various constraints and/or shape-matching mechanisms to approximate elastic deformations of objects, including rods (Kugelstadt and Schömer, 2016; Umetani et al., 2014); however, they place a reduced emphasis on accuracy compared to the standard continuum

mechanics-based approaches that we build upon, and they do not consider elastic bending and twisting at point connections outside the context of pure rods.

Dispensing with an explicit mesh, Martin et al. (2010) proposed a meshless elastic model known as *elastons*. Their approach can model continuous elastic bodies, but it is complex, is slower than the classic single-type models we build on, and does not address the more general connections and embeddings we consider. Specifically, the elastons model assumes a smooth body of material, whereas our method supports (1) singular point or segment connections with stiffness independent of the connected materials, and (2) embedded structures that allow sharp jumps in stiffness and the ability to disable twisting and/or bending coupling at connections. These features can straightforwardly support diverse behaviors for the same geometry (e.g., see our frisbee, toy tunnel, and sandwich composite examples in Figures 6.3, 6.4, 6.6.)

Moreover, the computational overhead of the mesh-free setting generally exceeds that of existing mesh-based alternatives. A more efficient alternative was proposed by Faure, Gilles, and co-workers (Faure et al., 2011; Gilles et al., 2011), which assigns sparse local coordinate frames to points on the object and uses modified shape functions to construct a continuum mechanics formulation for elasticity. However, this method assumes a volumetric representation of the solid and does not consider rods and shells.

Zhu et al. (2015; 2014) explored the simulation of liquids on non-manifold simplicial meshes. Our work is distinct in that we focus on purely elastic solids rather than fluids. In addition, their lower-dimensional models (i.e., threads and sheets) neglect bending and twisting effects altogether, which play a critical role in the distinctive behaviors of elastica.

3.4 Specialized Elastic Model Coupling

Instances of mixed-dimensional interactions, similar in spirit to our work, have been considered in a number of special cases. Li et al. (2014) used impulses to couple cloth and rods to model an umbrella. Chentanez et al. (2009) modeled prostate brachytherapy by coupling embedded elastic rod-based needles to a tetrahedral solid with Lagrange multipliers that allow sliding. Rémillard and Kry (2013) coupled a much higher resolution shell-based skin model to a lower-resolution volumetric simulation to accurately model skin wrinkling effects. Bergou et al. (2008) used Lagrange multipliers to affix a rigid body to the end of a rod with matching orientation. The position-based rod model of Umetani et al. (2014) supports attaching a rod's endpoint to frames, triangles, or rigid bodies. Xu et al. (2018) used equality constraints on the twisting angles to connect different pairs of wire pieces. Pérez

et al. (2017) proposed a computational design framework for Kirchhoff-Plateau surfaces using a cloth model augmented with elastic rods embedded in the plane of the cloth. We aim to develop a more broadly applicable unified framework. In engineering, joint or interface elements have been proposed for finite element methods to connect different materials across interfaces, often for shell-shell or solid-shell surface contacts (similar to our embedded models). For example, this approach has been used in geomechanics for modeling joints or fractures in rock structures (Beer, 1985; Schellekens and De Borst, 1993).

3.5 Constraint-Based Coupling

We assume that the elastic object to be modeled consists of a single conforming simplicial mesh. A typical alternative is to model each single-dimensional component in isolation and introduce specialized position and/or orientation constraints to tie components back together in a desired fashion; such constraints could be enforced by either Lagrange multipliers (Platt and Barr, 1988) or penalty methods (Witkin et al., 1988) (i.e., "hard" or "soft" constraints). For example, simple spring forces have long been used to model joint forces and/or joint limits in the context of articulated rigid bodies (Isaacs and Cohen, 1987; Wilhelms, 1987). Such an approach can be highly flexible: meshes need not even be geometrically conforming and constraints can be tailored to particular tasks. However, this strategy introduces some mesh redundancy and may be more complex than necessary for many common elastic body scenarios. Our mixed-dimensional elastic coupling energies can be interpreted as particular instances of penalty methods for enforcing coupling of coordinate frames, although the coupling of positions is guaranteed implicitly through the use of a single conforming mesh. Tournier et al. (2015) recently presented a method that conceptually unifies elasticity and constraints in order to handle constrained systems in a stable fashion; we expect that this technique could be beneficially applied alongside the elastic connection energies that we propose. Another coupling approach, related to Lagrange multipliers, is to directly "bind" or "embed" some particles such that their motion is driven strictly by that of a parent (Sifakis et al., 2007; Twigg and Kacic-Alesic, 2010); this can also be interpreted as a kind of reduced coordinate model, in that the positions of bound particles are described only in relation to their parent object's vertices or reference frame, rather than as truly independent degrees of freedom. In our unified representation, all vertices belong to a single conforming simplicial mesh, so no explicit binding is necessary.

Chapter 4

Elastic Energies for Single-Type Models

$\mathbf{e}$	edge vector
$\mathbf{r}_\alpha$	rod's reference directors ($\alpha \in \{1,2,3\}$). $\mathbf{r}_3$ refers to the director along rod's centerline.
$\mathbf{m}_\alpha$	rod's material directors ($\alpha \in \{1,2,3\}$). $\mathbf{m}_3$ refers to the director along rod's centerline.
m	rod's twist between two adjacent edge segments
$\boldsymbol{\kappa}$	rod's discrete material curvature
G	Green's strain tensor
$\boldsymbol{\phi}$	deformation map
$\mathbf{F}$	deformation gradient
J	determinant of deformation gradient
Ψ	energy density of a shell or a sold
$\mathbf{X}$	undeformed vertex position
$\mathbf{x}$	current (deformed) vertex position
λ, μ	Lamé's first and second parameter
Y	Young's modulus
ν	Poisson's ratio
h	shell's thickness

Table 4.1: Summary of Notations

In this chapter, we briefly review elastic energies for each single-type model (i.e., 1D

rods, 2D shells, and 3D solids) that we intend to couple in the next chapters. Interestingly, as the dimension of the model increases (i.e., rods to shells to 3D solids), the number of types of energies each model is subject to decreases. 1D rod segments are subject to elastic stretching, bending, and twisting but for 2D manifold triangles, there are only in-plane membrane deformation and out-of-plane bending between triangles. A 3D tetrahedral mesh solely has per-tetrahedron deformation; two adjacent tetrahedra connected by a face do not generate additional motions such as bending between triangles for the shell case. For 1D rods, per-segment motion is a fairly simple one dimensional stretching but the other two motions (i.e., bending and twisting) are relatively hard to express. For 2D and 3D models, more complicated elementwise (i.e., per triangle or per tetrahedron) deformation occurs, but they can be generalized to the same expressions except with different dimensionality. Thus, we group the energy models for shells and solids together with additional explanations for bending which happens only for 2D shells.

Given the energy models in each dimension, we can express the corresponding forces $\mathbf{f}$ by taking a negative gradient of the potential energies E,

$$\mathbf{f} = -\frac{\partial E}{\partial \mathbf{x}}. \tag{4.1}$$

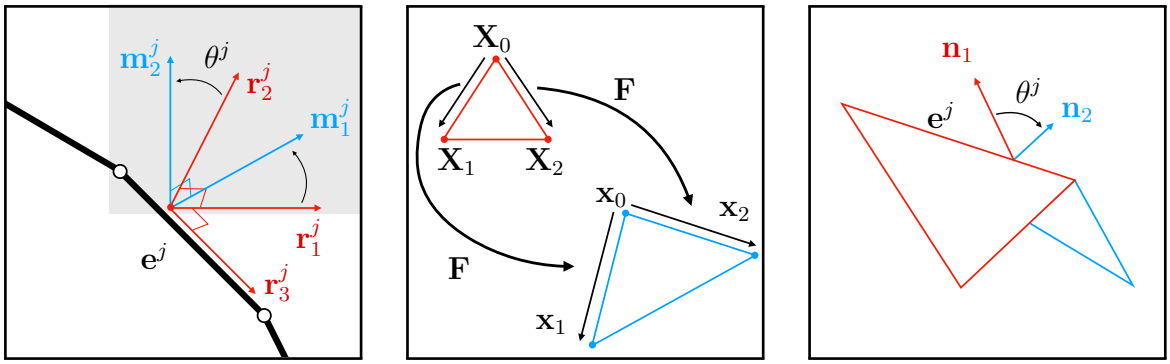
The vertex positions of elastic objects are updated by using the forces with a time integration scheme. We specifically adapt the famous (semi-)implicit time integration method popularized in graphics by Baraff and Witkin (1998), which enables taking large steps stably:

$$\begin{aligned} \left(\mathbf{M} - \Delta t^2 \frac{\partial \mathbf{f}^n}{\partial \mathbf{x}}\right) \Delta \mathbf{x}^{n+1} &= \Delta t \mathbf{M} \mathbf{v}^n + \Delta t^2 \mathbf{f}^n \\ \mathbf{x}^{n+1} &= \mathbf{x}^n + \Delta \mathbf{x}^{n+1}. \end{aligned} \tag{4.2}$$

Here, $\mathbf{M}$ is the mass matrix, Δt is the timestep, $\mathbf{f}$ is the force vector, $\mathbf{x}$ is the position vector, and $\mathbf{v}$ is the velocity vector. Two differences of (4.2) from the original formulation are, first, we solve for $\Delta \mathbf{x}$ not $\Delta \mathbf{v}$, and second, we ignore the damping terms in the original formulation but use the Stokes-Rayleigh analogy to derive the damping forces (Batty et al., 2012; Bergou et al., 2010).

4.1 Discrete Elastic Rods

In the discrete setting, rods are treated as piecewise linear segment meshes (Figure 4.1(a)). In this section, we assume the discrete rod has $n + 1$ segments and n joint vertices. We



(a) Elastic rods (b) Elastic shells (c) Hinged-based bending

Figure 4.1: **Elastic Energy Models.** (a) Elastic rod configuration. $\mathbf{r}_\alpha^j$ is a reference director and $\mathbf{m}_\alpha^j$ is a material director on the j^{th} edge. $\mathbf{r}_3^j$ is along the edge vector $\mathbf{e}^j$. The shaded region (grey) represents a plane perpendicular to $\mathbf{r}_3^j$. The material directors can be obtained by rotating the reference directors by θ^j around the edge vector $\mathbf{e}^j$. (b) Deformation gradient of one triangle within an elastic shell. $\mathbf{X_0}$, $\mathbf{X_1}$, and $\mathbf{X_2}$ are vertex positions in the rest (undeformed) configuration, and $\mathbf{x_0}$, $\mathbf{x_1}$, and $\mathbf{x_2}$ are the vertex positions in the current (deformed) configuration. The edge vectors in the rest configuration are mapped to the ones in the current configuration by deformation gradient $\mathbf{F}$. (c) Bending configuration using a dihedral angle between two triangles. $\mathbf{n}_1$ and $\mathbf{n}_2$ represent the normal vectors of the triangles and θ^j is the angle between them.

use e^j for the j^{th} edge segment ($0 \leq j \leq n$), $\mathbf{x}_i$ to indicate the i^{th} vertex position, and $\mathbf{e}^j$ to represent the edge vector of e^j, $\mathbf{e}^j = \mathbf{x}_{j+1} - \mathbf{x}_j$.

The first and simplest elastic energy in rods is a stretching energy. Using the model of Bergou et al. (2010), the stretching energy is given by

$$E_s = \frac{1}{2}\sum_{j=0}^{n} k_s^j \left(\epsilon^j\right)^2 \left|\bar{\mathbf{e}}^j\right|, \tag{4.3}$$

where j refers to an edge index, overlines denote rest state quantities, k_s is the stretching stiffness, $|\mathbf{e}|$ is the edge length, and ϵ means the relative axial strain, $\epsilon^j = \frac{|\mathbf{e}^j|}{|\bar{\mathbf{e}}^j|} - 1$. For rods, the elementwise deformation occurs in one direction which simplifies the strain and the corresponding stretching energy compared to shells or solids. Unlike a twisting and bending energy, the stretching energy is contained in each edge and the orientation of each edge is not considered in computation.

For twisting and bending energies, the orientation needs to be considered and the orientation is represented by the angle deviation from a *reference* frame around the edge vector (Figure 4.1(a)). The frame that displays the actual orientation of the edge segment is called a *material* frame. Here, each frame has three orthonormal basis vectors called directors (e.g., reference directors, material directors), and we use $\mathbf{r}_\alpha^j$ and $\mathbf{m}_\alpha^j$ ($\alpha \in \{1, 2, 3\}$) to denote the reference directors and material directors on the j^{th} edge, respectively. Note that the third reference director $\mathbf{r}_3^j$ and the third material director $\mathbf{m}_3^j$ are identical; they are along the edge vector $\mathbf{e}^j$. There are different choices possible for the reference frame. One can choose the geometrically most natural reference frame (Bishop frame) (Bergou et al., 2008) or using a space-time analogy, one can choose a natural frame in time (Bergou et al., 2010; Kaldor et al., 2010). We use the time-parallel reference frame for efficiency. With this time-parallel reference frame, the amount of *twist* between $(i-1)^{th}$ edge and i^{th} edge can be expressed as $m_i = \Delta\theta_i + r_i$, where $\Delta\theta_i = \theta^i - \theta^{i-1}$, θ^i is the angle between the reference frame and material frame on the i^{th} edge, and r_i represents the twist of the reference frames between the edges. The corresponding twisting energy is

$$E_t = \frac{1}{2}\sum_{i=1}^{n} k_{t\_i} \frac{(m_i - \bar{m}_i)^2}{\bar{l}_i} \tag{4.4}$$

where k_{t_i} is the twisting stiffness, $\bar{l}_i$ is the Voronoi length associated to the joint vertex between e^{i-1} and e^i.

The bending energy is measured by using a discrete material curvature $\boldsymbol{\kappa}$:

$$E_b = \frac{1}{2}\sum_{i=1}^{n} \frac{1}{\bar{l}_i}(\boldsymbol{\kappa}_i - \bar{\boldsymbol{\kappa}}_i)^T K_{b\_i}(\boldsymbol{\kappa}_i - \bar{\boldsymbol{\kappa}}_i) \tag{4.5}$$

where K_{b_i} is the bending stiffness tensor. Using the method of Bergou et al. (2010, 2008), the discrete curvature vector $\boldsymbol{\kappa}$ at a particular vertex along the centerline is

$$\boldsymbol{\kappa}_i = \frac{1}{2} \sum_{j=i-1}^{i} \left((\kappa \mathbf{b})_i \cdot \mathbf{m}_2^j, -(\kappa \mathbf{b})_i \cdot \mathbf{m}_1^j \right)^T \tag{4.6}$$

where $(\kappa \mathbf{b})_i = \frac{2\mathbf{r}_3^{i-1} \times \mathbf{r}_3^i}{1 + \mathbf{r}_3^{i-1} \cdot \mathbf{r}_3^i}$. Thus, the discrete curvature essentially depends on the joint angles between the edges. The material properties (e.g., Young's modulus, shear modulus) and the shape of the rod (e.g., cross-section area, major and minor radii of ellipse) are folded into the stiffness coefficients for simplicity. (Refer to the work of Bergou et al. (2010) for the full formulation of the stiffness coefficient.)

4.2 Discrete Elastic Shells and Solids

Unlike a simple axial strain in rods, shells and solids present more complicated elementwise deformation. To capture such deformation, we use the concept of deformation map and deformation gradient from continuum mechanics. A deformable object can be characterized by the time-dependent deformation map $\boldsymbol{\phi}(\mathbf{X}, t)$ from the position in the undeformed (rest) configuration, $\mathbf{X}$, to the one in the deformed (current) configuration, $\mathbf{x}$, or $\mathbf{x} = \boldsymbol{\phi}(\mathbf{X}, t)$ (Figure 4.1(b)). The Jacobian of the deformation map, or *deformation gradient* $\mathbf{F}$, effectively captures local deformation of a material or how small displacement in the rest configuration is mapped to the corresponding displacement in the current configuration.

$$\mathbf{F} = \frac{\partial \boldsymbol{\phi}(\mathbf{X}, t)}{\partial \mathbf{X}} = \frac{\partial \mathbf{x}(\mathbf{X}, t)}{\partial \mathbf{X}} \tag{4.7}$$

In the discrete setting, the usage of $\mathbf{F}$ is straightforward: for the triangle in Figure 4.1(b), each edge corresponds to the small displacement in (4.7)

$$\begin{aligned} \mathbf{x}_1 - \mathbf{x}_0 &= \mathbf{F}(\mathbf{X}_1 - \mathbf{X}_0) \\ \mathbf{x}_2 - \mathbf{x}_0 &= \mathbf{F}(\mathbf{X}_2 - \mathbf{X}_0) \end{aligned} \tag{4.8}$$

or in the matrix form

$$\mathbf{D}_c = \mathbf{F}\mathbf{D}_r \tag{4.9}$$

where $\mathbf{D}_r = [\mathbf{X}_1 - \mathbf{X}_0 \mid \mathbf{X}_2 - \mathbf{X}_0]$ and $\mathbf{D}_c = [\mathbf{x}_1 - \mathbf{x}_0 \mid \mathbf{x}_2 - \mathbf{x}_0]$. Even for a triangle in 3D, we still use a planar rest configuration, i.e., $\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2 \in \mathbb{R}^2$, and consequently the

deformation gradient $\mathbf{F}$ is a 3×2 matrix. The third edge is linearly dependent on the other two, thus omitted. For a tetrahedron, a simple modification to the triangle model is required: one additional equation is added to (4.8) as there are four corner vertices, i.e., $\mathbf{x}_3 - \mathbf{x}_0 = \mathbf{F}(\mathbf{X}_3 - \mathbf{X}_0)$, where $\mathbf{X}_3$ and $\mathbf{x}_3$ are the position of the new (fourth) vertex in the rest and current configuration. Also, the rest configuration is in $\mathbb{R}^3$ which makes the dimensions of the deformation gradient $\mathbf{F}$ 3×3. Given the deformation gradient $\mathbf{F} = \mathbf{D}_c\mathbf{D}_r^{-1}$, we can adapt different FEM-based energy functions by plugging in the deformation gradient to the energy models. For a solid, we use a St.Venant-Kirchhoff material which can be written as

$$\Psi(\mathbf{F}) = \mu||G||_F^2 + \frac{\lambda}{2}\mathrm{tr}^2 G \tag{4.10}$$

where λ and ν are Lamé's first and second parameter, and G is a quadratic Green's strain, $G = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right)$. For a shell, we use an isotropic linear elastic model (Gingold et al., 2004)

$$\Psi(\mathbf{F}) = \frac{1}{2}\frac{Yh}{(1-\nu^2)}((1-\nu)\mathrm{Tr}(G^2) + \nu(\mathrm{Tr}G)^2) \tag{4.11}$$

where Y is Young's modulus, ν is Poisson's ratio, and h is the thickness of the shell. Note that $\Psi(\mathbf{F})$ is an energy density, thus it must be multiplied by the area of a triangle or the volume of a tetrahedron to get the final energy formulation for each element.

4.2.1 Dihedral Bending for Shells

For shells, an additional energy must be applied to capture the bending behavior. We use a simple and efficient dihedral bending model of Grinspun et al. (2003).

$$E_b = \frac{1}{2}\sum_j k_b^j(\theta^j - \bar{\theta}^j)^2 \tag{4.12}$$

where j refers to the edge index between two adjacent triangles, k_b is the bending stiffness, θ is the angle between the two triangles, and $\bar{\theta}$ is the angle at the rest state. Again, we fold the material properties and the shape of the triangles into the stiffness coefficient k_b. (Refer to the work of Grinspun et al. (2003) and Batty et al. (2012) for the full expression.)

Chapter 5

Connecting Single-Type Models

Having described all of the single-type models, we return to the question of how to treat connections among them to model complex non-manifold simplicial structures. We consider each of the three basic connection types in turn: shared point, shared curve, and shared surface. Such connections may be treated as either stiff/hard constraints, in which the initial relative orientation of the two components remains unchanged, or as soft elastic connections, which allow modeling of flexibly deforming joints. Our discussion and evaluation focuses on the latter elastic penalty-like scenario, since it enables more general deformations and introduces interesting modeling challenges. However, in the case of truly "hard" constraints, a Lagrange multiplier formulation is often preferable, so our discussions will briefly touch on this variation.

5.1 Point Connections

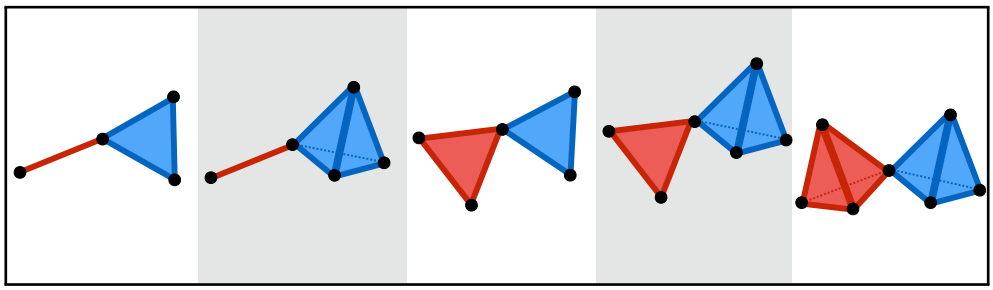


Figure 5.1: **Different Point Connection Types.** From left to right: rod-shell, rod-solid, shell-shell, shell-solid, and solid-solid coupling at a joint vertex.

We begin with point connections, of which there are five cases: rod-shell, rod-solid, shell-shell, shell-solid, and solid-solid (Figure 5.1). Despite this variety, we handle them consistently: we construct coordinate frames for each side of a connection at the shared point, and design appropriate deformation energies based on the *change* in their relative configurations. The main technical hurdle arises in accurately measuring these changes.

Since our geometry is a single conforming mesh, there is no need to explicitly enforce positional coincidence of shared vertices; their shared degrees of freedom implicitly yield ball-joint behavior in the absence of additional forces. This leaves only deviations in orientation to consider. For hard constraints, we can simply constrain the axes (or directors) of one coordinate frame to be fixed with respect to the other, by requiring fixed dot products. Denoting the two coordinate frames with their director vectors, $\{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\}$ and $\{\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3\}$ (e.g., Figure 5.2), we have the constraint

$$\begin{bmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{bmatrix} \begin{bmatrix} \mathbf{n}_1, \mathbf{n}_2 \end{bmatrix} - \begin{bmatrix} \overline{\mathbf{m}}_1^T \\ \overline{\mathbf{m}}_2^T \\ \overline{\mathbf{m}}_3^T \end{bmatrix} \begin{bmatrix} \overline{\mathbf{n}}_1, \overline{\mathbf{n}}_2 \end{bmatrix} = \mathbf{0} \tag{5.1}$$

where overlines denote rest state quantities. Constraining only two directors, $\overline{\mathbf{n}}_1$ and $\overline{\mathbf{n}}_2$, suffices because the third is orthogonal.

For softer elastic connections, a first obvious choice would be to form an elastic potential by squaring the left side of (5.1) and multiplying by a stiffness parameter. This is undesirable for two reasons. First, this choice cannot support accumulated twisting angles that exceed π; in such a case, the joint will suddenly begin rotating in the *opposite* direction, since this is the quickest path to realigning the directors, despite the fact that it

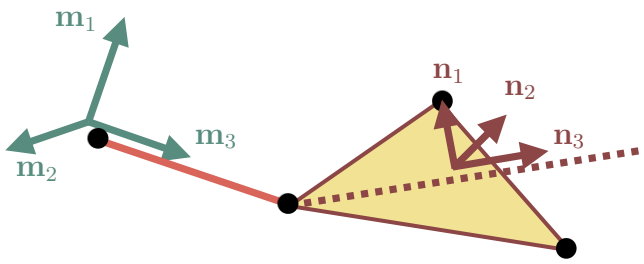


Figure 5.2: **A rod-shell point connection with coordinate frames.** The dashed line represents the shell's twisting axis.

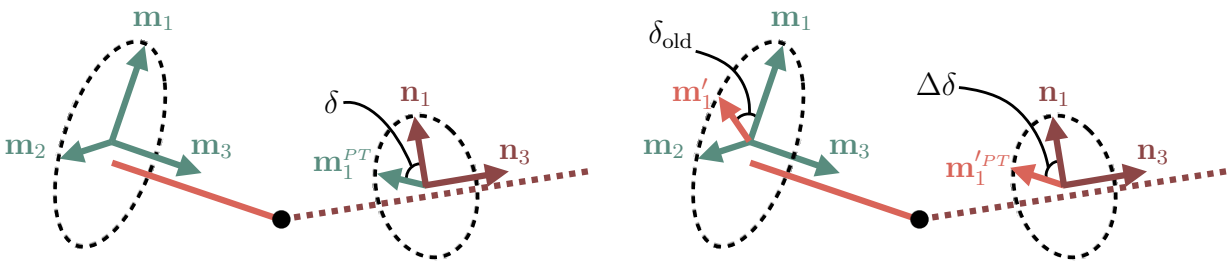
actually further *increases* the true net twist. Second, this formulation does not explicitly separate twisting and bending deformations into orthogonal modes, which implies that their stiffness parameters cannot be assigned independently. Our experience suggests that the ability to separately control bending and twisting behavior is a feature that artists find useful. Therefore, we instead took inspiration from the dynamics of elastic rods.

5.1.1 Twisting Energy

Twisting energy is accumulated when the two sides of a point connection become twisted with respect to their undeformed relative configurations. While a rod inherently possesses a centerline around which twisting is measured, shell and solid models do not. To ensure a compatible interface we must construct appropriate twisting axes and associated coordinate frames, but we temporarily defer this discussion to Section 5.1.3.

For now, consider two simplices sharing a single vertex, with associated orthonormal coordinate frames, $\{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\}$ and $\{\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3\}$, where the simplices' centerlines (rods) or chosen twisting axes (shell, solid) are assumed to lie along $\mathbf{m_3}$ and $\mathbf{n_3}$, respectively, as in Figure 5.2. Unfortunately, these two axes will not necessarily be mutually aligned, so the twist angle *cannot* be correctly measured by simply examining the change in angle between their perpendicular directors, $\mathbf{m}_1$ and $\mathbf{n}_1$. Instead, we adapt ideas from the elastic rods of Bergou et al. (2008).

Recall that discrete *parallel transport* applies the minimal rotation about the binormal that keeps a tangent vector tangential to the implied curve, as we move from one segment to the next. By parallel transporting the perpendicular director $\mathbf{m}_1$ into the other coordinate frame, we find a vector $\mathbf{m}_1^{PT}$ which *can* be safely compared to $\mathbf{n}_1$ to determine the actual



(a) Direct angle update. A subsequent post-processing step is required to ensure the correct angle increment is determined (**Incremental I**).

(b) Incremental angle update (**Incremental II**).

Figure 5.3: **Point connection angle updates using parallel transport.**

twist angle, δ (Figure 5.3a). Given the angle δ, we define the point's twisting energy as

$$E_t = \frac{1}{2} k_t (\delta - \bar{\delta})^2, \tag{5.2}$$

where δ is the deformed angle, $\bar{\delta}$ is its undeformed counterpart, and k_t is the twisting stiffness coefficient. Since the parallel transport operation is perfectly reversible, the choice of which simplex coordinate frame to start from is arbitrary.

Remark This energy is intentionally defined to be concentrated at the singular non-manifold point, rather than as an integral over a local region. This allows for greater consistency and flexibility for general deformable connections. Specifically, we prefer that a point connection: (1) yield consistent deformation behavior independent of the resolution (length/area/volume) of incident simplices; (2) support an arbitrary choice of stiffness independent of the incident material type(s). (By contrast, point-based models such as Elastons (Martin et al., 2010) require uniform material behavior across all (implied) dimension transitions.)

Incremental angle update The remaining shortcoming of this approach is that if the angle deviation δ is (re-)computed directly from the coordinate frames at each step, the resulting angle (and energy) will suffer from discontinuous jumps as δ wraps around in the space of angles; for example, the relevant trigonometric functions cannot naturally distinguish π from 3π based on the current coordinate frames alone. Prior rod-only models do not typically suffer from this issue due to their choice of twist representation, such

as quaternions or persistent scalar twisting angles (Bergou et al., 2008; Spillmann and Teschner, 2007). The absence of this information in general simplices necessitates our new approach.

We propose two possible incremental angle update schemes that can overcome this by instead solving for only the *incremental* change in δ, while properly accounting for parallel transport:

- **Incremental I:** After performing the direct angle update (Figure 5.3a), one can apply an additional post-processing step to circumvent the discontinuous angle jumps. Instead of using the computed angle directly, we first add (or subtract) the multiple of 2π that yields the closest angle to the angle from the previous time step.
- **Incremental II:** Alternatively, one can pre-rotate the director ($\mathbf{m}_1$) by the previous accumulated angle (δ_{old}), before parallel transporting to compare with the other director. Consider the twisting angle at a rod-shell connection as an example (Figure 5.3b). Before parallel transporting $\mathbf{m}_1$ to the other frame, we rotate $\mathbf{m}_1$ around $\mathbf{m}_3$ by δ_{old} to yield $\mathbf{m}_1'$, as shown in Figure 5.3b. We then apply parallel transport to get $(\mathbf{m}_1')^{PT}$, and compute the angle increment as the angle between $(\mathbf{m}_1')^{PT}$ and $\mathbf{n}_1$. This process yields an update to δ of

$$\delta = \delta_{\text{old}} + \text{arctan2}((\mathbf{m}_1')^{PT} \cdot \mathbf{n}_2,\ (\mathbf{m}_1')^{PT} \cdot \mathbf{n}_1). \tag{5.3}$$

The above methods work equally well and share the same mild limitation: the angle increment cannot exceed π on a single timestep. We use the first approach (Incremental I) in all the examples.

5.1.2 Bending Energy

Bending also measures an angle deviation, but in the directions orthogonal to twisting. To calculate the bending force, we track the degree to which the second coordinate frame's twisting axis, $\mathbf{n}_3$, deviates from its rest state in the first coordinate frame. Let $\mathbf{d}$ be defined as

$$\mathbf{d} = (\mathbf{n}_3 \cdot \mathbf{m}_1, \mathbf{n}_3 \cdot \mathbf{m}_2, \mathbf{n}_3 \cdot \mathbf{m}_3) \tag{5.4}$$

which is simply the second frame's twisting axis $\mathbf{n}_3$ expressed in the first coordinate frame.

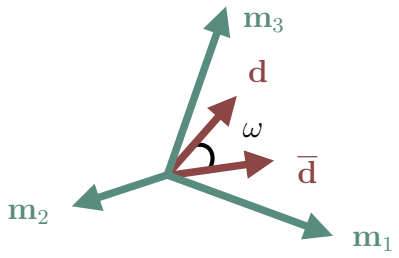


Figure 5.4: **Bending at a point connection.** When expressed in the other coordinate frame (green) the twisting axis is denoted by $\mathbf{d}$ and compared with its rest state counterpart, $\overline{\mathbf{d}}$, to determine the bending angle ω.

The angle difference between $\mathbf{d}$ and its undeformed counterpart $\overline{\mathbf{d}}$ gives the bending angle ω, as illustrated in Figure 5.4. Consequently, the bending energy has the form

$$E_b = \frac{1}{2} k_b \omega^2. \tag{5.5}$$

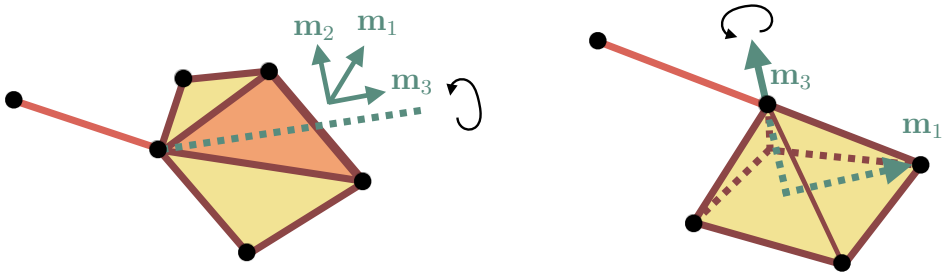
We do not treat the angle update for ω incrementally because unlike twisting there is no single axis around which this bending occurs, i.e., $\mathbf{d}$ can deviate from $\overline{\mathbf{d}}$ in multiple directions.

5.1.3 Choosing Coordinate Frames at Vertices

Applying our twisting and bending energies at a point requires defining appropriate orthonormal coordinate frames at the shared vertex, with their third director aligned along the centerline (rods) or along an appropriately chosen twisting axis (shells and solids). For rods, the natural coordinate frame is its inherent material frame. For shells or solids, twisting axes could be chosen by the user to enable various behaviors. We suggest some natural choices below.

Shells

For a shell, the point connection may either be on the shell's outer boundary (Figure 5.5a) or in the interior (Figure 5.5b). For the shell boundary case, we set the twisting axis, $\mathbf{m}_3$, to be the tangent vector that bisects the total angle formed by all the incident shell triangles, in their rest configuration (Figure 5.5a). The triangle's normal can be used as $\mathbf{m}_1$, and the remaining vector $\mathbf{m}_2$ is determined from the cross-product of the other two.



(a) Boundary point. (b) Interior point.

Figure 5.5: **Twisting axis $\mathbf{m}_3$ choices for a shell at a rod-shell point connection.**

For the shell interior case, we define the twisting axis $\mathbf{m}_3$ to be the vertex normal of the triangle mesh, which is a function of the one-ring of vertices around the point. A second director $\mathbf{m}_1$ can be found by taking one edge of an incident triangle and projecting out its component along $\mathbf{m}_3$; their cross-product provides $\mathbf{m}_2$. Using only a single edge in this case can slightly bias the behavior with respect to that edge, in the sense that only changes in that edge influence $\mathbf{m}_1$ and $\mathbf{m}_2$. If this bias is deemed undesirable, it can be avoided by simply creating a duplicate twisting energy for each incident edge, and re-scaling the stiffness coefficients to compensate. (Since bending involves only the single $\mathbf{n}_3$ vector on one side, this issue is absent for the bending energy, except when both sides are shells or solids.)

Solids

For the solid case, two natural options present themselves. The first is to compute a surface vertex normal as the twisting axis, and connect it to the single tetrahedron penetrated by that vector; this approach is essentially a 3D extension of the shell boundary approach of Figure 5.5a. The second possibility is to mimic the shell interior case, where the solid's exterior triangulation takes the place of the shell triangulation. From an implementation standpoint, the former is more attractive since the stencil involves only one simplex on each side, whereas the latter involves the whole one-ring of triangles.

Remark The choices above yield behavior that is, by design, analogous to elastic rods in the manner in which twisting propagates between elements. While this is a natural choice, it is important to understand its effect. Consider a shell joined *at an angle* to a rod

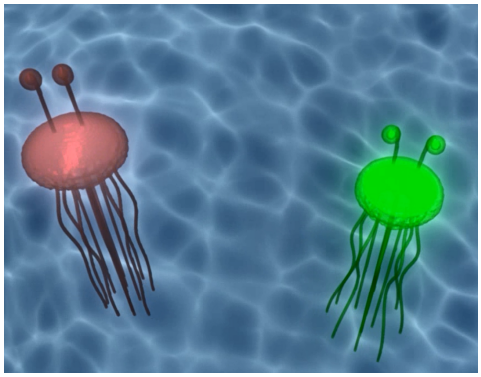


Figure 5.6: **Jellyfish:** An alien jellyfish species composed of rod, shell, and solid components joined by point connections.

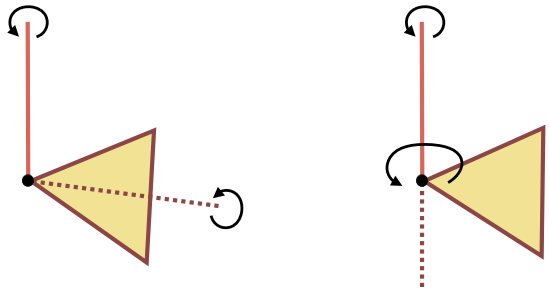


Figure 5.7: **Influence of twisting axis choice**: when the rod is rotated, our proposed twisting axis causes the shell to rotate in a manner analogous to an elastic rod, i.e., about its own centerline (left). An alternative would be to use the rod's axis for the shell frame, which leads to different behavior (right).

(Figure 5.7). Under the approach outlined above, rotating the rod will cause the shell to rotate *about its own twisting axis*, akin to elastic rods or bevel gears that meet at an angle, rather than, for example, rotating about the centerline of the rod. If the latter option were desired, one could construct the coordinate frame for the shell based on the rod, though this would be inconsistent with the behavior of regular rod-rod connections.

5.2 Curve Connections

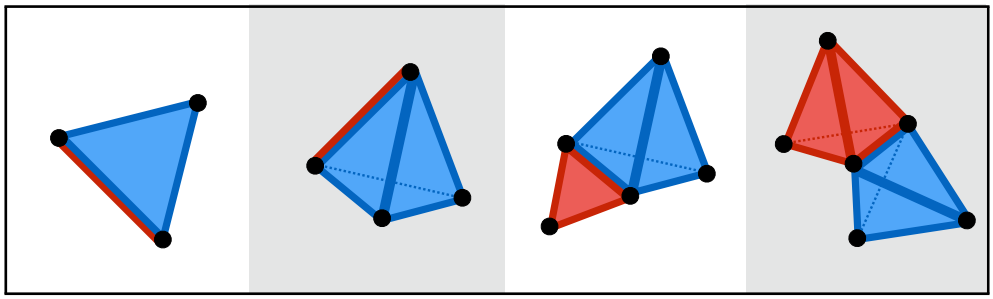


Figure 5.8: **Different Curve Connection Types.** From left to right: embedded rod-shell, embedded rod-solid, shell-solid, and solid-solid coupling at a shared edge.

A curve connection between two models can arise in several cases: two solids sharing a surface curve, a solid sharing a surface curve with a shell's boundary or interior, or a rod embedded within either the boundary or the interior of a shell or solid (Figure 5.8). Our assumption of a unified conforming mesh ensures that both sides of the joint share the exact same curve edges; therefore, no relative tangential sliding can occur. Furthermore, we do not need to penalize deformations of the shared curve itself, since these can be directly handled by adding a rod model, if such a behavior is desired. Therefore the only relevant deformation mode at the shared curve is relative rotation of the two sides around the curve. Our conforming mesh immediately guarantees hinge-like behavior by default; in the embedded rod case, this manifests as free rotation within its containing shell or solid, like a tent pole through a fabric sleeve (see e.g., Figure 6.3, middle). Thus, our conforming mesh can be a useful modeling choice in its own right, but if we do wish to penalize relative deformations around the edge, we need an appropriate energy. We assume that coordinate frames for each side of the joint are chosen such that $\mathbf{m}_3$ and $\mathbf{n}_3$ are aligned along the shared curve (unlike point connections). We defer a detailed discussion of coordinate frame construction to Section 5.2.

For a hard constraint, the relative orientation of the coordinate frame directors perpendicular to the curve should not drift over time; that is, the dot products should be constant, using

$$\begin{bmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{bmatrix} \mathbf{n}_1 - \begin{bmatrix} \overline{\mathbf{m}}_1^T \\ \overline{\mathbf{m}}_2^T \\ \overline{\mathbf{m}}_3^T \end{bmatrix} \overline{\mathbf{n}}_1 = \mathbf{0}. \tag{5.6}$$

where again overline notation indicates rest configuration quantities. As described for point

connections, however, simply squaring the left side of this expression does not provide a robust elastic energy, if we wish to support large accumulated deformations (e.g., a shell rotating around a rod by more than π radians). We therefore introduce an angle-based elastic energy to penalize relative rotational deformation between frames around a shared axis. We determine the angle δ between $\mathbf{m}_1$ and $\mathbf{n}_1$ (Figure 5.9) and use it to construct a quadratic energy that is integrated along the length of the shared curve. Letting $\overline{\delta}$ be the desired rest angle and k_b the stiffness of the connection, the resulting smooth energy integral is

$$\int_C \frac{1}{2} k_b (\delta - \overline{\delta})^2, \tag{5.7}$$

where C is a shared curve between two models. Similar to the point connection case, we prioritize consistency and flexibility by assigning this energy an independent stiffness parameter and concentrating it on the singular non-manifold curve. That is, the potential is integrated only along the curve's length, rather than over some local finite volume.

Discretization

Discretizing the energy in (5.7) along the relevant edges of our discrete simplicial mesh, we arrive at a bending-like energy that is a sum over the edges comprising the curve,

$$E_b = \sum_i \frac{1}{2} k_b \|\mathbf{e}^i\| \left(\delta^i - \overline{\delta^i}\right)^2, \tag{5.8}$$

where the superscript i indicates the index of the edge, and $\|\mathbf{e}^i\|$ indicates the length of the edge $\mathbf{e}^i$. It remains only to define and construct the angle δ^i. Since we assumed that $\mathbf{m}_3$ and $\mathbf{n}_3$ are mutually aligned along the shared curve, we only need to measure the angle between $\mathbf{m}_1$ and $\mathbf{n}_1$:

$$\delta = \operatorname{arctan2}(\mathbf{n}_1 \cdot \mathbf{m}_2, \mathbf{n}_1 \cdot \mathbf{m}_1). \tag{5.9}$$

The approach above will still suffer from discontinuous jumps in angle unless an incremental approach is adopted. As described in Section 5.1, we have two choices of incremental update. The first approach (Incremental I) can be applied in the same way as in the point connections: calculate the angle δ naïvely, and then apply a post-processing step to find the true angle. To use the second approach (Incremental II), we again pre-rotate the director $\mathbf{m}_1$ with the previous angle (δ_{old}) and compare it with $\mathbf{n}_1$ to get the angle increment. This is conceptually consistent with our point connection treatment, but simpler because parallel transport is not needed.

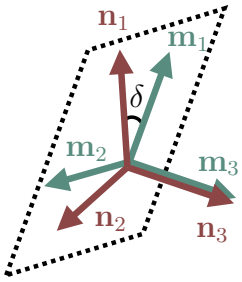


Figure 5.9: **Curve Connection:** Two frames with axes $\mathbf{m}_3$ and $\mathbf{n}_3$ aligned along the shared edge are compared by examining the angle between $\mathbf{m}_1$ and $\mathbf{n}_1$.

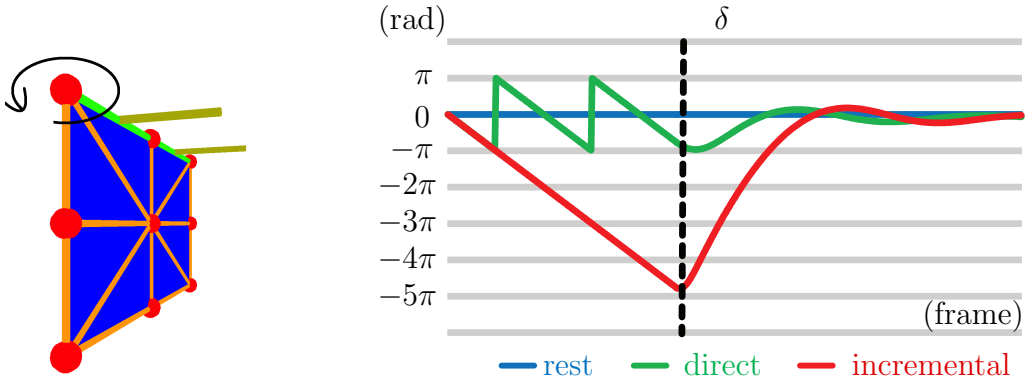


Figure 5.10: **A comparison between the computed angle δ using direct and incremental updates, for a green rod with a large prescribed rotational velocity embedded in a blue shell, under the given rest configuration (left).** We hold the shell vertices fixed for the frames preceding the dashed line while the rod rotates to store up energy. The direct update does not capture the correct angle between the rod's material director (yellow) and the shell's normal vector, so when we release the constrained vertices (vertical dashed line), the shell does not rotate correspondingly. However, the incremental update (red) remembers the history of the deformation, and therefore the shell shows proper angular acceleration to untwist the object at the connection.

As a concrete illustration of our incremental updates, we initialize a rod edge in a single shell triangle with the rest angle $\bar{\delta}$ set to 0 radians as shown in Figure 5.10, left. In the test scenario, we constrain the movement of the shell for the early frames and later release it, while the rod is prescribed to rotate with a constant angular velocity throughout. We observe the values of δ as we rotate the rod while applying the elastic forces described above. As in Figure 5.10, right, we see that the direct angle updates (green curve) show unphysical sudden jumps and the direct updates are also unaware of the accumulated bending angle (i.e. 3π deformation is regarded as π). By contrast, our incremental update (red curve) exhibits smooth changes in δ and it tracks the complete angular deformation.

Choosing Coordinate Frames at Edges

The coordinate frame for a rod is simply its material frame; $\mathbf{m}_3$ is the vector along the (shared) centerline curve. For shells, a coordinate frame can be constructed by setting $\mathbf{m}_3$ as the unit tangent vector along the shared edge. We define $\mathbf{m}_1$ as the normal vector at the edge, which may be either the normal of a single triangle for shell boundary edges, or the average normal of two incident triangles for interior edges. The cross-product yields $\mathbf{m}_2$.

For solids, there are two situations to consider. First, for a solid with an embedded rod, we construct an independent energy for each triangle from the set of tetrahedra incident on the shared edge to minimize bias that would arise in coupling only a single rod-tetrahedron pair. As such, the necessary coordinate frames are constructed for each individual triangle in the same manner as for the shell boundary case above. Second, for a solid sharing a surface curve with a shell or another solid, we use the exterior surface triangulation to construct the coordinate frame, as we did for the interior case of the shell. That is, $\mathbf{m}_3$ lies along the edge, $\mathbf{m}_1$ is the average edge normal from the two incident triangles, and the cross-product is again used to find $\mathbf{m}_2$.

5.3 Surface Connections

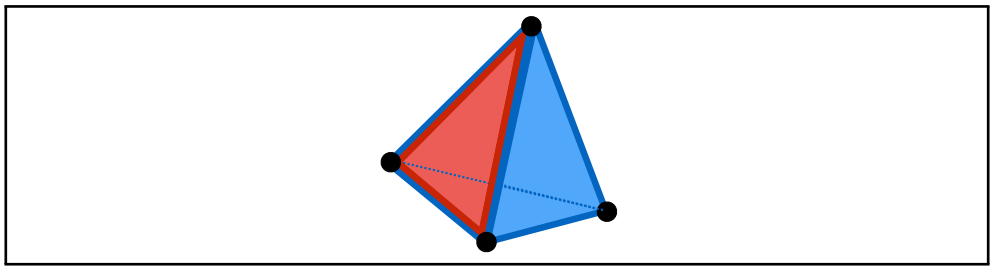


Figure 5.11: **Surface Connection Types:** there is a single case where a triangle (red) is embedded on the surface of a tetrahedron (blue).

The final case, in which two different models share a surface, arises only when a shell is embedded within or on the surface of a solid object. The shell imposes its additional stretching, shearing, and bending forces on the surface shared with the solid. Since our framework assumes a single conforming mesh (without sliding), this type of connection is trivial: we label the appropriate simplices as standard shell or solid elements, and apply their associated forces to the vertices as usual. Unlike point or curve connections, no remaining degrees of freedom exist.

Now that we have all the connection energies between single-type models, we can sum up and apply the energies in a numerical simulation for our conformal mesh in the same manner as for single-type meshes, as discussed at the start of Chapter 4 (e.g., equations 4.1 and 4.2). We solved the resulting nonlinear systems using a standard Newton solver with conjugate gradient for the inner linear solves, although it would be interesting to explore alternatives for greater efficiency (Li et al., 2019). We employed Bridson's collision detection and resolution strategy (Bridson et al., 2002).

Chapter 6

Results

All of the animation examples that follow were computed on a 2.8 GHz Intel Core i7 processor. Single-threaded performance data is listed in Table 6.1; however, there likely remain significant opportunities for optimizing our prototype implementation, as our focus is primarily on the flexibility of the system rather than speed. Moreover, since cross-dimensional connections typically comprise a small subset of the domain compared to single-type regions, our modifications are unlikely to be a bottleneck unless the elastic coupling at the connection is far stiffer than the surrounding material.

Each example is constructed by first designing a single non-manifold geometric mesh, and then tagging simplices and simplex-pairs with model types and connection types, respectively.

Note that the coverage of our method is different from the previous coupling approaches between different simplicial meshes. The previous methods only cover a small subset of the couplings (e.g., rigid-body attachment to a rod (Bergou et al., 2008)), while we present a complete set of coupling energies that allows simulating any shapes of objects (Figure 2.2).

Didactic Examples Our supplemental video includes several animations that exercise the various connection types in isolation. For each connection type, a scenario is shown with and without the associated energy being applied, to highlight the behavior that it assigns to the simplicial mesh. Figure 2.2 shows a scenario where a single complex non-manifold simplicial mesh is assigned many individual connection types. In the video, the tetrahedron on the right end is scripted to rotate, leading the entire shape to rotate, bounce, and deform elastically

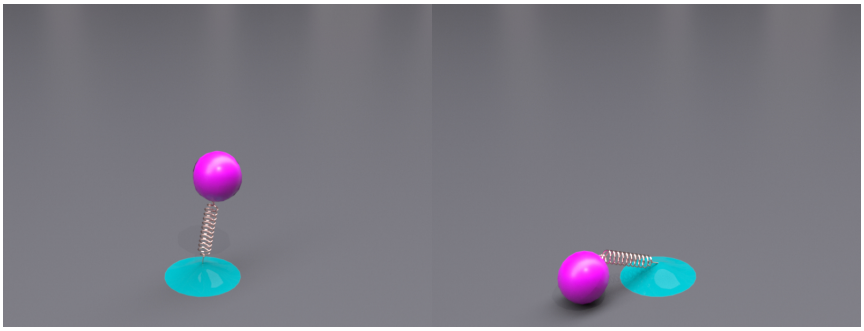


Figure 6.1: **Toy Ball:** A ball-and-spring toy composed of three point-connected parts. The behavior is different with proper point connection energies (left) in comparison with the one without any coupling energies (right). In the latter case, the sphere collapses to the ground.

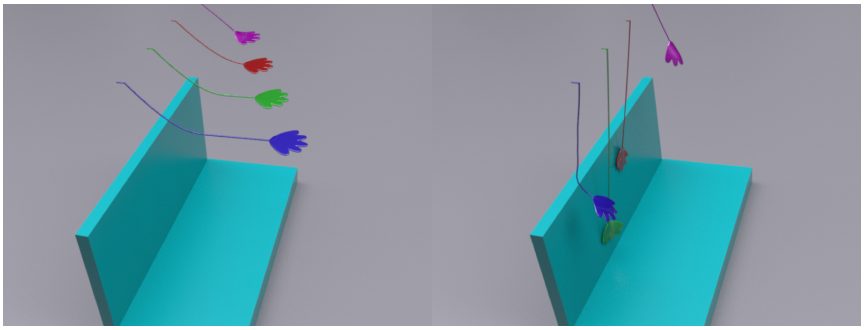


Figure 6.2: **Sticky Hands:** Rubber children's toys modeled as non-manifold meshes comprised of rod and shell components.

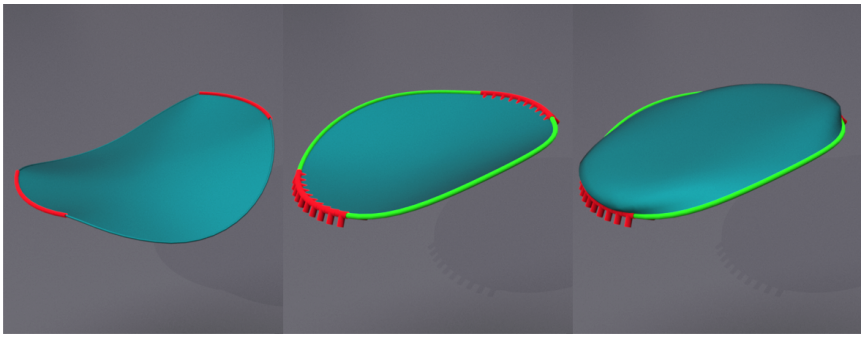


Figure 6.3: **Nylon Frisbee:** A circular frisbee composed of a shell with different rod embeddings along its boundary. We prescribe an axial rotation of two sides of the boundary curve (red), and observe the resulting behavior. Left: Without a rod. Middle: With a rotationally uncoupled embedded rod (green). Right: With a rotationally coupled embedded rod.

Toy Ball In the basic example of Figure 6.1, a single mesh is used to represent three distinct components that have been point-connected together: rod (spring), shell (platform/suction cup), and solid (ball). With properly connecting the three distinct components, we can simulate a springy-ball toy (left), but in the absence of any coupling energies, the sphere collapses to the ground (right).

Sticky Hands The sticky hand toy example of Figure 6.2 shows an elastic object composed of a single material featuring rod-like and shell-like parts joined through stiff point connections.

Jellyfish Our alien jellyfish example (Figure 5.6) consists of components of various dimensions representing the tentacles, eye stalks, eyeballs, and thin body of the jelly fish. To animate it, we prescribe the motion of the body, and allow the various appendages to deform freely.

Nylon Frisbee Our nylon frisbee model (Figure 6.3) is comprised of a shell with stiff rubber rim (rod), to illustrate three distinct possible behaviors designed by decorating a simple circular mesh geometry with different materials and connections. We rotate the red region of the rod, and observe the effect of the rod on the interior cloth. In the absence of the rod, the shell hangs limply (left). With the rod added but not rotationally coupled

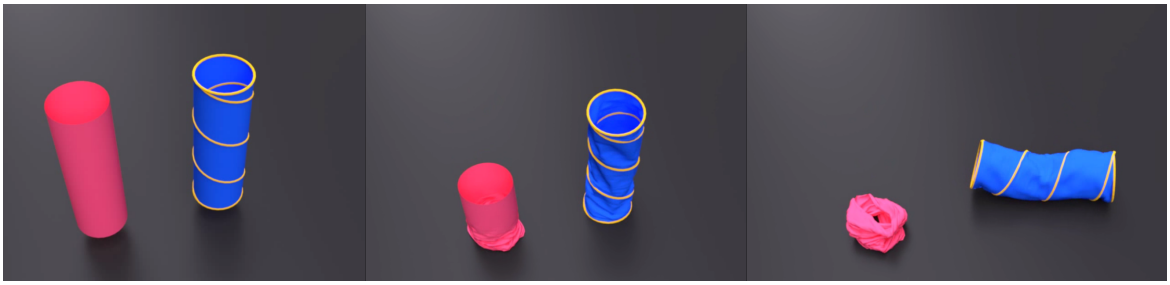


Figure 6.4: **Toy Tunnel:** A falling cylindrical triangle mesh behaves differently with its elements labeled as pure cloth (pink) as compared to cloth with embedded rod segments (blue and yellow).

Figure 6.5: **Umbrella:** An umbrella closed, open, and shown with only the wireframe rod-edges rendered. The rod sub-component of the mesh contains non-manifold triple-junctions.

to the cloth (middle), the rod can freely rotate so that the only effect on the cloth is through the deformation induced along the free section of the rod, that is, it no longer sags unsupported. With our rotational coupling added (right), the rod is instead glued to the cloth such that its rotation induces direct (bulging) deformations on the connected cloth, as expected. Depending on the intended scenario, any of these three may be the desired behavior; thus our method provides significant additional flexibility to the artist or user, while using the exact same underlying mesh.

Toy Tunnel In Figure 6.4 we construct a children's toy tunnel from a cylindrical cloth (shell) mesh by labeling a subset of its existing edges as connected rods in a spiral pattern. Compared to a piece of pure cloth, the version with embedded rods clearly better captures the desired spring-like behavior.

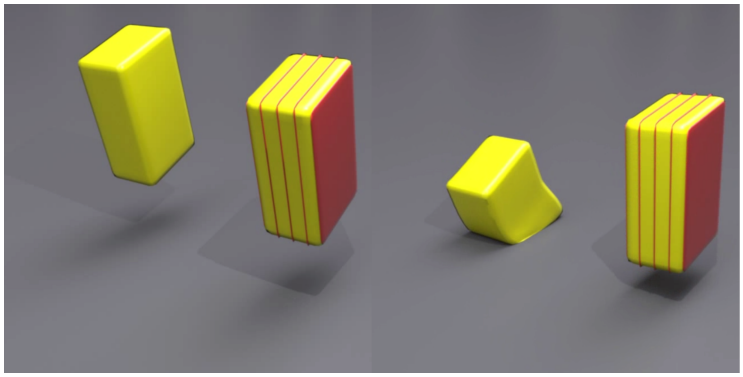


Figure 6.6: **Sandwich composite:** By identifying several layers of faces within the tetrahedral mesh as stiff shells, the two same volumetric solids present totally different behaviors.

Umbrella Figure 6.5 shows a rendered umbrella, and a separate visualization of the rod edges of the umbrella alone. Here we see that in addition to embedded rods through the umbrella's cloth shell, the rod structure itself also contains non-manifold triple-joints, which our framework supports straightforwardly.

Sandwich Composite Figure 6.6 shows the effect of stiff shells embedded within a softer solid volume, by labeling several layers of the tetrahedral mesh's interior triangular faces to be stiff shells.

Anglerfish Our cartoon anglerfish example of Figure 6.7 contains both point connections and curve connections. The body of the anglerfish is composed of 3D tetrahedra, while fins (2D) and an antenna (1D) are connected to the body forming a curve connection and a point connection, respectively. At the end of the antenna, a light-emitting ball (3D) is attached via another point connection.

Continuous Uniform Objects While we focus on more arbitrary elastic connections where the components on either side of the connection may have entirely distinct properties, our framework can also plausibly model smoothly continuous objects consisting of a single material. For example, if a thick solid block gradually narrows to become quite thin, at some point along its length it might be better represented as a thin shell with identical Lamé parameters.

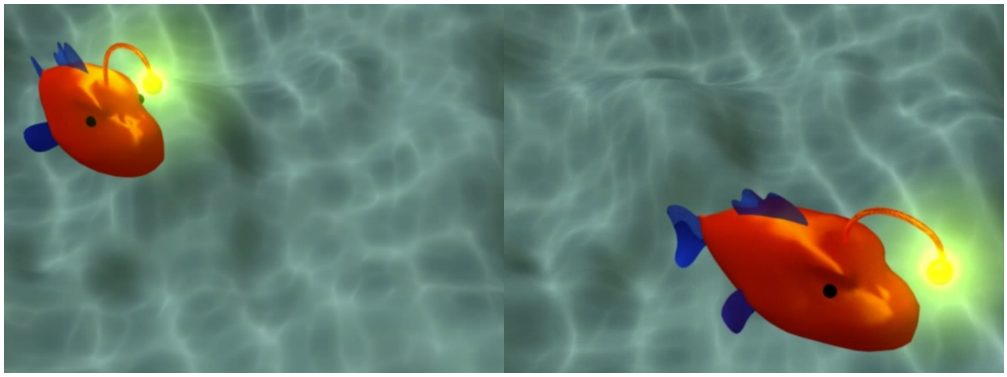
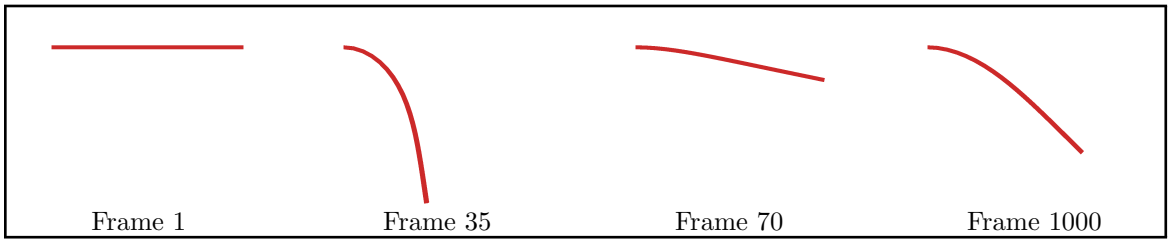
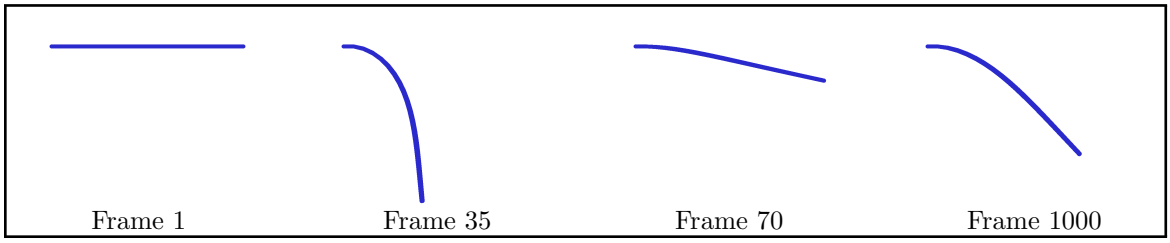


Figure 6.7: **Anglerfish:** The anglerfish model is composed of 3D tetrahedra (body, light ball), 2D triangles (fins), and 1D rods (antenna) connected by both point and curve connections. The head motion is prescribed and the rest of the body's motion is induced by our elastic energies.

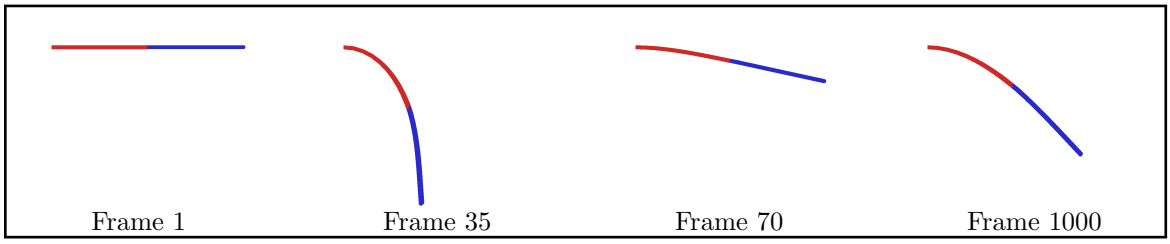
This can be conveniently modeled by ensuring that the connection point/curve has no discontinuous kink in the implied continuous geometry; i.e., the point connection twisting axes or curve connection planes are aligned on both sides. We treat this with a high stiffness connection so that the components' relative configurations remain essentially unchanged (this contrasts with the softer elastic joints with potentially nonzero rest angles used elsewhere). This amalgamates the two incident elements into a smoothly interfacing *super-element*. While this yields a "flat" connection, it is no different than a flat triangle; both their regions of influence naturally shrink under refinement to approximate smooth behavior. (For a coarse mesh, if the resulting "flat" region is deemed too large relative to the surrounding elements, the two elements comprising this super-element could simply be modeled as half the size). In addition to simplicity, an advantage of this approach is that while the material parameters and thickness/diameter of each single-type model must be set to correspond, there is no need to tune the connection's stiffness parameter to precisely match as well; it is simply acting as a strong penalty constraint rather than an elastic energy. Of course, if an elastic object is simulated with models of different dimensions (i.e., a thin solid object can be treated as either a 3D solid model with a small thickness or a 2D shell model), they might not behave perfectly consistently, depending on the choice of the single-type physical models, their discretizations, and/or the level of mesh refinement. However, as long as the two separate models have matching behaviors, connecting them with the method above does not damage the apparent homogeneity of the material. In Figure 6.8 we simulate a thin solid object with a 3D solid model, 2D shell model, and combinations of the two models by connecting the two half-length models with high stiffness.



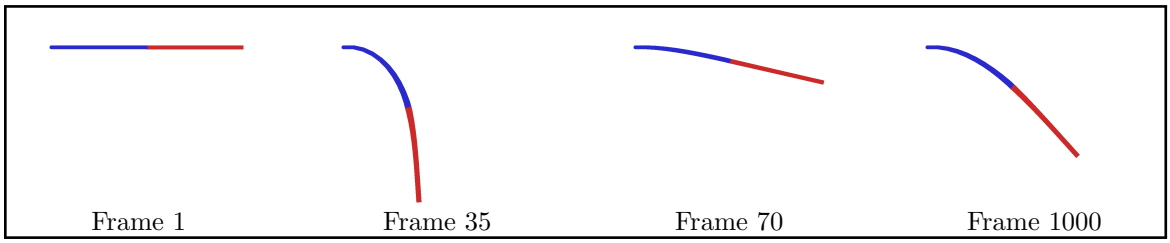
(a) Pure 3D solid model



(b) Pure 2D shell model



(c) Half solid(left)-half shell(right) model



(d) Half shell(left)-half solid(right) model

Figure 6.8: **Comparison between single- and mixed-dimensional models** (red: solid model, blue: shell model). Our method smoothly connects two different-dimensional models of the same material while preserving uniform behavior.

Scene	#vertices	#rod edges	#shell faces	#tetrahedra	#frames	Total time (s)
Toy ball	229	203	96	226	1300	218.301
Sticky hands (purple)	200	19	292	-	1000	115.605
Sticky hands (red)	200	19	292	-	1000	200.095
Sticky hands (green)	200	19	292	-	1000	212.095
Sticky hands (blue)	200	19	292	-	2000	221.073
Jellyfish (single)	1465	1273	-	454	200	528.271
Nylon frisbee	289	64	512	-	700	24.8301
Umbrella	3168	198	6144	-	600	4364.83
Sandwich composite	395	-	1510	600	5000	1311.2

Table 6.1: Simulation Statistics

The observed behaviors are qualitatively indistinguishable.

Part II

Curl-Flow: A Novel Divergence-Free Velocity Interpolation Method in Fluid Animation

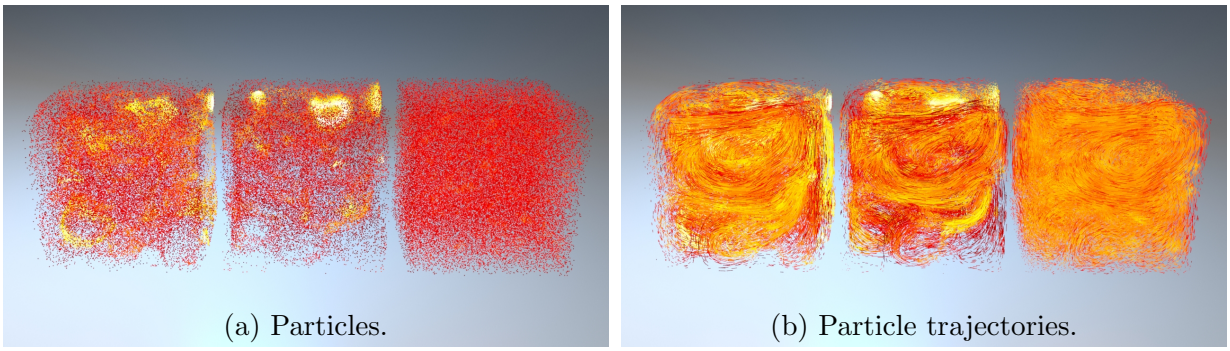
Chapter 7

Introduction

$\mathbf{u}$	velocity vector
$\mathbf{f}$	external forces
$\boldsymbol{\psi}$	vector potential
$\mathbf{H}$	harmonic vector field: curl-free and divergence-free
u, v, w	x-, y-, and z-component of velocity vector
ρ	fluid density
p	pressure
μ	dynamic viscosity
ψ	stream function
h	grid cell width and height (and depth in 3D)
Δt	simulation timestep
τ	viscous shear stress tensor

Table 7.1: Summary of Notations

The assumption of incompressibility is pervasive in computer animation of fluids. Since compressive effects are imperceptible in many (but not all) visually relevant liquid and gas scenarios, neglecting fast-moving compression waves is often justified in practice and yields significant efficiency gains. Mathematically, incompressibility implies that the fluid velocity field $\mathbf{u}$ should be divergence-free: $\nabla \cdot \mathbf{u} = 0$. Popular staggered grid-based schemes rely on this assumption, using finite difference, finite volume, or discrete exterior calculus ideas, combined with Lagrangian or semi-Lagrangian advection methods, to transform the continuous incompressible flow equations into discrete, computable algorithms (Bridson,



(a) Particles. (b) Particle trajectories.

Figure 7.1: **Effect of Interpolants on Particle Distribution in 3D:** The results of identical, initially uniform particle distributions advected through the same $5 \times 5 \times 5$ *discretely* incompressible field for 300 frames using different velocity interpolants. In each trio, left is standard trilinear velocity interpolation, center is monotonic cubic velocity interpolation (Fritsch and Carlson, 1980), and right is our Curl-Flow method. The coloring indicates the per-particle local density estimate, with yellow/white being highest density. Standard velocity interpolants lead to undesirable severe clustering and spreading, while our pointwise incompressible Curl-Flow interpolation preserves close to uniform particle distributions over long time periods.

2015). This fertile mathematical soil has sprouted diverse numerical tools for visual simulation of drifting cigarette smoke, coiling honey, crashing ocean waves, and more (Enright et al., 2002; Fedkiw et al., 2001; Larionov et al., 2017), which are widely integrated into industrial software like *Houdini*, *Cinema4D*, and *Blender*. Yet despite the widespread use of grid-based incompressible flow animation techniques, the pointwise velocity vector fields they provide are not, in the strictest sense, incompressible.

To elucidate this statement, we distinguish *discrete* incompressibility from its *continuous* counterpart. Under a finite volume approach, the discrete velocity components stored at cell face midpoints will indeed satisfy discrete incompressibility: the flux across each cell's boundary sums to zero. However, interpolation is often required to provide *pointwise* velocity values everywhere in the simulation domain, to support popular Lagrangian and semi-Lagrangian discretizations of advection for density, velocity, temperature, passive tracer particles, and so on (Jiang et al., 2015; Stam, 1999; Zhu and Bridson, 2005). Applying basic polynomial interpolants to discretely incompressible grid data affords no guarantee that the interpolated velocity fields will be *pointwise analytically incompressible*, and in practice they are not.

Spurious compressibility has non-negligible implications. Advecting particles through

vector fields with artificial sources and sinks damages volume conservation and causes uniformly distributed particles to clump and spread. Alternatively, reducing discretization error by significantly increasing grid resolution fails to address the root cause and is too costly regardless: simulation time scales cubically or worse with grid resolution. Irregular solid boundaries further exacerbate the issue because standard grid-based interpolants are essentially oblivious to obstacles; obstacle-aware cut-cell interpolants (Azevedo et al., 2016) improve enforcement of the no-normal-flow condition at the cost of worsening compression artifacts.

To guarantee an analytically divergence-free interpolated velocity field by construction, we instead form the velocity field $\mathbf{u}$ from the curl ($\nabla\times$) of a second vector field, $\boldsymbol{\psi}$, known as a *vector potential*:

$$\mathbf{u} = \nabla \times \boldsymbol{\psi}. \tag{7.1}$$

Incompressibility is thus enforced by a basic vector calculus identity,

$$\nabla \cdot \mathbf{u} = \nabla \cdot \nabla \times \boldsymbol{\psi} = 0. \tag{7.2}$$

We exploit this relationship in the discrete and continuous settings to ensure pointwise incompressibility through three steps:

1. Given discretely incompressible fluid velocity values, construct corresponding discrete vector potential values.

2. Interpolate the discrete vector potential values to yield a pointwise vector potential field.

3. Take the analytical curl of the interpolated vector potential field to yield a pointwise incompressible velocity field.

Bao et al. (2017) proposed this three-step recipe in deriving discrete delta functions for the immersed boundary method to support weakly coupled fluid-structure interaction in computational fluid dynamics. However, their method is limited to perfectly uniform rectangular domains (i.e., no irregular boundaries) with periodic boundary conditions. Even so, their potential reconstruction approach requires the solution of a costly vector Poisson problem, making it several times more expensive than pressure projection in a standard fluid solver.

We extend this conceptual framework to be practical for animation applications in two and three dimensions by overcoming several theoretical and algorithmic challenges, including enabling more efficient recovery of discrete vector potentials, enforcing exact

no-flow or prescribed flux boundary conditions for exterior boundaries, and, especially, providing pervasive support for irregular cut-cell solid obstacles. We refer to our approach as the *Curl-Flow* method.

Our divergence-free interpolation method for staggered grid fluid animation includes the following specific technical contributions:

- A more efficient strategy to recover the discrete vector potential field corresponding to a given discrete velocity field, based on a new parallel sweeping technique and a gauge correction approach.
- An additive ramping strategy for improved free-slip boundaries in the "Curl-Noise" method of Bridson et al. (2007).
- Application of this ramping method during interpolation to enforce divergence-free, *exact* no-normal-flow conditions on axis-aligned exterior domain boundaries and irregular (cut-cell) solids in both 2D and 3D.

Chapter 8

Related Work

8.1 Fluid Simulation Methods

Depending on the application, the Navier-Stokes equations have been solved with different primary variables, e.g., stream functions (2D) or vector potentials (3D), velocities, and vorticities, with different representations of fluid, e.g., particles, grids, and meshes. Different combinations form different simulation methods and our method is based on grid structures for fluid representation and velocities (with pressure) as a primary variable, which has been the prevalent choice for fluid animation in graphics.

8.1.1 Grid-Based Eulerian Methods

Techniques for fluid simulation are often categorized into being either Eulerian or Lagrangian. Loosely speaking, Eulerian refers to approaches where the fluid data is stored in a stationary grid or mesh, with quantities at those locations updated to reflect the local fluid properties as it flows past. By contrast, Lagrangian refers to methods where the fluid data is stored on particles or meshes that move dynamically along with the flow. Certain hybrid techniques involve elements of both. Grid-based Eulerian methods are often favoured in fluid simulation in graphics and computational fluid dynamics (CFD) communities, and their easiness in enforcing global incompressibility contributes to their popularity. In particular, a staggered grid (Foster and Metaxas, 1996; Harlow and Welch, 1965) and the operator splitting approach (Bridson et al., 2007; Chorin, 1968; Stam, 1999) have been widely adapted in the graphics community. Using this basic premise with necessary modifications, the early work of Stam (1999) has been steadily enhanced until now:

the methods for solving each part of the Navier-Stokes equations (e.g., pressure projection, advection, viscosity, etc.) have been developed independently or in a combined manner. As our primary focus is obtaining divergence-free velocity interpolation using discretely divergence-free fields, the most relevant parts are advection and projection. We describe numerical techniques for these stages in Chapter 9, and summarize recent related work below.

For advection, the semi-Lagrangian scheme (Stam, 1999) has been commonly used due to its stability and simplicity. However, accumulated interpolation errors during the advection steps cause excessive numerical dissipation. To alleviate the numerical damping, higher order accurate schemes and/or secondary particles have been introduced. Fedkiw et al. (2001) used higher order (cubic) interpolation methods in space and reduced overshooting problems. The overshooting behaviors can be fully eliminated using the method of Fritsch and Carlson (1980). As an alternative to such monotonic cubic approaches, Weighted Essentially Non-Oscillatory (WENO) schemes can also be used (Kim et al., 2013). The back and forth error compensation and correction method (BFECC) of Dupont and Liu (2003) effectively increases the order of accuracy both in time and space to the second order by advecting the solution forward and backward in time to estimate and correct error. The MacCormack scheme (Selle et al., 2008a) obtains the same order of accuracy using a reduced number of advection steps (three to two) thus computationally more attractive. Also, long-term backtracing (Qu et al., 2019; Sato et al., 2018a) can ease the incremental numerical damping caused by repeated interpolation under semi-Lagrangian schemes. Alternatively, secondary particles have been heavily used in graphics. Zhu and Bridson (2005) used a linear combination of the Particle-in-Cell (PIC) method (Harlow and Welch, 1965) and the Fluid-Implicit-Particle (FLIP) method (J.U.Brackbill and H.M.Ruppel, 1986), and their approach has been prevailing for the advection step in fluid simulation. The Zhu-Bridson method has been further improved for better accuracy and less numerical damping by augmenting each particle with locally affine (Ding et al., 2020; Jiang et al., 2015), or polynomial (Fu et al., 2017) approximations to the grid velocity. These approaches all interpolate the velocity directly, and do not consider the presence of absence of divergence.

For pressure projection, the divergent terms in the Helmholtz-Hodge decomposition are projected away to get a divergence-free velocity field by solving a Poisson problem. Starting from the underlying theory, the projection method has improved and branched out for better accuracy, less computational cost, and broader applications. Fluid fractions (in a cell or on a face) were introduced in the solve (Batty et al., 2007; Ng et al., 2009) to better capture the flow with solid obstacles. We employ this approach in our work. These cut-cell methods are further expanded upon to simulate fluid flows with thin obstacles

(Azevedo et al., 2016) and/or thin liquid surfaces (Chen et al., 2020). Different enhanced structures have been introduced for speed-up such as octrees (Aanjaneya et al., 2017; Losasso et al., 2006, 2004; Setaluri et al., 2014) and tiles (Goldade et al., 2020). While each individual component of fluid simulation has been developed and improved within its own methodology, there have also been notable improvements by combining these individual components. Advection-reflection solver (Narain et al., 2019; Zehnder et al., 2018) reduces energy dissipation with minimal modifications (mostly reordering) of the traditional *project and advect* framework, resulting in better preservation of fine level details of smoke. Mullen et al. (2009) used a purely Eulerian advection method solved simultaneously with pressure to preserve the kinetic energy of fluids over long time periods. Similarly, pressure and viscosity steps can be combined: Larionov et al. (2017) solved them together in a single step and reproduced the coiling instability that was not possible with previous grid-based approaches.

Though our method uses velocities (with pressure) as a primary variable of the simulation, stream function (2D) or vector potential (3D) (Hou and Wetton, 2009; Yu and Tian, 2019), and vorticities (Elcott et al., 2007; Mullen et al., 2009) are also valid choices for grid-based methods.

8.1.2 Lagrangian Methods

Smoothed Particle Hydrodynamics (SPH) is a representative Lagrangian method that has rivaled Eulerian methods for many years in graphics. Ever since Müller et al. (2003) introduced SPH to fluid animations, it has been steadily adapted and improved like the grid-based methods. Although particle-based methods have advantages in their intuitive representations (e.g., analogy to molecules), and especially the exact mass conservation during advection, it has limitations regarding solving the physics equations accurately: the pioneering work by Müller et al. (2003) uses repulsion forces to approximate incompressibility, which can cause visible compression or "bouncing" artifacts. Consequently, there has been subsequent work focusing on reducing spurious divergence, e.g., weakly compressible SPH (Becker and Teschner, 2007), predictive-corrective SPH (Solenthaler and Pajarola, 2009), divergence-free SPH (Bender and Koschier, 2015). Akinci et al. (2012) further made thin solid obstacles (i.e., a single layer of particles) possible in the SPH framework, which makes the SPH framework more flexible. The coverage of SPH is expanding just like grid-based methods: we can simulate incompressible elastic solids (Peer et al., 2017), rigid bodies (Gissler et al., 2019), snow (Gissler et al., 2020), and even ferrofluids (Huang et al., 2019) using SPH.

There are other particle-based methods, for example, De Goes et al. (2015) used volumetric parcels that partition the fluid domain using a power diagram to better preserve fluid volume.

One can also use an explicit representation of the fluid surface (e.g., triangle meshes) and move the explicit surfaces over time (Keeler and Bridson, 2015; Pfaff et al., 2012). The explicit surface representation can be advantageous for preserving fine-level details and simulating thin liquid volumes, but *remeshing* is required as meshes often become tangled or ill-shaped (Brochu and Bridson, 2009; Da et al., 2014).

8.2 Divergence-Free Fields

8.2.1 Vector Potentials

Recovering Discrete Vector Potentials

A few vector potential-based Eulerian or hybrid solvers have been proposed in the fluid animation literature, but most operate solely in the discrete realm and require solving a costly *vector* Poisson equation. Elcott et al. (2007) solved a vector Poisson problem to recover the vector potential from vorticity within a simplicial discretization of the vorticity equation, whereas Ando et al. (2015) developed a vector Poisson-based alternative to the standard grid-based pressure projection. Notably, Ando et al. (2015) suggested employing vector potential interpolation as interesting future work. Similarily, Sato et al. also recover the vector potential from a velocity field in fluid control applications (Sato et al., 2021, 2015) by solving the vector Poisson equation. Our proposed vector potential reconstruction strategy offers a significantly more efficient solution, finding the desired potential at essentially the cost of a single scalar Poisson solve if the input velocity field is divergence-free, or two otherwise, even in the presence of cut-cell boundaries.

For 2D flow visualization, Biswas et al. (2016) recovered the discrete (scalar) stream function from velocity data and used marching squares to construct streamlines. They proposed an axis-based sweeping approach for 2D stream function recovery; our proposed sweeping approach generalizes this idea to irregular cut-cell boundaries and three dimensional vector potentials.

More generally, discrete vector field (Hodge) decomposition techniques have long been of interest in graphics (Tong et al., 2003). Recently, more elaborate five-component discrete decompositions were considered by Poelke and Polthier (2016) for piecewise constant vector

fields on 3D triangulated surfaces and by Zhao et al. (2019) for face and edge-based tetrahedral discrete vector fields, including various choices of gauge and boundary conditions. These approaches also determine the potentials directly from the vector field via a vector Poisson solve (and in the five-component case, further require solving eigenproblems).

Gauge Conditions

The methods most similar to ours are those that use explicit vector potentials. Since the vector potential for a given velocity field is not unique, various gauge conditions are used to select a particular potential from this null space. The gauge choice provides one dimension along which to categorize the schemes discussed below. Another is whether they apply a local approach to find the vector potential, based on a cell-by-cell (possibly branching) traversal, or find a simultaneous global solution by solving a linear system. We straddle these categories: our parallel sweeping approach is a fast cell-by-cell scheme that yields an initial potential, which we correct with a relatively inexpensive scalar Poisson solve to ensure smoothness by enforcing the Coulomb gauge, $\nabla \cdot \boldsymbol{\psi} = 0$. Furthermore, beyond our innovations in the uniform grid setting, our work is unique in considering interior solid geometry via cut-cells.

The Coulomb gauge condition, which picks out a unique smooth vector potential by enforcing the potential to be divergence-free, is perhaps the most common gauge and has been used in graphics, (e.g., (Ando et al., 2015)). The implicit gauge condition used in our 3D parallel sweeping method is conceptually similar to that of Ravu et al. (2016), who sought to directly construct spline-based vector potentials. Like our initial vector potential construction, they set the ψ_z component to zero. However, their overall approach differs in that (a) their velocity and vector potential degrees of freedom are colocated at nodes, (b) they directly seek the coefficients of continuous cubic spline functions rather than edge-based discrete vector potential values (hence four times as many unknowns), and (c) compared to our fast parallel sweeping, they solve an expensive global linear system to find the many required coefficients.

Another implicit gauge condition falls out from the tree-cotree approach, explored for edge-based finite element methods by Albanese and Rubinacci (1990) on grids and Manges and Cendes (1995) on meshes. This global approach determines a spanning tree of an edge-based grid or mesh which one can safely set to zero, in order to explicitly eliminate the null space of the matrix representing the discrete curl operator. This idea has natural connections to the traversal patterns of cell-by-cell approaches.

Bao et al. (2017) proposed a divergence-free interpolation strategy on uniform staggered

grids that reconstructs a discrete vector potential field from velocities, to improve coupling and reduce volume conservation errors in Peskin's classic immersed boundary (IB) method (Peskin, 2002). (IB uses smeared delta functions for approximate coupling with solids, rather than the sharp cut-cell approach we advocate.) However, similar to methods in graphics (Ando et al., 2015; Tong et al., 2003), they recover the vector potential under the Coulomb gauge using a vector Poisson system; they also assume periodic boundary conditions, which are undesirable for many graphics applications, highlighting non-periodic domains as important future work. Their restriction to a perfect uniform grid with periodic boundaries decouples the vector problem into three scalar Poisson solves that they treat via the fast Fourier transform (FFT). Our method recovers the same potential with an efficient sweep process and just one scalar Poisson solve, which can likewise be done by FFT in the periodic case. For the actual interpolation on uniform grid regions, they use tensor-product B-spline kernels similar to ours. Casquero et al. (2018) also address the immersed boundary method on uniform grids, but like Evans and Hughes (2013) and in contrast to Bao et al. (2017), they perform the entire simulation using divergence-free B-spline basis functions rather than post-processing staggered grid data.

Silberman et al. (2019) also adopt a staggered uniform grid configuration and seek a discrete edge-based vector potential, with the primary aim of interconverting between electromagnetics solvers based on a vector potential $\mathbf{A}$ and a magnetic field $\mathbf{B} = \nabla \times \mathbf{A}$. They propose a cell-by-cell flooding strategy that is sequential in nature and more costly than ours. Specifically, our axis-based sweeping safely assumes zero values in the z-component; this reduces the computation needed per cell, the total number of unknowns (by one third), and the number of special cases, while enabling a solution based on easily parallelized 1D sweeps. Like us, they use a Poisson-based gauge correction to subsequently enforce the Coulomb gauge, but like Bao et al. (2017) they support only uniform grids and simple boundary conditions to allow solution by FFTs. Silberman et al. (2019) also suggest an alternative "global linear algebra" approach that is loosely similar to the vector Poisson method of Ando et al. (2015) in that it both removes divergence and recovers a potential satisfying the Coulomb gauge; they observe that this method has worse scaling than their flood-then-correct approach.

Pointwise Divergence-Free Fields Using Vector Potentials

The Curl-Noise method (Bridson et al., 2007) uses a continuous vector potential for procedural design of animated divergence-free vector fields. Divergence-free sub-grid turbulence models subsequently built on these ideas (Kim et al., 2008; Schechter and Bridson, 2008). Incidentally, Schechter and Bridson (2008) also used (but did not present or describe) a

divergence-free hybrid constant-linear velocity interpolant (see their supplemental videos at time 2:18-2:27, left), which we show can be derived from bilinear stream function interpolation. This is the *only* prior instance of divergence-free grid-based interpolation in computer animation of which we are aware. Vector potentials have also been used in geometry processing for shape interpolation (Eisenberger et al., 2018).

8.2.2 Dual Stream Functions

Just like the velocity fields obtained by applying curl to the stream functions or vector potentials are divergence-free, there is another similar way to build such divergence-free fields branching from the same root: the cross product of two gradient functions is divergence-free by construction. This can be simply verified: for scalar functions λ, μ and ϕ, if we define the vector potential as $\boldsymbol{\psi} = \lambda\nabla\mu - \nabla\phi$, then the corresponding velocity becomes $\mathbf{u} = \nabla \times \boldsymbol{\psi} = \nabla\lambda \times \nabla\mu$. This approach is often called *dual stream functions* (Frewer et al., 2014; Li and Mallinson, 2006). In fluid animation, DeWolf (2006) used the dual stream function approach to generate divergence-free noise for turbulence, and in geometry processing, Von Funck et al. (2006) used it for volume-preserving deformation for use in 3D modeling of shapes.

8.2.3 Vortex Methods

Like our work, Lagrangian vorticity-based simulation methods also employ a secondary vector variable to construct analytically divergence-free velocity fields; specifically, velocity is computed from vorticity via the Biot-Savart law. These methods come in many forms, including Lagrangian particles (Park and Kim, 2005), filaments (Angelidis and Neyret, 2005), sheets (Brochu et al., 2012; Pfaff et al., 2012), and bubbles (Da et al., 2015); closely related boundary integral/element (surface-only) methods similarly offer divergence-free fields by construction (Da et al., 2016). Model-reduced fluid simulation methods based on Laplacian eigenfunctions can offer pointwise divergence-free fields by using analytical basis functions (Cui et al., 2018; De Witt et al., 2012), albeit in restricted domains (e.g., boxes).

8.2.4 Matrix-Valued Radial Basis Functions

An alternative divergence-free interpolation strategy for mesh-free point data is provided by McNally (2011) and Lowitzsch (2005), who exploit matrix-valued radial basis functions (RBF) and properties of the vector Laplacian. They apply the double-curl operator

$(\nabla\nabla^T - \nabla^2\mathbf{I})$ to a scalar RBF making the interpolant generate smooth and divergence-free fields. Note that the matrix-valued RBF implicitly satisfies the Coulomb gauge condition. Although this mesh-free method offers divergence-free fields, it struggles to enforce boundaries precisely and, more critically, we found it to be far more costly, since it requires a global solve to simultaneously determine coefficients for all of the radial basis functions. Using global kernels induces a fully dense system, and substituting local kernels also has limitations: a sufficient support radius is needed to avoid local oscillations but also makes the system matrix unattractively dense. The matrix-valued RBF also requires colocated vector velocity data and each sample has three DOFs which can further scale up the size of the system matrix.

8.2.5 Divergence-Free Finite Element Methods

In applied mathematics, a wide range of discontinuous Galerkin (DG) and other non-conforming finite element schemes have been developed to offer pointwise divergence-free fields, often by adopting carefully designed incompressible basis functions on each element (e.g., (Cockburn et al., 2004; Lehrenfeld and Schöberl, 2016; Rhebergen and Wells, 2018)). However, such methods tend to have a more complex implementation and do not naturally integrate with standard approaches in fluid animation; moreover, they possess pervasive field discontinuities at element borders by their nature, which can be problematic for visual applications. Guzmán and Neilan (2014) tackled the rather more challenging task of designing a fully conforming/continuous Stokes finite element method yielding divergence-free solutions on 3D tetrahedra. Doing so required a specialized finite element basis consisting of a combination of cubic polynomials and divergence-free rational functions, and due to the construction's complexity the authors note that "practical significance of the proposed elements may be questionable". In isogeometric analysis, Evans and Hughes (2013) developed an exactly divergence-free Navier-Stokes simulation framework based on geometrically mapped rectangular B-spline grids; approaches in this vein require complex mesh construction for non-trivial domains, in contrast to the simpler and more efficient cut-cell techniques often preferred in animation. In fact, all of the approaches above would necessitate replacing the entire Navier-Stokes simulator, while ours provides a convenient plug-in upgrade to the advection phase of industry-standard visual effects methods. Thus, somewhat closer to our approach are methods that post-process flow solver solutions to exactly recover a divergence-free velocity field. For example, Linke (2012) converted the solution of a staggered triangulated discretization into a strictly divergence-free field in terms of Raviart-Thomas elements. Lederer et al. (2017) proposed a velocity reconstruction operator that maps discretely divergence-free fields of Taylor-Hood and mini elements

to exactly divergence-free ones, and suggested using their method as a postprocessing of the discrete solution in the conclusion. However, like the DG methods discussed above, the resulting finite element spaces (i.e., velocity fields) are not continuous between elements.

8.2.6 Direct Interpolation of Finite Volume Solutions

In a computational fluid dynamics context, Jenny and colleagues (Jenny et al., 2001; Meyer and Jenny, 2004) derived 2D divergence-free node-based velocity interpolants for uniform grids and demonstrated that these offer improved particle distributions in particle-in-cell schemes. In geodynamics, Wang et al. (2015) developed the 3D extension, and Pusok et al. (2017) adapted it to face-based data by first averaging to the nodes. These approaches augment multilinear interpolants with correction terms that upgrade them to satisfy pointwise incompressibility. In astrophysics (magnetohydrodynamics), Balsara (2001, 2004) proposed similar divergence-free vector field reconstruction strategies based on local piecewise quadratic fitting on each cell or tetrahedron. In this strategy, one presupposes a polynomial basis and uses the divergence-free condition to determine constraints on it. Nearby vector field values or derivatives are used to solve for the coefficients. Balsara (2009) later presented an extension of this approach to Cartesian grid WENO schemes. These approaches do not consider cut-cell geometries, and exhibit kinks between cells.

The approaches above do not require recovering a vector potential at all. On the other hand, the recovered fields exhibit pervasive discontinuities between cells, and to achieve a given order of accuracy or support a particular geometry, they must be carefully designed on a case-by-case basis. Our choice to instead recover and exploit an explicit discrete vector potential makes it comparatively straightforward to derive generalizations to higher order accuracy, global continuity/smoothness, or different element shapes, using *standard* interpolation techniques, including spline interpolants and mesh-free methods. Having the vector potential available can also enable useful fluid control tools (Sato et al., 2021, 2015).

8.2.7 Subdivision Schemes

Subdivision schemes based on discrete exterior calculus consider how to preserve key properties of discrete differential operators on triangulated surface meshes undergoing mesh refinement (De Goes et al., 2016; Wang et al., 2006). These papers focus on surfaces and do not consider volumetric uniform grids or polyhedral cut-cells. More fundamentally, they achieve pointwise incompressibility only in the limit of infinite refinement. It would be interesting to explore whether these ideas can be adapted to general polyhedra.

Chapter 9

Preliminaries

In this chapter, we briefly review grid-based fluid simulation methods which our work is based on. Since our method specifically focuses on the advection step in grid-based methods, and assumes the input velocity fields are discretely divergence-free (which is typically achieved in the pressure projection step), the advection and projection steps are mainly addressed. Also, our method leverages secondary variables called stream functions in 2D, or vector potentials in 3D, for pointwise divergence-free interpolation, so we review the usage of discrete and continuous vector potentials in graphics.

9.1 The Equations of Fluids

The incompressible Navier-Stokes equations have the form

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -\mathbf{u}\cdot\nabla\mathbf{u} + \frac{\mathbf{f}}{\rho} - \frac{\nabla p}{\rho} + \frac{\nabla\cdot\tau}{\rho}, \\ \nabla\cdot\mathbf{u} &= 0, \\ \tau &= \mu\big(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\big), \end{aligned} \tag{9.1}$$

where $\mathbf{u}$ is velocity, $\mathbf{f}$ is external forces, p is pressure, τ is the viscous shear stress tensor, ρ is fluid density, and μ is dynamic viscosity. Given the non-linear characteristics of the advection term ($-\mathbf{u}\cdot\nabla\mathbf{u}$), each term in (9.1) is often solved independently (Bridson, 2015; Chorin, 1968; Stam, 1999): applying external forces, solving for pressure and viscosity in one step (Larionov et al., 2017) or in different steps (Batty and Bridson, 2008), and advecting fluid as shown in Figure 9.1.

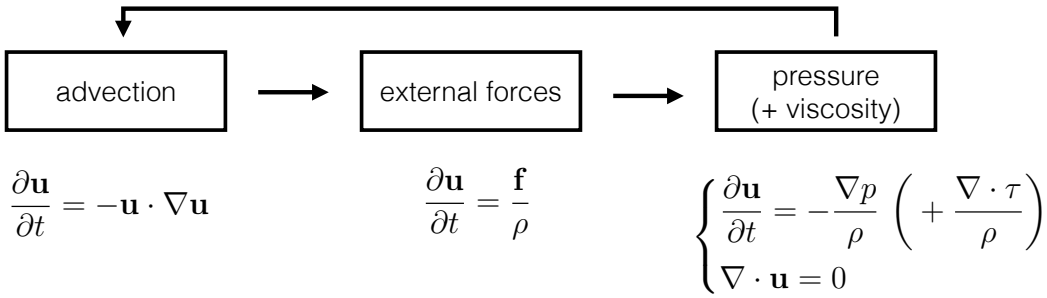


Figure 9.1: **Operator splitting approach to the incompressible Navier-Stokes equations.**

The Navier-Stokes equations without the viscosity term are called the Euler equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u} \cdot \nabla \mathbf{u} + \frac{\mathbf{f}}{\rho} - \frac{\nabla p}{\rho} \tag{9.2}$$

and we use the Euler equations throughout the thesis for simplicity, but adding viscosity or additional forces does not affect our method as they are solved in a separate step. This leaves us with just two steps, discussed further below: advection, which transports the fluid velocity along with the flow itself, and pressure projection, so called because it projects the velocities to be incompressible.

In Algorithm 1, the pseudocode for simulating such inviscid fluids is provided. To use our Curl-Flow method, one needs to replace previous particle advection approaches (line 8) to our method (line 5, 6). Our Curl-Flow method can also be applied to velocity advection (line 10), but we only use it for advecting passive particles to isolate the effects of interpolation for comparison (Section 10, second paragraph). This algorithm can be further extended to incorporate other grid quantities (except grid velocity), viscous fluids, etc.: just like in Algorithm 1, only the advection steps differ from existing methods.

9.2 Pressure Projection

Using the Helmholtz-Hodge decomposition, a vector field can be decomposed into three terms,

$$\mathbf{u} = \nabla \theta + \nabla \times \boldsymbol{\psi} + \mathbf{H}. \tag{9.3}$$

Algorithm 1: Pseudocode for Simulating Inviscid Fluids.

```
for each timestep do
    apply external forces                          ▷ gravity, buoyancy, etc.
    perform pressure projection                    ▷ enforce discrete incompressiblilty
    if using Curl-Flow                             ▷ using our Curl-Flow method
        build desired interpolant
        advect particles with our interpolant
    else                                           ▷ using previous methods
        advect particles using direct velocity interpolantion
    end
    advect velocity (semi-Lagrangian)
end
```

where the first term ($\nabla\theta$) is curl-free (i.e., zero curl), the second term ($\nabla\times\boldsymbol{\psi}$) is divergence-free (i.e., zero divergence), and the third term ($\mathbf{H}$) is harmonic which is both curl-free and divergence-free. There are two complementary ways to obtain divergence-free fields:

- find the divergence-free components in (9.3) directly, or
- find the divergent component first (i.e., $\nabla\theta$), and subtract the divergent field from the input field, i.e., $\mathbf{u}_{div-free} = \mathbf{u} - \nabla\theta$.

They may sound similar but the system of equations and variables are completely different in the discrete setting. The latter is used in this section, while the former is addressed in the following section (Section 9.3). Considering only the velocity change due to pressure, we have

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\nabla p}{\rho}, \tag{9.4}$$

and if we discretize (9.4) in time using forward Euler, we obtain

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t \frac{1}{\rho}\nabla p. \tag{9.5}$$

This indeed has the shape of subtracting the divergent part in (9.3) from the input field $\mathbf{u}^n$, where $\left(\Delta t\frac{1}{\rho}\right)p$ corresponds to θ. Intuitively, the pressure gradient, ∇p, acts as a force to

enforce the divergence-free condition ($\nabla \cdot \mathbf{u}^{n+1} = 0$). To find such p (or θ), the divergence-free condition is applied to (9.5):

$$\frac{\Delta t}{\rho} \nabla \cdot \nabla p = \nabla \cdot \mathbf{u}^n \tag{9.6}$$

which forms a *scalar Poisson* equation. For simplicity, we first consider the spatial discretization for a uniform Cartesian grid in 2D. We discretize the Poisson equation (9.5) in space using centered finite differences with the sample locations in Figure 9.2, left:

$$\begin{aligned} \frac{\Delta t}{\rho h^2}(p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}) = \\ \frac{1}{h}(u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j} + v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}). \end{aligned} \tag{9.7}$$

Here, h is the width of each grid cell, which we assume to be a square (a cube in 3D), Δt is a simulation timestep, ρ is the density of the fluid, and p, u, and v are the samples of pressure, x- and y-component of velocity, respectively. The subscripted indices indicate the row/column positions within the grid; integer indices correspond to cell centers, with half-integer indices used for face locations. In 3D, we have different sample locations but used the same subscripted index convention (Figure 9.3, left). In this way, the discretization in 2D naturally extends to 3D with additional modifications due to z-directional components.

$$\begin{aligned} \frac{\Delta t}{\rho h^2}(p_{i+1,j,k} + p_{i-1,j,k} + p_{i,j+1,k} + p_{i,j-1,k} + p_{i,j,k+1} + p_{i,j,k-1} - 6p_{i,j,k}) = \\ \frac{1}{h}(u_{i+\frac{1}{2},j,k} - u_{i-\frac{1}{2},j,k} + v_{i,j+\frac{1}{2},k} - v_{i,j-\frac{1}{2},k} + w_{i,j,k+\frac{1}{2}} - w_{i,j,k-\frac{1}{2}}) \end{aligned} \tag{9.8}$$

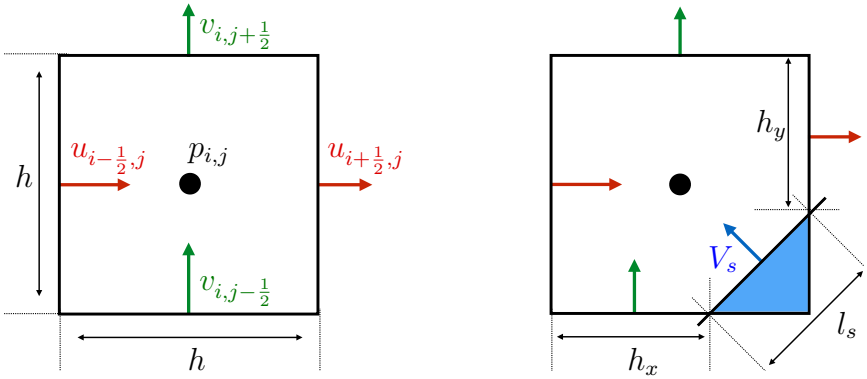


Figure 9.2: **Fluid Discretization in 2D.** Left: pure fluid cell. Right: fluid cell partially filled by a solid obstacle (blue). V_s refers to the velocity flux on the solid boundary.

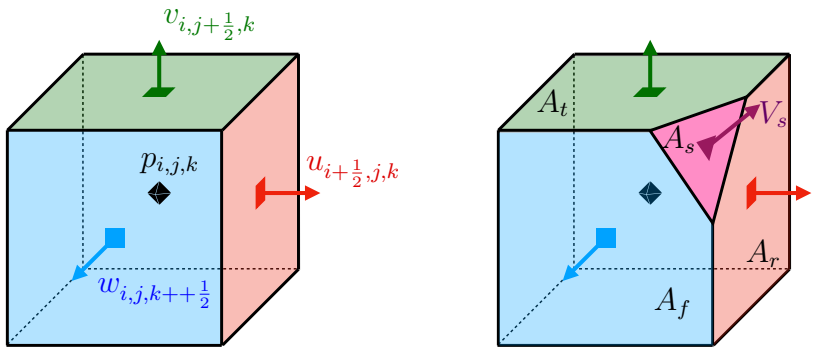


Figure 9.3: **Fluid Discretization in 3D.** Left: pure fluid cell. Right: fluid cell clipped against a solid obstacle. A_r, A_t, A_f, and A_s are the area of the right, top, front, and solid face, respectively. V_s refers to the velocity flux on the solid boundary.

9.2.1 Boundary Conditions

Ignoring two-way interactions between fluids and dynamically simulated solids, which is outside of the scope of our work, there are two typical boundary conditions depending on what forms the interface.

$$\begin{cases} \mathbf{u}^{n+1} \cdot \mathbf{n} = \mathbf{u}_{solid} \cdot \mathbf{n}, & \text{fluid-solid boundary} \\ p = 0, & \text{free surface} \end{cases}$$

As our primary target of simulation is smoke, we omit the free surface condition that is imposed at the interface between a liquid and air. The solid boundary condition describes the free-slip condition: the fluid velocity component in the normal direction ($\mathbf{u}^{n+1} \cdot \mathbf{n}$) is forced to match that of the solid obstacle ($\mathbf{u}_{solid} \cdot \mathbf{n}$) while fluid can *freely slip* in the tangential direction. As a side note, enforcing this condition on the pressure projection implies fluids at the interface do not penetrate into or cleave solid obstacles, but *only in the discrete setting*. When using pointwise velocities (e.g., by interpolating discrete velocity samples), this can easily be violated: fluid particles easily pierce solid obstacles thus extra treatments are required. As a simple remedy, the penetrated particles are often pushed back to the surface, but we highlight a potential problem of this simple approach and propose an improved solution (Section 11.2.3, 12.3.2).

For the cells where solid obstacles cut through, or *cut-cells* (Figure 9.2, right), the Poisson equation (9.7) requires additional modifications to incorporate the influence of the solid. The solid geometry is obtained from marching cubes (marching squares in 2D) of

a solid level set. We use the divergence theorem to effectively enforce the incompressible condition in the presence of solid obstacles:

$$\iint_A \nabla \cdot \mathbf{u} \; dA = \int_C \mathbf{u} \cdot d\mathbf{n}$$

where A is a region in the plane whose boundary is a closed curve C. Using the divergence theorem, the discrete divergence in the example of Figure 9.2, right, becomes

$$[\nabla \cdot \mathbf{u}]_{(i,j)} = \frac{1}{h^2}\Big(- h_x v_{i,j-\frac{1}{2}} - h u_{i-\frac{1}{2},j} + h v_{i,j+\frac{1}{2}} + h_y u_{i+\frac{1}{2},j} - l_s V_s \Big). \tag{9.9}$$

Thus for the cut-cells, the right hand side of the Poisson equation (9.6) can be obtained by plugging $\mathbf{u}^n$ into (9.9). The left hand side can also be computed in a similar manner by putting ∇p in place of $\mathbf{u}$ in (9.6) with necessary modifications:

$$[\nabla \cdot \nabla \mathbf{p}]_{(i,j)} = \frac{1}{h^2}\Big(- h_x \frac{p_{i,j} - p_{i,j-1}}{h} - (p_{i,j} - p_{i-1,j}) + (p_{i,j+1} - p_{i,j}) + h_y \frac{p_{i+1,j} - p_{i,j}}{h} \Big).$$

This coincides with the finite volume approach of Ng et al. (2009). This approach naturally extends to 3D; we can simply use the 3D version of the divergence theorem,

$$\iiint_V \nabla \cdot \mathbf{u} \; dV = \iint_S \mathbf{u} \cdot d\mathbf{S}$$

where V is a solid region and S is the boundary surface of V. Like the 2D case, the discrete divergence in the case of Figure 9.3, right, can be defined as

$$\begin{aligned}[\nabla \cdot \mathbf{u}]_{(i,j,k)} = \frac{1}{h^3}\Big(& - h^2 u_{i-\frac{1}{2},j,k} - h^2 v_{i,j-\frac{1}{2},k} - h^2 w_{i,j,k-\frac{1}{2}} \\ & + A_r u_{i+\frac{1}{2},j,k} + A_t v_{i,j+\frac{1}{2},k} + A_f w_{i,j,k+\frac{1}{2}} + A_s V_s \Big)\end{aligned} \tag{9.10}$$

where A_r, A_t, A_f, and A_s refer to the area of right, top, front, and solid face, respectively, and V_s is the normal flux on the solid face. Similarly to the 2D case, we can discretize the Poisson equation (9.6) for cut-cells using (9.10). The expressions for the cut-cells can be viewed as a generalization for the regular cell cases: they roll back to the equations for the regular cells in the absence of solid obstacles. Having solved the resulting linear system for pressure, we determine the new discretely incompressible velocity field by evaluating (9.5), $\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t \frac{1}{\rho} \nabla p$. Note that when we refer to a vector field as "discretely incompressible" or "discretely divergence-free", it means that (9.9) or (9.10) evaluates to zero.

9.3 Vector Potentials

As alluded to in the previous section (Section 9.2), there are different ways of enforcing discrete incompressibility. In this section, we use a different approach from the previous section: we find divergence-free fields directly using a variable called a vector potential (or $\boldsymbol{\psi}$) in 3D, or a stream function (or ψ) in 2D, instead of finding the pressure values and subtracting the pressure gradients from the original field. Since we assume a simply connected closed domain with static boundaries, the harmonic component in (9.3) will be zero, thus a vector field can be decomposed as

$$\mathbf{u} = \nabla\theta + \nabla \times \boldsymbol{\psi}. \tag{9.11}$$

(If a nonzero harmonic field is required, e.g., a high-genus geometry, one needs to precompute and reinject the harmonic part (Elcott et al., 2007).) Now the complementary relation between the pressure (or θ) and the vector potential is more obvious, and the divergence-free vector field can be written as $\mathbf{u}_{div-free} = \nabla \times \boldsymbol{\psi}$. To find such $\boldsymbol{\psi}$, we apply the curl operator to (9.11):

$$\nabla \times (\nabla \times \boldsymbol{\psi}) = \nabla(\nabla \cdot \boldsymbol{\psi}) - \nabla^2\boldsymbol{\psi} = \nabla \times \mathbf{u}. \tag{9.12}$$

Since the solution $\boldsymbol{\psi}$ is the inverse curl of $\mathbf{u}_{div-free}$, there exists an infinite number of solutions, thus additional gauge conditions are required to uniquely define $\boldsymbol{\psi}$. Perhaps the most popular choice for the gauge is the *Coulomb gauge*, $\nabla \cdot \boldsymbol{\psi} = 0$, which gives a smooth $\boldsymbol{\psi}$ field. With this condition, (9.12) is further simplified to a *vector* Poisson problem,

$$\nabla^2\boldsymbol{\psi} = -\nabla \times \mathbf{u}. \tag{9.13}$$

Note that in (9.13), whether the input field $\mathbf{u}$ is divergence-free is not important; the divergent term in (9.11) vanishes with the curl operator.

Ando et al. (2012) arrived at the same results by solving a minimization problem

$$\underset{\boldsymbol{\psi}}{\text{argmin}} \qquad \int_\Omega \frac{1}{2}||\mathbf{u} - \nabla \times \boldsymbol{\psi}||^2 dV + \frac{1}{2}(\nabla \cdot \boldsymbol{\psi})^2 dV, \tag{9.14}$$

where we assume a uniform density of fluid for simplicity. The first term is about finding $\boldsymbol{\psi}$ where $\nabla \times \boldsymbol{\psi}$ is as close to $\mathbf{u}$ as possible. The second term is a regularizer to remove the null space in the first term. The regularizer encodes the Coulomb gauge condition. If we discretize (9.14), take the derivative with respect to $\boldsymbol{\psi}$, and set it to zero, we obtain a linear system

$$([\nabla\times]^T[\nabla\times] + [\nabla\cdot]^T[\nabla\cdot])[\boldsymbol{\psi}] = [\nabla]^2[\boldsymbol{\psi}] = [\nabla\times]^T[\mathbf{u}]$$

where $[\cdot]$ means discrete terms (a matrix or a vector). We assume the grid cells are pure fluid cells (i.e., smoke in regular cells) for simplicity, but we can easily adapt this method to consider volume fractions of the fluid in each cell. In general, the vector Laplacian of $\nabla^2\boldsymbol{\psi}$ in Cartesian coordinates reduces to the much simpler form:

$$\nabla^2\boldsymbol{\psi} = (\nabla^2\psi_x, \nabla^2\psi_y, \nabla^2\psi_z) \tag{9.15}$$

which enables solving for each component of $\boldsymbol{\psi}$ independently unless other constraints (e.g., boundary conditions) tie them together.

9.3.1 Applications of Vector Potentials in Graphics

In this section we briefly review two closely related papers in graphics that make use of either discrete or continuous vector potentials.

Liquid Simulations Using Discrete Vector Potentials

Ando et al. (2015) used a *discrete* vector potential method to simulate liquids with bubbles. With the same method, one can simulate incompressible smoke also, but when it is applied to liquids it yields a special effect: despite solving only for variables on or inside the liquid volume, the volume of the ambient air or bubbles are automatically preserved. The volume conservation comes from having $\boldsymbol{\psi}$ samples at the surface of the bubbles or (closed) air regions: the net flux of the encapsulated bubbles are enforced to be zero (Figure 9.4). The $\boldsymbol{\psi}$ samples at the surface effectively capture the volume-preserving motions of the bubbles such as the glugging effect of liquid flowing through the neck of a bottle or watercooler. Although the pressure-based methods do not give such effects when naively used, the use of a (variation of) vector Poisson solve makes this method less attractive due to its computational cost. Furthermore the vector Poisson solve cannot be decomposed into three independent scalar Poisson solves due to their boundary conditions which makes the size of the linear system approximately three times larger than a scalar Poisson solve.

Procedural Design of Turbulent Fluid Using Continuous Vector Potentials

Bridson et al. (2007) used a *continuous* vector potential for procedural design (as opposed to physical simulation) of random and divergence-free vector fields, which is referred to as the Curl-Noise method. Vector potentials are a great ingredient to build divergence-free

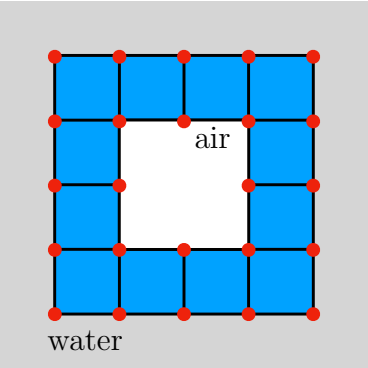


Figure 9.4: **Air cells (white) surrounded by water cells (blue).** The red circles at the nodes represent ψ samples in 2D. Even though the interior nodes in the air do not have the degrees of freedom of ψ, the ψ values on the interface ensures the total volume of the encapsulated air cells is preserved.

velocity fields but they are relatively unwieldy to control in comparison with directly controlling velocities. Therefore, Bridson et al. further provided ways to process or prescribe velocity fields using vector potentials to achieve specific behaviors such as obtaining a rigid body motion and making velocity fields not penetrate solid obstacles. Our exact solid boundary enforcement (Section 11.2.3, 12.3.2) stems from their ramping strategies toward a solid obstacle, where the normal velocity is enforced to be zero on the solid surface. Taking the 2D case as an example, the velocity recovered from the curl of ψ is

$$\mathbf{u} = \left(\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right) = \nabla \psi^{\perp} \tag{9.16}$$

where $(\cdot)^{\perp}$ means a 90° rotation. Since the normal velocity is determined by $\nabla\psi$ in the tangential direction, they proposed modulating ψ with a ramp to a constant value of zero along the solid boundary (Figure 9.5(a)),

$$\psi'(\mathbf{x}) = \alpha(\mathbf{x})\psi(\mathbf{x}), \tag{9.17}$$

where α is a smooth function, $\alpha = \text{ramp}(d(\mathbf{x})/d_0)$, d_0 is an influence radius, and $d(\cdot)$ is the distance to the surface. We adopt Bridson's ramp function

$$\text{ramp}(r) = \begin{cases} 1, & \text{if } r \geq 1 \\ \frac{15}{8}r - \frac{10}{8}r^3 + \frac{3}{8}r^5, & \text{if } -1 < r < 1 \\ -1, & \text{if } r \leq -1 \end{cases}$$

and use the width of a grid cell for d_0 in the examples of Figure 9.5. Although this approach effectively prevents boundary-penetrating velocity fields, it has two potential problems: first, setting the target ψ value blindly to zero on all boundaries can distort the velocity field. In Figure 9.5, the value of ψ along the left and right boundaries varies from 0 to 0.5 (bottom to top) to simulate a horizontal flow. Applying ramping on top of this field, to force ψ to match the target value of zero on the boundary, can detrimentally change the flow in both direction and magnitude (Figure 9.5(b)). Assuming we know the desired target value ψ_t (see Section 11.2.2), we can modify the ramping function (9.17) as

$$\psi'(\mathbf{x}) = \alpha(\mathbf{x})\psi(\mathbf{x}) + (1 - \alpha(\mathbf{x}))\psi_t(\mathbf{x}), \tag{9.18}$$

to ramp to ψ_t instead of zero. Knowing the function for the target ψ_t, the major source of the velocity distortions goes away. Taking a closer look, one can notice the ramping function (9.18) modifies the value of ψ by multiplying by a smooth function α (followed by adding an additional term if ψ_t is not zero). This ramping strategy effectively constrains the normal velocity on the solid surface, but at the same time, it steepens and flattens the curve of ψ in the normal direction, leaving unnatural patterns of velocity magnitude toward the solid obstacle. Figure 9.5(c) is the plot of the velocity magnitude where yellow represents large values in magnitude while red means zero magnitude. In this figure, the ramping pattern near the solid or exterior boundary is pronounced: see the bands of yellow near the boundaries followed by red regions. The unramped velocity fields in (d) do not display such patterns, which proves that the unnatural patterns are from the ramping scheme (although the unramped fields also do not fully respect the boundary). The same problems also arise in 3D. We introduce improved ramping strategies for both 2D and 3D in Sections 11.2.3, 12.3.

9.4 Advection

The advection part in the Euler equations (9.2) is

$$\frac{\partial q}{\partial t} + \mathbf{u} \cdot \nabla q = 0, \tag{9.19}$$

where $\mathbf{u}$ is a *divergence-free* velocity field and q is any scalar (e.g., density) or vector (e.g., velocity) quantity to be advected. For a typical smoke simulation, there are two quantities to be advected: the velocity field itself, as part of the Euler or Navier-Stokes equations, and the smoke/soot representation used for visualization, typically either particles or a scalar density field. The need for divergence-free velocity fields is obvious; we would not want to

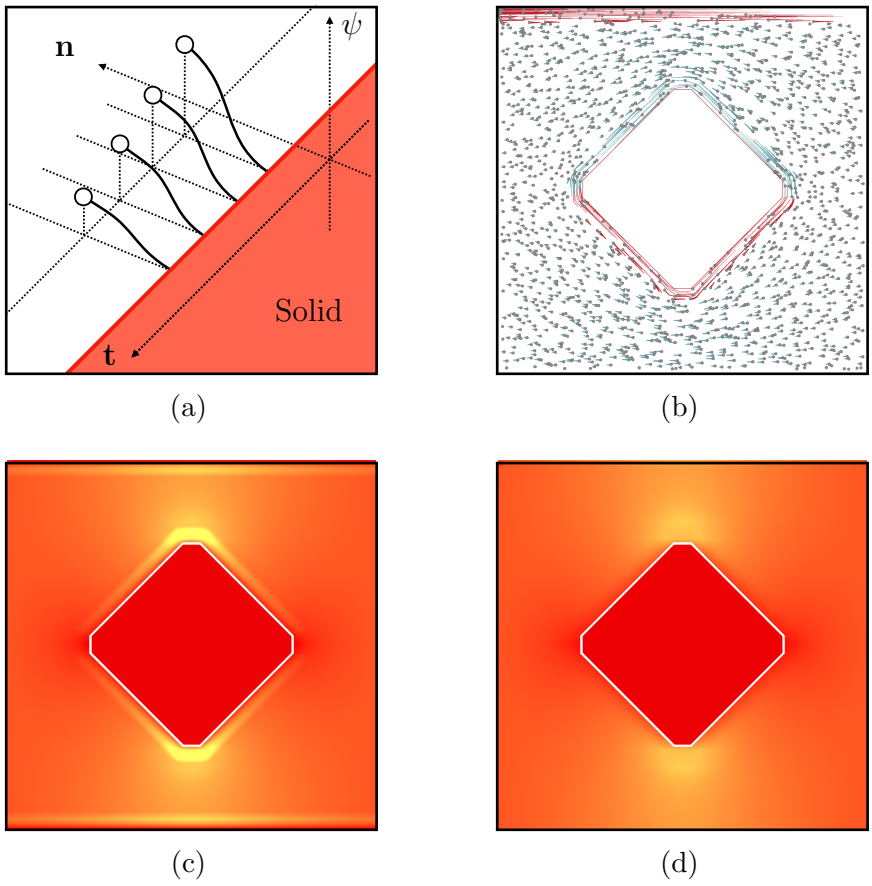


Figure 9.5: **Curl-Noise Ramping:** (a) Curl-Noise ramping enforces the value of ψ to be zero on the solid boundary. The ramping effectively enforces zero gradient in the tangential direction, but the curve of ψ in the normal direction is *flattened* at the ends and *steepened* in the middle. (b) Original Curl-Noise ramping with the target value of zero applied to a steady rightward flow. Red trails represent flow in the reverse (-x) direction. The constant inflow/outflow rate is described by a vertical linear stream function value from 0 to 0.5 from bottom to top. Naively ramping to zero distorts the direction and magnitude of the flow although the velocity field remains boundary-respecting. (c) The velocity magnitude plot with a correct (nonzero) target ψ_t from a solver. The magnitude increases from red to yellow. Modifying the target value to correct one in this way removes the artifacts shown in (b). However, there remains a pronounced unnatural pattern in $\nabla\psi$ (or velocity) near the solid and exterior domain boundary. (d) The same velocity magnitude plot but without ramping. This comparison indicates the spurious patterns in (c) are from the ramping scheme.

gain or lose the volume (or other quantities) of the fluid during advection for incompressible fluids. Thus, when using a Chorin-style advection-projection scheme (Bridson et al., 2007; Chorin, 1968; Stam, 1999), it is often preferred to place advection immediately after the projection step, so the quantities can flow under a divergence-free velocity field. Semi-Lagrangian type schemes have been particularly popular for advection as they provide unconditional stability unlike explicit Eulerian time integration schemes (e.g., forward Euler or upwinding schemes). To update the grid quantity q_G at the grid position $\mathbf{x}_G$ during the advection step, we first *backtrace* the trajectory from $\mathbf{x}_G$ by one timestep to find the hypothetical departure position $\mathbf{x}_D$ that lands at $\mathbf{x}_G$ according to the given velocity field. Just like a Lagrangian approach, we assume the quantity does not change during the advection step, so we can simply copy the quantity at $\mathbf{x}_D$ (q_D^n) to the quantity at the grid $\mathbf{x}_G$ (q_G^{n+1}), and consequently, (9.19) is trivially satisfied. However, since the quantity lies only on the grid (e.g., cell centers, cell faces, cell vertices, etc.) we do not know the value of the quantity at an arbitrary position $\mathbf{x}_D$ which necessitates interpolating the nearby grid quantities. So this is an Eulerian method using the Lagrangian framework for backtracing, thus *semi-Lagrangian* (Figure 9.6). With a simple forward Euler time discretization for the backtrace, the quantity at the grid is updated as

$$q_G^{n+1} = \text{interpolate}(q_G^n,\ \mathbf{x}_G - \Delta t \mathbf{u}^n(\mathbf{x}_G)). \tag{9.20}$$

We can further improve the backtracing in (9.20): one can use higher order Runge-Kutta schemes (e.g., RK2, RK3, RK4, etc.) to more accurately trace the quantity in time. Likewise one can use higher order interpolation methods in space (e.g., cubic) while reducing or avoiding potential overshooting problems (Fedkiw et al., 2001; Fritsch and Carlson, 1980). Alternatively, one can use high order accurate upwind methods such as WENO schemes (Kim et al., 2013), or take multiple backward and forward advection steps (Kim et al., 2007, 2005; Selle et al., 2008a) to reduce error.

Yet another popular strategy is to use secondary particles for velocity advection. A widely used method is a linear combination of Particle-in-Cell (PIC) method and Fluid-Implicit-Particle (FLIP) method (Zhu and Bridson, 2005). When using these particle-based advection methods, we need to transfer the quantities such as velocities back and forth between the grid and the particles, because the underlying physics is computed on a grid. In Figure 9.7, the particle velocities are transferred to the grid ($\mathbf{u}_G^n$), since the subsequent steps are performed on the grid (e.g., applying external forces and pressure projection). Once we apply the forces, the grid velocities are updated ($\mathbf{u}_G^{n+1}$) and transferred back to the particles ($\mathbf{u}_P^{n+1}$) to advect. In PIC, the final particle velocities ($\mathbf{u}_P^{n+1}$) are simply interpolated from the final grid velocities ($\mathbf{u}_G^{n+1}$), while in FLIP, the grid velocity increments ($\mathbf{u}_G^{n+1} - \mathbf{u}_G^n$) are interpolated to the particles, and added to the previous particle velocities

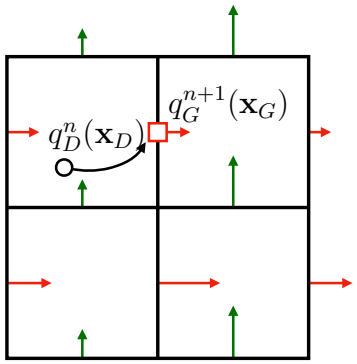
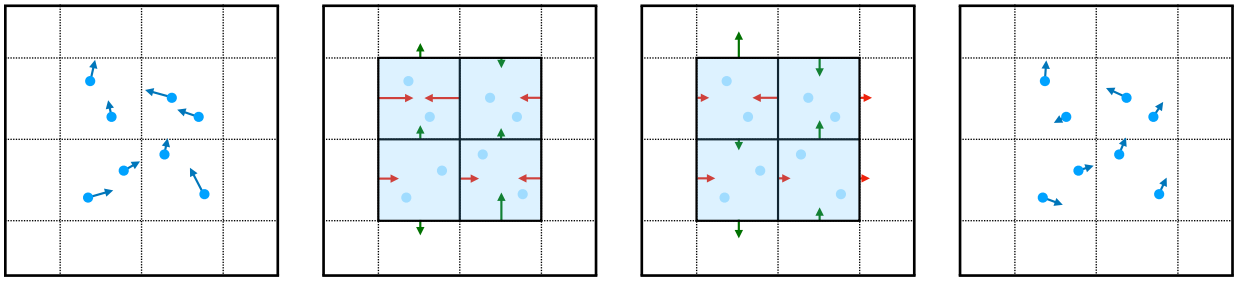


Figure 9.6: **Semi-Lagrangian Advection:** To update the grid quantity at $\mathbf{x}_G$, first back-trace the position where $\mathbf{x}_G$ would be in the previous frame using the underlying velocity field. Then copy the (interpolated) value of $q_D^n(\mathbf{x}_D)$ to $q_G^{n+1}(\mathbf{x}_G)$.



(a) $\mathbf{u}_P^n$ (b) $\mathbf{u}_G^n$ (c) $\mathbf{u}_G^{n+1}$ (d) $\mathbf{u}_P^{n+1}$

Figure 9.7: **Velocity Transfer between Particles and Grid.** (a) initial particle velocities. (b) grid velocities transferred from particle velocities. (c) final grid velocities after applying all forces (e.g., external forces, pressure force, etc.). (d) updated particle velocities transferred back from the grid velocities.

($\mathbf{u}_P^n$). This increment is done to avoid the blurring out of velocity and resulting artificial dissipation due to repeated interpolation in PIC. In practice, PIC and FLIP are combined with a regularization parameter α (typically $\alpha \approx 0.98$) to balance the overly noisy behavior of FLIP with the over-smoothing of PIC.

$$\mathbf{u}_P^{n+1} = (1.0 - \alpha) \cdot \mathbf{u}_{PIC} + \alpha \cdot \mathbf{u}_{FLIP}, \ \alpha \in [0, 1] \tag{9.21}$$

Once the velocities are returned to the particles, the particles passively flow with the (interpolated) background velocity fields ($\mathbf{u}_G^{n+1}$). The conversion is typically done by using a kernel function. For simplicity, assuming particle quantities are being transferred to the grid in 2D, let us denote the kernel function for the grid index (i, j) as $N_{i,j}(\mathbf{x})$. For higher (than one) dimensional interpolation, we use multiplication of one dimensional kernel functions (Steffen et al., 2008),

$$N_{i,j}(\mathbf{x}_P) = N\left(\frac{1}{\Delta x}(x_p - x_i)\right) N\left(\frac{1}{\Delta y}(y_p - y_j)\right), \text{ in 2D}, \tag{9.22}$$

where N represents a one-dimensional kernel function, $\mathbf{x}_P = (x_p, y_p)$, (x_i, y_j) refers to the grid sample location, and Δx and Δy are the width and height of the grid cell, respectively. The choice of N depends on the application. A linear kernel is often used for fluid simulation while quadratic or cubic functions are preferred in the Material-Point-Method (MPM) (Jiang et al., 2016), as it uses $\nabla N_{i,j}(\mathbf{x})$ to compute elastic forces and discontinuous $\nabla N_{i,j}(\mathbf{x})$ causes unwanted jumps in the forces. In our interpolation method, we also use the quadratic

kernel to ensure continuous $\nabla \times N_{i,j}(\mathbf{x})$ (see Section 10). The shape of the kernels are:

$$N_{linear}(x) = \begin{cases} 1 - |x|, & \text{if } 0 \leq |x| < 1 \\ 0, & \text{otherwise} \end{cases}$$

$$N_{quadratic}(x) = \begin{cases} \frac{3}{4} - |x|^2, & \text{if } 0 \leq |x| < \frac{1}{2} \\ \frac{1}{2}\left(\frac{3}{2} - |x|\right)^2, & \text{if } \frac{1}{2} \leq |x| < \frac{3}{2} \\ 0, & \text{otherwise} \end{cases} \tag{9.23}$$

$$N_{cubic}(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3}, & \text{if } 0 \leq |x| < 1 \\ \frac{1}{6}\left(2 - |x|\right)^3, & \text{if } 1 \leq |x| < 2 \\ 0, & \text{otherwise.} \end{cases}$$

With the kernel functions (9.23), we can transfer a quantity from particles to grid in a weighted average manner. For example, the x component of the velocity, u, on the staggered grid becomes

$$u_{i+\frac{1}{2},j} = \frac{\Sigma_p u_p N_{i+\frac{1}{2},j}(\mathbf{x}_P)}{\Sigma_p N_{i+\frac{1}{2},j}(\mathbf{x}_P)}, \tag{9.24}$$

where u_p is the x component of the particle velocity.

Based on the discussion above, we can see that many modern advection schemes are tightly related to interpolation of grid data: to obtain the pointwise velocity at an arbitrary point in the domain as required to trace particle trajectories, we need to interpolate nearby grid velocities. However, these existing advection schemes have a potential problem: interpolating the discrete velocity samples fails to preserve the incompressibility constraint, even if the grid data satisfies it. To see the problem with standard interpolants, we define a linear interpolation function $lerp(a, b, t) = (1 - t)a + tb$ and construct the bilinearly interpolated velocity at a point $P = (x, y)$ (see Figure 9.8) as

$$\begin{aligned} u(x, y) &= lerp(lerp(u_0, u_1, \alpha_u(x)), lerp(u_2, u_3, \alpha_u(x)), \beta_u(y)) \\ v(x, y) &= lerp(lerp(v_0, v_1, \alpha_v(x)), lerp(v_2, v_3, \alpha_v(x)), \beta_v(y)) \end{aligned} \tag{9.25}$$

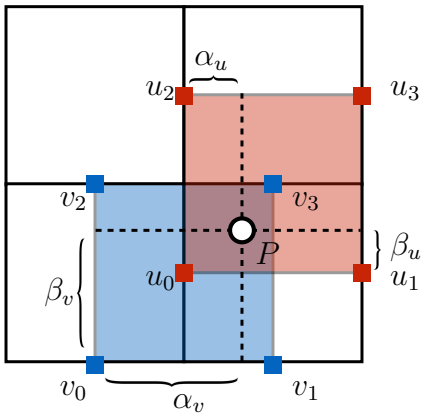


Figure 9.8: **Direct Velocity Interpolation:** To compute a pointwise velocity at point P in a staggered velocity grid, one (bilinearly) interpolates the nearby velocity samples component by component.

where the α and β functions return edge fractions ($0 \leq \alpha, \beta \leq 1$) for the data indicated by their subscripts. This interpolated velocity is not analytically divergence-free in general:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= \frac{lerp(u_1 - u_0, u_3 - u_2, \beta_u(y)) + lerp(v_2 - v_0, v_3 - v_1, \alpha_v(x))}{h} \\ &\neq 0 \end{aligned} \tag{9.26}$$

where the grid cell width h appears due to the derivatives of α and β. We present improved divergence-free interpolation schemes in Section 11.2.3, which will in turn lead to improved advection behavior.

Chapter 10

Problem Setting

In this chapter, we present the problems of direct velocity interpolation which produces divergent pointwise velocity as shown in (9.26). We build on a standard staggered grid-based fluid solver for the incompressible Euler equations (Bridson, 2015; Fedkiw et al., 2001) as reviewed in the preceding chapter. We represent solids using node-based level sets, leading to simple marching cubes cut-cells. For cut-cell pressure projection, we use the finite volume Poisson stencil of Ng et al. (2009). For velocity advection, we use semi-Lagrangian backtracing with multilinear interpolation and third order Runge-Kutta.

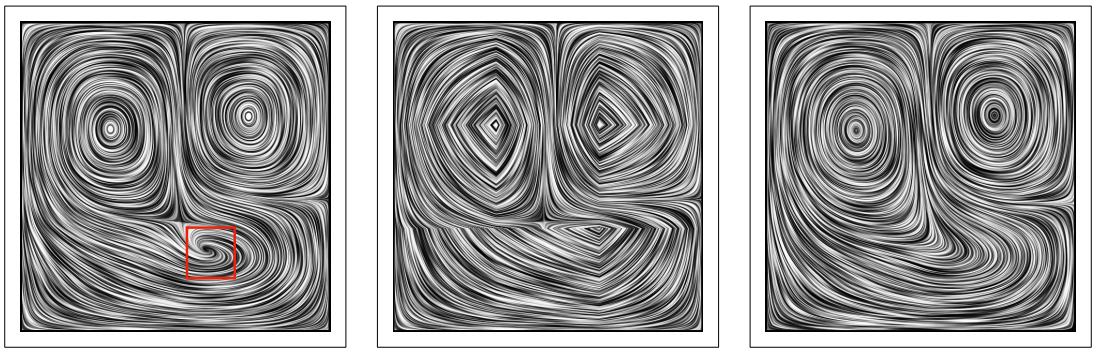
We develop a new velocity interpolant for use during advection. Common advection discretizations use semi-Lagrangian (Stam, 1999) or Lagrangian (Zhu and Bridson, 2005) methods that trace trajectories through the flow and require interpolation to query velocity at arbitrary locations. We focus on advecting passive particles, rather than velocity or other variables affecting the dynamics; this lets us use identical simulated discrete velocity fields in our evaluations, isolating the effects of incompressible interpolation.

Advection occurs after incompressibility enforcement (i.e., pressure projection) so that $\mathbf{u}$ is (discretely) divergence-free. Discrete velocity components are placed at grid face centers, so the discrete divergence-free condition is

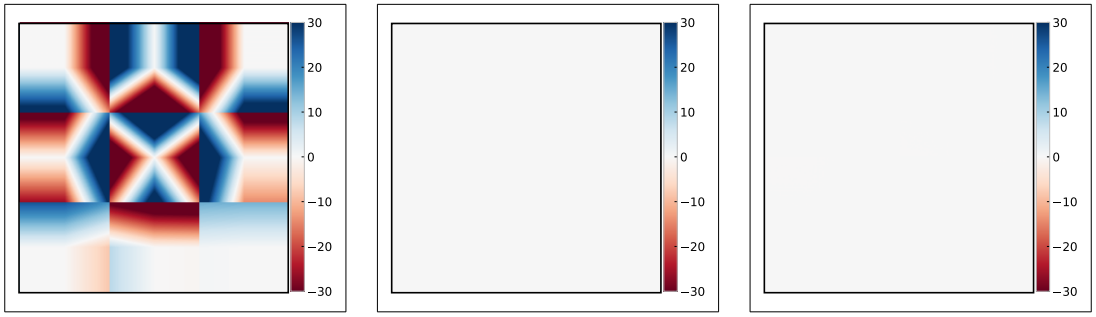
$$\sum_{\text{faces } f} A_f u_f = 0, \tag{10.1}$$

where A_f is face area and u_f is the face's outward oriented normal velocity component (see Section 9.2 for details).

The simplest standard approach applies bilinear interpolation on each staggered velocity component, but the resulting vector field's analytical divergence is nonzero in general



(a) Interpolated velocity fields using standard bilinear (left), linear Curl-Flow (middle), and quadratic Curl-Flow (right).



(b) Corresponding pointwise divergence plots (units: 1/s)

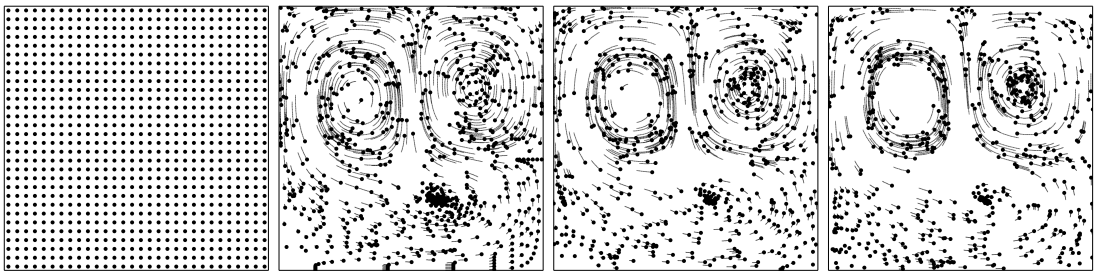
.

Figure 10.1: **Pointwise Divergence Comparison:** Left: Bilinear interpolation of discretely divergence free velocity data on a 3×3 grid. The red box in (a) highlights a spurious sink. Middle: *Linear* Curl-Flow interpolation is perfectly divergence-free, but has piecewise constant components that induce kinks. Right: *Quadratic* Curl-Flow interpolation is divergence-free and smooth.

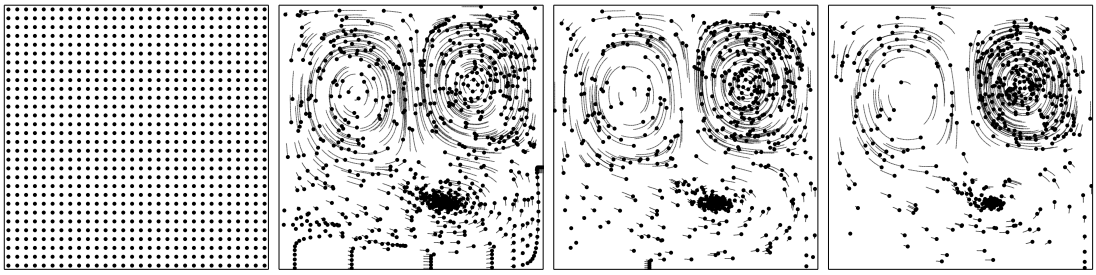
(Section 9.4). Figure 10.1, left column, gives an illustration; advecting uniformly sampled particles through this field (Figure 10.2, top row), a sink region absorbs a large number of particles while many more cluster into large and small rings (Bridson et al. (2007) call these "gutters"). Unfortunately, neither higher order interpolants (Figure 10.2, middle row) nor using node-based/colocated velocity data resolve these issues, since such interpolants remain divergence-oblivious. We use staggered multilinear interpolation as our baseline direct velocity interpolant throughout the thesis, unless stated otherwise.

The middle and right columns in Figure 10.1 show the velocity and pointwise divergence fields using the linear and quadratic variants of our approach, respectively, developed in the next section. Since the linear case exhibits velocity kinks we prefer the quadratic, but neither contain sinks or sources and their pointwise divergence fields are exactly zero.

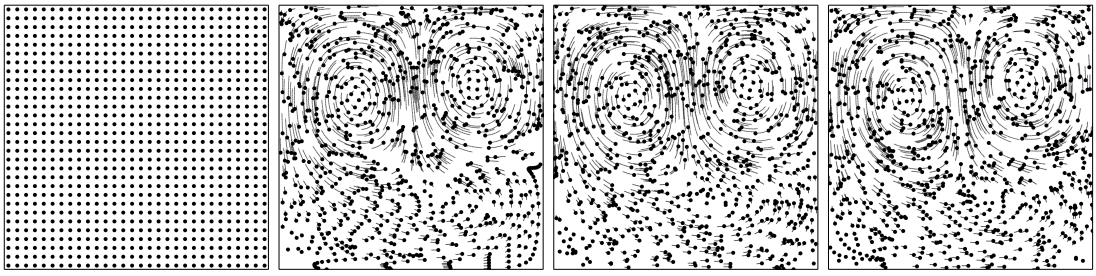
Direct incompressible velocity interpolation As discussed in Section 8.2.6, several existing approaches aim for divergence-free interpolation (or "conservative velocity interpolation") using velocity data directly, rather than using intermediate potentials. In particular, Jenny et al. (2001) and Balsara (2001) used the same staggered grid layout as us (and in practice yield identical fields to one another). Although these methods also produce pointwise incompressible fields within each cell, velocity kinks are visible *between* cells due to their piecewise construction and use of the minmod limiter (Jenny et al., 2001), unlike our (quadratic) Curl-Flow method (Figure 10.3).



(a) Bilinear velocity interpolation

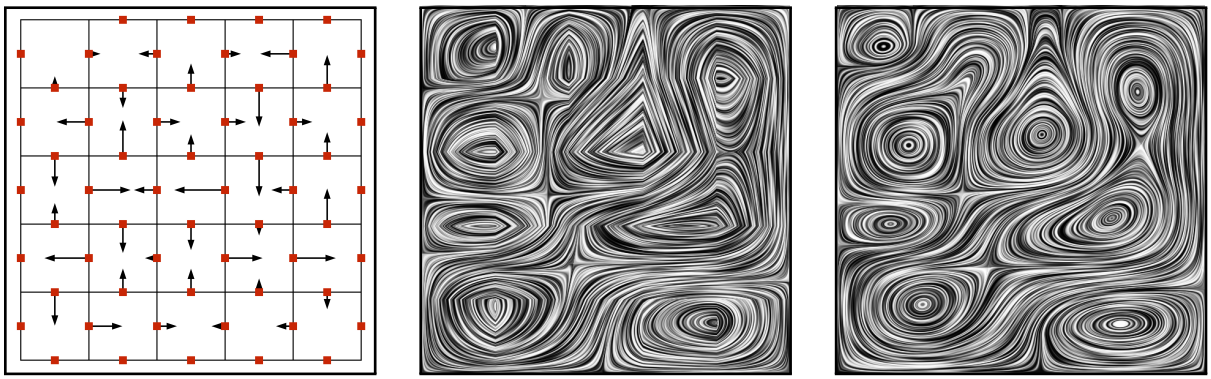


(b) Bicubic velocity interpolation



(c) Curl-Flow interpolation

Figure 10.2: **Effect on Particle Distribution:** Initially uniform particles advected through a static 2D vector field on a 3×3 grid. Frames 1, 100, 200, and 300 are shown from left to right. (a) With direct bilinear velocity interpolation, particles become heavily clustered leaving large empty voids in the flow. (b) With higher order direct velocity interpolation (monotonic cubic (Fedkiw et al., 2001; Fritsch and Carlson, 1980)), clustering and spreading remain significant. (c) With our Curl-Flow interpolation using a quadratic kernel, particles remain much more uniformly distributed, as expected.



(a) (b) (c)

Figure 10.3: **Direct Incompressible Velocity Interpolation vs. Curl-Flow.** (a) Discrete and discretely divergence-free velocity field to interpolate. (b) Direct incompressible velocity interpolation (Balsara, 2001; Jenny et al., 2001). The velocity field inside each cell is incompressible, but discontinuities between cells are pronounced. (c) Curl-Flow interpolation is pointwise incompressible and smooth both inside and across cells.

Chapter 11

Curl-Flow Interpolation In 2D

In this chapter, we present how to use our Curl-Flow method in 2D, which produces pointwise divergence-free velocity. In 2D, velocity $\mathbf{u}$ has two components, u and v, and the vector potential has only one scalar component, ψ_z, called the *stream function* and denoted by non-bolded ψ. The relationship (7.1) simplifies to

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}.$$

That is, $\mathbf{u} = \nabla \psi^{\perp}$ as shown in (9.16). To discretize, we place ψ samples at cell vertices (Figure 11.1) and assume ψ varies linearly along edges. Figure 11.1(c) diagrams the relationship between two nodal ψ values and the normal component of velocity, v_n, on their shared edge e_{ij} having unit tangent $\mathbf{e}_{ij}$ and length l. The gradient theorem,

$$\int_{e_{ij}} \nabla \psi(\mathbf{r}) \cdot d\mathbf{r} = \psi(\mathbf{x}_j) - \psi(\mathbf{x}_i), \tag{11.1}$$

discretized on each edge, gives the relationship

$$\frac{\psi_j - \psi_i}{l} = \nabla \psi \cdot \mathbf{e}_{ij} = \nabla \psi^{\perp} \cdot \mathbf{e}_{ij}^{\perp} = v_n(\mathbf{n} \cdot \mathbf{e}_{ij}^{\perp}). \tag{11.2}$$

Since $\mathbf{e}_{ij}^{\perp}$ is oriented, unit length, and matches $\mathbf{n}$ up to a sign flip, the dot product simply determines the sign.

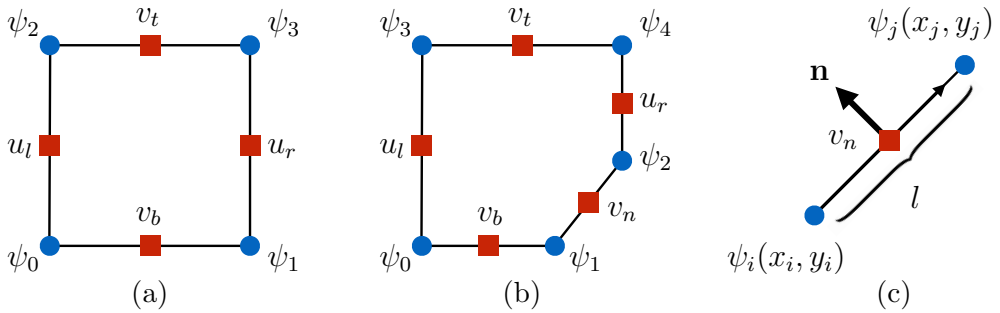


Figure 11.1: **Discretization in 2D:** Red squares represent discrete velocity samples and blue circles represent discrete stream function samples. (a) A uniform grid cell. (b) A cut-cell induced by clipping with a solid object. (c) The relationship between edge normal velocity and stream function samples. For this case, $v_n = -(\psi_j - \psi_i)/l$.

11.1 Uniform Grids in 2D

11.1.1 Recovering Discrete ψ

For a single uniform grid cell as in Figure 11.1(a), equation 11.2 leads to

$$\begin{aligned} u_l &= \frac{\psi_2 - \psi_0}{h}, \quad & u_r &= \frac{\psi_3 - \psi_1}{h}, \\ v_b &= -\frac{\psi_1 - \psi_0}{h}, \quad & v_t &= -\frac{\psi_3 - \psi_2}{h}, \end{aligned} \tag{11.3}$$

consistent with finite differences on (9.16). We seek ψ_i satisfying the given velocities. These equations have a 1D null space: constant offsets of all ψ_i do not change the velocities. (Physically, discrete incompressibility ensures one flux is the negated sum of the others, implying one equation is linearly dependent.) We select a unique solution by arbitrarily setting $\psi_0 = 0$. We determine the other ψ_i by traversing the cell's edge graph, computing each subsequent ψ value from its predecessor on the edge and the edge's velocity component. Discrete incompressibility ensures discrete integrability, i.e., looping back to ψ_0 gives a consistent result.

This graph traversal is also effective when applied to an entire uniform grid of cells, thereby avoiding a traditional (and costly) global Poisson solve for ψ based on (7.2). Any ordering suffices, but we aim to maximize parallelism. As proposed by Biswas et al. (2016) and illustrated in Figure 11.2, we initially set $\psi_0 = 0$, and sweep vertically to obtain all the ψ values at $x = x_0$ (i.e., $\psi_1 = \psi_0 + hu_0$ and so on). We then sweep horizontally, in parallel, to obtain all remaining ψ values. This approach is compatible with any standard exterior

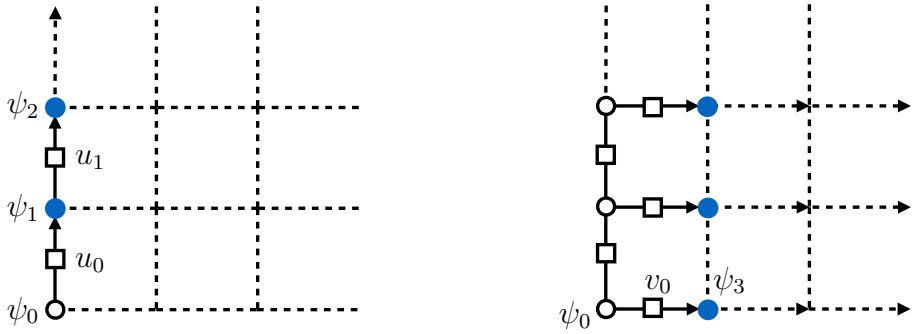


Figure 11.2: **Parallel Sweeping** efficiently recovers a 2D uniform grid discrete stream function field. Left: First, sequentially compute all ψ values at $x = x_0$, starting from $\psi_0 = 0$ and sweeping in the y direction. Right: Next, compute the remaining ψ values by sweeping in the x direction in parallel.

domain boundary conditions (inflow/outflow/open/closed), since it only needs the velocity data. Section 11.2 presents our proposed extension of this method to cut-cell irregular solids.

11.1.2 Interpolating ψ

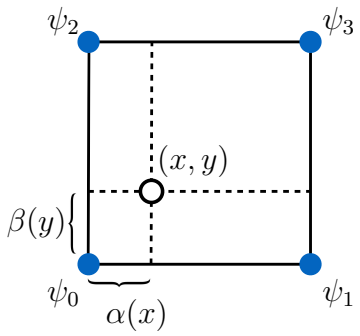


Figure 11.3: Interpolating ψ in uniform grid

Interpolation of the grid ψ values provides an analytical stream function at every point. Applying the analytical curl, we obtain a velocity field which is *pointwise* divergence-free by construction. For example, for bilinear interpolation (referring to the inset figure),

$$\begin{aligned} \psi(x,y) &= lerp(lerp(\psi_0, \psi_2, \beta(y)), lerp(\psi_1, \psi_3, \beta(y)), \alpha(x)), \\ u(x,y) &= \frac{lerp(\psi_2 - \psi_0, \psi_3 - \psi_1, \alpha(x))}{h} = lerp(u_l, u_r, \alpha(x)), \\ v(x,y) &= -\frac{lerp(\psi_1 - \psi_0, \psi_3 - \psi_2, \beta(y))}{h} = lerp(v_b, v_t, \beta(y)), \end{aligned} \tag{11.4}$$

where $lerp(a, b, t) = (1 - t)a + tb$ and the rightmost equalities follow from (11.3). In this specific case, the analytical divergence is clearly zero because it exactly equals the finite difference divergence:

$$\nabla \cdot \mathbf{u} = \frac{u_r - u_l + v_t - v_b}{h} = 0. \tag{11.5}$$

Notably, the derivatives in the curl operator inherently induce a difference in polynomial degree between axes: here, velocity components are piecewise constant in one axis and piecewise linear in the other, introducing undesired tangential discontinuities between cells (Figure 10.1, middle).

Fortunately, because we have the discrete ψ field at hand, we can recover velocity continuity simply by upgrading the interpolation to a higher polynomial degree (Figure 10.1, right). Although many choices are possible, we adopt the quadratic dyadic B-spline kernels popular in material point methods (Jiang et al., 2016; Steffen et al., 2008). The quadratic kernels are sufficiently smooth, their analytical curl is straightforward to derive, and they possess relatively small stencils for efficiency. Bao et al. (2017) similarly used tensor-product B-splines to construct regularized Dirac delta functions in their divergence-free immersed boundary method.

On 2D uniform grids, a velocity interpolant constructed in this way will conveniently reduce back to a simple function of the original discrete u, v velocities, as seen in (11.4), eliminating the need for explicit ψ values. This occurs because, in 2D, each velocity component is defined by a *single* derivative of ψ and the associated null space is a constant shift. However, deriving simple and general direct incompressible velocity interpolants in this way becomes unwieldy or impossible in the presence of cut-cell geometry and especially in 3D, where each velocity component depends on two vector potential components and the null space is nontrivial.

11.2 Cut-Cells in 2D

11.2.1 Recovering Discrete ψ

Considering cut-cell solids during stream function reconstruction requires accounting for truncated grid edges and non-axis-aligned "cut edges". Applying (11.2) we get an update rule for sweeping through such edges:

$$\psi_{i+1} = \psi_i + l_{i,i+1} v_{i,i+1} (\mathbf{e}^{\perp}_{i,i+1} \cdot \mathbf{n}_{i,i+1}). \tag{11.6}$$

For static (zero velocity) solids, the recovered ψ will be constant on the surface, i.e., an isocontour along which particles should slide in a free-slip manner. Static obstacles also preserve the simplicity of parallel sweeping: nodal stream function values at the entry and exit point of a grid line through a solid obstacle are identical.

11.2.2 Interpolating ψ

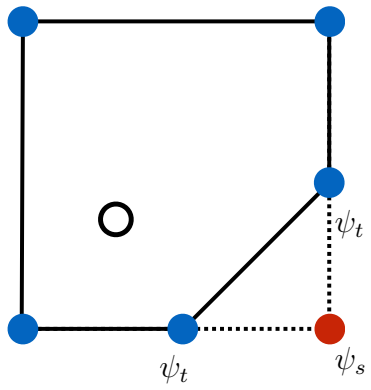


Figure 11.4: Interpolating ψ in cut-cell

Cut-cell solids cause the nodal ψ values (blue in the inset) to no longer lie in a convenient uniform grid, thereby complicating interpolation. One could use moving least squares (MLS) interpolation based on ψ data in some neighborhood, but MLS requires solving small dense linear systems and computing the necessary analytical derivatives is nontrivial (Huerta et al., 2004). Instead we extrapolate ψ values to uniform grid nodes inside the solid (i.e., ψ_s in the inset) and use quadratic B-spline kernels as in the uniform grid case. This extrapolation can be done in various ways, such as using MLS or simply copying (e.g., from ψ_t to ψ_s), assuming discrete flux is zero on interior solid edges (dashed lines); we adopt the latter for simplicity. At the outer axis-aligned domain boundaries, the quadratic stencil is likewise missing data samples, so we extrapolate an additional uniform layer of ψ samples outside the domain. However, regardless of these extrapolation and interpolation choices near boundaries, the no-normal-flow condition is so far enforced only approximately; we propose an additional correction below.

11.2.3 A Curl-Noise Enhancement for Exact 2D Boundary Enforcement

To ensure that the interpolated ψ will be *strictly* constant along the polygonal boundary for static obstacles, we adapt ideas from the Curl-Noise method (Bridson et al., 2007). That method modulates the existing ψ value near obstacles, forcing it to zero on the boundary by multiplying against a smooth ramp function based on boundary proximity. Defining d_0 as an influence radius (we use h), $d(\cdot)$ as the distance to the surface, and $ramp(\cdot)$ as Bridson's ramp function, the necessary correction is the product $\psi'(\mathbf{x}) = \alpha\psi(\mathbf{x})$, where $\alpha = ramp(d(\mathbf{x})/d_0)$. We further modify the original ramping strategy to remove dramatic spurious flow deviation (Figure 9.5(b)) using a desired target boundary value:

$$\psi'(\mathbf{x}) = \alpha\psi(\mathbf{x}) + (1-\alpha)\psi_t. \tag{11.7}$$

(See Section 9.3.1 for details.) However, the initial multiplication still appreciably damages the *normal derivatives* of ψ. Since $\mathbf{u} = \nabla\psi^{\perp}$ implies $(\mathbf{u}\cdot\mathbf{t})\mathbf{t} = \frac{\partial\psi}{\partial\mathbf{n}}$, where $\mathbf{t}$ is the boundary

tangent vector, this induces undesired tangential damping, acceleration, or no-slip behavior (Figure 9.5(c)).

We propose a novel purely *additive* ramping procedure that instead enforces ψ_t by computing and adding a compensating offset. The required offset is determined by interpolating the existing ψ at the closest boundary point and subtracting it from ψ_t. Letting $cp(\cdot)$ be a function returning the closest boundary point, we modify ψ as:

$$\psi'(\mathbf{x}) = \psi(\mathbf{x}) + (\psi_t - \psi(cp(\mathbf{x})))(1 - \alpha). \tag{11.8}$$

This expression ramps the full correction precisely on at the boundary ($d(\mathbf{x})/d_0 = 0$) and blends it smoothly off at the edge of the influence region ($d(\mathbf{x})/d_0 = 1$). The inset shows an example input ψ curve (black), and the result after using multiplicative (red) and our additive (green) ramping to $\psi_t = 0$. Both methods correct the value, but additive ramping better preserves the derivative near the boundary and thus yields more faithful free-slip flow (Figure 11.6, top half). When querying velocity, as we did for the interpolation kernels, we use the analytical curl to determine this added contribution. Note that our ramping approach with polygonal solid/exterior boundaries can introduce discontinuities in velocity near the boundaries. This is because the recovered velocity $\mathbf{u}' = \nabla\psi'^{\perp}$ contains $\nabla\alpha$ term from (11.8), and $\nabla\alpha$ is not continuous over different edge segments in general. Despite the discontinuities, our Curl-Flow method produces boundary-respecting and divergence-free flows, and because the required ψ correction is typically small, the discontinuities are often not perceptible in practice.

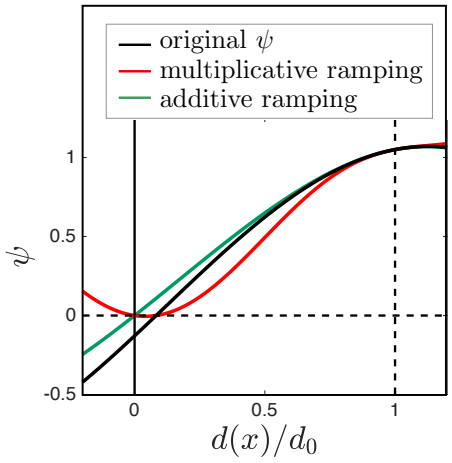
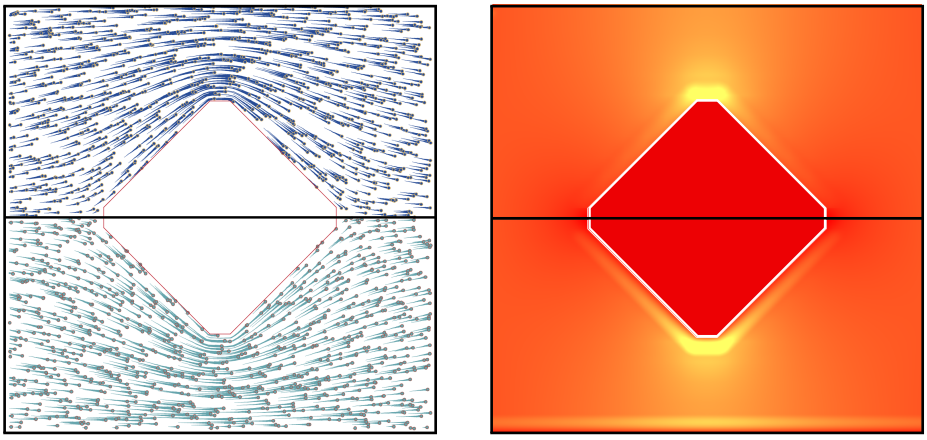


Figure 11.5: Ramping comparison

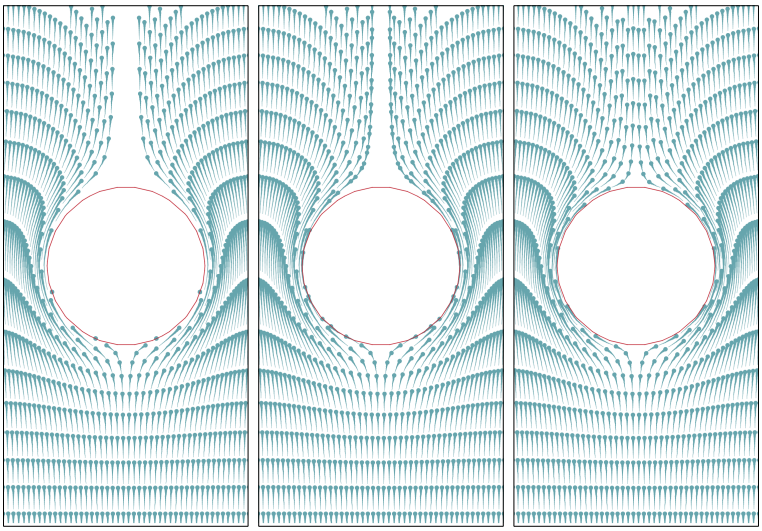
We apply this ramping strategy to both exterior domain boundaries and solid surfaces. For a static solid or closed domain boundary, ψ_t is a constant value since the normal flux on the boundary is zero. For prescribed velocity inflow/outflow or open exterior boundaries, the ψ endpoint values differ (e.g., ψ_{c_0}, ψ_{c_1}). One could set the ramping target value ψ_t to the linearly interpolated value of ψ_{c_0} and ψ_{c_1} at the closest point to achieve desired constant flow in the normal direction across that edge. However, we opted not to apply ramping for such domain boundaries, since exact enforcement there is not as visually critical.



(a) (b)

Figure 11.6: **Additive vs. Multiplicative Ramping**: A steady rightward flow with boundary ψ_t value adjustment. Left: Particles with speed-dependent trail lengths. Right: Velocity magnitudes. Top half: Our additive ramping exactly satisfies the boundary with no apparent damping of free-slip velocities. Bottom half: Despite modifying Bridson's multiplicative ramp to target the correct ψ_t value, tangential damping occurs along the bottom wall and near the solid, since $\partial\psi/\partial\mathbf{n}$ is damaged. Note the more pronounced banding in the bottom magnitude plot.

Figure 11.7 compares direct (bilinear) velocity interpolation vs. our Curl-Flow method (with additive ramping boundary correction) on a steady flow past a disk. To approximate free-slip near solids for the direct case, we extrapolate fluid velocities to grid samples inside the solid (Houston et al., 2003; Rasmussen et al., 2004).



(a) (b) (c)

Figure 11.7: **Flow Near Solids:** Particle trajectories flowing from bottom to top under a fixed (steady state) flow past a solid disk in 2D. Grid resolution: 30×15. (a) Direct velocity interpolation induces spurious trailing gaps, when trajectories colliding with the solid are terminated. (b) With the same velocity interpolant, projecting penetrating particles back to the surface does not significantly help divergent trajectories to "close up" and unwanted particle clumping occurs. (c) Our Curl-Flow method better respects solid boundaries and produces less spurious empty space behind the solid. Notice the trajectories are simply isocontours of the 2D ψ field.

Chapter 12

Curl-Flow Interpolation In 3D

In this chapter, we extend the 2D Curl-Flow method to 3D. In three dimensions, the scalar stream function ψ is replaced by the vector potential $\boldsymbol{\psi} = (\psi_x, \psi_y, \psi_z)$; each velocity component is now dictated by the *interactions* of two ψ components' derivatives (e.g., $u = \partial\psi_z/\partial y - \partial\psi_y/\partial z$). This interaction complicates boundary enforcement, especially for irregular geometry. The curl operator also possesses a multi-dimensional null space, necessitating enforcement of a *gauge condition* to find an appropriate unique $\boldsymbol{\psi}$.

12.1 Uniform Grids in 3D

We place vector potential components on cell edges and velocity normal components on cell faces (Ando et al., 2015; Bao et al., 2017; Elcott et al., 2007) (Figure 12.1). Given discretely divergence-free face velocities, we seek corresponding edge vector potential values.

Considering only static obstacles, we assume that the input axis-aligned exterior domain boundaries have zero normal velocity, and we use a constant value (i.e., 0) for the tangential components of boundary $\boldsymbol{\psi}$, or $\boldsymbol{\psi}_{tan}$, to enforce zero *pointwise* normal velocity when interpolated. (See Section 12.3 for details.) Thus, the continuous problem we are solving is

$$\begin{aligned} \nabla \times \boldsymbol{\psi} &= \mathbf{u} \quad \text{in} \quad \Omega, \\ \boldsymbol{\psi}_{tan} &= \mathbf{0} \quad \text{on} \quad \partial\Omega, \end{aligned} \tag{12.1}$$

with $\mathbf{u}$ given and $\boldsymbol{\psi}$ unknown. The flux across a given surface, in terms of the vector

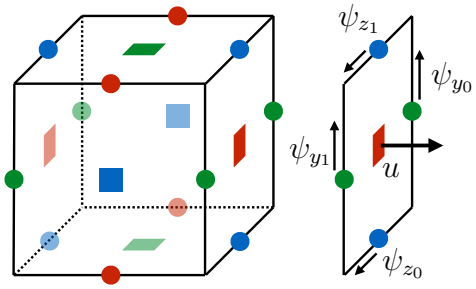


Figure 12.1: **Discretization in 3D:** Discrete vector potential samples are located on cell edges (circles) and velocity samples are on cell faces (squares). Red, green, and blue represent x, y, and z components, respectively.

potential, is

$$\iint_S \mathbf{u} \cdot d\mathbf{S} = \iint_S (\nabla \times \boldsymbol{\psi}) \cdot d\mathbf{S} = \int_C \boldsymbol{\psi} \cdot d\mathbf{r}, \tag{12.2}$$

where S is an oriented smooth surface, C is the oriented boundary curve of the surface, and $\mathbf{r}$ is the boundary tangent direction. (The second equality holds by Stokes' theorem.) Discretizing (12.2) with midpoint quadrature for the single face in Figure 12.1, right, we have

$$u_f h^2 = \sum_{e \in E} \boldsymbol{\psi}_e \cdot (h\mathbf{e}_e) = h(\psi_{y_0} + \psi_{z_1} - \psi_{y_1} - \psi_{z_0}), \tag{12.3}$$

where h is a cell width and $\mathbf{e}_e$ is the oriented unit vector along an edge. (Finite differences on (12.1) yields the same.) Equation 12.3 relates four unknown edge ψ_e values to one face u_f velocity. Stacking the equations for all grid faces yields a global sparse linear system.

Equation 12.1 is an inverse curl problem with infinitely many solutions, e.g., one cell has a seven-dimensional null space: 12 edge ψ_e and five linearly independent face u_f (the sixth is redundant by incompressibility). We select a unique solution using the popular *Coulomb* gauge condition (with our boundary condition), which enforces zero divergence of the vector potential ($\nabla \cdot \boldsymbol{\psi} = 0$). The Coulomb gauge offers maximal smoothness of the $\boldsymbol{\psi}$ field, which will be attractive for interpolation.

Direct application of the Coulomb gauge condition yields

$$\nabla \times (\nabla \times \boldsymbol{\psi}) = -\nabla^2 \boldsymbol{\psi} = \nabla \times \mathbf{u}, \tag{12.4}$$

i.e., a *vector* Poisson problem for $\boldsymbol{\psi}$, around three times larger than the pressure projection (e.g., (Ando et al., 2015)). Fortunately, unlike Ando et al., our discrete velocities are *already*

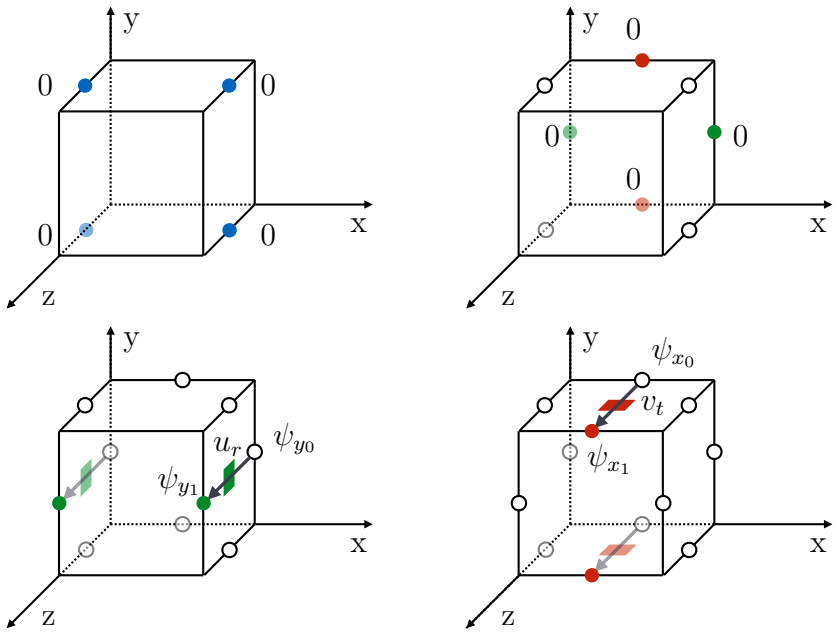


Figure 12.2: **Parallel Sweeping in 3D:** Top-left: Set all ψ_z values to be zero. Top-right: Compute satisfying ψ_x and ψ_y values at $z = z_{min}$. In this example they are all set to zero. Bottom-left: Starting from ψ_y values at $z = z_{min}$, we can compute all ψ_y values in the entire domain in one sweep using the relation $\psi_{y_1} = \psi_{y_0} - hu_r$. Bottom-right: Starting from ψ_x values at $z = z_{min}$, we can compute all ψ_x values in the entire domain in one sweep using the relation $\psi_{x_1} = \psi_{x_0} + hv_t$.

incompressible, enabling us to to solve (12.1) much more efficiently. Observe that

$$\mathbf{u} = \nabla \times \boldsymbol{\psi} = \nabla \times (\boldsymbol{\psi} + \nabla\phi) = \nabla \times \boldsymbol{\psi}', \tag{12.5}$$

where ϕ is an arbitrary scalar field. Since $\nabla \times \nabla\phi$ is always zero (a vector calculus identity), defining $\boldsymbol{\psi}' = \boldsymbol{\psi} + \nabla\phi$ gives another valid vector potential field. Leveraging this characterization of the null space, we propose a two step approach. First, find a *valid but arbitrary* $\boldsymbol{\psi}$ field through an efficient parallel sweeping scheme (Section 12.1.1), like our 2D method. Second, modify this $\boldsymbol{\psi}$ field to satisfy the Coulomb gauge $\nabla \cdot \boldsymbol{\psi} = 0$ and our boundary condition $\boldsymbol{\psi}_{tan} = \mathbf{0}$ using a carefully constructed ϕ (Section 12.1.2).

12.1.1 Recovering a Vector Potential by Parallel Sweeping

Our first goal is to efficiently find a velocity-consistent discrete vector potential field on a box-shaped domain, irrespective of boundary conditions or gauge choice. Our proposed fast 3D parallel sweeping strategy is illustrated in Figure 12.2.

1. Set $\psi_z = 0$ (or a constant) *everywhere* in the domain (Figure 12.2, top-left). This is safe because the remaining two components (ψ_x, ψ_y) still suffice to represent any set of three velocity components (u, v, w) (Ravu et al., 2016).

2. Compute velocity-satisfying vector potential values for the $z = z_{min}$ boundary plane. Assuming zero boundary normal flux, we can simply set all ψ_x and ψ_y values to zero (Figure 12.2, top-right). For different boundary conditions (e.g., nonzero normal flux), boundary $\boldsymbol{\psi}$ values need to be modified, accordingly.

3. Compute the remaining vector potential values by parallel sweeping. Equation 12.3 and $\psi_z = 0$ give $\psi_{y_1} = \psi_{y_0} - hu$ (Figure 12.2, bottom-left), and likewise $\psi_{x_1} = \psi_{x_0} + hv$ (Figure 12.2, bottom-right). In this manner, the values of ψ_{x_n} and ψ_{y_n}, along with the discrete velocities, dictate $\psi_{x_{n+1}}$ and $\psi_{y_{n+1}}$ as we sweep across the entire domain. Each 1D line of variables can be computed independently.

At the conclusion of this process the discrete velocity condition (12.3) is met on all grid faces, $\psi_z = 0$ everywhere, and we have zero ψ_x and ψ_y values on all the outer boundaries, except the $z = z_{max}$ boundary plane.

Next, we would like to modify this $\boldsymbol{\psi}$ to enforce the desired gauge and boundary conditions.

12.1.2 Boundary Conditions and Gauge Correction

To satisfy the boundary condition $\boldsymbol{\psi}_{tan} = \mathbf{0}$, we first construct a corrective discrete scalar field ϕ_{BC}, which is defined at nodes of cells so components of the discrete $\nabla\phi_{BC}$ coincide with edge-based vector potential components. The desired boundary condition $\boldsymbol{\psi}'_{tan} = 0$ gives

$$\boldsymbol{\psi}'_{tan} = \boldsymbol{\psi}_{tan} + \nabla\phi_{BC} = \mathbf{0} \quad \text{on } \partial\Omega. \tag{12.6}$$

Given $\boldsymbol{\psi}_{tan}$, already known from parallel sweeping, we must find ϕ_{BC} on boundary nodes. Furthermore, since our sweeping process ensured that the only nonzero boundary $\boldsymbol{\psi}_{tan}$ values left to be eliminated are ψ_x and ψ_y on the $z = z_{max}$ plane, finding nonzero ϕ_{BC}

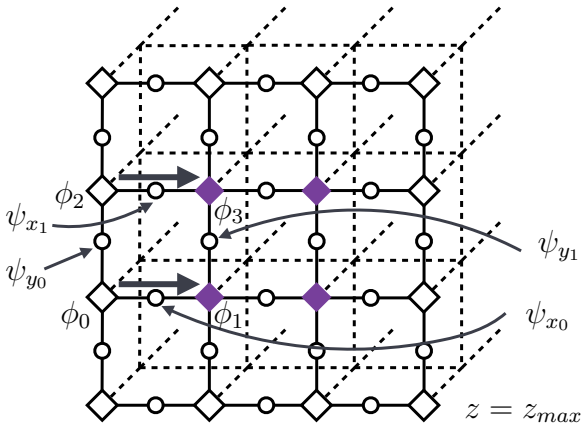


Figure 12.3: **Gauge Correction Boundary Conditions:** We construct $\nabla\phi_{BC}$ to update boundary $\boldsymbol{\psi}$ components to satisfy no-normal-flow. Black/white circles, black/white diamonds, and purple diamonds represent known vector potential values, pinned ϕ values, and unknown ϕ values, respectively. Unknown ϕ values are computed from the pinned or previously computed ϕ values ($\phi_1 = \phi_0 - h\psi_{x_0}$, $\phi_3 = \phi_2 - h\psi_{x_1}$).

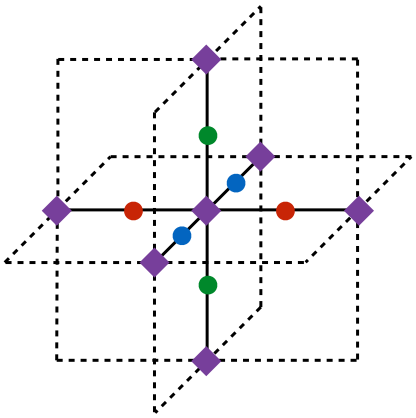


Figure 12.4: **Gauge Correction:** A uniform grid *scalar* Poisson solve is used to find a nodal scalar field ϕ (purple diamonds). Adding $\nabla\phi$ to the vector potential values, ψ_x (red disks), ψ_y (green disks), and ψ_z (blue disks), satisfies the Coulomb gauge condition, $\nabla \cdot \boldsymbol{\psi}' = \nabla \cdot (\boldsymbol{\psi} + \nabla\phi) = 0$.

values only on $z = z_{max}$ suffices. We set the outer boundary loop of nodal ϕ_{BC} values on this plane (black/white diamonds in Figure 12.3) to zero, and compute the interior ϕ_{BC} values in the plane by traversing the interior edges using $\phi_{next} = \phi_{prev} - h\psi_e$.

Adding $\nabla\phi_{BC}$ directly to $\boldsymbol{\psi}$ (including any interior edges touching the nonzero ϕ values) would satisfy our exterior boundary condition without breaking consistency between the discrete $\mathbf{u}$ and $\boldsymbol{\psi}$. However, the (arbitrary) gauge of the resulting field offers no guarantees on the presence or absence of large, discontinuous jumps (Silberman et al., 2019). Thus, while applying interpolation and the analytical curl will yield *some* pointwise divergence-free velocity field, its observed pointwise behavior can be irregular depending on interactions between the discrete data jumps and componentwise interpolants, because velocity is determined by differences of vector potential component derivatives. Fortunately, enforcing the Coulomb gauge condition $\nabla \cdot \boldsymbol{\psi} = 0$ on the interior will provide an optimally smooth vector potential field for interpolation.

Defining a new global nodal ϕ field, we apply a gauge correction similar to Silberman et al. (2019) (although they consider different boundary conditions designed to enable a Fourier-based solution that cannot handle interior solids). The Coulomb condition is $\nabla \cdot \boldsymbol{\psi}' = \nabla \cdot (\boldsymbol{\psi} + \nabla\phi) = 0$, giving a node-based *scalar* Poisson problem for ϕ with Dirichlet boundary conditions:

$$\begin{aligned} \nabla \cdot \nabla\phi &= -\nabla \cdot \boldsymbol{\psi} \quad && \text{in } \Omega, \\ \phi &= \phi_{BC} && \text{on } \partial\Omega. \end{aligned} \tag{12.7}$$

After solving for ϕ we update the vector potential with $\boldsymbol{\psi}' = \boldsymbol{\psi} + \nabla\phi$.

12.1.3 Interpolation

Our choice for 3D uniform grid interpolation is a natural generalization from 2D: we apply low order dyadic spline kernels, separately on each staggered component of $\boldsymbol{\psi}$, choosing their polynomial degrees to avoid velocity kinks as follows. The curl operator applied to one component of $\boldsymbol{\psi}$ (e.g., ψ_x) involves its partial derivatives in the other two axes (e.g., $\partial/\partial y$ and $\partial/\partial z$). Therefore to ensure the resulting velocity is at least (piecewise) linear in all directions, we use a mix of linear and quadratic kernels (in our ψ_x example, linear x, quadratic y and z). Uniformly quadratic or higher order interpolants would also suffice, in exchange for higher cost.

12.2 Cut-Cells in 3D

The interference of cut-cell solid objects requires adaptations to our sweeping and gauge correction steps, and a modified interpolation strategy to exactly enforce the desired boundary behavior.

12.2.1 Parallel Sweeping with 3D Cut-Cells

By carefully removing redundant DOFs and choosing traversal orders, we efficiently obtained valid $\boldsymbol{\psi}$ values on the uniform grids. With the interference of cut-cells, we encounter a different number and different directions of vector potentials on a face, thus a different traversal rule is required. For this purpose, we further assume that the solid portion of partial cut-edges (e.g., two axis-aligned edges of the three purple edges in Figure 12.5(a)) have zero vector potential on the edge. Setting these two values to zero effectively enforces the vector potential on the diagonal edge to zero for the static solid. By pinning such DOFs, we can safely apply parallel sweeping as before.

For example, since we assumed zero values for the purple edges, we discretize the top face in Figure 12.5(a), left, according to (12.2) as

$$v_t A W_t = h\psi_{z_0} + l_0\psi_{x_1} - l_1\psi_{z_1} - h\psi_{x_0} \tag{12.8}$$

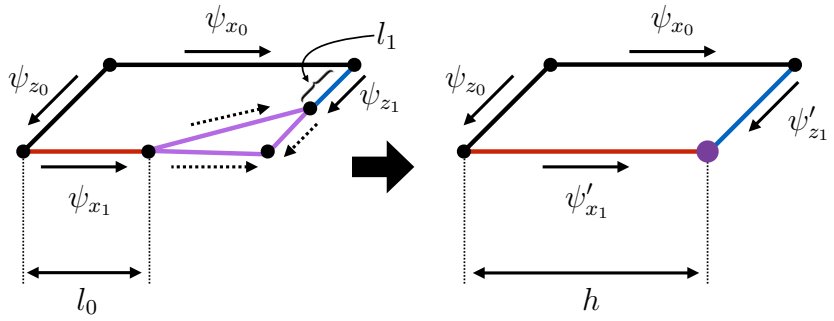
where v_t is the velocity normal component on the top face, $A = h^2$ is the area of a regular (non-cut) cell face, the l_i are the lengths of partial fluid edges, and W_t is the fluid area fraction of the top face.

This face-centric approach extends to cover all geometries of axis-aligned cut-faces (i.e., the marching squares cases). In Figure 12.5(b), consider that the original top face consists of a fluid part (nonzero flux) and a solid part (zero flux). These two (sub-)faces imply two equations,

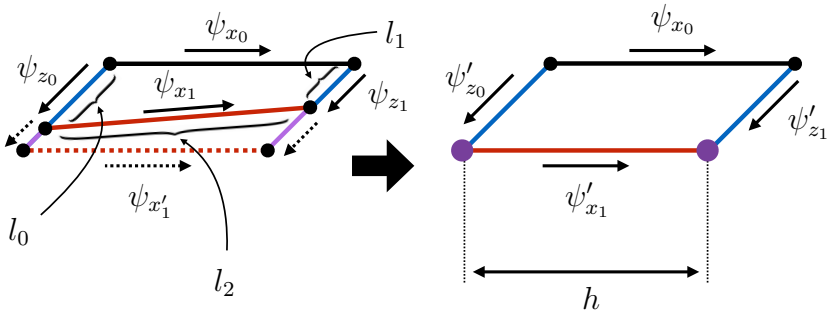
$$\begin{aligned} v_t A W_t &= l_0\psi_{z_0} + l_2\psi_{x_1} - l_1\psi_{z_1} - h\psi_{x_0}, \\ 0 &= l_2\psi_{x_1} - h\psi'_{x_1}. \end{aligned} \tag{12.9}$$

where the second equation relies on assigning zero to the two short purple edges. Therefore, we can find ψ_{x_1} and ψ'_{x_1} via sweeping. The second equation in (12.9) can be omitted if we do not use ψ'_{x_1}.

This approach can be viewed as "stretching" fluid edges. We can obtain the same discrete formulation as (12.8) and (12.9), if we lengthen cut fluid edges to the regular ones but scale down the vector potential values such that the edge lengths times vector potential



(a) Purple triangle (left) is collapsed to the purple point (right) and the new ψ' values in regular cells are length fractions times the original ψ values ($\psi'_{x_1} = \frac{l_0}{h}\psi_{x_1}$, $\psi'_{z_1} = \frac{l_1}{h}\psi_{z_1}$).



(b) Two purple edges (left) are collapsed (right) and the ψ values are rescaled accordingly ($\psi'_{z_0} = \frac{l_0}{h}\psi_{z_0}$, $\psi'_{z_1} = \frac{l_1}{h}\psi_{z_1}$, $\psi'_{x_1} = \frac{l_2}{h}\psi_{x_1}$).

Figure 12.5: **Adapting Parallel Sweeping to Cut-Cells:** Fluid cut-edges are converted to uniform edges by assigning zero ψ to solid purple edges and (conceptually) collapsing them, enabling uniform grid sweeping to proceed.

values remain the same. Likewise, the velocity flux across the partial face area is the same as the scaled (down) velocity flux across the entire face area. Using the approaches above, our parallel sweeping method provides both ψ values on the non-regular faces (e.g., ψ_{x_1}, ψ_{z_1}) and ψ' values on the conceptually uniform faces (e.g., ψ'_{x_1}, ψ'_{z_1}).

12.2.2 Approximate Gauge Correction with 3D Cut-Cells

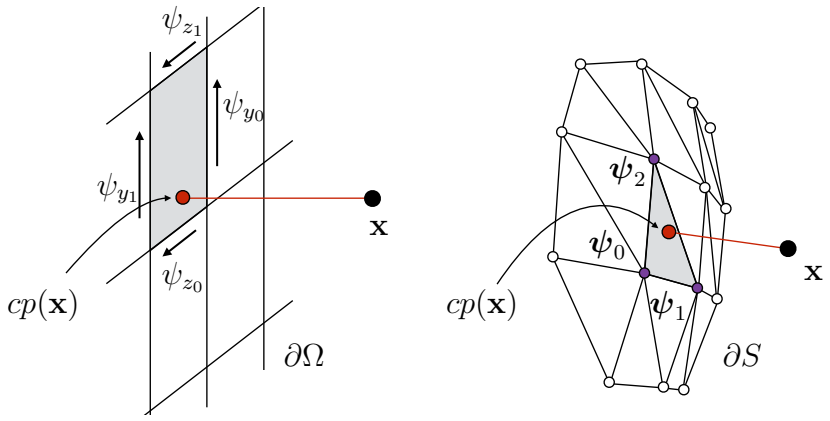
Applying gauge correction is more difficult with polyhedral cut-cell solids, since discrete vector potential components can correspond to arbitrary, rather than Cartesian, directions; this could potentially be handled with a polyhedral PDE solver, e.g., mimetic finite differences (Lipnikov et al., 2006). However, interpolating the resulting edge components would also be unwieldy and, as we saw in 2D, will nevertheless still require an additional correction to fully respect the boundary (i.e., at the pointwise level). We therefore enforce the gauge in a simple but approximate manner: we use the "stretched" vector potentials and conceptually uniform faces determined above (Figure 12.5), and perform gauge correction as before on the entire uniform grid, including through the interior of solids by assuming zero flux on interior solid faces. This provides gauge-corrected (i.e., smooth) axis-aligned $\boldsymbol{\psi}$ values everywhere, which we use for B-spline kernel interpolation as in the uniform case.

12.3 Exact Boundary Enforcement in 3D

As in 2D, we propose an additive ramping-based approach for exact boundary enforcement method in 3D, applied as a correction to the approximate values produced by grid interpolation. The 3D adaptation requires caution in treating the interacting $\boldsymbol{\psi}$ components.

12.3.1 Closed Exterior Domain Boundaries

The domain exterior consists of axis-aligned planes with two tangential components of $\boldsymbol{\psi}$ lying on each boundary plane (Figure 12.6(a)). Recall that for a u-face, the flux is determined by ψ_y and ψ_z through $u = \frac{\partial \psi_z}{\partial y} - \frac{\partial \psi_y}{\partial z}$. To achieve zero flux everywhere on the plane, these terms must precisely cancel: we set the discrete ψ_y and ψ_z values to zero for simplicity, and consequently ramp the pointwise ψ_y and ψ_z values to zero as a particle approaches the border. (A less careful choice of the ramp values can badly distort the velocity, as seen in Figure 9.5(b).) Since we apply ramping to each component of $\boldsymbol{\psi}$ independently, this ramping strategy is identical to the 2D case (Section 11.2.3). The



(a) Exterior boundary (b) Solid boundary

Figure 12.6: **Boundary enforcement:** The black and red disks represent a particle near the boundary and its closest point on the boundary, respectively. (a) For planar exterior boundaries, the discrete ψ values (e.g., ψ_{y_0}, ψ_{y_1}, ψ_{z_0}, ψ_{z_1}) are zero by our boundary conditions, and the relevant axis components ($\psi_y(\mathbf{x})$ and $\psi_z(\mathbf{x})$) are ramped to zero for exact enforcement. (b) For triangulated solids, we first find discrete $\boldsymbol{\psi}$ values at the triangle vertices (e.g., $\boldsymbol{\psi}_0$, $\boldsymbol{\psi}_1$, $\boldsymbol{\psi}_2$) which imply a perfectly tangential surface velocity, under barycentric $\boldsymbol{\psi}$ interpolation. When querying velocities, we ramp the full $\boldsymbol{\psi}$ vector towards this interpolated surface $\boldsymbol{\psi}$ at the closest point.

extension to open domain boundaries can be done similarly to the 2D case. For example, to prescribe u velocities on the x boundary, we can use linear ψ_z values in y direction on the boundary, while keeping ψ_y values to zero. This becomes identical to the ramping in 2D: we apply ramping to ψ_z only to the closed boundaries.

12.3.2 Solid Obstacles

Although we use the solid geometry obtained from marching cubes for pressure projection and $\boldsymbol{\psi}$ reconstruction on the grid, we use the original triangle mesh (with a similar resolution to the simulation grid) for ramping. This is to avoid ramping toward ill-shaped or even near-degenerate triangles (i.e., point-like, or edge-like) that marching cubes can create.

A key component of exact boundary enforcement with ramping is the proper choice of target $\boldsymbol{\psi}$ values (i.e., $\boldsymbol{\psi}_t$) on the solid surface. Specifically, $\boldsymbol{\psi}_t$ should produce a zero normal velocity,

$$\mathbf{u} \cdot \mathbf{n} = (\nabla \times \boldsymbol{\psi}_t) \cdot \mathbf{n} = 0, \tag{12.10}$$

where $\mathbf{n}$ represents the solid normal vector. For exterior boundaries, we used constant values which trivially satisfies (12.10) as discussed in Section 12.3.1. However, one cannot find a single constant vector $\boldsymbol{\psi}_t$ that precisely compensates the existing (spatially varying) ambient interpolated $\boldsymbol{\psi}$ on the solid; moreover, an arbitrary choice of $\boldsymbol{\psi}_t$ can still severely distort the tangential velocities, even if it satisfies (12.10).

We instead leverage the characteristics of barycentric coordinates: the gradient of the barycentric interpolant is constant, thus $\nabla \times \boldsymbol{\psi}_t$ yields a constant velocity (per triangle). Therefore, an appropriate choice of $\boldsymbol{\psi}$ at the triangle vertices ($\boldsymbol{\psi}_0$, $\boldsymbol{\psi}_1$, $\boldsymbol{\psi}_2$ in Figure 12.6(b)), can satisfy (12.10), exactly and continuously. At the same time, we wish to minimize the perturbation of the velocity field induced by ramping, which we can do by encouraging the discrete $\boldsymbol{\psi}_i$ values at triangle vertices to be as close to the ambient $\boldsymbol{\psi}$ (i.e., the interpolated pointwise $\boldsymbol{\psi}$ before applying ramping) as possible. This yields an equality constrained quadratic minimization problem:

$$\begin{aligned} \underset{\mathbf{y}}{\text{argmin}} \quad & \frac{1}{2}(\mathbf{y} - \boldsymbol{\psi}_{amb})^T(\mathbf{y} - \boldsymbol{\psi}_{amb}) \\ \text{subject to} \quad & \left(\nabla \times \sum_{i \in \{0,1,2\}} w_i \mathbf{y}_{t_i}\right) \cdot \mathbf{n}_t = 0 \qquad \text{for } t \in T, \end{aligned}$$

where T is a set of (connected) solid triangles, w_i are the barycentric coordinates of the i^{th} vertex, t_i is the global vertex index of the i^{th} vertex in the t^{th} triangle, $\mathbf{n}_t$ is the

triangle normal, and $\boldsymbol{\psi}_{amb}$ is a stack of the ambient $\boldsymbol{\psi}$ values at the triangle vertices (before ramping). If we write the constraints as $A\mathbf{y} = \mathbf{0}$, the optimality condition gives rise to the linear system

$$\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\psi}_{amb} \\ \mathbf{0} \end{bmatrix} \tag{12.11}$$

or $AA^T\boldsymbol{\lambda} = A\boldsymbol{\psi}_{amb}$. We can find the discrete $\boldsymbol{\psi}_i$ at the solid vertices via $\mathbf{y} = \boldsymbol{\psi}_{amb} - A^T\boldsymbol{\lambda}$. For ramping, $\boldsymbol{\psi}_t$ can be queried by interpolating $\mathbf{y}$ in the closest triangle using barycentric interpolation.

The final modified pointwise $\boldsymbol{\psi}$ with ramping is

$$\boldsymbol{\psi}'(\mathbf{x}) = \boldsymbol{\psi}(\mathbf{x}) - (1 - \alpha(\mathbf{x}))(\boldsymbol{\psi}(cp(\mathbf{x})) - \boldsymbol{\psi}_t(cp(\mathbf{x}))). \tag{12.12}$$

Again, we use $cp(\mathbf{x})$ to represent the closest point on the solid triangles and $\alpha(\mathbf{x}) = \text{ramp}(d(\mathbf{x}/d_0))$ for a smooth transition toward $\boldsymbol{\psi}_t$. Here, d_0 is the influence radius (h in our case), and $\text{d}(\cdot)$ is the distance to the closest point $cp(\mathbf{x})$. We employ a standard *smoothstep* function for $ramp(\cdot)$ in 3D, but other smooth functions would also suffice. This approach is similar to (11.8) in 2D, but $\boldsymbol{\psi}_t$ is no longer a constant function of $\mathbf{x}$, since we use barycentric interpolation, and we ramp all three components of $\boldsymbol{\psi}$. The corresponding divergence-free free-slip velocity is

$$\begin{aligned} \mathbf{u}'(\mathbf{x}) = \nabla \times \boldsymbol{\psi}'(\mathbf{x}) = \mathbf{u}(\mathbf{x}) \\ + \nabla\alpha(\mathbf{x}) \times (\boldsymbol{\psi}(cp(\mathbf{x})) - \boldsymbol{\psi}_t(cp(\mathbf{x}))) \\ - (1 - \alpha(\mathbf{x}))\nabla \times (\boldsymbol{\psi}(cp(\mathbf{x})) - \boldsymbol{\psi}_t(cp(\mathbf{x}))). \end{aligned} \tag{12.13}$$

Due to barycentric interpolation, $\nabla \times \boldsymbol{\psi}_t(cp(\mathbf{x}))$ in (12.13) is constant per triangle, allowing for exact zero flux on the solid surface. This piecewise constant term can add additional discontinuities in velocity to the existing discontinuities from $\nabla\alpha$ (Section 11.2.3). Since the surface is triangulated and its velocity is piecewise constant, there will necessarily be velocity discontinuities when the closest triangle changes near solid obstacles; we found this to be a reasonable tradeoff for incompressibility and precise boundary satisfaction.

To illustrate the improved behavior of 3D Curl-Flow with cut-cells, we simulate a flow past a solid sphere with a fixed velocity field (Figure 12.7), and release a single planar layer of passive particles. With trilinear advection, many of the particles collide with the obstacle, leaving a large gap in the particle field. With Curl-Flow, the particles instead flow more naturally around the obstacle.

The pseudocode for constructing discrete $\boldsymbol{\psi}$ is given in Algorithm 2. This corresponds to building desired interpolants (line 5) in the fluid simulation pipeline of Algorithm 1.

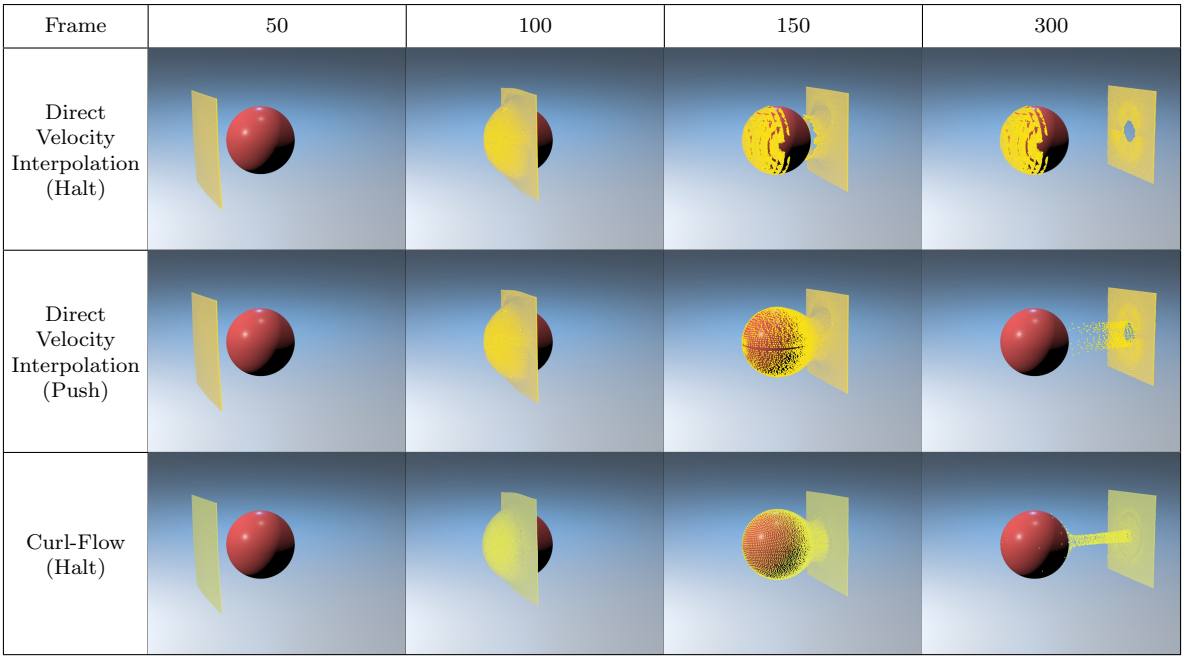


Figure 12.7: **Flow Past A Sphere:** A layer of particles travels left to right under a steady state flow field past a sphere; particles are halted at the back wall for illustration. Grid resolution: $40 \times 20 \times 20$. Top: direct velocity interpolation, with colliding particles frozen upon contact. Middle: direct velocity interpolation, but with colliding particles projected out of the obstacle at each step. Bottom: Curl-Flow interpolation, with colliding particles frozen upon contact. Direct velocity interpolation (top, middle) produces worse results than Curl-Flow (bottom), regardless of how particle collisions are handled. Notice the large gap in particles on the back wall.

Algorithm 2: Building Discrete $\boldsymbol{\psi}$.	
parallel sweeping for initial $\boldsymbol{\psi}$ construction	▷ Section 12.1.1, 12.2.1
enforcing exterior domain boundary condition	▷ Section 12.1.2
applying gauge correction	▷ Section 12.1.2, 12.2.2
computing $\boldsymbol{\psi}$ at solid triangle vertices	▷ Section 12.3.2

Chapter 13

Results and Discussion

To consistently compare our Curl-Flow method against (divergence-oblivious) direct velocity interpolation (i.e., bi/trilinear unless otherwise noted) while isolating the effects of interpolation, we always use the same simulation settings, changing only the interpolation of velocity used for *passive particle tracing*. Velocity advection uses basic semi-Lagrangian (Stam, 1999) with multilinear velocity interpolation, unless stated otherwise. Only *particle advection* differs, even for time-dependent (unsteady) flows. When a particle incorrectly penetrates a solid object due to poor quality velocity fields, large timesteps, or insufficiently accurate path integration, a typical choice is to "resolve" the collision by projecting the particle back out of the solid, although this exacerbates clumping. To highlight such errors, we freeze penetrating particles in place, forever, unless otherwise indicated.

Simulation timings were gathered on a 2.8 GHz, 4-Core Intel Core i7 processor for Tables 13.1 and 13.4, and an 8-core, Apple M1 processor for Tables 13.2 and 13.3.

13.1 Particle Distribution Comparisons in 2D

2D Curl-Flow is particularly attractive because it does not require the linear solve needed for gauge correction in 3D. The earlier 2D results of Figures 9.5(b), 10.2, 11.6, and 11.7 used static velocity fields for illustration. Below, we consider time-evolving simulations and again observe how our approach affects passive particle motions and the resulting distributions.

In Figure 13.3, we initially seed the particles with blue noise sampling and advect the particles under a dynamic flow on a coarse 20×20 grid. Here, we supply an upward

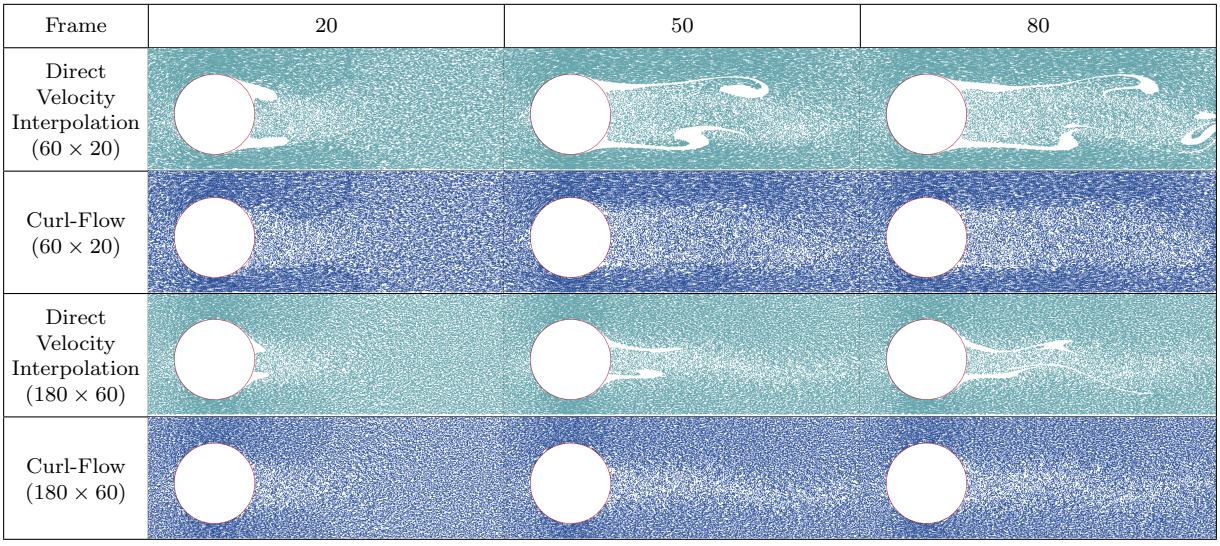


Figure 13.1: **Smooth Obstacle Comparison:** Particles undergoing a time-dependent horizontal flow past a circular object. Particles colliding with objects are halted in place for illustration. Regardless of the grid resolution, direct velocity interpolation creates spurious gaps (cyan), but our Curl-Flow interpolation tightly follows the solid object significantly reducing the gaps in the flow (dark blue).

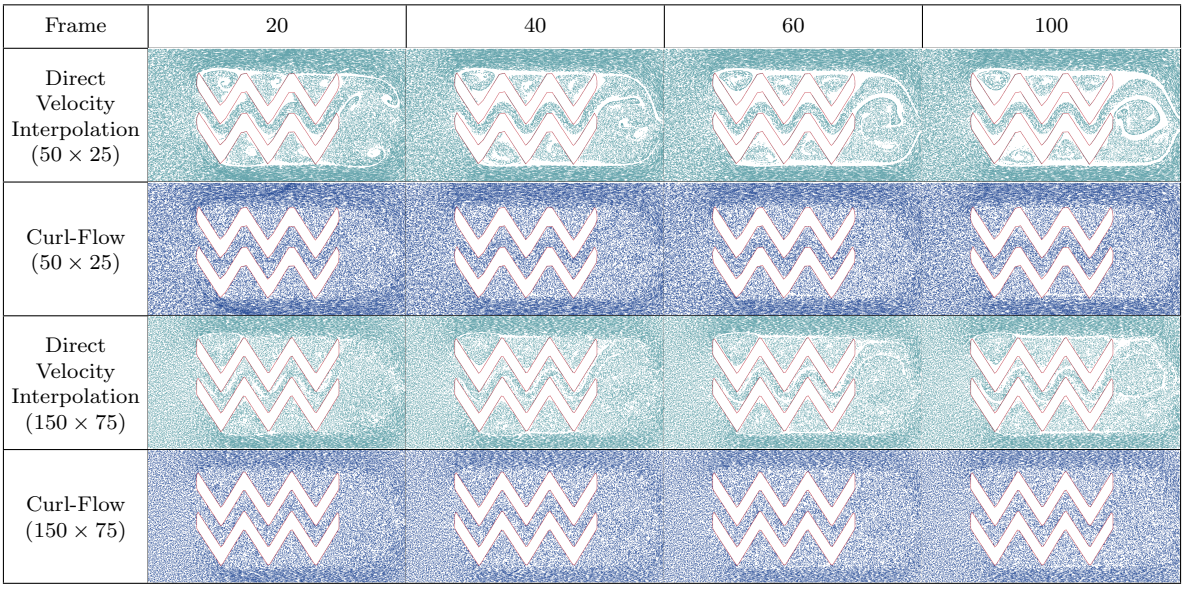
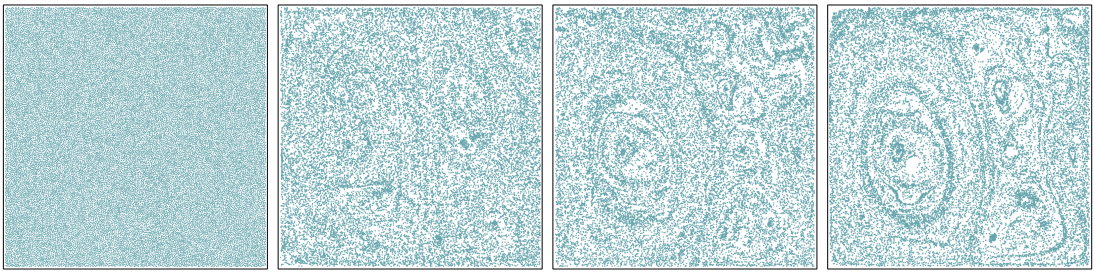
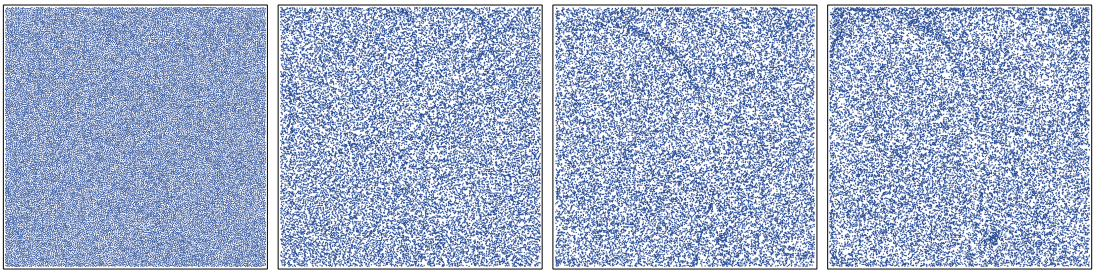


Figure 13.2: **Jagged Obstacle Comparison:** Particles in a time-dependent horizontal flow with jagged solid objects. Curl-Flow interpolation (cyan) tightly follows the jagged obstacle, leaving significantly fewer gaps than when using direct velocity interpolation (dark blue).



(a) Bilinear velocity interpolation

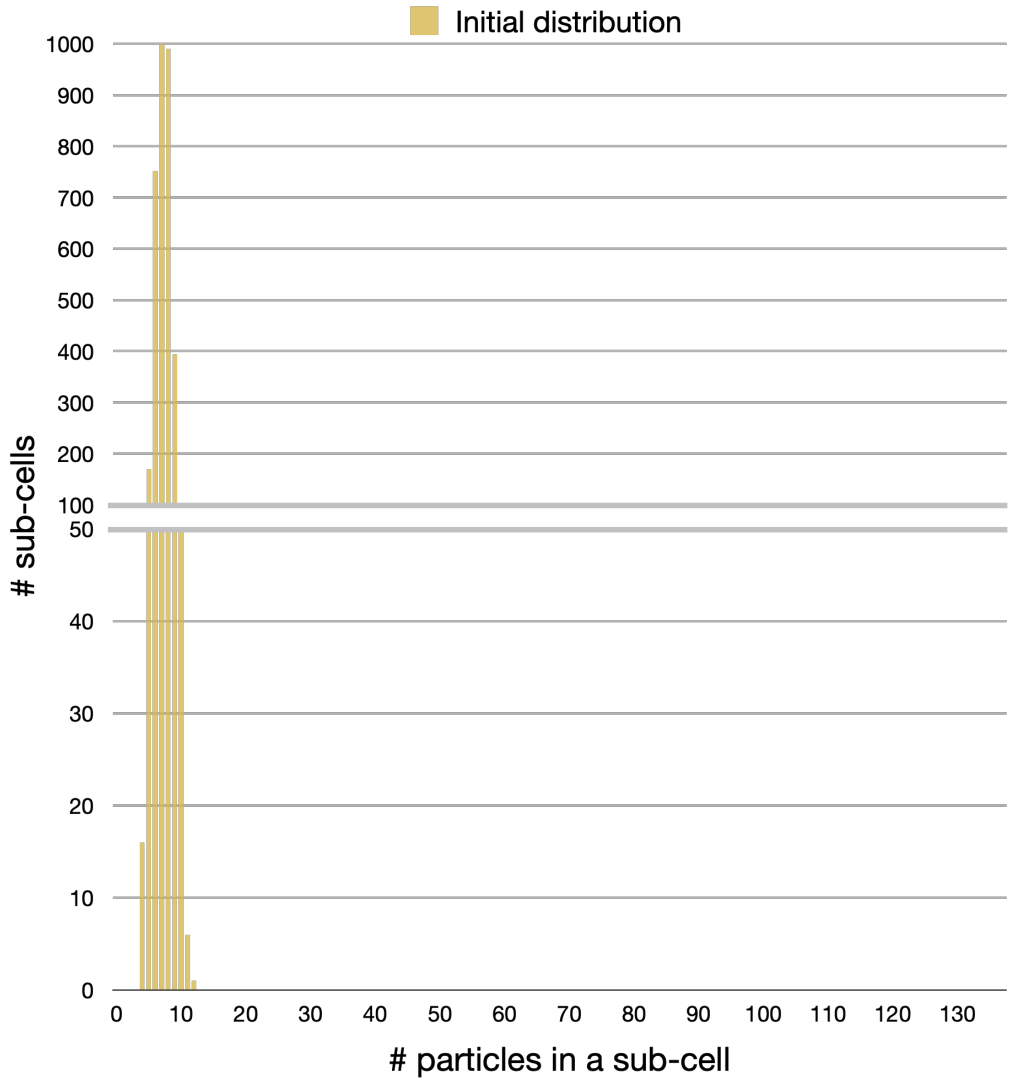


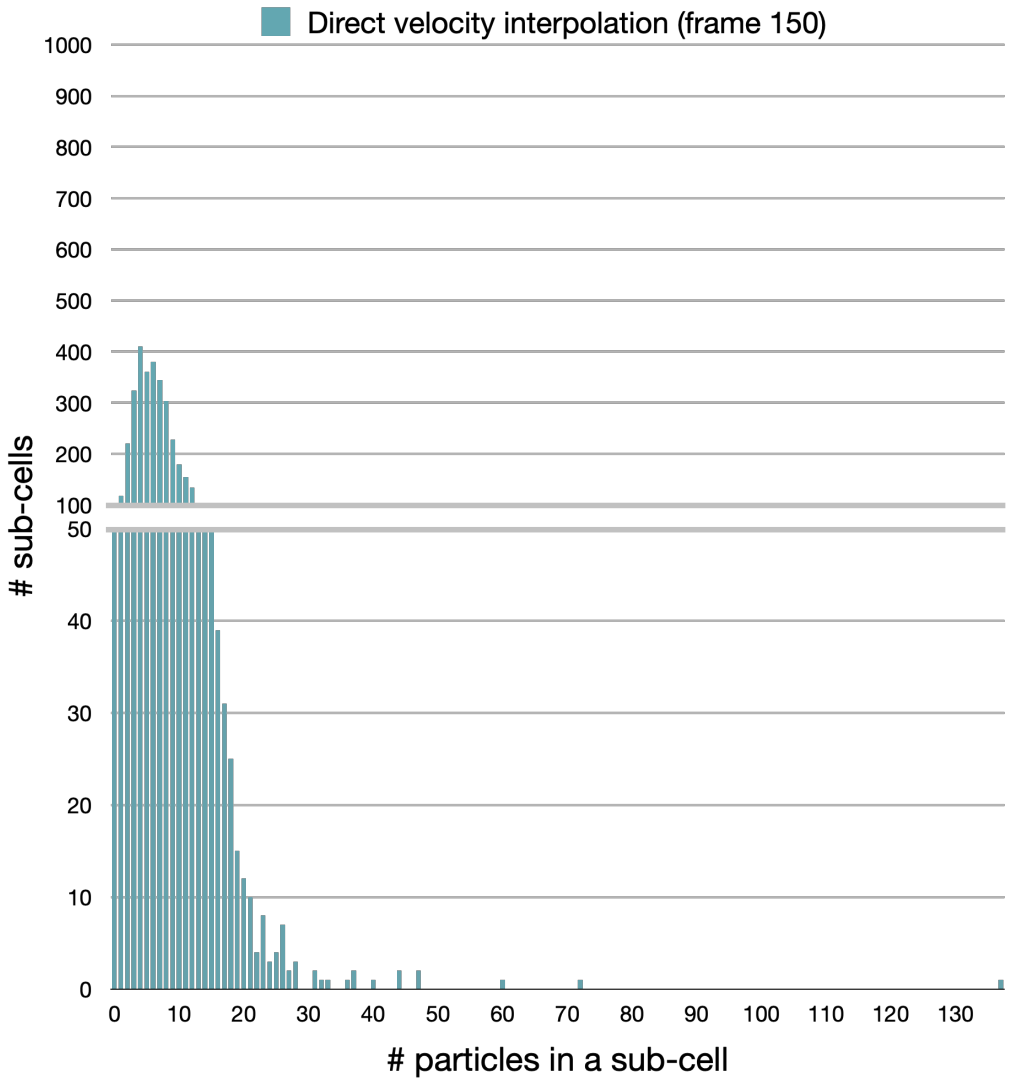
(b) Curl-Flow interpolation

Figure 13.3: **Dynamic 2D Flow Comparison:** Initially uniform particles advected through a dynamic 2D vector field on a 20×20 grid. Frames 1, 50, 100, and 150 are shown from left to right. Particle distribution remains more uniform with our Curl-Flow method.

velocity at the bottom-center of the domain and random velocities (noise) in the entire domain in each frame. The resulting particle distribution is more uniform with our Curl-Flow method. To better quantify this effect, we conceptually lay a finer grid of sub-cells over the domain and count the number of particles per sub-cell in the last frame (frame 150) of the same example, where 3×3 sub-cells form a single grid cell (Figure 13.4). Here, the x-axis indicates the number of particles in a sub-cell, and the y-axis denotes the number of the sub-cells that have the specified number of particles. Initially, most sub-cells contain 6–8 particles, and the distribution spreads over time from the center. In Figure 13.4, the graph for direct velocity interpolation does not decay from the center as rapidly as Curl-Flow, indicating less uniform particle distribution. At the extreme ends, there are 50 empty sub-cells and a sub-cell that contains excessive number of particles (137 particles).

In the *Smooth Obstacle* test of Figure 13.1 (top half), we observe particles under a dynamic horizontal flow past a large circular solid on a coarse 60×20 grid. Spurious gaps


Initial distribution
1000
900
800
700
600
500
400
300
200
100
50
40
30
20
10
0
sub-cells
0
10
20
30
40
50
60
70
80
90
100
110
120
130
particles in a sub-cell


Direct velocity interpolation (frame 150)
1000
900
800
700
600
500
400
300
200
100
50
40
30
20
10
0
sub-cells
0
10
20
30
40
50
60
70
80
90
100
110
120
130
particles in a sub-cell

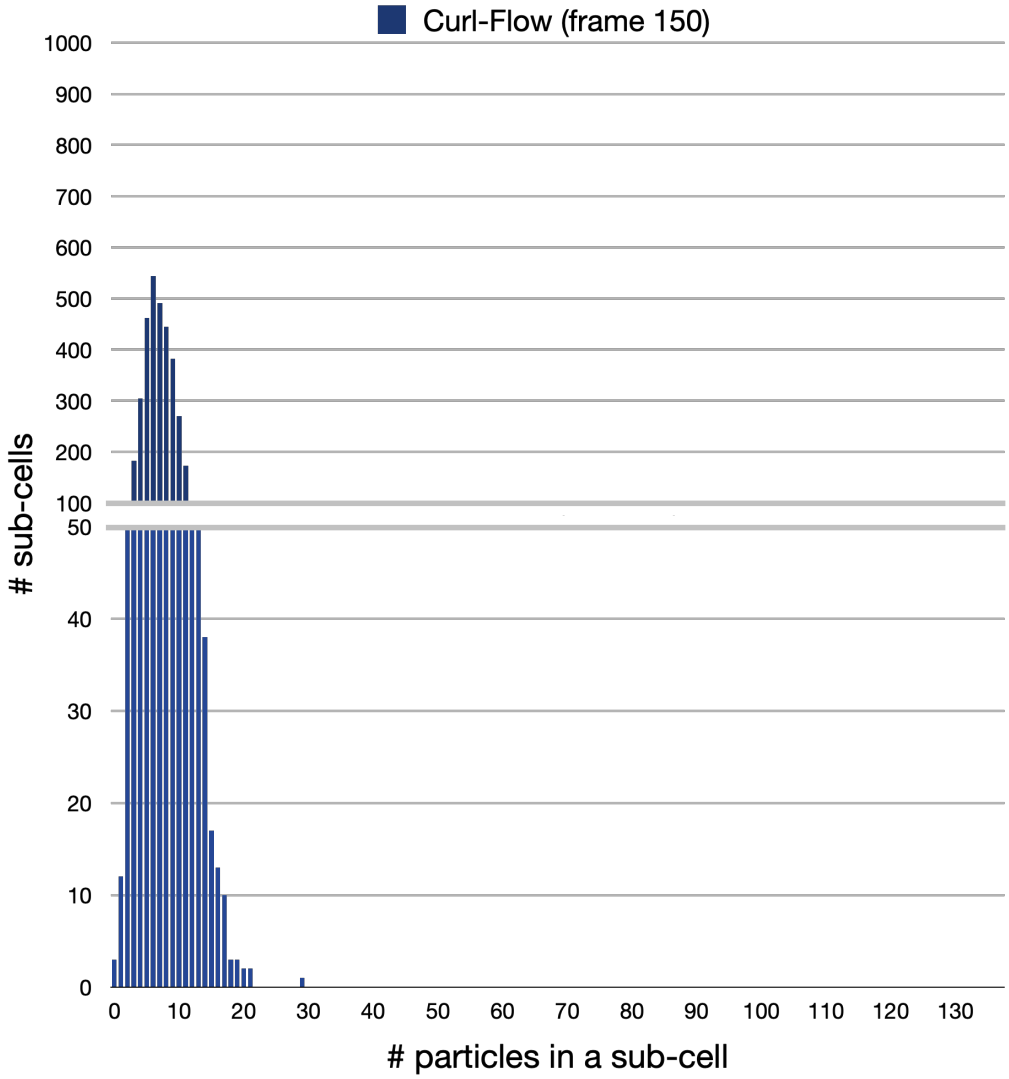


Figure 13.4: **Particle Count Distribution in Sub-cells:** The number of particles in sub-cells is counted in the last frame (frame 150) of the example in Figure 13.3. The width of the sub-cells is 1/3 of the cell width. The x-axis indicates the number of particles in a sub-cell, and the y-axis represents the number of the sub-cells that have the specified number of particles. The graph for direct velocity interpolation shows a wider range of particle counts per sub-cell, indicating less uniform particle distribution as shown in Figure 13.3.

in the initially dense particle sampling quickly arise with direct velocity interpolation. Curl-Flow precisely obeys the boundary while being strictly incompressible, thus dramatically reducing the presence of gaps in the flow. Because interpolation errors depend on cell size, tripling the grid resolution (Figure 13.1, bottom half) proportionally reduces the gap size, but incurs significant cost and such errors persist for any finite resolution.

Our *Jagged Obstacle* test shows the same effect for non-smooth geometry: Figure 13.2 shows a test case with two nearby jagged solid objects in a horizontal flow. With direct velocity interpolation, significant gaps arise behind the obstacle and near high curvature features; Curl-Flow significantly reduces these artifacts.

13.2 Particle Distribution Comparisons in 3D

Although the interaction of three vector potential components somewhat increases the complexity and computational cost of 3D Curl-Flow, we nevertheless obtain a pointwise divergence-free and boundary respecting flow.

Static Flow In A Box To illustrate the effects of incompressibility at a fine scale, Figure 7.1 compares our full Curl-Flow method (including boundary ramping) against direct velocity interpolants with both (componentwise) trilinear and monotonic tricubic (Fedkiw et al., 2001; Fritsch and Carlson, 1980), as these are relatively common choices in fluid animation. The data is a static and coarse $5\times5\times5$ discretely incompressible staggered grid discrete vector field. The direct interpolants quickly exhibit clustering and thinning out of particles, leaving visibly low densities in some regions and denser ring-like patterns in others. By contrast, the Curl-Flow result remains remarkably uniformly distributed for the length of the animation, with no particle resampling, perturbation, or other remedies applied.

Dynamic Wind Tunnel At a larger scale, analogous to the 2D case of Figure 13.1, we compare Curl-Flow to direct velocity interpolation under a 3D dynamic horizontal flow past a sphere (Figure 13.5 shows the particles in a narrow slice plane). Although our particles are purely passive, for illustration we seed them throughout the domain at a density representative of typical particle-in-cell schemes, i.e., 8 per cell (Zhu and Bridson, 2005). The particle distribution soon becomes non-uniform when using direct velocity interpolation, but with our Curl-Flow interpolation the particle distribution remains more uniform, regardless of grid resolution.

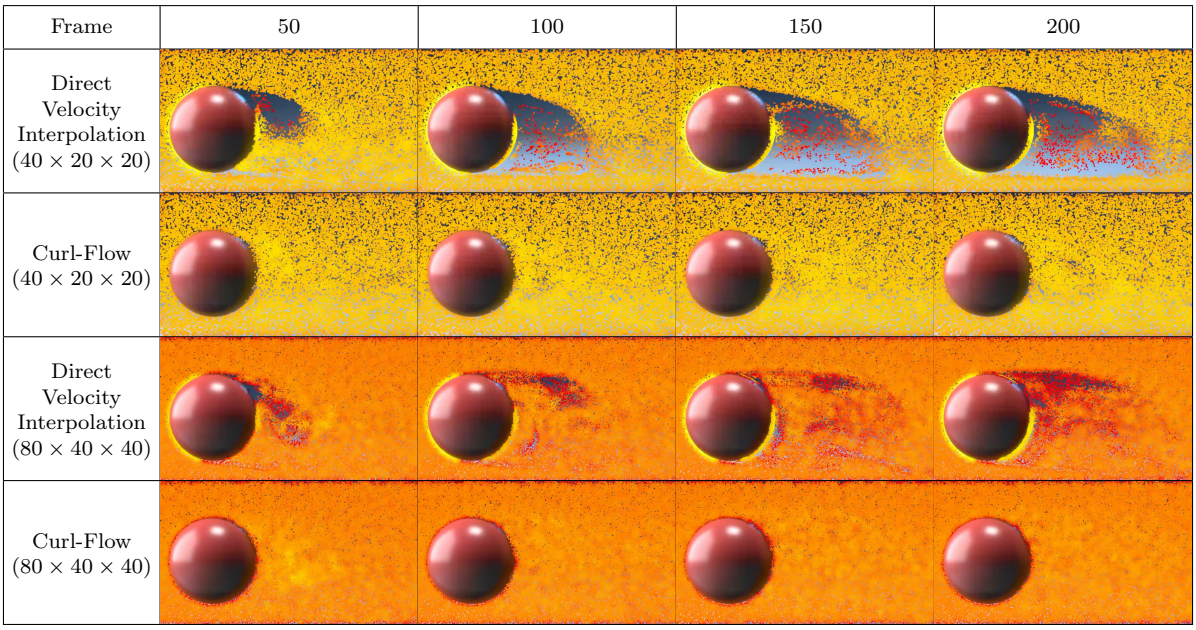


Figure 13.5: **Dynamic 3D Fluid Flow in A Wind Tunnels Past A Sphere:** A slice-plane view of particles undergoing a time-dependent horizontal flow past a spherical object in 3D. The color of the particles represents the particle densities: the density increases from red to yellow. Particles colliding into objects are halted in place for illustration. Regardless of the grid resolution, direct velocity interpolation creates spurious gaps, while our Curl-Flow interpolation tightly follows the solid object, significantly reducing the gaps in the flow and maintaining better particle distribution. The bright yellow regions around the solid obstacle highlight the large number of collisions incurred without using Curl-Flow.

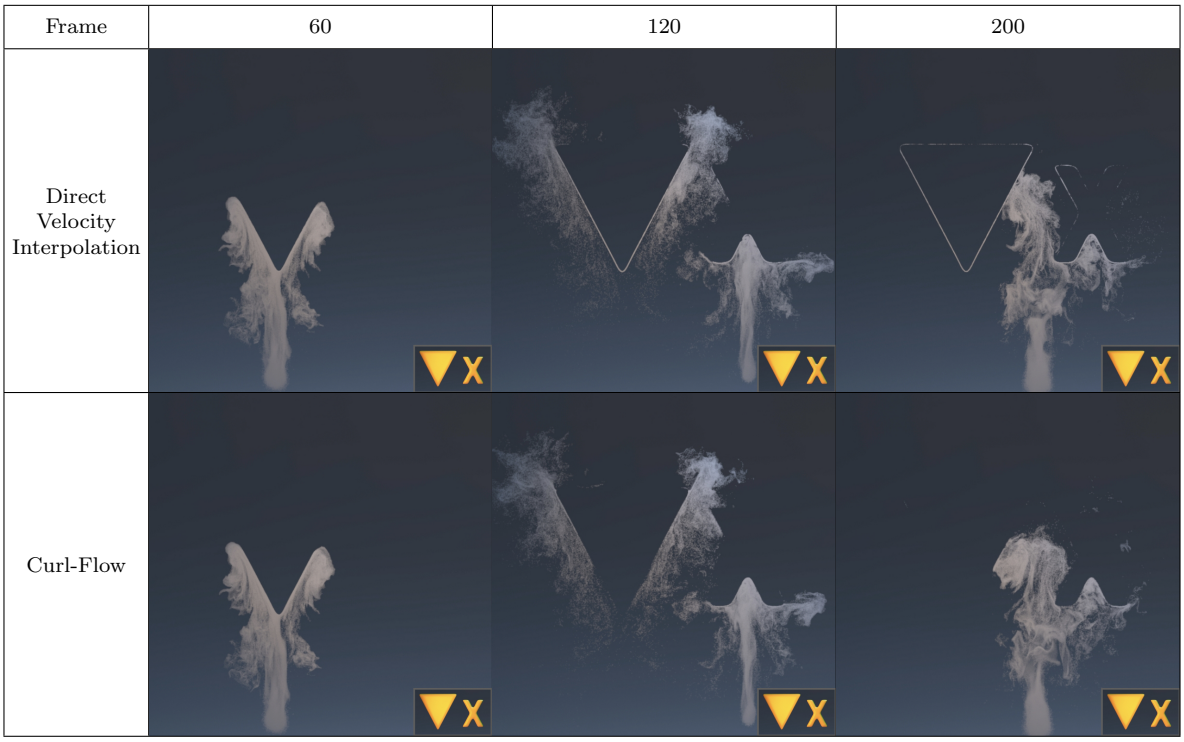


Figure 13.6: **A Smoke Plume Simulation with A Solid Object Shaped like "$\nabla\times$"**. Only a thin slice of smoke is visualized and the obstacles not rendered to better present the boundary behaviors. (The shape of the obstacles are shown at the bottom-right corner.) In the direct velocity interpolation case, a large number of particles erroneously collide with the solid leaving the dense smoke outline around the solid.

13.3 Smoke Simulations and Performance

We consider two rising smoke plume scenarios with different obstacles: a simple object shaped like the curl operator ($\nabla\times$) (Figure 13.6) and a more complex dragon-shaped obstacle (Figure 13.7). We use a 150^3 and 180^3 grid respectively, and the smoke particle counts steadily increase up to 4M. The smoke particles are splatted to a grid for rendering in both examples. Penetrating particles are halted upon collision and these particles do not vanish to highlight the collision. In both examples, Curl-Flow shows better boundary-respecting behavior: direct velocity interpolation yields a dense smoke outline from the hit particles around the solid.

Our ramping method enforces a pointwise boundary-respecting velocity field under

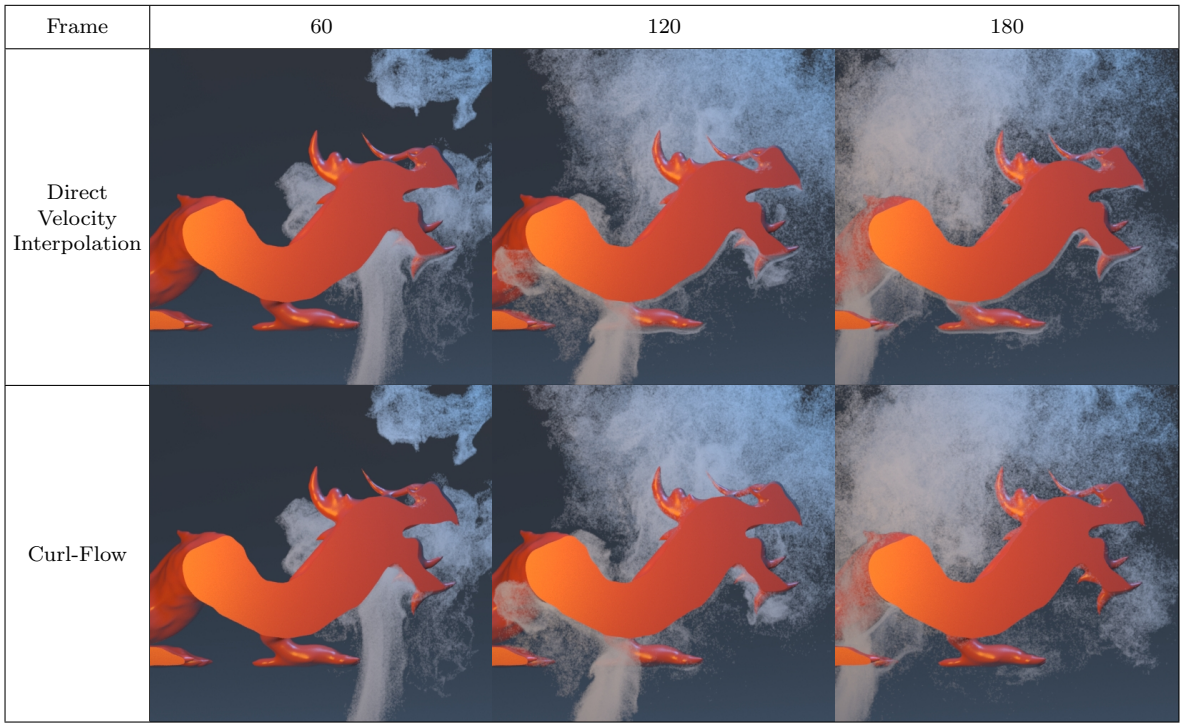


Figure 13.7: **A Smoke Plume Simulation with A Dragon-Shaped Solid**. Half the domain is cut off for better visualization. Curl-Flow method shows better boundary-respecting behavior: the outline of dense smoke from hit particles is conspicuous in the direct velocity interpolation case.

Curl-Flow, but particles advected with discrete timesteps can still collide with solid obstacles, especially under a fast dynamic flow and/or with a complex solid. For the smoke examples, we apply substepping to (only) particle advection to partially ameliorate this innate problem, constraining the timestep so that particles cannot move more than half of a grid cell width per substep. We further use additional substepping upon collision: if a particle hits a solid obstacle, the particle is slightly pushed back from the hit position (10% of the distance the particle moved) and completes a new partial substep from that point for the remaining time. We use this extra substep only once, and if the particle still collides after the extra substep, the particle halts at the hit position. Using even more substeps would further reduce collisions, at the cost of increasing computational time. With direct velocity interpolation we likewise performed basic substepping, but since its vector field is not boundary-respecting, the additional post-collision substepping did not reduce collisions, so we did not use it. In both cases, if the particle lies inside the obstacle after a

	Pressure Projection (s)	Grid $\boldsymbol{\psi}$ Construction (s)	Solid $\boldsymbol{\psi}$ Construction (s)	Particle Advection (s)	Total (s)
Direct Velocity Interpolation	28.816	–	–	0.64722	31.174
Curl-Flow	28.896	20.944	0.20970	6.4323	58.437

Table 13.1: Average computational time per timestep for a smoke plume simulation with a solid object shaped like the curl operator ($\nabla\times$) (Figure 13.6). Grid resolution: 150^3, #solid triangles: 21,568, #solid vertices: 10,788.

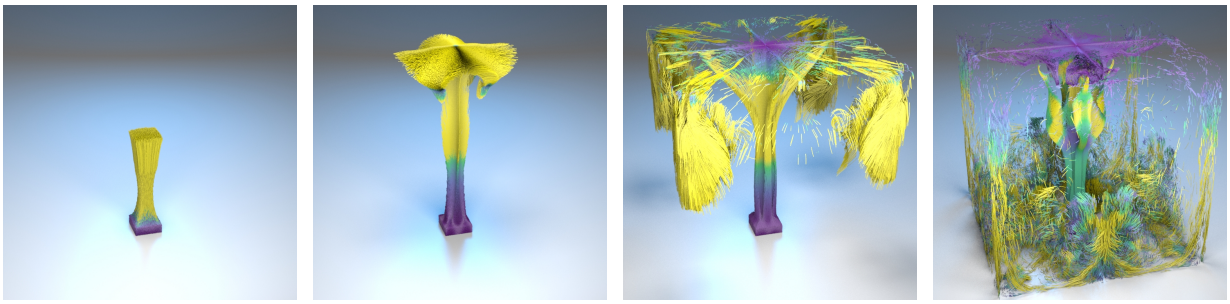
	Pressure Projection (s)	Grid $\boldsymbol{\psi}$ Construction (s)	Solid $\boldsymbol{\psi}$ Construction (s)	Particle Advection (s)	Total (s)
Direct Velocity Interpolation	27.902	–	–	1.214	31.788
Curl-Flow	27.459	20.158	0.241	8.935	59.407

Table 13.2: Average computational time per timestep for a smoke plume simulation with a dragon-shaped solid (Figure 13.7). Grid resolution: 180^3, #solid triangles: 43,872, #solid vertices: 21,936.

(sub)step, it is frozen in place.

To demonstrate Curl-Flow's practicality, these smoke plume tests were carried out by implementing our method into Houdini's smoke solver (*Pyro*), with appropriate modifications (e.g., we use cut-cell methods for pressure projection). Unlike the prior examples, we use a MacCormack scheme (the default in Houdini) for velocity advection. In both examples, Curl-Flow shows superior boundary-respecting behavior: fewer incidents of particle collision with solid obstacles are observed. Compared to standard interpolation, Curl-Flow pays the additional cost of finding discrete vector potentials on the regular grid and on solid vertices. As is evident from Tables 13.1 and 13.2, the net cost on these scenarios was about twice the standard approach. The decision to make this cost-quality tradeoff will be application-dependent, but we consider it worthwhile for scenarios where particle density and boundary fidelity are paramount.

The dominant additional computational expense is enforcing the Coulomb gauge condition. Although this is much faster than the vector Poisson solve of Ando et al., it still requires solving the scalar Poisson equation (12.7), for which we used standard conjugate gradient. Particle advection also contributes to the cost, because Curl-Flow has



Frame 15 Frame 30 Frame 60 Frame 120

Figure 13.8: **A Plume of Particles Using Curl-Flow** : Four different $\boldsymbol{\psi}$ construction methods are adapted in Curl-Flow to produce the rising particle plume scene. All four methods yield identical results, with different computational costs (Table 13.3).

a larger stencil size for interpolation (quadratic vs. linear), and our ramping strategy requires additional operations, e.g., finding the closest point on the solid, evaluating a second interpolation, etc. This cost will be heavily dependent on the number of particles being advected, and could become the bottleneck for massive particle counts. We expect that interpolation could be further optimized. Most importantly, the particle advection can be done in a fully parallel manner. Genuinely passive particles can also often be traced in an entirely parallel post-process, after simulation completes.

To find the vector potential values at the solid vertices ("Solid Ψ Construction" in the Table), we used a direct solver since the system matrix in (12.11) can be ill-conditioned. Nevertheless, the cost of solving (12.11) is relatively small: we can solve it for each closed solid obstacle independently, the solid obstacles only cover part of the simulation domain, and the degrees of freedom lie only on the solid surfaces rather than throughout the volume.

13.4 Vector Potential Reconstruction Comparisons

Several alternative, but more costly, methods to reconstruct edge-based vector potential fields from face-based velocity (or magnetic) fields have previously been proposed (Ando et al., 2015; Bao et al., 2017; Sato et al., 2015; Silberman et al., 2019). Since those methods mostly do not handle cut-cell obstacles, to fairly compare computational costs we consider a flow in an empty box domain. We adapted the methods of Ando et al. and Silberman et al. into our framework as replacements for the $\boldsymbol{\psi}$ reconstruction step, as discussed further below. We consider the combined cost of pressure projection and $\boldsymbol{\psi}$ reconstruction for each

	Pressure Projection (s)	Initial $\boldsymbol{\psi}$ Construction (s)	Gauge Correction (s)	Total $\boldsymbol{\psi}$ Construction (s)
Vector Poisson Dimensionally-Coupled	-	-	-	69.599
Vector Poisson Componentwise	-	-	-	42.960
Pressure Projection + Gauge Correction	17.809	3.398	13.054	34.261
Ours: Pressure Projection + Gauge Correction + Parallel Sweeping	17.825	0.024	11.895	29.744

Table 13.3: Average computational time per timestep for discrete $\boldsymbol{\psi}$ construction on the 150^3 grid simulation of Figure 13.8.

method in Table 13.3, since Ando et al. achieve both simultaneously. Despite the different computational costs, the solutions are numerically consistent (Figure 13.8).

Comparison to Ando et al. (2015): The method of Ando et al. solves a single, dimensionally-coupled vector Poisson problem to recover $\boldsymbol{\psi}$ directly from a *divergent* velocity field, avoiding a separate pressure projection step. We modified this approach to consider our exterior domain boundary conditions (Section 12.3.1). As shown in Table 13.3, its computational cost (topmost row) is expensive compared to our method (bottommost row): about 2.34× slower for the 150^3 grid of Figure 13.8. Unless boundary conditions tie them together (as is often the case), the dimensionally-coupled vector Poisson equation can be decoupled into three independent scalar Poisson equations for efficiency. Applying this special case optimization to our test case, the decoupling gives a speedup (about 62% faster) as compared to solving the coupled vector Poisson equation (Table 13.3, second row), but still about 1.44× slower than ours.

Comparison to Silberman et al. (2019): The reconstruction method of Silberman et al. is most similar to ours: their method recovers the desired vector potential from an initially divergence-free input vector field (albeit in the electromagnetic setting), using a cell-by-cell construction of an initial vector potential followed by Poisson solve-based gauge correction. Silberman et al. assumed the absence of boundary conditions to enable solution by FFT. We adapted this basic idea into our framework by considering our desired boundary conditions and using the same linear solver (conjugate gradient). Like our method, the net cost is two scalar Poisson solves (projection, gauge correction) plus the cost of

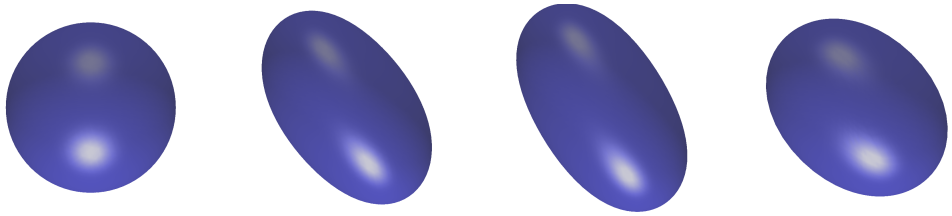


Figure 13.9: **Deforming elastic membrane test from (Bao et al., 2017) :** A spherical membrane immersed in fluid is deformed with an initial velocity and surface tension returns it to a spherical shape. We replace Bao's vector Poisson solve with our $\boldsymbol{\psi}$ construction method. It produces identical results to the original method, but our construction is faster (Table 13.4).

the initial $\boldsymbol{\psi}$ construction (Table 13.3, third row). However, as compared to our sweeping approach, the initial cell-by-cell $\boldsymbol{\psi}$ construction of Silberman et al. is slower because it is inherently serial and it finds all three components of $\boldsymbol{\psi}$ rather than the two needed by our scheme. It also requires more complicated case-by-case code: it first constructs 12 edge values on a cell, and then adjusts them for consistency depending on how many neighbor cells have previously been processed. In practice, the cost of our parallel sweeping strategy is *two orders of magnitude faster* than Silberman et al. (Table 13.3, third column), and essentially negligible compared to pressure projection or gauge correction. Overall, the (adapted) method of Silberman et al. (2019) is 1.15× slower than ours; we emphasize that they also did not handle our boundary conditions nor interior obstacles.

Comparison to Bao et al. (2017): Bao et al. use a vector potential in their immersed boundary method to reduce volume error caused by velocity interpolation and force spreading on the regular grid. Like Silberman et al. (2019) they assume a divergence-free input field, but like Ando et al. (2015) they solve a vector Poisson equation. They assume periodic boundaries and solve using the FFT. Their provided code further exploits the lack of obstacles to decompose the vector problem into three scalar Poisson problems. For a fair comparison, we used their (MATLAB) code and replaced only the discrete $\boldsymbol{\psi}$ construction step with our method. While the vector Poisson solve is not a bottleneck in their problem domain, our method nevertheless gives a speedup for $\boldsymbol{\psi}$ construction of around 2.5× as shown in Table 13.4.

While our application is fluid velocity interpolation, vector potential reconstruction has already been used in graphics (for fluid control (Sato et al., 2021, 2015) and visualization

	$\boldsymbol{\psi}$ construction time (s)
Bao et al. (2017)	2.3239×10^{-3}
Curl-Flow	0.9397×10^{-3}

Table 13.4: Average computational time for constructing $\boldsymbol{\psi}$, within the code of Bao et al. (2017) for an immersed deforming membrane. For the "Curl-Flow" comparison, we replace only the $\boldsymbol{\psi}$ construction part.

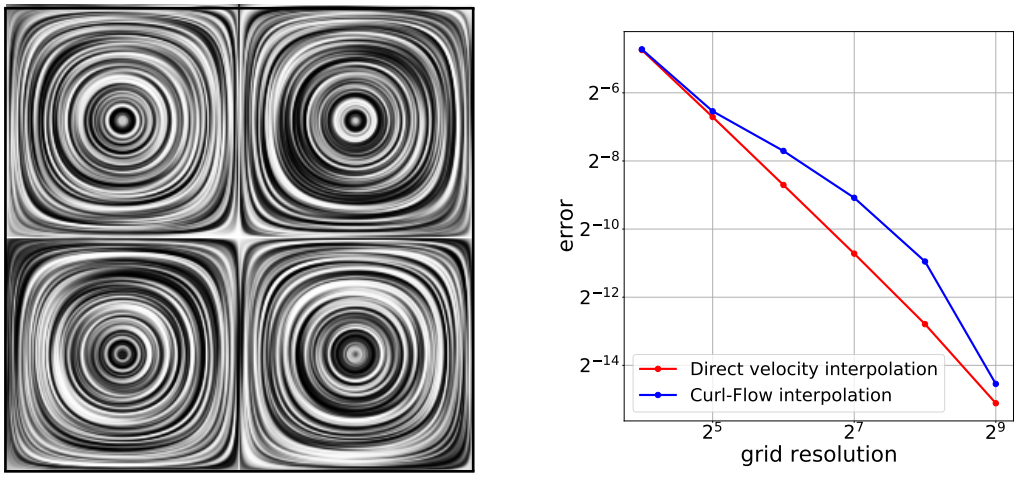
Grid Resolution	16^2	32^2	64^2	128^2	256^2	512^2
Direct Velocity Interpolation (L_∞)	3.733e-02 (–)	9.554e-03 (1.966)	2.396e-03 (1.996)	5.927e-04 (2.015)	1.412e-04 (2.070)	2.824e-05 (2.322)
Curl-Flow (L_∞)	3.787e-02 (–)	1.073e-02 (1.819)	4.789e-03 (1.165)	1.845e-03 (1.376)	5.050e-04 (1.870)	4.183e-05 (3.594)

Table 13.5: Data for convergence test of Figure 13.10: error and the order of convergence (within parentheses) are measured using a L_∞ norm under grid refinement. Both direct (bilinear) velocity interpolation and Curl-Flow interpolation yield similar convergence.

(Biswas et al., 2016)) as well as electromagnetics and other fields. Our new approach can thus offer immediate speedups in all of these domains. e.g., potentially more than doubling the speed of the most expensive step in the recent method of (Sato et al., 2021), even without adopting more elaborate optimizations (e.g., multigrid, GPU implementation, etc.)

13.5 Convergence of Curl-Flow Velocity Interpolation

Since our Curl-Flow interpolation method uses intermediate vector potentials rather than directly interpolating grid velocities, it is reasonable to ask if it is a valid interpolant at all; that is, does the resulting continuous field converge? We study this question by evaluating how well our interpolated field agrees with an input analytically divergence-free field under refinement. Since the input to interpolation should be a discretely divergence-free velocity field defined on the staggered grid, we first sample grid velocities from the input analytical field, enforce discrete incompressibility on them via pressure projection, apply the chosen interpolation scheme (at a much finer sampling of points), and evaluate their error with



(a) Analytical solution

(b) L_∞ norm

Figure 13.10: **Convergence of Velocity Interpolation:** We sample discrete grid velocities from the analytically divergence-free field of (a), enforce discrete incompressibility, interpolate using direct (bilinear) velocity interpolation and Curl-Flow interpolation, and measure the error. Both methods show approximately second order convergence. The error data is provided in Table 13.5.

respect to the input. For the analytically divergence-free field, we use the velocity field

$$\mathbf{u}(x, y) = (\sin(2\pi x)\cos(2\pi y), -\cos(2\pi x)\sin(2\pi y)),$$

shown in Figure 13.10(a). Since we use a quadratic kernel for Curl-Flow interpolation, we often need to query the samples outside the domain. We linearly extrapolate ψ values to the samples outside the domain for the convergence test. Direct (bilinear) velocity interpolation yields second order convergence, and Curl-Flow also shows similar convergence (while additionally providing exact incompressibility) (Figure 13.10).

Chapter 14

Conclusions

We have presented novel approaches for pointwise divergence-free fluid advection and unified elastic object simulation. These topics have been barely studied in graphics, but they are of importance as they provide necessary or desired features that increase the versatility and quality of animation results that effects artists can achieve. Also, our proposed approaches require minimal modifications to industrially prevalent solvers so they have the potential to be easily adopted into existing animation software. Specifically, our elastic coupling strategies do not depend on any particular single-type solvers and they are designed to integrate naturally with familiar methods for single-type solvers. Our new fluid interpolation method only influences one component of the entire fluid simulation pipeline, which is the advection step, thus the remaining parts of the solvers can be left untouched.

14.1 Unified Elastic Objects

We have presented an expressive and practical approach for designing and simulating diverse non-manifold and mixed-dimensional elastic objects that effectively leverages well-studied continuum models of single-dimensional elastica. The user simply models an object's desired geometry with a single conforming possibly non-manifold simplicial mesh, and then labels its simplices with model and connection types to achieve the desired target behavior.

Our assumption of a conforming mesh limits us to situations in which the two sides of a connection (or the connection itself) cannot slide relative to the bodies. In previous work, Chentanez et al. (2009) explored a method to allow a Lagrangian rod to slide while

constrained within a tetrahedral mesh, and Weidner et al. (2018) enabled smooth sliding of Lagrangian cloth relative to contact points; these and related Eulerian-on-Lagrangian simulation concepts (Fan et al., 2013) may prove useful in tackling this challenge. The conforming mesh assumption also disallows connections in the interior of an element, such as a rod endpoint joined at the middle of a shell triangle.

We adopted three specific single-type elastic models for rods, shells, and solids. We expect that other options would integrate equally well, since our coupling connections place few limits on the chosen models beyond requiring simplicial meshes. Similarly, extending the single-type models with additional features or constitutive laws (e.g., strain limiting, incompressibility, plasticity) would be largely orthogonal to the proposed approach. However, designing connections which themselves have exotic elastoviscoplasticity or angle limits may be interesting to consider in future.

Another exciting avenue to explore is the extension of our framework to mixed-dimensional *liquids*, analogous to work on viscous sheets and threads (Batty et al., 2012; Bergou et al., 2010). While non-manifold simplicial liquid models have been suggested (Zhu et al., 2015, 2014), an approach along the lines of our framework could naturally support the characteristic twisting and bending forces (and associated buckling and coiling effects) of viscous and non-Newtonian sheets and threads, which Zhu et al. omitted. Naturally this would require incorporating mixed-dimensional dynamic remeshing, as Zhu et al. (2014) have previously demonstrated.

Despite the attractive simplicity of our point, curve, and surface connections and the commonality among the various energies, the construction of appropriate coordinate frames near connections nevertheless involves special cases that depend on the particular configuration. It is interesting to consider whether it may be possible to find an even more concise set of "atomic" stencils or energies that can be re-assembled to recover all possible models and couplings, thereby further streamlining the methodology. Finally, our approach also reveals some user interface challenges: geometric modeling tools for general non-manifold objects are less well-studied than for surface meshes, and achieving the desired deformation behavior by tagging simplices with model and connection types remains a somewhat labor-intensive process.

14.2 Curl-Flow

Large time steps in particle trajectory integration, poorly enforced boundaries, and (optionally) inadequate pressure solver tolerances have long been known to cause density/volume

drift in grid-based fluid animation; accordingly, a host of post-compensation strategies have been developed (Ando et al., 2012; Kugelstadt et al., 2019; Sato et al., 2018c; Takahashi and Lin, 2019). However, we believe our work is the first in computer graphics to identify divergent velocity interpolation as another contributing factor. Our Curl-Flow interpolation framework, tailored to plug into popular grid-based cut-cell fluid animation tools, addresses this issue by globally guaranteeing pointwise divergence-free velocities. The resulting flows offer better long-term particle distributions and natural motions around obstacles. We believe our work opens a previously unexplored dimension in the design of advection schemes for fluid animation; our method's current limitations suggest exciting questions for investigation.

We considered only static obstacles. It is straightforward to support nonzero boundary fluxes during vector potential reconstruction; however, the Lagrangian nature of obstacle motion raises intriguing questions about collision-safe time integration of trajectories, especially for intermediate substeps of Runge-Kutta. A deforming mesh strategy that carries the potential with it is a possible avenue. Similarly, we have only discussed grid domains that are either fully open, fully closed, or "wind-tunnel"-like. More general boundary conditions would be an obvious extension, including free surface boundaries for liquid animation where our method could improve volume conservation.

We focused on the effects of incompressible interpolation on passive particle trajectories. However, our stronger enforcement of the continuity equation might also offer visual benefits for advection of the velocity field itself and/or grid-based scalar fields like temperature or density. Because particles remain better distributed, we likely will not have to frequently resample to avoid empty regions that do not get assigned valid velocities (Maljaars et al., 2018).

A minor drawback of Curl-Noise-based boundary treatments is that, though always incompressible, they can yield velocity discontinuities when the closest polygon facet changes. Also, when ramping $\boldsymbol{\psi}$ towards solid surfaces, we sought a minimal perturbation to $\boldsymbol{\psi}$ and found that this criterion yields visually natural results. However, different criteria could be applied (e.g., controlling $\nabla \times \boldsymbol{\psi}_t$) to the optimization.

While not an issue for passive particle advection, interpolation of higher order than linear can yield overshooting. Using this method for velocity advection will neccessitate adopting some form of limiting. Simple velocity clamping as done by Selle et al. (2008a) is straightforward, but sacrifices incompressibility. A more attractive option may be to fall back to the linear variant of Curl-Flow in offending regions, preserving incompressibility at the cost of localized kinks.

Lastly, because of the high cost and limited controllability of fluid simulations, the

ability to edit simulations in an *efficient* post-process remains highly desirable (Pan et al., 2013; Sato et al., 2018b). Our vector potential-based boundary correction approach, as well as the work of Sato et al. (2021, 2015), suggests that exploiting vector potentials may enable other fast, divergence-free editing capabilities by combining vector potential-based *procedural* tools with vector potential-based *simulation* tools.

References

Mridul Aanjaneya, Ming Gao, Haixiang Liu, Christopher Batty, and Eftychios Sifakis. 2017. Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Transactions on Graphics* 36 (07 2017), 1–12. https://doi.org/10.1145/3072959.3073625

Nadir Akinci, Markus Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner. 2012. Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics (TOG)* 31 (2012), 1 – 8.

R. Albanese and G. Rubinacci. 1990. Magnetostatic field computations in terms of two-component vector potentials. *Internat. J. Numer. Methods Engrg.* 29, 3 (March 1990), 515–532. https://doi.org/10.1002/nme.1620290305

Ryoichi Ando, Nils Thuerey, and Chris Wojtan. 2015. A stream function solver for liquid simulations. *ACM Transactions on Graphics* 34, 4 (jul 2015), 53:1–53:9. https://doi.org/10.1145/2766935

Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. 2012. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE transactions on visualization and computer graphics* 18, 8 (2012), 1202–1214.

Alexis Angelidis and Fabrice Neyret. 2005. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 87–96.

Baptiste Angles, Daniel Rebain, Miles Macklin, Brian Wyvill, Loic Barthe, J P Lewis, Javier Von Der Pahlen, Shahram Izadi, Julien Valentin, Sofien Bouaziz, and Andrea Tagliasacchi. 2019. VIPER: Volume Invariant Position-based Elastic Rods. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* (2019). https://doi.org/10.1145/3340260

Vinicius C Azevedo, Christopher Batty, and Manuel M Oliveira. 2016. Preserving geometry and topology for fluid flows with thin obstacles and narrow gaps. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 97.

Dinshaw S Balsara. 2001. Divergence-free adaptive mesh refinement for magnetohydrodynamics. *J. Comput. Phys.* 174, 2 (2001), 614–648.

Dinshaw S Balsara. 2004. Second-order-accurate schemes for magnetohydrodynamics with divergence-free reconstruction. *The Astrophysical Journal Supplement Series* 151, 1 (2004), 149.

Dinshaw S Balsara. 2009. Divergence-free reconstruction of magnetic fields and WENO schemes for magnetohydrodynamics. *J. Comput. Phys.* 228, 14 (2009), 5040–5056.

Yuanxun Bao, Aleksandar Donev, Boyce E Griffith, David M McQueen, and Charles S Peskin. 2017. An Immersed Boundary method with divergence-free velocity interpolation and force spreading. *Journal of computational physics* 347 (2017), 183–206.

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, New York, NY, USA, 43–54. https://doi.org/10.1145/280814.280821

Christopher Batty, Florence Bertails, and Robert Bridson. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.* 26, 3 (2007), 100.

Christopher Batty and Robert Bridson. 2008. Accurate Viscous Free Surfaces for Buckling, Coiling, and Rotating Liquids. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Dublin, Ireland) *(SCA '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 219–228. http://dl.acm.org/citation.cfm?id=1632592.1632624

Christopher Batty, Andres Uribe, Basile Audoly, and Eitan Grinspun. 2012. Discrete viscous sheets. *ACM Trans. Graph. (SIGGRAPH)* 31, 4 (2012), 113.

Markus Becker and Matthias Teschner. 2007. Weakly Compressible SPH for Free Surface Flows. *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* 9, 209–217. https://doi.org/10.1145/1272690.1272719

G. Beer. 1985. An isoparametric joint/interface element for finite element analysis. *Internat. J. Numer. Methods Engrg.* 21, 4 (1985), 585–600.

Jan Bender and Crispin Deul. 2013. Adaptive cloth simulation using corotational finite elements. *Computers & Graphics* 37, 7 (2013), 820–829. https://doi.org/10.1016/j.cag.2013.04.008

Jan Bender and Dan Koschier. 2015. Divergence-Free Smoothed Particle Hydrodynamics. https://doi.org/10.1145/2786784.2786796

Miklos Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. 2010. Discrete viscous threads. *ACM Trans. Graph. (SIGGRAPH)* 29, 4 (2010), 116.

Miklos Bergou, Max Wardetzky, Stephen Robinson, Basile Audoly, and Eitan Grinspun. 2008. Discrete elastic rods. *ACM Trans. Graph. (SIGGRAPH)* 27, 3 (2008), 63.

Florence Bertails. 2009. Linear time super-helices. *Computer Graphics Forum (Eurographics)* 28, 2 (2009), 417–426.

Florence Bertails, Basile Audoly, Marie-Paule Cani, Frédéric Leroy, Bernard Querleux, and Jean-Luc Lévêque. 2006. Super-helices for predicting the dynamics of natural hair. *ACM Trans. Graph. (SIGGRAPH)* 25, 3 (jul 2006), 1180–1187. https://doi.org/10.1145/1141911.1142012

Ayan Biswas, Richard Strelitz, Jonathan Woodring, Chun-Ming Chen, and Han-Wei Shen. 2016. A scalable streamline generation algorithm via flux-based isocontour extraction. In *Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization*. 69–78.

Javier Bonet and Richard D. Wood. 2008. *Nonlinear Continuum Mechanics for Finite Element Analysis* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511755446

Allan Bower. 2009. *Applied Mechanics of Solids*. 1–795 pages. https://doi.org/10.1201/9781439802489

Robert Bridson. 2015. *Fluid Simulation for Computer Graphics, Second Edition*. Taylor & Francis.

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. (SIGGRAPH)* 21, 3 (2002), 594–603.

Robert Bridson, Jim Houriham, and Marcus Nordenstam. 2007. Curl-Noise for Procedural Fluid Flow. *ACM Trans. Graph.* 26, 3 (July 2007), 46–es. https://doi.org/10.1145/1276377.1276435

Robert Bridson, Sebastian Marino, and Ronald Fedkiw. 2003. Simulation of clothing with folds and wrinkles. In *Symposium on Computer Animation*. Eurographics Association, 28–36.

Tyson Brochu and Robert Bridson. 2009. Robust Topological Operations for Dynamic Explicit Surfaces. *SIAM J. Scientific Computing* 31 (01 2009), 2472–2493. https://doi.org/10.1137/080737617

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

Hugo Casquero, Yongjie Jessica Zhang, Carles Bona-Casas, Lisandro Dalcin, and Hector Gomez. 2018. Non-body-fitted fluid–structure interaction: Divergence-conforming B-splines, fully-implicit dynamics, and variational formulation. *J. Comput. Phys.* 374 (2018), 625–653.

Jumyung Chang, Vinicius C. Azevedo, and Christopher Batty. 2021. Curl-Flow: Pointwise Incompressible Velocity Interpolation for Grid-Based Fluids. *CoRR* abs/2104.00867 (2021). arXiv:2104.00867 https://arxiv.org/abs/2104.00867

Jumyung Chang, Fang Da, Eitan Grinspun, and Christopher Batty. 2019. A Unified Simplicial Model for Mixed-Dimensional and Non-Manifold Deformable Elastic Objects. *Proc. ACM Comput. Graph. Interact. Tech.* 2, 2, Article 11 (July 2019), 18 pages. https://doi.org/10.1145/3340252

Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A Simple Geometric Model for Elastic Deformations. 29, 4 (2010). https://doi.org/10.1145/1778765.1778775

Yi-Lu Chen, Jonathan Meier, B. Solenthaler, and V. C. Azevedo. 2020. An extended cut-cell method for sub-grid liquids tracking with surface tension. *ACM Transactions on Graphics (TOG)* 39 (2020), 1 – 13.

Nuttapong Chentanez, Ron Alterovitz, Daniel Ritchie, Lita Cho, Kris K. Hauser, Ken Goldberg, Jonathan R. Shewchuk, and James F. O'Brien. 2009. Interactive simulation

of surgical needle insertion and steering. *ACM Trans. Graph. (SIGGRAPH)* 28, 3 (2009), 88.

Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but Responsive Cloth. *ACM Transactions on Graphics* 21 (07 2002). https://doi.org/10.1145/1198555.1198571

Alexandre Joel Chorin. 1968. Numerical Solution of the Navier-Stokes Equations. *Math. Comp.* 22, 104 (1968), 745–762. http://www.jstor.org/stable/2004575

Fehmi Cirak and Quan Long. 2011. Subdivision shells with exact boundary control and non-manifold geometry. *Int. J. Numer. Methods Eng.* 88, 9 (2011), 897–923.

Bernardo Cockburn, Fengyan Li, and Chi-Wang Shu. 2004. Locally divergence-free discontinuous Galerkin methods for the Maxwell equations. *J. Comput. Phys.* 194, 2 (2004), 588–610.

Qiaodong Cui, Pradeep Sen, and Theodore Kim. 2018. Scalable laplacian eigenfluids. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 87.

Fang Da, Christopher Batty, and Eitan Grinspun. 2014. Multimaterial Mesh-Based Surface Tracking. *ACM Trans. on Graphics (SIGGRAPH 2014)* (2014).

Fang Da, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2015. Double bubbles sans toil and trouble: Discrete circulation-preserving vortex sheets for soap films and foams. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 149.

Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2016. Surface-only liquids. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–12.

Fernando De Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. 2016. Subdivision exterior calculus for geometry processing. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Fernando De Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 50:1–50:11.

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid simulation using Laplacian eigenfunctions. *ACM Transactions on Graphics (TOG)* 31, 1 (2012), 10.

Ivan DeWolf. 2006. *Divergence-free noise.* Technical Report. Technical report, Martian Labs., 2005.

Ounan Ding, Tamar Shinar, and Craig Schroeder. 2020. Affine particle in cell method for MAC grids and fluid simulation. *J. Comput. Phys.* 408 (2020), 109311. https://doi.org/10.1016/j.jcp.2020.109311

Todd F. Dupont and Yingjie Liu. 2003. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *J. Comput. Phys.* 190, 1 (2003), 311–324. https://doi.org/10.1016/S0021-9991(03)00276-6

David Eberle. 2018. Better collisions and faster cloth for Pixar's Coco. 1–2. https://doi.org/10.1145/3214745.3214801

Marvin Eisenberger, Zorah Lähner, and Daniel Cremers. 2018. Divergence-Free Shape Interpolation and Correspondence. *arXiv preprint arXiv:1806.10417* (2018).

Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. 2007. Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics (TOG)* 26, 1 (2007), 4.

Douglas Enright, Stephen Marschner, and Ronald Fedkiw. 2002. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 736–744.

Olaf Etzmuß, Michael Keckeisen, and Wolfgang Straßer. 2003. A Fast Finite Element Solution for Cloth Modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG '03)*. IEEE Computer Society, USA, 244.

John A Evans and Thomas JR Hughes. 2013. Isogeometric divergence-conforming B-splines for the unsteady Navier–Stokes equations. *J. Comput. Phys.* 241 (2013), 141–167.

Ye Fan, Joshua Litven, David I. W. Levin, and Dinesh K. Pai. 2013. Eulerian-on-Lagrangian simulation. *ACM Trans. Graph. (SIGGRAPH)* 32, 3 (2013), 22.

François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K. Pai. 2011. Sparse meshless models of complex deformable solids. *ACM Trans. Graph. (SIGGRAPH)* 30, 4 (2011), 73.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual Simulation of Smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 15–22. https://doi.org/10.1145/383259.383260

Nick Foster and Dimitris Metaxas. 1996. Realistic Animation of Liquids. *CVGIP: Graphical Model and Image Processing* 58 (01 1996), 471–483.

Michael Frewer, Martin Oberlack, and Vladimir Grebenev. 2014. The Dual Stream Function Representation of an Ideal Steady Fluid Flow and its Local Geometric Structure. *Mathematical Physics Analysis and Geometry* 17 (03 2014). https://doi.org/10.1007/s11040-014-9138-5

F. N. Fritsch and R. E. Carlson. 1980. Monotone Piecewise Cubic Interpolation. *SIAM J. Numer. Anal.* 17, 2 (1980), 238–246.

Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. 2017. A Polynomial Particle-in-cell Method. *ACM Trans. Graph.* 36, 6, Article 222 (Nov. 2017), 12 pages. https://doi.org/10.1145/3130800.3130878

Benjamin Gilles, Guillaume Bousquet, François Faure, and Dinesh K. Pai. 2011. Frame-based elastic models. *ACM Trans. Graph.* 30, 2 (2011), 15.

Yotam Gingold, Adrian Secord, Jefferson Y. Han, Eitan Grinspun, and Denis Zorin. 2004. *A discrete model for inelastic deformation of thin shells.* Technical Report. New York University. 12 pages.

Christoph Gissler, A. Henne, S. Band, A. Peer, and M. Teschner. 2020. An implicit compressible SPH solver for snow simulation. *ACM Transactions on Graphics (TOG)* 39 (2020), 36:1 – 36:16.

Christoph Gissler, Andreas Peer, Stefan Band, Jan Bender, and Matthias Teschner. 2019. Interlinked SPH Pressure Solvers for Strong Fluid-Rigid Coupling. *ACM Transactions on Graphics* 38 (01 2019), 1–13. https://doi.org/10.1145/3284980

Ryan Goldade, Mridul Aanjaneya, and Christopher Batty. 2020. Constraint bubbles and affine regions: reduced fluid models for efficient immersed bubbles and flexible spatial coarsening. *ACM Transactions on Graphics* 39 (07 2020). https://doi.org/10.1145/3386569.3392455

Eitan Grinspun, Anil N. Hirani, Peter Schröder, and Mathieu Desbrun. 2003. Discrete shells. In *Symposium on Computer Animation*. Eurographics Association, 62–67.

Johnny Guzmán and Michael Neilan. 2014. Conforming and divergence-free Stokes elements in three dimensions. *IMA J. Numer. Anal.* 34, 4 (2014), 1489–1508.

Francis H. Harlow and J. Eddie Welch. 1965. Numerical Calculation of Time Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The Physics of Fluids* 8, 12 (1965), 2182–2189. https://doi.org/10.1063/1.1761178

Thomas Hou and Brian Wetton. 2009. Stable Fourth Order Stream-Function Methods for Incompressible Flows with Boundaries. *Journal of Computational Mathematics* 27 (07 2009). https://doi.org/10.4208/jcm.2009.27.4.012

Ben Houston, Chris Bond, and Mark Wiebe. 2003. A unified approach for modeling complex occlusions in fluid simulations. In *ACM SIGGRAPH 2003 Sketches & Applications*. 1–1.

Libo Huang, Torsten Hädrich, and Dominik Michels. 2019. On the accurate large-scale simulation of ferrofluids. *ACM Transactions on Graphics* 38 (07 2019), 1–15. https://doi.org/10.1145/3306346.3322973

Antonio Huerta, Yolanda Vidal, and Pierre Villon. 2004. Pseudo-divergence-free element free Galerkin method for incompressible fluid flow. *Computer Methods in Applied Mechanics and Engineering* 193, 12 (2004), 1119 – 1136. https://doi.org/10.1016/j.cma.2003.12.010 Meshfree Methods: Recent Advances and New Applications.

Geoffrey Irving, J. Teran, and Ronald Fedkiw. 2004. Invertible finite elements for robust simulation of large deformation. In *SCA '04*.

Paul M Isaacs and Michael F Cohen. 1987. Controlling dynamic simulation with kinematic constraints. *ACM SIGGRAPH* 21, 4 (1987), 215–224.

P Jenny, SB Pope, M Muradoglu, and DA Caughey. 2001. A hybrid algorithm for the joint PDF equation of turbulent reactive flows. *J. Comput. Phys.* 166, 2 (2001), 218–252.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 51.

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

J.U.Brackbill and H.M.Ruppel. 1986. FLIP: a method for adaptively zoned, particle-in-cell calculuations of fluid flows in two dimensions. *J. Comput. Phys.* 65, 2 (1986), 314–343.

Jonathan M. Kaldor, Doug L. James, and S. Marschner. 2010. Efficient yarn-based cloth with adaptive contact linearization. *ACM SIGGRAPH 2010 papers* (2010).

T. Keeler and R. Bridson. 2015. Ocean waves animation using boundary integral equations and explicit mesh tracking. In *SCA '14*.

Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. 2007. Simulation of bubbles in foam with the volume control method. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 98–es.

ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. 2005. FlowFixer: Using BFECC for Fluid Simulation. In *Eurographics Workshop on Natural Phenomena*, Pierre Poulin and Eric Galin (Eds.). The Eurographics Association. https://doi.org/10.2312/NPH/NPH05/051-056

Theodore Kim. 2020. A Finite Element Formulation of Baraff-Witkin Cloth. *Computer Graphics Forum* 39 (12 2020), 171–179. https://doi.org/10.1111/cgf.14111

Theodore Kim and D. Eberle. 2020. Dynamic deformables: implementation and production practicalities. *ACM SIGGRAPH 2020 Courses* (2020).

Theodore Kim, Jerry Tessendorf, and Nils Thürey. 2013. Closest Point Turbulence for Liquid Surfaces. 32, 2 (2013). https://doi.org/10.1145/2451236.2451241

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 2008. Wavelet Turbulence for Fluid Simulation. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–6. https://doi.org/10.1145/1360612.1360649

Tassilo Kugelstadt, Dan Koschier, and Jan Bender. 2018. Fast Corotated FEM using Operator Splitting. *Computer Graphics Forum (SCA)* 37, 8 (2018).

Tassilo Kugelstadt, Andreas Longva, Nils Thurey, and Jan Bender. 2019. Implicit Density Projection for Volume Conserving Liquids. *IEEE Computer Architecture Letters* 01 (2019), 1–1.

T. Kugelstadt and E. Schömer. 2016. Position and orientation based Cosserat rods. In *Symposium on Computer Animation*. 169–178.

Egor Larionov, Christopher Batty, and Robert Bridson. 2017. Variational stokes: a unified pressure-viscosity solver for accurate viscous liquids. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 101.

Philip L Lederer, Alexander Linke, Christian Merdon, and Joachim Schoberl. 2017. Divergence-free reconstruction operators for pressure-robust Stokes discretizations with continuous pressure finite elements. *SIAM J. Numer. Anal.* 55, 3 (2017), 1291–1314.

Christoph Lehrenfeld and Joachim Schöberl. 2016. High order exactly divergence-free hybrid discontinuous Galerkin methods for unsteady incompressible flows. *Computer Methods in Applied Mechanics and Engineering* 307 (2016), 339–361.

Faming Li, Xiaowu Chen, Lin Wang, and Qinping Zhao. 2014. Canopy-frame interactions for umbrella simulation. *Computers and Graphics* 38 (2014), 320–327.

Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. 2019. Decomposed Optimization Time Integrator for Large-Step Elastodynamics. *ACM Trans. Graph. (SIGGRAPH)* 31 (2019).

Zhenquan Li and Gordon Mallinson. 2006. Dual stream function visualization of flows fields dependent on two variables. *Computing and Visualization in Science* 9, 1 (2006), 33–41. https://doi.org/10.1007/s00791-006-0015-z

Alexander Linke. 2012. A divergence-free velocity reconstruction for incompressible flows. *Comptes Rendus Mathematique* 350, 17-18 (2012), 837–840.

Konstantin Lipnikov, Mikhail Shashkov, and Daniil Svyatskiy. 2006. The mimetic finite difference discretization of diffusion problem on unstructured polyhedral meshes. *J. Comput. Phys.* 211, 2 (2006), 473–491.

Frank Losasso, Ronald Fedkiw, and Stanley Osher. 2006. Spatially adaptive techniques for level set methods and incompressible flow. *Computers & Fluids* 35, 10 (2006), 995–1010. https://doi.org/10.1016/j.compfluid.2005.01.006

Frank Losasso, Frederic Gibou, and Ronald Fedkiw. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph.* 23 (08 2004), 457–462. https://doi.org/10.1145/1015706.1015745

Svenja Lowitzsch. 2005. Matrix-valued radial basis functions: stability estimates and applications. *Advances in Computational Mathematics* 23, 3 (2005), 299–315. https://doi.org/10.1007/s10444-004-1786-8

Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real-time applications. *ACM Trans. Graph. (SIGGRAPH)* 33, 4 (2014), 153.

Jakob Maljaars, Robert Labeur, and Matthias Möller. 2018. A hybridized discontinuous Galerkin framework for high-order particle-mesh operator splitting of the incompressible Navier-Stokes equations. *J. Comput. Phys.* 358 (04 2018). https://doi.org/10.1016/j.jcp.2017.12.036

J.B. Manges and Z.J. Cendes. 1995. A generalized tree-cotree gauge for magnetic field computation. *IEEE Transactions on Magnetics* 31, 3 (may 1995), 1342–1347. https://doi.org/10.1109/20.376275

Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. 2010. Unified simulation of elastic rods, shells, and solids. *ACM Trans. Graph. (SIGGRAPH)* 29, 4 (2010), 39.

Aleka McAdams, Andrew Selle, Kelly Ward, Eftychios Sifakis, and Joseph Teran. 2009. Detail Preserving Continuum Simulation of Straight Hair. *ACM Trans. Graph.* 28 (07 2009). https://doi.org/10.1145/1531326.1531368

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. 30, 4 (2011). https://doi.org/10.1145/2010324.1964932

Colin P McNally. 2011. Divergence-free interpolation of vector fields from point values—exact div-B= 0 in numerical simulations. *Monthly Notices of the Royal Astronomical Society: Letters* 413, 1 (2011), L76–L80.

DW Meyer and P Jenny. 2004. Conservative velocity interpolation for PDF methods. In *PAMM: Proceedings in Applied Mathematics and Mechanics*, Vol. 4. Wiley Online Library, 466–467.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. 2009. Energy-Preserving Integrators for Fluid Animation. *ACM Trans. Graph.* 28, 3, Article 38 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531344

Matthias Müller and Nuttapong Chentanez. 2011. Solid simulation with oriented particles. *ACM Trans. Graph. (SIGGRAPH)* 30, 4 (2011), 92.

Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. 2002. Stable Real-Time Deformations. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/545261.545269

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. *Fluid Dynamics* 2003, 154–159.

Matthias Müller, Nuttapong Chentanez, Tae Kim, and Miles Macklin. 2014. Strain Based Dynamics. https://doi.org/10.2312/sca.20141133

Rahul Narain, Jonas Zehnder, and B. Thomaszewski. 2019. A Second-Order Advection-Reflection Solver. *Proc. ACM Comput. Graph. Interact. Tech.* 2 (2019), 16:1–16:14.

Yen Ting Ng, Chohong Min, and Frédéric Gibou. 2009. An Efficient Fluid-solid Coupling Algorithm for Single-phase Flows. *J. Comput. Phys.* 228, 23 (Dec. 2009), 8807–8829.

James F. O'Brien and Jessica K. Hodgins. 1999. Graphical modeling and animation of brittle fracture. In *SIGGRAPH*. ACM Press/Addison-Wesley Publishing Co., 137–146.

D. Pai. 2002. STRANDS: Interactive Simulation of Thin Solids using Cosserat Models. *Computer Graphics Forum* 21 (2002).

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive localized liquid motion editing. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–10.

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

Andreas Peer, Christoph Gissler, Stefan Band, and Matthias Teschner. 2017. An Implicit SPH Formulation for Incompressible Linearly Elastic Solids: Implicit Elastic SPH Solids. *Computer Graphics Forum* 37 (12 2017). https://doi.org/10.1111/cgf.13317

Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. 2017. Computational design and automated fabrication of kirchhoff-plateau surfaces. *ACM Trans. Graph. (SIGGRAPH)* 36, 4 (jul 2017), 1–12. https://doi.org/10.1145/3072959.3073695

Jesús Pérez, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, José A. Canabal, Robert Sumner, and Miguel A. Otaduy. 2015. Design and Fabrication of Flexible Rod Meshes. *ACM Trans. Graph.* 34, 4, Article 138 (July 2015), 12 pages. https://doi.org/10.1145/2766998

Charles S Peskin. 2002. The immersed boundary method. *Acta numerica* 11 (2002), 479–517.

Tobias Pfaff, Nils Thuerey, and Markus Gross. 2012. Lagrangian vortex sheets for animating fluids. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 112.

Eric Plante, Marie-Paule Cani, and Pierre Poulin. 2002. Capturing the Complexity of Hair Motion. *Graphical Models* 64, 1 (Jan. 2002), 40–58. https://doi.org/10.1006/gmod.2002.0568 submitted Nov. 2001, accepted, June 2002.

John C Platt and Alan H Barr. 1988. Constraint methods for flexible models. *SIGGRAPH* 22, 4 (1988), 279–288.

Konstantin Poelke and Konrad Polthier. 2016. Boundary-aware Hodge decompositions for piecewise constant vector fields. *Computer-Aided Design* 78 (2016), 126–136.

Xavier Provot. 1995. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior.

Adina E Pusok, Boris JP Kaus, and Anton A Popov. 2017. On the quality of velocity interpolation schemes for marker-in-cell method and staggered grids. *Pure and Applied Geophysics* 174, 3 (2017), 1071–1089.

Ziyin Qu, Xinxin Zhang, Ming Gao, Chenfanfu Jiang, and Baoquan Chen. 2019. Efficient and Conservative Fluids Using Bidirectional Mapping. *ACM Trans. Graph.* 38, 4, Article 128 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322945

Nick Rasmussen, Douglas Enright, Duc Nguyen, Sebastian Marino, Nigel Sumner, Willi Geiger, Samir Hoon, and Ronald Fedkiw. 2004. Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 193–202.

Bharath Ravu, Murray Rudman, Guy Metcalfe, Daniel Lester, and Devang Khakhar. 2016. Creating analytically divergence-free velocity fields from grid-based data. *J. Comput. Phys.* 323 (07 2016). https://doi.org/10.1016/j.jcp.2016.07.018

Olivier Rémillard and Paul G. Kry. 2013. Embedded thin shells for wrinkle simulation. *ACM Trans. Graph. (SIGGRAPH)* 32, 4 (2013), 50.

Sander Rhebergen and Garth N Wells. 2018. A hybridizable discontinuous Galerkin method for the Navier–Stokes equations with pointwise divergence-free velocity field. *Journal of Scientific Computing* 76, 3 (2018), 1484–1501.

Robert E. Rosenblum, Wayne E. Carlson, and Edwin Tripp III. 1991. Simulating the structure and dynamics of human hair: Modelling, rendering and animation. *The Journal of Visualization and Computer Animation* 2, 4 (1991), 141–148. https://doi.org/10.1002/vis.4340020410 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/vis.4340020410

Syuhei Sato, Yoshinori Dobashi, and Theodore Kim. 2021. Stream-Guided Smoke Simulation. *ACM Transactions on Graphics (TOG)* 40, 4, Article 161 (2021).

Syuhei Sato, Yoshinori Dobashi, and Tomoyuki Nishita. 2018b. Editing fluid animation using flow interpolation. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 1–12.

Syuhei Sato, Yoshinori Dobashi, Yonghao Yue, Kei Iwasaki, and Tomoyuki Nishita. 2015. Incompressibility-preserving deformation for fluid flows using vector potentials. *The Visual Computer* 31, 6 (2015), 959–965.

Takahiro Sato, Christopher Batty, Takeo Igarashi, and Ryoichi Ando. 2018a. Spatially adaptive long-term semi-Lagrangian method for accurate velocity advection. *Computational Visual Media* (8 2018), 1–8. https://doi.org/10.1007/s41095-018-0117-9

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

Hagit Schechter and Robert Bridson. 2008. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 1–7.

J. C. J. Schellekens and René De Borst. 1993. On the numerical integration of interface elements. *Internat. J. Numer. Methods Engrg.* 36, 1 (1993), 43–66.

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008a. An unconditionally stable MacCormack method. *Journal of Scientific Computing* 35, 2-3 (2008), 350–371.

Andrew Selle, Michael Lentine, and Ronald Fedkiw. 2008b. A mass spring model for hair simulation. ACM Trans. Graphics SIGGRAPH. *ACM Trans. Graph.* 27 (08 2008). https://doi.org/10.1145/1360612.1360663

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: a sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph.* 33 (2014), 205:1–205:12.

Eftychios Sifakis and Jernej Barbic. 2012. FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. *ACM SIGGRAPH 2012 Courses, SIGGRAPH'12* (08 2012). https://doi.org/10.1145/2343483.2343501

Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ron Fedkiw. 2007. Hybrid simulation of deformable solids. In *Symposium on Computer Animation*. 81–90.

Zachary J. Silberman, Thomas R. Adams, Joshua A. Faber, Zachariah B. Etienne, and Ian Ruchlin. 2019. Numerical generation of vector potentials from specified magnetic fields. *J. Comput. Phys.* 379 (2019), 421 – 437. https://doi.org/10.1016/j.jcp.2018.12.006

Breannan Smith, Fernando Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Transactions on Graphics* 37 (03 2018), 1–15. https://doi.org/10.1145/3180491

B. Solenthaler and R. Pajarola. 2009. Predictive-corrective incompressible SPH. In *SIGGRAPH 2009*.

Jonas Spillmann and Matthias Teschner. 2007. CORDE: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. In *Symposium on Computer Animation*. 63–72.

Jonas Spillmann and Matthias Teschner. 2009. Cosserat nets. *IEEE TVCG* 15, 2 (2009), 325–338.

Jos Stam. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128. https://doi.org/10.1145/311535.311548

Jos Stam. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *Computer-Aided Design and Computer Graphics*. 1–11.

Michael Steffen, Robert M Kirby, and Martin Berzins. 2008. Analysis and reduction of quadrature errors in the material point method (MPM). *International journal for numerical methods in engineering* 76, 6 (2008), 922–948.

Alexey Stomakhin, Russell Howes, Craig Schroeder, and Joseph M. Teran. 2012. Energetically Consistent Invertible Elasticity. Eurographics Association, Goslar, DEU.

Tetsuya Takahashi and Ming C Lin. 2019. A geometrically consistent viscous fluid solver with two-way fluid-solid coupling. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 49–58.

Rasmus Tamstorf, Toby Jones, and Steve McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Transactions on Graphics* 34 (10 2015), 1–13. https://doi.org/10.1145/2816795.2818081

Yiying Tong, Santiago Lombeyda, Anil N Hirani, and Mathieu Desbrun. 2003. Discrete multiscale vector field decomposition. *ACM transactions on graphics (TOG)* 22, 3 (2003), 445–452.

Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and Francois Faure. 2015. Stable constrained dynamics. *ACM Trans. Graph. (SIGGRAPH)* 34, 4 (2015), 132.

Christopher D. Twigg and Zoran Kacic-Alesic. 2010. Point cloud glue: Constraining simulations using the Procrustes transform. In *Symposium on Computer Animation*. 45–54.

Nobuyuki Umetani, Ryan Schmidt, and Jos Stam. 2014. Position-based elastic rods. , 21–30 pages. https://dl.acm.org/citation.cfm?id=2849522

Wolfram Von Funck, Holger Theisel, and Hans-Peter Seidel. 2006. Vector field based shape deformations. In *ACM Transactions on Graphics (TOG)*, Vol. 25. ACM, 1118–1125.

Hongliang Wang, Roberto Agrusta, and Jeroen van Hunen. 2015. Advantages of a conservative velocity interpolation (CVI) scheme for particle-in-cell methods with application in geodynamic modeling. *Geochemistry, Geophysics, Geosystems* 16, 6 (2015), 2015–2023.

Huamin Wang and Yin Yang. 2016. Descent Methods for Elastic Body Simulation on the GPU. 35, 6 (2016). https://doi.org/10.1145/2980179.2980236

Ke Wang, Yiying Tong, Mathieu Desbrun, and Peter Schröder. 2006. Edge subdivision schemes and the construction of smooth vector fields. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 1041–1048.

Nicholas J. Weidner, Kyle Piddington, David I. W. Levin, and Shinjiro Sueda. 2018. Eulerian-on-lagrangian cloth simulation. *ACM Trans. Graph. (SIGGRAPH)* 37, 4 (jul 2018), 1–11. https://doi.org/10.1145/3197517.3201281

Jane Wilhelms. 1987. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications* 7, 6 (1987), 12–27.

Andrew Witkin, Kurt Fleischer, and Alan Barr. 1988. Energy constraints on parameterized models. In *SIGGRAPH*. 225–232.

Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2018. Bend-it: Design and Fabrication of Kinetic Wire Characters. *ACM Trans. Graph.* 37, 6, Article 239 (Dec. 2018), 15 pages. https://doi.org/10.1145/3272127.3275089

P. Yu and Zhen Tian. 2019. A high-order compact scheme for the pure streamfunction (vector potential) formulation of the 3D steady incompressible Navier-Stokes equations. *J. Comput. Phys.* 382 (04 2019). https://doi.org/10.1016/j.jcp.2018.12.027

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

Rundong Zhao, Mathieu Desbrun, Guo-Wei Wei, and Yiying Tong. 2019. 3D Hodge decompositions of edge-and face-based vector fields. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

Bo Zhu, Minjae Lee, Ed Quigley, and Ronald Fedkiw. 2015. Codimensional non-Newtonian fluids. *ACM Trans. Graph. (SIGGRAPH)* 34, 4 (2015), 115.

Bo Zhu, Ed Quigley, Matthew Cong, Justin Solomon, and Ronald Fedkiw. 2014. Codimensional surface tension flow on simplicial complexes. *ACM Trans. Graph. (SIGGRAPH)* 33, 4 (2014), 111.

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