HOLA: a High-Order Lie Advection of Discrete Differential Forms

With Applications in Fluid Dynamics

Thesis by

Alexander McKenzie

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

California Institute of Technology

Pasadena, California

2007

(Submitted May 25, 2007)

Acknowledgements

My sincerest thanks to Mathieu Desbrun. Special thanks to research collaborators Yiying Tong, Dmitry Pavlov, Patrick Mullen, Eva Kanso and Jerrold E. Marsden. Additional thanks for interesting classes by Michael Ortiz and Tim Colonius. My gratitude to Peter Schröder for generous provision of lab space and computing resources, and deepest appreciation to support staff Maria Lopez, Patrick Cahalan, Kathryn Moran and Diane Goodfellow. Thank you all very much.

This work was supported by National Science Foundation (CAREER CCR-0133983, and ITR DMS-0453145), Department of Energy (DE-FG02-04ER25657), and Pixar Animation Studios.

Abstract

The Lie derivative, and Exterior Calculus in general, is ubiquitous in the elegant geometric interpretation of many dynamical systems. We extend recent trends towards a *Discrete Exterior Calculus* by introducing a discrete framework for the Lie derivative defined on differential forms, including a WENO based numerical scheme for its implementation. The usefulness of this operator is demonstrated through the advection of scalar and vector valued fields (arbitrary discrete k-forms) in a desirable intrinsic and metric independent fashion. Examples include Lie advection of fluid flow vorticity, and we conclude with a significant discussion on the conservative Lie advection of fluid mass density for robust free surface flows in computer graphics.

Contents

Chapter 1

High-Order Lie Advection

1.1 Introduction

Simulation and scientific computing is an increasingly crucial component of research in various disciplines. However, deeply-rooted assumptions about smoothness and differentiability of most continuous laws of physics often clash with the inherently discrete nature of computing on modern architectures. Consequently, a number of computational techniques have been proposed to *discretize* differential equations, and numerical analysis—an important underpinning of computational science—is used to prove properties of the resulting schemes, such as stability, accuracy, and convergence. Over the past few years, the value of *geometrically-derived* techniques has often been shown superior to traditional, purely numerical-analytic approaches in a number of applications such as electromagnetism [7], discrete mechanics [43], and fluids [23]. As the essence of a mechanical system is characterized (and thus, modeled) by its symmetries and invariants (*e.g.*, momenta), preserving these geometric notions in the computational realm is of paramount importance [33]. In particular, symplectic integrators deriving from a discrete Hamilton's principle are noteworthy in their ability to properly capture the underlying continuous motion, often independently of the order of accuracy used in the computations.

In this thesis we follow such a geometric point of view to introduce a high-order numerical scheme for the Lie advection equation,

$$\frac{\partial \omega}{\partial t} + \mathscr{L}_X \omega = 0, \tag{1.1}$$

where ω is an arbitrary discrete differential k-form [4, 20] defined on a discrete manifold $\mathscr{M}$, and X a discrete vector field living on this manifold. The spatial Lie derivative $\mathscr{L}_X$, ubiquitous in most advection phenomena, is approximated through the use of a combinatorial exterior derivative and a high-order WENO-based definition of the interior product (or *contraction*). We will show that our high-order scheme can be seen as a generalization of the traditional Finite-Volume WENO schemes.

1.1.1 Background on Lie Derivative

The notion of Lie derivative $\mathscr{L}_X$ in Elie Cartan's Exterior Calculus extends the usual concept of derivative of a function along a vector field X. Although a formal definition of this operator can be made purely algebraically (Chap. 5.3, [1]), its nature is better elucidated from a dynamical perspective (Chap. 5.4, [1]). Consequently, the spatial Lie derivative (along with its closely related time-dependent version) is an underlying element in all areas of mechanics: *e.g.*, the rate of strain tensor in elasticity and the vorticity advection equation in fluid dynamics are both nicely described using Lie derivatives.

Lie Advection of Scalar Fields. A particularly simple context where a Lie derivative describes a physical evolution is in the advection of scalar fields. For instance, given a scalar field ρ, the conventional conservation-of-mass equation can be written as

$$\frac{\partial \rho}{\partial t} + \mathscr{L}_{\mathbf{v}}\rho = 0.$$

The case of divergence-free vector fields $\mathbf{v}$ (*i.e.*, $\nabla \cdot \mathbf{v} = 0$) has been the subject of extensive investigation over the past several decades, leading to several numerical schemes for solving these types of hyperbolic conservation laws. So-called "upwind" methods of advection were presented in [24] to approximate the conservation form of the advection equation. This initial work quickly gave rise to more sophisticated upwind schemes, first through the use of higher-order "least"-oscillatory local polynomial reconstruction (ENO [56]), and later refined through a weighted combination of ENO polynomials to further improve accuracy (WENO [40]). WENO schemes were derived in a finite-volume context (where they yield cell average preserving approximations to the unknowns rather than point value interpolations [54, 60]) as well as in a finite-difference context. Both types of schemes have been widely used in applications such as large-eddy simulations in the presence of strong shocks [35]. This line of research in WENO schemes also turned out to be crucial in the recent developments of accurate level set methods [50], where the Hamilton-Jacobi equation is solved in order to advect the zero contour of a signed distance function [52].

While these schemes have been successfully used for now over a decade, they seem to have been almost solely used to treat scalar fields, whether functions (as in LSM), or densities (as in FVM). To the authors' knowledge, Lie advection of non-scalar entities such as vorticity for fluids has yet to benefit from these advances.

1.1.2 Emergence of Structure-Preserving Computations

Concurrent to the development of high-order methods for scalar advection, new computational methods have emerged that faithfully respect the underlying geometric structures, gaining acceptance among engineers as well as mathematicians [3]. Computational electromagnetism [7], mimetic (or natural) discretizations [49, 6], and more recently Discrete Exterior Calculus (DEC, [36, 20]) and Finite Element Exterior Calculus (FEEC, [4]) have all proposed similar discrete structures that mimic (discretely preserve) vector calculus identities in order to offer improved numerics. In particular, the relevance of exterior calculus (Cartan's calculus of differential forms [15]) and algebraic topology (see, *e.g.*, Munkres 1984 [47]) for use in numerical simulations has come to light.

Differential Forms as Computational Building Blocks. Exterior calculus is a concise formalism to express differential *and* integral equations on smooth and curved spaces in a consistent manner, while revealing the geometrical invariants at play. At its root is the notion of differential forms, denoting antisymmetric tensors of arbitrary order. As integration of differential forms is an abstraction of the measurement process, this calculus of forms provides an intrinsic, coordinate-free approach to the concise description a multitude of physical models [11, 1, 42, 27, 45, 14, 30]. One of the key insights from the theory of differential forms is rather simple and intuitive: one needs to recognize that different physical quantities have different properties. Fluid mechanics or electromagnetism, for instance, make heavy use of line integrals, as well as surface and volume integrals. Similarly, even physical measurements are performed as specific local integrations or averages (think flux for magnetic field, or current for electricity) over some small surface of the measuring instrument. Pointwise values for such quantities do not have physical meaning; instead, one should manipulate those quantities only as geometrically-meaningful entities integrable over appropriate submanifolds—these entities are the so-called differential forms. Note also that integral values are generally speaking more robust to noise in measurement than pointwise evaluations from a numerical standpoint, making them particularly suitable for computations. Algebraic topology, specifically the notion of chains and cochains (see *e.g.*, [65, 47]), can then be used to emulate exterior calculus on finite grids, offering a natural discrete analog to differential forms: a set of values on vertices, edges, faces, and cells are proper discrete versions of pointwise functions, line integrals, surface integrals, and volume integrals respectively. Notice that this point of view is entirely compatible with the treatment of volume integrals in Finite Volume methods, or scalar functions in Finite Element methods; but it also involves the "edge elements" and "facet elements" as introduced in E&M as special H_{div} and H_{curl} bases elements [48]. Equipped with such discrete forms of arbitrary degree, Stokes' theorem (which connects differentiation and integration) is automatically enforced if one thinks of differentiation as the dual of the boundary operator—a particularly simple operator on meshes. With these basic building blocks, important structures and invariants of the continuous setting directly carry over to the discrete world, culminating in a discrete Hodge theory. As a consequence, the emerging notion of a discrete exterior calculus has proven useful in many areas such as electromagnetism [7], fluid simulation [23], surface parameterization [32], and remeshing [62] to mention a few.

Unfortunately, the contraction and Lie derivative of arbitrary discrete forms—two important operators in exterior calculus—have received very little attention, with the exception of the discrete contraction of Bossavit in [8]. The general idea behind his approach (that is reviewed in upcoming Section 1.3.1) was to notice that extrusion and contraction are dual operators, resulting in an integral definition of the interior product that fits the existing foundations. While a discrete contraction was derived using linear "Whitney" elements, no notion of high-order ENO-like treatment was proposed. Furthermore, the Lie derivative was not discussed.

1.1.3 Contributions

In this thesis, we introduce the first WENO-based upwind discretizations of contraction and Lie advection of an arbitrary discrete form. Our contribution therefore extends the DEC machinery to include a high-order accurate interior product and Lie derivative, and also extends conventional high-order WENO schemes to allow for the advection of forms and vector fields. In particular, our scheme in 3D is a generalization of finite-volume WENO techniques, where not only *cell-averages* are used, but also *face-* and *edge-averages*, in addition to vertex values.

1.2 Mathematical Tools

Before introducing our contribution, we briefly review the existing mathematical tools we will need in order to derive a high-order Lie advection: after discussing our setup, we describe the necessary operators of Discrete Exterior Calculus, before reviewing the foundations of WENO methods for high-order advection.

1.2.1 Discrete Setup

Space Discretization. Throughout the exposition of our approach, we assume regular Cartesian grid discretization of space[1]. This grid forms an orientable and manifold cell complex $K = (V, E, F, C)$ with vertex set $V = \{v_i\}$, edge set $E = \{e_{ij}\}$, as well as face set F and cell set C. Each cell, face and edge is assigned an arbitrary yet fixed intrinsic orientation, while vertices and cells always have a positive orientation. By convention, if a particular edge e_{ij} is positively oriented then e_{ji} refers to the same edge with negative orientation, and similar rules apply for higher dimensional mesh elements given even vs. odd permutations of their vertex indexing.

Boundary Operators. If we now assume all mesh elements in K are enumerated with an arbitrary (but fixed) indexing, the incidence matrices of K now define the boundary operators. For example, we let ∂^1 denote the $|V| \times |E|$ matrix with $(\partial^1)_{ve} = 1$ (resp., -1) if vertex v is incident to edge e and the edge orientation points towards (resp., away from) v, and zero otherwise. Similarly, ∂^2 denotes the $|E| \times |F|$ incidence matrix of edges to faces with $(\partial^1)_{ef} = 1$ (resp., -1) if edge e is incident to face f and their orientations agree (resp., disagree), and zero otherwise. The incidence matrix of faces to cells ∂^3 is defined in a similar way. See [47] for details.

[1]A discussion on how to extend HOLA to arbitrary simplicial complexes will be briefly discussed in Section 1.5.

1.2.2 Calculus of Discrete Forms

Guided by Cartan's exterior calculus of differential forms on smooth manifolds, DEC offers a calculus on discrete manifolds that maintains the covariant nature of the quantities involved.

Chains and Cochains At the core of this computational tool is the notion of *chains*, defined as a linear combination of mesh elements; a 0-chain is a weighted sum of vertices, a 1-chain is a weighted sum of edges, etc. Since each k-dim cell has a well-defined notion of boundary (in fact its boundary is a chain itself; the boundary of a face, for example, is the signed sum of its edges), the boundary operator naturally extends to chains by linearity. A *discrete form* is now simply defined as the dual of a chain, or *cochain*, that is to say, a linear mapping that assigns each chain a real number. Thus, a 0-cochain (that we will abusively call a 0-form sometimes) amounts to one value per 0-dim cell, such that any 0-chain can naturally pair with this form. More generally, k-chains are defined by one value per k-cell, and they naturally pair with k-chains. The resulting pairing of a k-cochain α^k and a k-chain σ_k is the discrete equivalent of the integration of a continuous k-form $\boldsymbol{\alpha}^k$ over a k-dimensional submanifold $\boldsymbol{\sigma}_k$:

$$\int_{\boldsymbol{\sigma}_k} \boldsymbol{\alpha}^k \equiv \langle \alpha^k, \sigma_k \rangle.$$

The chain and cochain representations are not only attractive from a computational perspective due to their conceptual simplicity and elegance; they are deeply rooted in a theoretical framework defined by [65], who introduced the Whitney and deRham maps that establish an isomorphism between simplicial cochains and Lipschitz differential forms. With these theoretical foundations, chains and cochains are used as basic building blocks for direct discretizations of important geometric structures such as the deRham complex through the introduction of two simple operators.

Discrete Exterior Derivative. The differential $\mathbf{d}$ (called exterior derivative) is the only existing DEC operator that we will need in our construction of a Lie derivative. This discrete exterior derivative $\mathbf{d}$ is constructed to satisfy Stokes' theorem, which elucidates the duality between the exterior derivative and the boundary operator. In the continuous sense, it is written

$$\int_{\sigma} \mathbf{d}\alpha = \int_{\partial\sigma} \alpha. \tag{1.2}$$

Consequently, if α is a discrete differential k-form, then the $(k+1)$-form $\mathbf{d}\alpha$ is defined on any $(k+1)$-chain σ by

$$\langle \mathbf{d}\alpha, \sigma \rangle = \langle \alpha, \partial \sigma \rangle\,, \tag{1.3}$$

where $\partial\sigma$ is the (k-chain) boundary of σ, as defined in Section 1.2.1. Thus, the discrete differential $\mathbf{d}$, mapping k-forms to $(k+1)$-forms, is given by the co-boundary operator, *i.e.*, the transpose of the signed incidence matrices of the complex K: $\mathbf{d}_0 = (\partial^1)^T$ maps 0-forms to 1-forms, while $\mathbf{d}_1 = (\partial^2)^T$ maps 1-forms to 2-forms. More generally in nD, $\mathbf{d}_k = (\partial^{k+1})^T$. In relation to standard 3D vector calculus, $\mathbf{d}_0 \equiv \nabla$, $\mathbf{d}_1 \equiv \nabla\times$, and $\mathbf{d}_2 \equiv \nabla\cdot$. The fact that the boundary of a boundary is empty results in $\mathbf{dd} = 0$, which corresponds to the vector calculus identities of $\nabla \times \nabla = \nabla \cdot \nabla\times = 0$. Notice that this operator is defined purely combinatorially, and thus does *not* need a high-order definition, unlike the operators we will introduce later.

Finally, we must point out that the notion of mesh duality (in the sense of Voronoi duality) can be used to introduce a discrete Hodge star that defines a natural correspondence between primal k-forms and dual $(n\text{-}k)$-forms, *i.e.*, forms living on dual $(n\text{-}k)$ cells [20]. This operator is indispensable to define a discrete Hodge decomposition of forms; however, we will not need this operator in the remainder of our derivation of the discrete contraction and Lie derivative operators.

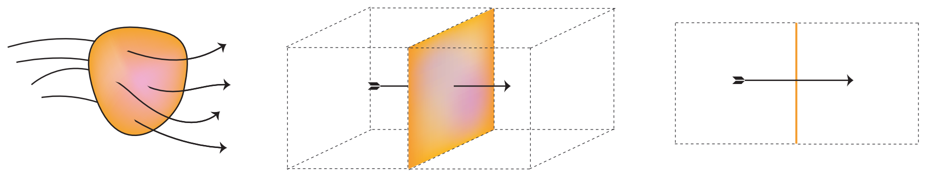

Figure 1.1: In the discrete setting, a vector field can be represented as a $(n$-1$)$-form in space, which intutively represents the flux induced by this vector field through hyperfaces (edge in 2D, face in 3D).

A Note about Vector Fields. Vector fields are often defined through their coordinates at mesh nodes, requiring the choice of a coordinate system. Discrete forms can, however, be advantageously used as proxies of vector fields, removing the need for coordinates. Indeed, in the continuous world and assuming that space is equipped with a volume form *Vol*, one can convert a vector field X into a $(n$-1$)$-form through the contraction $\mathbf{i}_X Vol$ of the volume form along X. This form can be seen as the “flux” induced by the vector field X. In computations, a vector field can thus be represented as a (n-1)-form, *i.e.*, one (flux) value per (n-1)-face of the primal mesh. This flux-based point of view is actually behind the routinely used Marker-And-Cell (MAC) "staggered" grids [34].

1.2.3 Principles of WENO

A last mathematical, numerical tool we will need is Weighted Essentially Non-Oscilloatory (WENO) shock-capturing schemes, first introduced in [40]. While we will provide a brief overview of these schemes, we refer the reader to [57] and [50] for further details and applications. The purpose of WENO schemes is to advect a function $u(x)$ by a velocity field $v(x)$ using a reconstruct, evolve, and average (REA) approach. In one dimension, we can define the cell average of a function $u(x)$ over cell C_i with width Δx as

$$\bar{u}_i = \frac{1}{\Delta x} \int_{C_i} u(x)\, dx \quad i = 1, 2, \ldots, N.$$

Given k adjacent cell averages, there is a unique polynomial $p(x)$ of degree at most $k-1$ such that the average of $p(x)$ in each of the k cells is equal to the average of $u(x)$ in those cells. For such a reconstruction, we can define a smoothness function, generally based on the magnitude of the derivatives, to evaluate how oscillatory the reconstruction is (see [10] in addition to above references for examples).

For a given cell boundary, there are k different choices of k adjacent cells that include the adjacent upwind cell (see Figure 1.7). Note that inclusion of the upwind cell (the cell from which the boundary flux comes) follows the physical intuition that we should include the information from the cell containing the source of the flux. Comparing the smoothness of these reconstructions and choosing the least oscillatory results in an Essentially Non-Oscillatory (ENO) scheme. WENO schemes, however, are based on the realization that all ENO reconstructions are good in smooth regions, and they can be further combined to result in a higher-order essentially non-oscillatory approximation; in regions of sharp changes, one must instead stick to the least oscillatory lower-order approximation provided by ENO. Once a WENO reconstruction is chosen, the flux across the boundary can be computed by integrating the reconstruction over the upwind region that will be transferred across the boundary (see Figure 1.4).

1.3 Discrete Interior Product and Discrete Lie Derivative

In keeping with the spirit of Discrete Exterior Calculus, we present the continuous interior product and Lie derivative operators in their "integral" form, *i.e.*, we present continuous definitions of $\mathbf{i}_X\omega$ and $\mathscr{L}_X\omega$ *integrated* over some small submanifold, precisely the objects we will have discretized on the cells of a discrete mesh by the end of this section.

1.3.1 Towards a Dynamic Definition of Lie Derivative

Interior Product through Extrusion. As pointed out by [8], the *extrusion* of objects under the flow of a vector field can be used to give an intuitive dynamic definition of the interior product. If $\mathscr{M}$ is an n-dim smooth manifold and $X \in \mathfrak{X}(\mathscr{M})$ a smooth (tangent) vector field on the manifold, let $\mathscr{S}$ be a k-dimensional submanifold on $\mathscr{M}$ with $k < n$. The flow φ of the vector field X is simply a function φ: $\mathscr{M} \times \mathbb{R} \to \mathscr{M}$ consistent with the one-parameter (time) group structure, that is, such that $\varphi(\varphi(\mathscr{S},t),s) = \varphi(\mathscr{S},s+t)$ with $\varphi(\mathscr{S},0) = \mathscr{S}$ for all $s,t \in \mathbb{R}$. Now imagine that $\mathscr{S}$ is carried by this flow of X for a time t; we denote the resultant "flowed out" submanifold $\mathscr{S}(t)$, which is equivalent to the image of $\mathscr{S}$ under the mapping φ, *i.e.*, $\mathscr{S}(t) \equiv \varphi(\mathscr{S},t)$. The extrusion $E_X(\mathscr{S},t)$ is then the $(k+1)$-dim submanifold formed by the advection of $\mathscr{S}$ over the time t to its final position $\mathscr{S}(t)$; it is the "extruded" (or "swept out") submanifold. This can be expressed formally as a union of flowed out manifolds,

$$E_X(\mathscr{S},t) = \bigcup_{\tau\in[0,t]} \mathscr{S}(\tau).$$

These geometric notions are visualized in Figure 1.2, where the submanifold $\mathscr{S}$ is presented as a 1-dim curve, flowed out to form $\mathscr{S}(t)$, or alternatively, extruded to form $E_X(\mathscr{S},t)$.

Using this setup, the interior product $\mathbf{i}_X$ of a time-independent form ω evaluated on $\mathscr{S}$ can now be defined as the instantaneous change of ω evaluated on $E_X(\mathscr{S},t)$, or more formally

$$\int_{\mathscr{S}} \mathbf{i}_X\omega = \left.\frac{d}{dt}\right|_{t=0} \int_{E_X(\mathscr{S},t)} \omega. \tag{1.4}$$

While this equation is coherent with the discrete spatial picture, we also wish to integrate $\mathbf{i}_X\omega$ over a small time interval as well, in order to get a continuous expression consistent with the integral form we will represent discretely. Hence, by taking the integral of both sides of Eqn. 1.4 over the interval $[0, \Delta t]$, the first fundamental theorem of calculus gives us

$$\int_0^{\Delta t} \left[\int_{\mathscr{S}(t)} \mathbf{i}_X \omega \right] dt = \int_{E_X(\mathscr{S}, \Delta t)} \omega, \tag{1.5}$$

which, as we will see, is a suitable candidate for discretization of the interior product.

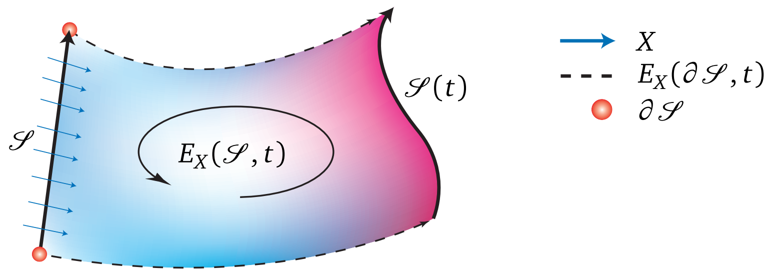

Figure 1.2: Geometric interpretation of the Lie derivative $\mathscr{L}_X \omega$ of a differential form ω in the direction of vector field X.

Algebraic and Flowed Out Lie Derivative. Using a similar setup, we can formulate a definition of Lie derivative based on the flowed out submanifold $\mathscr{S}(t)$. Remember that the Lie derivative is a generalization of the directional derivative to tensors, intuitively describing the change of ω in the direction of X. In fact, the Lie derivative $\mathscr{L}_X \omega$ evaluated on $\mathscr{S}$ is equivalent to the instantaneous change of ω evaluated on $\mathscr{S}(t)$, formally expressed by

$$\int_{\mathscr{S}} \mathscr{L}_X \omega = \left. \frac{d}{dt} \right|_{t=0} \int_{\mathscr{S}(t)} \omega, \tag{1.6}$$

as a direct consequence of the Lie derivative theorem (Theorem 6.4.1, [1]). As before, we can integrate Eqn. 1.6 over a small time interval $[0, \Delta t]$, applying the Newton-Leibnitz formula to find

$$\int_0^{\Delta t} \left[\int_{\mathscr{S}(t)} \mathscr{L}_X \omega \right] dt = \int_{\mathscr{S}(\Delta t)} \omega - \int_{\mathscr{S}} \omega. \tag{1.7}$$

Note that the formulation above, discretized through a semi-Lagrangian method, has been used by [23] to advect fluid vorticity; in that case the rhs of Eqn. 1.7 was evaluated by looking at the circulation through the boundary of the "backtracked" manifold. Rather than following their approach, we revert to discretizing the dynamic definition of the interior product in Eqn. 1.5 instead, and later constructing the Lie derivative algebraically. The primary motivation behind this modification is one of effective numerical implementation: we can apply a dimension-by-dimension WENO approximation scheme to obtain arbitrarily high order accuracy of the interior product, while the alternative—accurately computing integrals of ω over $\mathscr{S}(t)$ as required by Eqn. 1.7—is comparatively cumbersome and requires a computationally intensive multidimensional polynomial reconstruction to evaluate the required integrals to high accuracy.

Using Figure 1.2 as a reference, we now show how the Lie derivative and the interior product are linked through a simple algebraic relation known as Cartan's homotopy formula. In particular, this derivation requires repeated application of Stokes' theorem from Eqn. 1.2.

$$\begin{aligned}
\lim_{\Delta t\to 0}\frac{1}{\Delta t}\int_0^{\Delta t}\left[\int_{\mathscr{S}(t)}\mathscr{L}_X\omega\right]dt &= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left[\int_{\mathscr{S}(\Delta t)}\omega-\int_{\mathscr{S}}\omega\right] && (1.8)\\
&= \lim_{\Delta t\to 0}\frac{1}{\Delta t}\left[\int_{E_X(\mathscr{S},\Delta t)}\mathbf{d}\omega+\int_{E_X(\partial\mathscr{S},\Delta t)}\omega\right] && (1.9)\\
&= \int_{\mathscr{S}}\mathbf{i}_X\mathbf{d}\omega+\int_{\partial\mathscr{S}}\mathbf{i}_X\omega && (1.10)\\
&= \int_{\mathscr{S}}\mathbf{i}_X\mathbf{d}\omega+\int_{\mathscr{S}}\mathbf{d}\mathbf{i}_X\omega. && (1.11)
\end{aligned}$$

The submanifolds $\mathscr{S}$ and $\mathscr{S}(\Delta t)$ form a portion of the boundary of $E_X(\mathscr{S},\Delta t)$. Therefore, by Stokes' theorem, we can evaluate $\mathbf{d}\omega$ on the extrusion and subtract off the other portions of $\partial E_X(\mathscr{S},\Delta t)$ to obtain the desired quantity. This is how we proceed from line 1.8 to line 1.9 of the proof. The following line, Eqn. 1.10, is obtained by applying the dynamic definition of the interior product given in Eqn. 1.5 to each of two terms, leading us to our final result in line 1.11 through one final application of Stokes' theorem. What we have obtained is the Lie derivative expressed algebraically in terms of the exterior derivative and interior product. Notice that Eqn. 1.11 is the integral form of the celebrated identity called Cartan's homotopy (or *magic*) formula, most frequently written as

$$\mathscr{L}_X\omega=\mathbf{i}_X\mathbf{d}\omega+\mathbf{d}\mathbf{i}_X\omega. \qquad (1.12)$$

By defining our discrete Lie derivative through this relation, we ensure the algebraic definition holds true in the discrete sense by construction. It also implies that the Lie derivative can be directly defined through interior product and exterior derivative, without the need for its own discrete definition.

As mentioned, schemes discretizing the Lie derivative directly through Eqn. 1.7, have been proposed. In [23], the use of a backtracked flow intuitively corresponds to the vorticity being pulled forward from an earlier time, while higher-order approximations can be derived by mixing backwards *and* forward tracking [21]. Related Lagrangian advection techniques for conservative density advection can be found in, *e.g.*, [37, 22].

1.3.2 High-order Discrete Interior Product

The discrete interior product is computed by exploiting the principles of Eqn. 1.5, using the WENO machinery to achieve high-order accuracy. Given a discrete k-form α and a discrete vector field X, the interior product is approximated by extruding backwards in time every (k-1)-dim cell σ of the computational domain to form a new k-dim cell $E_X(\sigma, \Delta t)$. Evaluating the integral of α over the extrusion and pairing the resulting value with the original cell σ yields the mapping $\langle \mathbf{i}_X \alpha, \sigma \rangle$ integrated over a time step Δt. This procedure, once applied to all (k-1)-dim cells, assigns a value to every σ, hence giving the desired discrete (k-1)-form $\mathbf{i}_X \alpha$.

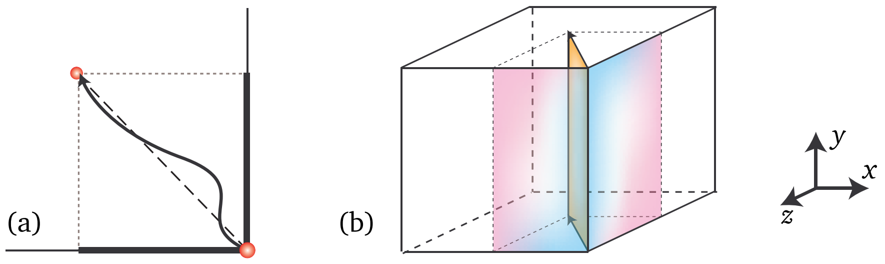

Figure 1.3: In the discrete setting, the extrusion of a 0-dim manifold (left), 1-dim manifold (right), or k-dim manifold is approximated by projecting the Lagrangian advection of the manifold into n number of 1-dim components.

Approximating the Extrusion. Instead of a Lagrangian advection of the cell σ to determine $E_X(\sigma, \Delta t)$, we have recourse to a fully Eulerian approach by projecting the advection to axis-aligned motions, allowing for a dimension-by-dimension WENO reconstruction. Consider the simple example illustrated in Figure 1.3(a) where a 0-dim cell is advected through the flow of X, creating a 1-dim extrusion between initial and final positions of the particle as indicated by the arrow. Rather than evaluating the integral of α over this geometrically non-trivial manifold, we project the extrusion into x- and y-components (seen in bold), evaluate $\int \alpha$ on each component independently, finally taking the sum of these terms as an approximation to the integral over the true extrusion. Note that this approximation corresponds to a *straightline approximation* of the extrusion as indicated with a dashed line in the figure. Similarly, the interior product of a higher dimensional form is displayed in Figure 1.3(b), showing how the extrusion of a 1-dim cell is approximated; note how the y-component projects to a line, hence one term of the integral approximation is automatically zero. The advantage of this dimension-wise decomposition, as hinted at earlier, is that it allows us to apply a high-order accurate 1-dim WENO stencil to each of the axis-aligned components of the extrusion that requires numerical integration of α, without resorting to a more expensive multidimensional WENO scheme—although such a treatment could also be attempted.

Note that the actual position of this dimension-wise advection of σ is computed through backtacking. The cell σ is flowed backwards in the velocity field for one discrete time step. The extrusion formed by this flow is precisely the manifold that will pass through σ during the *next* time step. In computations, we apply the velocity field defined at σ and perform a forward Euler time discretization for one negative time step. This is an extremely simple time update rule, and leads to an undesirable CFL style condition on the maximum time step Δt that can be used while maintaining a well behaved solution. If the velocity field required by the Euler step is not conveniently located with respect to appropriate k-cells, we interpolate the field (through linear interpolation, more accurate approximations could be employed). This entire bracktracking concept is highlighted in Figure 1.4, where an edge is advected backwards in time to find the extrusion on which we integrate.

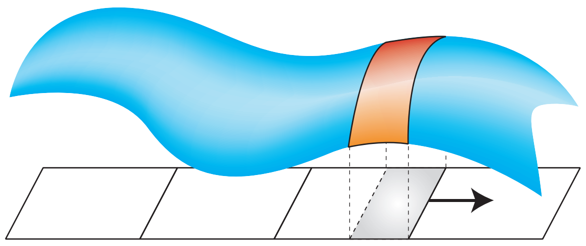

Figure 1.4: Upwind 1-dim polynomial reconstruction of ω used to compute $\mathbf{i}_X\omega$ by integrating ω over the backtracked extrusion $E_X(\sigma, t)$.

Upwind Approximation. The benefits of upwinding (*i.e.*, biasing computations in the direction determined by the local direction of the velocity) to discretize hyperbolic PDEs are well known: the method of characteristics suggests the use of information upstream from the area of evaluation, in an attempt to limit diffusion near discontinuities. We therefore use the upwind WENO approximation based on the sign of the velocity at the mesh element currently dealt with, as reviewed in Section 1.2.3. Specifically, this implies using more stencils in the upwind direction during the WENO reconstruction to compute the integral over the backtracked extrusion. Due to our dimension-wise decomposition of the extrustion, all WENO evaluates are essentially 1-dimensional in nature, even when integrating over arbitrary k-forms.

1.3.3 High-order Lie Advection

We now have all the ingredients to introduce a discrete Lie advection. Given a k-form α^k, we compute the $(k+1)$-form $\mathbf{d}_k\alpha^k$ by applying the transpose of the incidence matrix ∂^{k+1} as detailed in Section 1.2.2. We then compute the k-form $\mathbf{i}_X(\mathbf{d}\alpha^k)$, and the ($k$-1)-form $\mathbf{i}_X\alpha^k$. By applying $\mathbf{d}_{(k-1)}$ to the latter form and summing the resulting k-form with the other interior product, we finally get a high-order approximation of Cartan's homotopy formula of the Lie derivative.

1.4 Applications and Results

We now present a few direct applications of this discrete Lie advection scheme, both for illustrative and comparative purposes.

1.4.1 Interface Advection

Applying the HOLA approach to a standard finite volume scalar field discretization results in already well known advection schemes. Such schemes were used recently in [46] as the basis for advection and generalized gradient flows for density-based Eulerian surfaces. When only the interface is of concern, algorithms were presented to limit the computation of the Lie advection to a narrow band of cells nearby the surface to increase computational efficiency. This method can be used for a range of applications from geometry processing of foliations to mass-preserving fluid simulation (see Figure 2.2). In the latter case, the conservative nature of the scheme avoids the volume loss which plagues conventional level set methods. An in depth discussion of the Navier-Stokes fluid update, coupled with free surface, is presented in Chapter 2. Standard CFL time step restrictions can also be avoided to increase efficiency at the cost of a decreased order of accuracy as pointed out in [31].

1.4.2 Fluid Simulation through Vorticity Advection.

Consider an incompressible, homogeneous fluid on a manifold $\mathscr{D}$. The Euler equations for the velocity field $\mathbf{v}$ of the fluid, assuming a unit density ($\rho = 1$), are given by

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p \tag{1.13}$$

with the constraint that $\nabla \cdot \mathbf{v} = 0$ and that $\mathbf{v}$ is tangential to the boundary $\partial \mathscr{D}$. The equation above can be expressed in the language of exterior calculus, in terms of the Lie derivative and the (n-1)-form $v = \mathbf{i}_{\mathbf{v}} Vol$ (*i.e.*, the flux induced by the velocity), as

$$\frac{\partial v}{\partial t} + \mathscr{L}_{\mathbf{v}} v = \mathbf{d}(\tfrac{1}{2}|\mathbf{v}|^2 - p). \tag{1.14}$$

We define *vorticity* as $\omega = \mathbf{d} \star v$, and apply the exterior derivative to both sides of Eqn. 1.14. Since $\mathbf{d}$ and $\mathscr{L}_{\mathbf{v}}$ commute, this reduces to the well known Lie advection equation

$$\frac{\partial \omega}{\partial t} + \mathscr{L}_{\mathbf{v}} \omega = 0. \tag{1.15}$$

This simply states that vorticity is advected along the flow, and using the machinery described in the previous section, we can design an algorithm to simulate exactly that. Notice that working directly with vorticity avoids the numerical diffusion induced by projection on divergence-free fields [16]. We begin by discretizing our space $\mathscr{D}$ through a typical axis-aligned cartesian gird, and now consider the velocity form v as being the flux of our fluid (*i.e.*, a coordinate-free representation of the velocity). With this spatial approximation in place, we use a fractional step time integration of Eqn. 1.15 that proceeds as follows: (1) at each time step we advect the discrete form ω along the current velocity v using a forward Euler HOLA scheme, and (2) from the updated ω determine the new stream function, a (n-2)-form ϕ, through the Poisson equation $\Delta\phi = \omega$; the linear solver used here is the one detailed in [23]). The new velocity form can now be deduced via the relation $\mathbf{d}\phi = v$, and the procedure can be started anew. Note that the no-flux condition through $\partial\mathscr{D}$ leads to Dirichlet boundary conditions on the Poisson equation that gives the correct number of constraints. Also note that vorticity is an *exact* form since $\omega = d\star v$, therefore $\mathbf{d}\omega = 0$ and hence the first term of the Lie derivative, defined through the magic formula, Eqn. 1.12, drops out.

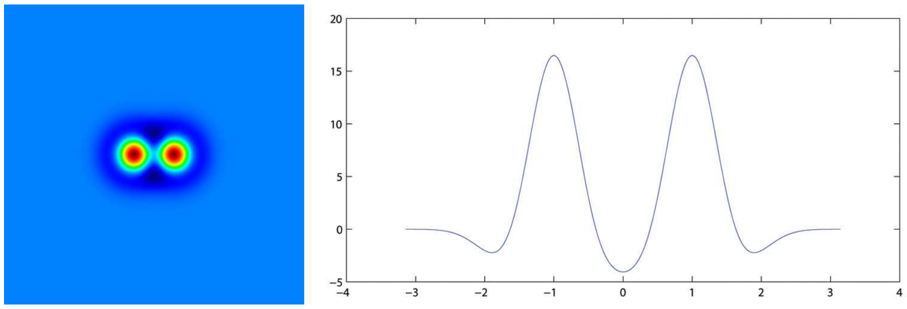

Figure 1.5: Vorticity advection. A 2D domain is initialized with two Taylor vortex distributions of same sign, visualized in false colors (*Left*); a 1D slice of vorticity is also presented (*Right*).

Numerical experiments are presented for comparison purposes—we solve the vorticity formulation of the Euler equations using a pseudospectral method with 3/2 dealiasing for the non-linear product [12], the circulation preserving scheme of [23], and using the HOLA scheme with two different types of stencils: a piecewise constant stencil (data from one collocation point required; HOLA-1), and a WENO piecewise cubic stencil (four stencils, seven collocation points total; HOLA-7). A 2D square domain with $x \in [-\pi, \pi]$ and $y \in [-\pi, \pi]$ and periodic boundary conditions is initialized with two Taylor vortices, the vorticity distribution of which is given by

$$\omega(x,y) = \frac{U}{a}\left(2 - \frac{r^2}{a^2}\right)\exp\left(\frac{1}{2}\left(1 - \frac{r^2}{a^2}\right)\right), \tag{1.16}$$

where r is the distance from (x,y) to the center of the vortex, and a is the core size of the vortex, and U is the maximum tangential velocity. We can specify multiple Taylor vortices by taking the sum of the contributions of ω from each vortex. Note that the distribution above ensures the integral of vorticity over the computational domain is zero, a necessary condition to allow comparison with Fourier based methods. In all the numerical tests presented, the spatial resolution of the grid is 128×128, with parameters $U = 1$ and $a = 0.3$ for two Taylor vortices separated by a distance of 0.8. These parameters yield vortices sufficiently small that their evolution is minimally influenced by their periodic mirrors. It is noted that such a twin Taylor vortex initialization exhibits a bifurcation in the dynamics as the separation distance increases from zero; the vortices merge for small separations but eventually diverge for larger

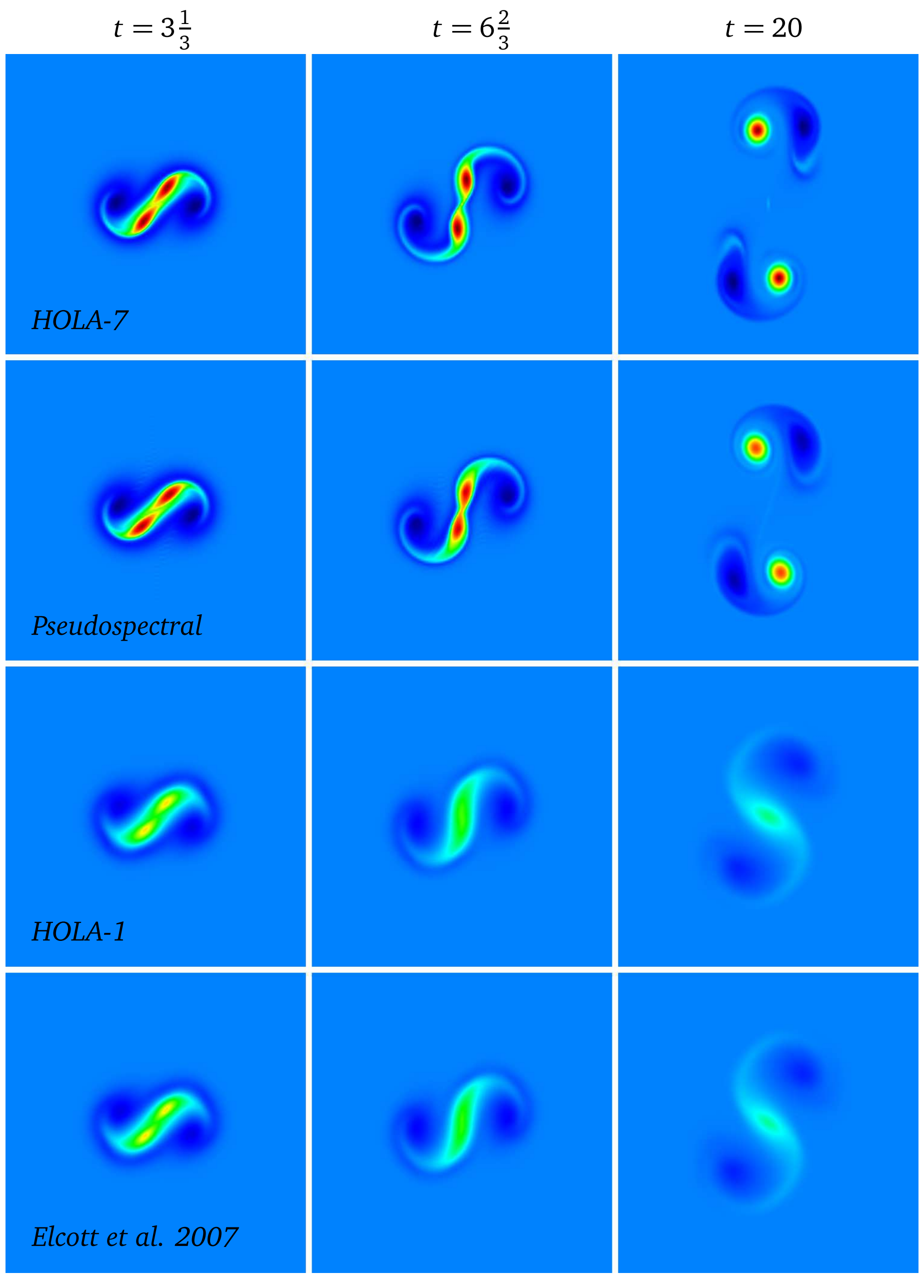

Figure 1.6: The vorticity distribution of Figure 1.5 is evolved in time using several vorticity advection schemes.

separations. In these numerical tests, the separation distance 0.8 is purposely very close to this critical bifurcation threshold.

Using the WENO advection scheme, $\int_{\mathscr{D}} \omega$ is properly conserved, as vorticity is simply exchanged between dual faces on the mesh, *i.e.* only one term of Cartan's formula is employed. HOLA-7 results in a vastly decreased numerical diffusion of vorticity compared to low-order schemes. However, numerical diffusion remains unavoidable over time, and manifests itself in the decrease of the total energy $\int_{\mathscr{D}} \frac{1}{2}|v|^2$. Observe also, in Figure 1.6, how the order of accuracy influences the *dynamics* of the vortices; the low order diffusive methods result in the vortices artificially merging.

1.5 Conclusions

In this thesis, we have introduced an extension of the classical WENO techniques to handle arbitrary discrete forms. The first WENO-based upwind discretization of both contraction and Lie derivative was presented, extending Discrete Exterior Calculus to include high-order accurate approximations to these operators. The advection of forms and vector fields are applicable in a multitude of problems, including conservative interface advection and conservative vorticity evolution.

Although WENO interpolation offers high-order accuracy at a relatively low computational cost, numerical diffusion is still present and can accumulate over time. In particular, the scheme is not variational in nature, *i.e.*, it is not derived from a variational principle, as we have seen.

In the future, we expect that extensions can be made to handle simplicial meshes, although this would require us to define a WENO based integration on irregular meshes. While some works already exist in this area, *e.g.*, [67, 39], the implementation is delicate with unavoidable increases in memory and computational expense.

1.6 Third Order Stencils

For completeness, we provide the conservative WENO weights. We give the stencils and weights used for a WENO advection scheme in our framework. Third order stencils are used, requiring 4 values per stencil. The 4 stencils can be combined using the weights $w_0 = \frac{1}{35}$, $w_1 = \frac{12}{35}$, $w_2 = \frac{18}{35}$, and $w_3 = \frac{4}{35}$ to obtain a 6th order approximation at a point in smooth regions. The following q_i (see Figure 1.7) compute the integral of the reconstructed ρ from the boundary between ρ_x and ρ_{x+1} to the boundary minus h for each stencil (generally $v \cdot dt$), where Δx is the width of each grid cell.

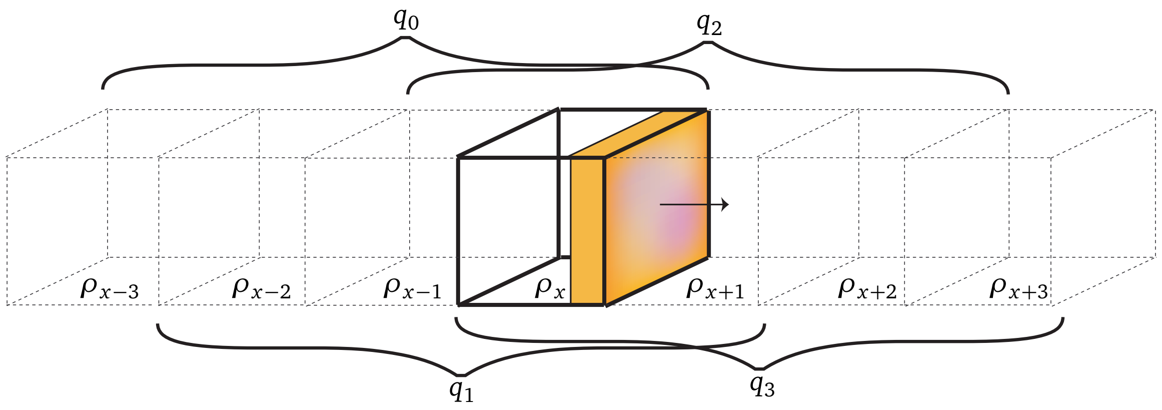

Figure 1.7: **WENO Stencils**: each stencil uses a different group of adjacent cells to compute the integral of a polynomial reconstruction over the orange region. Note all stencils used include the upwind (bold) cell ρ_x.

$$\begin{aligned}
q_0 &= \frac{h^4}{24\Delta x^3}(\rho_{x-3}-3\rho_{x-2}+3\rho_{x-1}-\rho_x) && +\frac{h^3}{12\Delta x^2}(-3\rho_{x-3}+11\rho_{x-2}-13\rho_{x-1}+5\rho_x)+\\
&\quad \frac{h^2}{24\Delta x}(11\rho_{x-3}-45\rho_{x-2}+69\rho_{x-1}-35\rho_x) && +\frac{h}{12}(-3\rho_{x-3}+13\rho_{x-2}-23\rho_{x-1}+25\rho_x)\\
q_1 &= \frac{h^4}{24\Delta x^3}(\rho_{x-2}-3\rho_{x-1}+3\rho_x-\rho_{x+1}) && -\frac{h^3}{12\Delta x^2}(\rho_{x-2}-5\rho_{x-1}+7\rho_x-3\rho_{x+1})-\\
&\quad \frac{h^2}{24\Delta x}(\rho_{x-2}-3\rho_{x-1}-9\rho_x+11\rho_{x+1}) && +\frac{h}{12}(\rho_{x-2}-5\rho_{x-1}+13\rho_x+3\rho_{x+1})\\
q_2 &= \frac{h^4}{24\Delta x^3}(\rho_{x-1}-3\rho_x+3\rho_{x+1}-\rho_{x+2}) && +\frac{h^3}{12\Delta x^2}(\rho_{x-1}-\rho_x-\rho_{x+1}+\rho_{x+2})+\\
&\quad \frac{h^2}{24\Delta x}(-\rho_{x-1}+15\rho_x-15\rho_{x+1}+\rho_{x+2}) && -\frac{h}{12}(\rho_{x-1}-7\rho_x-7\rho_{x+1}+\rho_{x+2})\\
q_3 &= \frac{h^4}{24\Delta x^3}(\rho_x-3\rho_{x+1}+3\rho_{x+2}-\rho_{x+3}) && +\frac{h^3}{12\Delta x^2}(3\rho_x-7\rho_{x+1}+5\rho_{x+2}-\rho_{x+3})+\\
&\quad \frac{h^2}{24\Delta x}(11\rho_x-9\rho_{x+1}-3\rho_{x+2}+\rho_{x+3}) && +\frac{h}{12}(3\rho_x+13\rho_{x+1}-5\rho_{x+2}+\rho_{x+3})
\end{aligned}$$

The smoothness function we used is the sum of the integral squared of each derivative of the reconstructed polynomial over the region of the reconstruction Ω, *i.e.*

$$S(\rho(x))=\sum_{k=1}^{3}\int_{\Omega}\left(\frac{\partial^k}{\partial x^k}\rho(x)\right)^2 dx. \tag{1.17}$$

For the cubic polynomial constructed from ρ_0, ρ_1, ρ_2, and ρ_3 this evaluates to

$$\begin{aligned}
S(\rho_0,\rho_1,\rho_2,\rho_3) &= \frac{1}{60}(867\rho_0^2+6083\rho_1^2-11606\rho_1\rho_2+\\
&\quad 6083\rho_2^2+3802\rho_1\rho_3-4362\rho_2\rho_3+867\rho_3^2-\\
&\quad 2\rho_0(2181\rho_1-1901\rho_2+587\rho_3))
\end{aligned}$$

We also tried several other smoothness functions, including weighting the derivatives based on their order and varying the range of integration, but found little to no qualitative differences in the results. The above equations are all used in standard WENO fashion to compute the final flux through the face between ρ_x and ρ_{x+1} (assuming $h=v\cdot dt$ is positive) as

$$\frac{1}{\alpha}\sum_{i=0}^{3}\frac{w_i q_i}{S(\rho_{x+i-3},\rho_{x+i-2},\rho_{x+i-1},\rho_{x+i})+\epsilon} \tag{1.18}$$

where

$$\alpha = \sum_{i=0}^{3} \frac{w_i}{S(\rho_{x+i-3}, \rho_{x+i-2}, \rho_{x+i-1}, \rho_{x+i}) + \epsilon} \tag{1.19}$$

is a normalization factor, and ϵ is a small number to avoid division by 0 (taken as 1×10^{-6} in our experiments).

Chapter 2

Mass-Conserving Fluid Flows

The governing equations of inviscid incompressible fluid motion are called the *Euler Equations*, and are also known, with an additional dissipative viscous term, as the *Navier-Stokes Equations*. There is an extensive history of solving Navier-Stokes (NS) in the field of computational fluid dynamics (CFD), yet the recent push towards realistic animation of fluids for computer graphics has led to renewed interest and new breakthroughs in this exciting and rich area of research. In graphics, the motivation is obvious: fluids, with their intricate and complex behavior, have long been viewed as the pinnacle aspiration of physically-based animation, due to this relative complexity. It is a drudging and laborious process for an artist to painstakingly model the motion of a liquid explicitly, in a frame by frame manner. Moreover, the results are often lackluster due to the complicated dynamics of even a relatively simple flow. The problem is also made difficult by the high expectations of a sophisticated audience who interact with fluids on a regular basis.

In this chapter, we continue to develop and improve upon physically-based animation and simulation methods by proposing and enforcing a discrete mass-conservation law; a computational counterpart analogous (for homogeneous flows) to the mass conservation constraint implicitly inherent to the Navier-Stokes Equations. In addition, we thoroughly describe all the background necessary for the reader to devise and create their own fluid solver with mass preserving properties. This chapter is an account of one implementation; specifically the marker-and-cell (MAC) method from Welch et al. [64] with Chorin's L_2 pressure projection step [17] on to the divergence free manifold. This style of fractional step integration has proven itself worthwhile for visual accuracy, simple implementation, and relatively fast computations (although real-time performance is still unpractical in the general case, even with GPGPU based implementations such as [19]). We shall begin with a brief introduction into state-of-the-art fluid techniques in graphics, and continue by introducing the Navier-Stokes Equations and then detail the specific finite difference discretizations of appropriate operators that allows us to solve for fluid motion on our discrete computational domain, and the numerical methods we employ to solve the partial differential equations of motion.

2.1 Background Literature

Computational Fluid Dynamics. Fluid mechanics has been studied for centuries in the scientific community, first analytically, and later (over the last fifty years) numerically as well. Computational fluid dynamics aims to discretize the governing equations, Navier-Stokes, through various techniques such as Finite Volume (FV), Volume of Fluid (VOF), Finite Element, Finite Differences (FD), and in very specific cases, Pseudo Spectral methods as well. While we will not overview all of these methods here, a comprehensive discussion of computational approaches can be found, for instance, in Langtangen et al. [38]. It is important to note, and generally acknowledged, that no one scheme is found to be ideal over the vast range of fluid flow problems encountered, *e.g.*, from inviscid to viscous, or from stable flows to turbulent flows. Practitioners in computer graphics have selected methods based on their suitability (*i.e.*, computational cost), robustness, and appealing visual behavior.

Computer Animation. Today's modern techniques are used for photorealism [51], or controlling liquid-like characters [66, 44, 55]. These advanced simulation systems are all derived from the basic setup of Foster and Metaxas [29]; they adopted the NS Equations for incompressible fluids as their primary physical model, and attempted to solve these equations in three dimensional domains. They advocated the use of a particle Marker-And-Cell (MAC) grid discretization of the computational domain, a concept earlier introduced by [64] in the CFD community, and now used for graphics. The type of immiscible fluid behavior being modeled, *e.g.*, pouring water into a glass, or perhaps an ocean wave breaking onto a sea shore, are example of *two phase* flows. Since the relative density of a typical liquid such as water to air is approximately a thousand to one, Foster and Metaxas were able to induce huge computational savings by opting to model the dense liquid volume only, with minor to negligible implications for the visual behavior in most cases. Assuming vacuum on the other side of the immiscible interface is commonly referred to as a *free surface* fluid simulation. Foster and Metaxas discuss the first steps in dealing with the free surface boundary conditions required by the discontinuous state variables near the interface, pressure and velocity in particular. The staggered, rectilinear grid of cells they use, allows for finite difference stencils to be applied in an attempt to appropriate discretize and solve the non-linearity of the NS PDE's in a computational tractable way. A spatial central differencing scheme was employed to account for the fluid convection (*aka* advection) and viscosity terms, and the entire system was integrated over time though a forward Euler step time discretization. Such explicit schemes proved to have significant drawbacks. Even for relatively inviscid substances, numerical stability is a problem due to the Courant-Fredreichs-Lewy (CFL) condition, which dictates that the largest stable time step Δt is proportional to the size of the grid cell, *i.e.*, that $\Delta t \sim h$ where h is the length of a cell. As the grid is refined for spatial accuracy, the impact of the CFL condition is to inflate the number of iterations required to maintain a well behaved solution. This restriction quickly becomes computationally prohibitive for graphics and animation. Jos Stam [58] published what has become a highly celebrated breakthrough, in which he advocated implicit integration as a way to circumvent the (sometimes unconditional) instabilities of alternative methods. His approach (*aka* "Stable Fluids") illustrated the use of a semi-Lagrangian *method of characteristics* that was first proposed in [18], and previously used in the meteorological community [59] for large time stepping. In addition, Stam documented an implicit approach to solving the dissipative viscous term (although at the cost of an additional linear system to solve), hence achieving a guaranteed unconditional stability at the cost of some numerical dissipation in the solution that is induced by these implicit schemes. Generally speaking, the unnatural dissipation has been an acceptable price to pay for arbitrarily large time steps, although several recent contributions such as [23] and [26] specifically try to address this problem—the vorticity preserving scheme of Elcott et al. [23] is preferable in this case, as the vorticity confinement approach [26] achieves this added visual complexity by reintroducing an unstable and uncontrollable term to the solver.

Augmenting the basic solver approach proposed by Foster and Metaxas [29], further advance was made in Foster and Fedkiw [28], allowing the first practical simulation of liquids (*i.e.* free surfaces) for use in offline applications, particularly in films. In addition to incorporating the Stable Fluids technique, [28] introduced a novel hybrid particle / level set representation of the immiscible interface as a mechanism to elegantly handle the dynamically changing topology of the interface as it is stretching and splashing. The fundamental philosophy behind the approach was to exploit the advantages of both an Eulerian representation of the interface *and* a Lagrangian representation of the fluid volume itself. Particles provide a smooth integration of the fluid's velocity field, and are used to maintain accuracy of the fluid position, since they are not bounded to the course resolution of the underlying grid. Conversely, the level set function is used primarily to implicitly capture the topology of the fluid / vacuum interface. Furthermore, while Foster and Metaxas [29] applied a slow linear solver, a Successive Over Relation (SOR) iteration to enforce fluid incompressibility via Chorin's pressure projection [17], this numerical method was replaced by a more effective (*i.e.*, efficient) Preconditioned Conjugate Gradient (PCG) algorithm to solve the sparse linear system in order to determine the pressure. Increases in speed, accuracy and ultimately, in the quality of the results, made this an important contribution, especially as a systems oriented approach suitable for commercial application. Perhaps the final significant improvement to the core fluid simulator was the paper of Enright et al. [25], which saw a general shift in emphasis away from modeling the liquid volume, to modeling the liquid interface. By using information gained from the level set function, valid velocity values are grown (extrapolated) out from the fluid volume into the open vacuum region (using the fast marching method proposed by Adalsteinsson and Sethian [2]), creating an upwind, "method of characteristics" style velocity field extrapolated at the interface. The particle / level set approach could then be symmetrized for interface advection and capture, by now adding particles to the vacuum side of the immiscible interface and allowing them to be advected by this extrapolated velocity field (usually the velocity is only determined *inside* the fluid). This final "Particle Level Set Method" (PLSM) as it is known, is currently the defacto standard for high quality tracking of dynamically evolving interfaces.

2.2 Mass Density based Fluid Representation

Departing from the hybrid Eulerian / Lagrangian schemes developed by Enright et al. [25] and others, we opt for a cell-centered Finite-Volume setup, where a single value per cell is stored. This setup falls in the category of *interface capturing methods*, as it defines the interface as a region of steep gradient of a characteristic-like function (as opposed to LSM-like *interface tracking methods* which treat the interface as a sharp discontinuity moving through a grid). This setup has an obvious physical interpretation: acknowledging the fact that explicitly maintaining a perfect Heaviside function of the fluid is very difficult in the discrete Eulerian setting, we

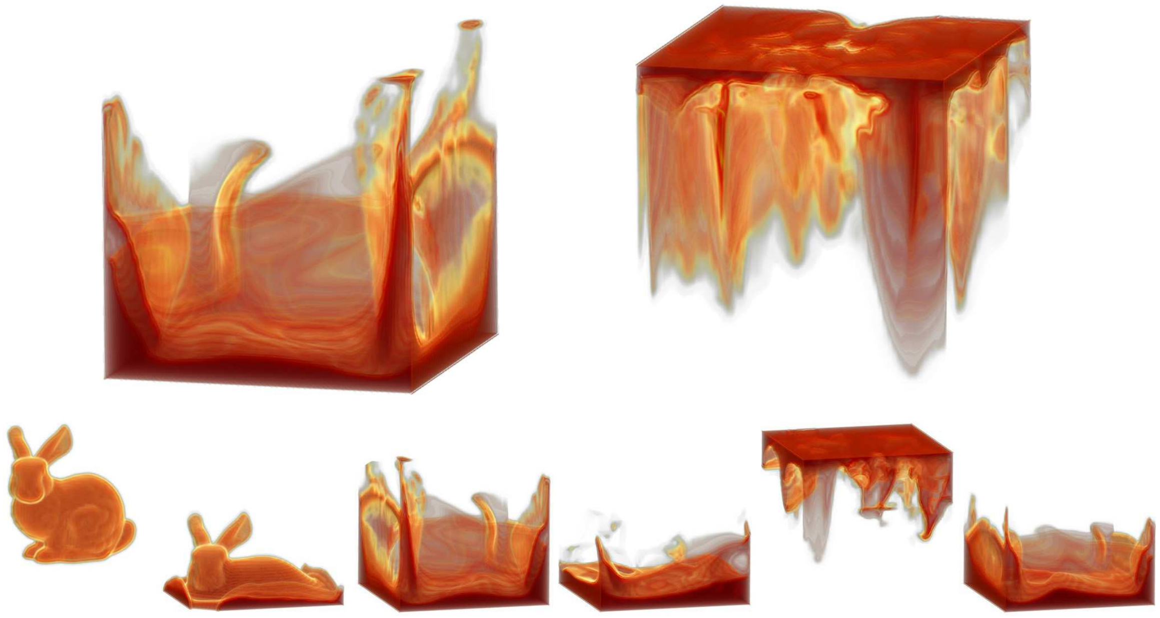
Figure 2.1: **Volumetric Rendering Of Fluid.** A bunny-shaped fluid is dropped into a box, preserving the total mass even after gravity is inverted arbitrarily, causing extreme deformation. Notice how the volumetric rendering reveals intricate details captured by all the isosurfaces throughout the animation, despite a very coarse grid resolution of only 64^3.

do not encode the exact surface, but instead store an approximate (*blurred*, in a sense) *mass density* of the fluid inside each cell. Thus, a cell with a value of 0 will be considered completely outside the fluid, a cell at 1 will be considered as completely inside, and the rest of the cells (with densities varying from 0 to 1) represent a smeared interface of the fluid, or alternately the proportion of fluid inside the cell (note that we will use the terms *density* and *mass density* interchangeably as our explanations will always use densities restricted to $[0, 1]$). Unlike VOF or PFM, we *do not restrict the profile* of our density function, avoiding the computational overhead incurred when a special profile needs to be maintained, as well as allowing the treatment of multiple isosurfaces with varying shape. This density-based setup will facilitate the use of this Eulerian representation in fluid simulation.

This mass density can be seen, in the context of HOLA, as a discrete differential n-form ρ which we use in 2- and 3-dimensions to represent the mass density of a fluid. This is used in the context of free surface flows, where we must explicitly keep track of where the fluid is and how it occupies the computational domain $\mathscr{D}$. The immiscible interface between the fluid and vacuum is encoded at the 1/2 isovalue, and shortly we will see how the fluid equations of motion are coupled to this interface, for example with boundary conditions that the pressure $p = 0$ at the $\rho = 1/2$ isovalue. The discrete form ρ is a characteristic function that is initialized to one inside the fluid, and zero outside the fluid. This mass density evolves in time through Lie advection, *i.e.*, it adheres to the time evolution

$$\frac{\partial \rho}{\partial t} + \mathscr{L}_{\vec{u}}\, \rho = 0, \tag{2.1}$$

where the velocity field has been derived from the Navier-Stokes Equations, as discussed later in this chapter. Our mass density representation accommodates incompressible free surface fluid flows seamlessly. Contrary to the level set approach, mass preservation is easily achieved through our advection scheme guaranteeing that mass is only *exchanged* through faces, and hence conserved, via the HOLA scheme, as seen in Chapter 1. For the mass density ρ, in order to integrate the advection, we must compute the Lie derivative via

$$\begin{aligned} \mathscr{L}_{\vec{u}}\rho &= \mathbf{d}\mathbf{i}_{\vec{u}}\, \rho + \mathbf{i}_{\vec{u}}\mathbf{d}\, \rho &(2.2)\\ &= \mathbf{d}\mathbf{i}_{\vec{u}}\, \rho. &(2.3)\end{aligned}$$

Note that the second term of Cartan's formula drops out in this special setting; it is automatically zero because we take the discrete differential $\mathbf{d}$ of the n-form ρ, yielding a $n+1$-form living on a n-dimensional discrete manifold, which is thus always zero. The intuition behind the remaining term is to compute the flux through the face, as described by $\mathbf{i}_{\vec{u}}\rho$, and then take the sum of these terms adjacent to each cell, *i.e.*, the discrete differential $\mathbf{d}$, to obtain the net change of mass density. Preserving $\int_{\mathscr{D}} \rho$ through time helps to avoid the visual artifacts of volume loss traditionally present in particle-free surface capturing schemes—and without the memory and computational overhead associated with particles. A simple forward Euler time discretization is used to update ρ from a time step n to step $n+1$,

$$\rho^{n+1} = \rho^{n} + \Delta t\, \mathscr{L}_{\vec{u}}\rho^{n}. \tag{2.4}$$

Having a mass density provides us with more intricate visualizations than the traditional single level-set visualization: Figure 2.1 depicts volume renderings of a long simulation run, showing the complexity of the details captured in the density function even on a coarse grid—and mass

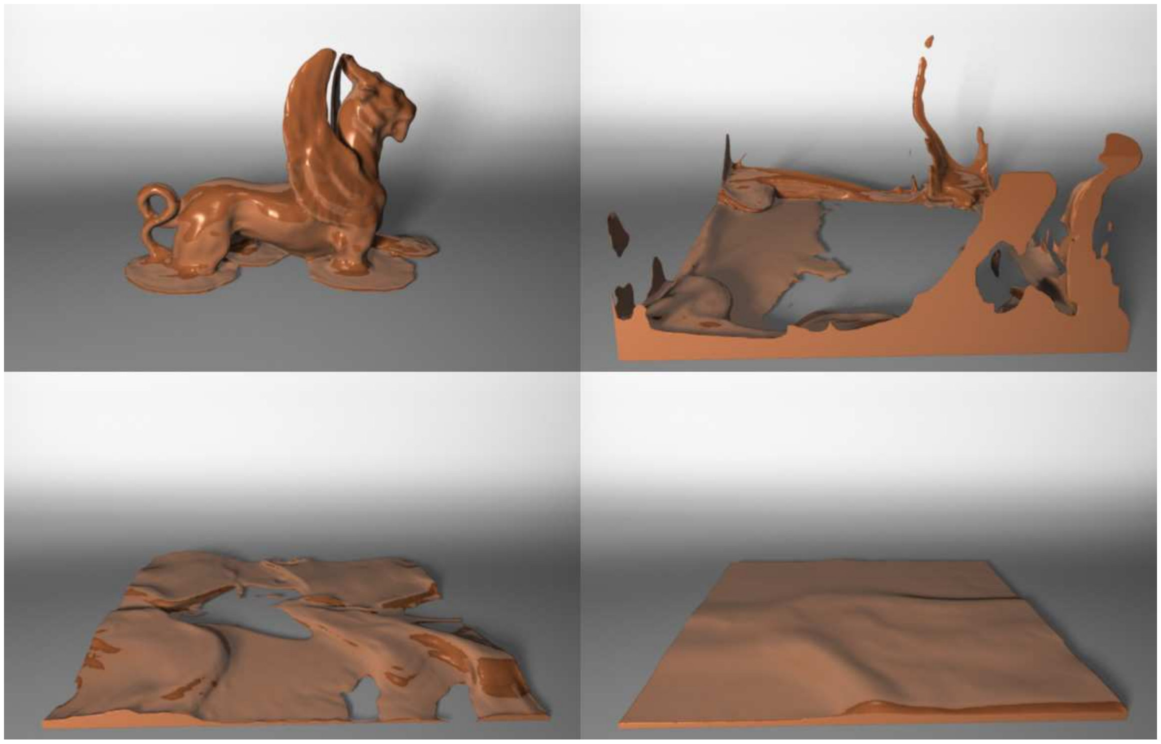

Figure 2.2: **Incompressible Fluid Simulation.** The feline flows into a thin layer of liquid, showing plenty of small-scale motion while preserving mass - all in the absence of any Lagrangian artifice. Grid Size: 128^3 for density, 64^3 for fluid solver.

is preserved throughout. Notice in Figure 2.2 that our density-based Eulerian formulation brings robustness to the simulation as even thin layers of fluid are treated appropriately. In the continuous setting, mass conservation and true volume preservation are synonymous under divergence freeness. While this is difficult to achieve in the discrete world, we can apply a sharpening procedure (as in [46]) which goes far to alleviate artifacts of spatial mass diffusion, but without the visual artifacts often seen with VOF methods. Note that our physically based interpretation of ρ as a mass density could also be amenable to high-speed *compressible* fluid models, while again maintaining mass conservation exactly.

2.2.1 Miscible Fluids

Miscible fluids can be simulated effortlessly as well, as multiple fluid densities are admissible in our representation. The total fluid mass density per cell is directly derived as the sum of these fluid densities. To demonstrate the mixing between liquids, we use a blending of colors to indicate the types of fluid (see Figure 2.3). In this particular case we initialize a ρ_r (red

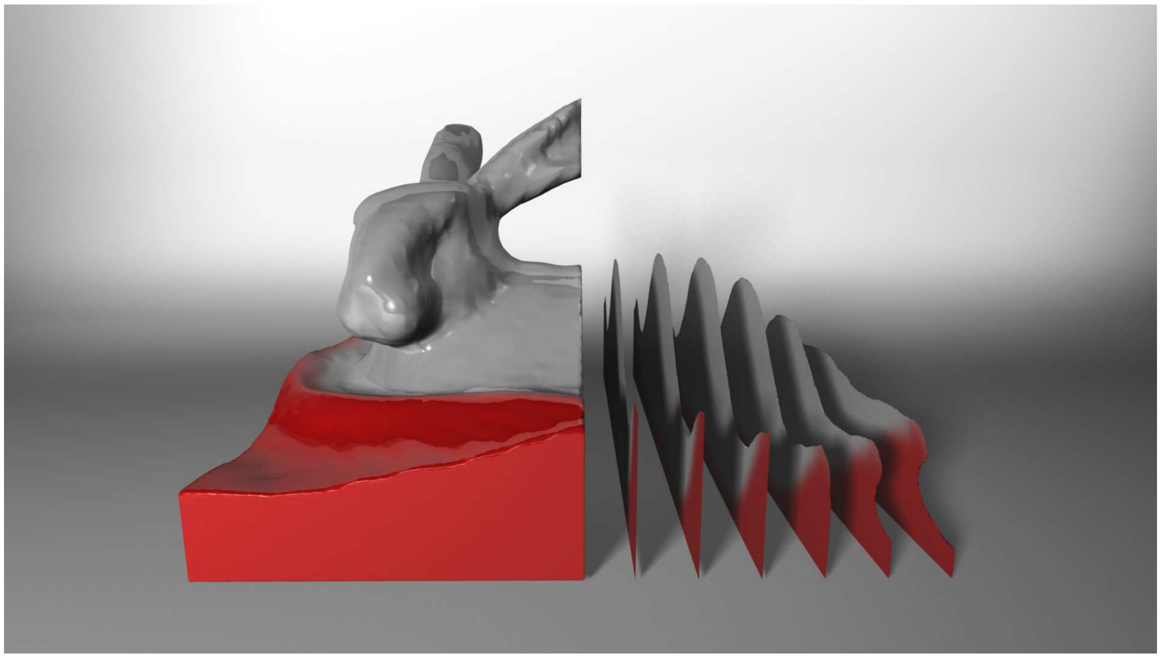

Figure 2.3: **Miscible Fluids.** Multiple fluid densities can be simulated in our density-based representation. Displayed are slices of two miscible fluids to demonstrate that mixing is easily achieved even on a very coarse grid size of 64^3.

fluid) and ρ_w (white fluid) individually, and evolve each in time by applying the Lie advection procedure to each mass density independently. In other words, we solve Equation 2.1 twice, once for ρ_r and once for ρ_w, to evolve the densities forward in time. Finally we take the union of the two densities, simply by summing the two $\rho = \rho_r + \rho_w$. This final ρ is then fed to the fluid solver in order to determine the velocity field for the next time step. If we wish to render the fluids, as in Figure 2.3, we extract the 1/2 isosurface through a Marching Cubes procedure [41], as is common for levelset contouring. For each vertex with position $\vec{x}$ in the extracted mesh, we evaluate the texture coordinate ζ,

$$\zeta(\vec{x}) = \frac{\rho_r(\vec{x})}{\rho_r(\vec{x}) + \rho_w(\vec{x})}, \tag{2.5}$$

as the proportion of the total fluid that is red at that location, whereby the ρ's are sampled using a simple trilinear interpolation over the cartesian grid. An appropriate one-dimensional texture map is then applied to the resulting triangle mesh surface during the rendering stage, in order to obtain the desired mixing effect in the final image.

2.3 Navier-Stokes Equations

In our attempts to numerically simulate the physical behavior of liquids, it is critical to have a thorough understanding of the dynamical system and underlying invariants and symmetries. The governing equations of incompressible fluid motion we consider here are the velocity formulation of the Navier-Stokes equations. A fluid of roughly constant density and temperature can be described as a velocity field $\vec{u}$ and a pressure field p. Assuming these quantities are known at some initial time $t = 0$, the evolution of these fields over time are described by a nonlinear set of PDE's, commonly written in condensed form as

$$\nabla \cdot \vec{u} = 0, \qquad \vec{u}|_{\partial \mathscr{D}} = 0, \tag{2.6}$$

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u} + \vec{g}, \tag{2.7}$$

where ν is the dissipative kinematic viscosity (*i.e.,* the "thickness"), ρ is the density and $\vec{g}$ defines external forces acting on the fluid (most commonly—gravity). In solving Equation 2.7, it is typical to label the terms by their physical interpretation, namely the *self-advection* term $\vec{u}(\nabla \cdot \vec{u})$, the *pressure* term $\frac{1}{\rho}\nabla p$ and the *diffusion* term $\nu \nabla^2 \vec{u}$. An important characteristic of liquids is conservation of mass; under the continuous model of Equations 2.6 and 2.7 this conservation law is synonymous with incompressibility under the divergence free constraint of Equation 2.6. In reality, liquids can vary in volume, especially in high-speed scenarios, but for practical purposes of the low-speed fluid scenarios we wish to model, this effect can be safely ignored. We will see in Section 2.4.2 that the incompressibility assumption accommodates a convenient solution of the pressure term in Equation 2.7. Meanwhile, the second part of Equation 2.6 simply states that the fluid should respect spatial boundary conditions of its enclosing domain $\mathscr{D}$; the perpendicular component of $\vec{u}$ at the boundary of the fluid domain $\partial \mathscr{D}$ is zero, and hence implies that no fluid will be advected through the solid boundaries of the computational domain, but can only run tangent to $\partial \mathscr{D}$. For a complete derivation of the Navier-Stokes Equations you are referred to the excellent fluid mechanics reference by Chorin and Marsden [17].

2.3.1 Domain Discretization

The Equations 2.6 and 2.7 lead to solutions in a continuous domain that are true in the point-wise sense. Clearly this is inconceivable for computational purposes, therefore it becomes nec-

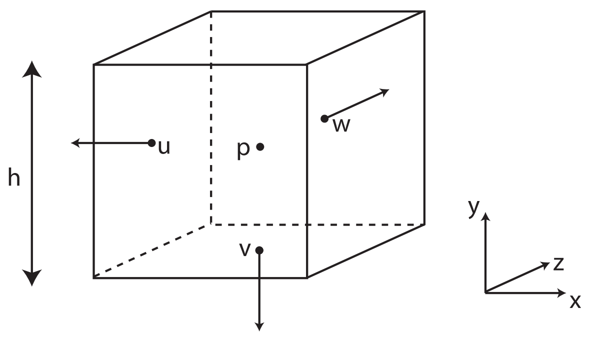

Figure 2.4: **MAC Grid.** For a given cell i, j, k, velocities are stored at the appropriate cell faces in a staggered grid formation, while pressure is stored at cell centers.

essary to choose a discrete domain on which to sample these continuous functions. The Marker-And-Cell grid stems from the simulating of fluids with free surfaces in CFD [64] and is perfect for our use with finite differencing spatial derivatives. Figure 2.4 shows a MAC cell of spacing h with the proper locations of pressure variables at the center, and a velocity vector whose x, y and z components (u, v, w, respectively) lie at the center of their respective cell face. These vector components are stored in what is known as a *staggered grid* formation, whereby the x-component is stored on the left face, the y-component on the bottom face and the z-component on the back face, and where each component can be thought of as a net *flux of mass* traveling through the representative face. The computational grid has dimensions $I \times J \times K$, indexed from zero. The scale of the solver is given by a *length* in meters for which $h = length / \min|I, J, K|$. As a general rule of thumb, all physical parameters are stored in SI units for simplicity and consistency.

Frequently velocity values will be required at locations that are not explicitly represented by our grid. In these situations, we apply piecewise linear basis functions to obtain a trilinear interpolation on all three velocity components to obtain an acceptable approximation of velocity at the desired location. One must be careful in implementing this, remembering to note that each component of the velocity is stored at a different location frame of reference (*i.e.*, all $x-$, $y-$, $z-$variables live on a different regular lattice). For instance, to find the velocity at the center of cell (i,j,k) we must evaluate

$$\vec{u}(ih, jh, kh) = \frac{1}{2}\begin{pmatrix} u_{i,j,k} + u_{i+1,j,k} \\ v_{i,j,k} + v_{i,j+1,k} \\ w_{i,j,k} + w_{i,j,k+1} \end{pmatrix}, \tag{2.8}$$

which is obviously not the same as the velocity stored in the MAC grid indexed by (i,j,k) which instead is $\vec{u}_{i,j,k} = (u_{i,j,k}, v_{i,j,k}, w_{i,j,k})$. For $\vec{u}_{i,j,k}$ the x-component is stored at $(ih - h/2, jh, kh)$, the y-component at $(ih, jh - h/2, kh)$ and the z-component at $(ih, jh, kh - h/2)$.

2.3.2 Spatial Derivatives through Finite Differencing

Finite differences are used to give approximations to spatial derivatives on a discrete lattice. Several derivatives will be required for solving Navier-Stokes, so we provide a brief overview of the discrete operators we employ. We begin with the continuous gradient operator ∇, for which a FD counterpart is defined. The discrete finite difference approximation of the gradient, for the pressure at cell (i,j,k), is the vector

$$(\nabla p)_{i,j,k} = \frac{1}{h}\begin{pmatrix} p_{i,j,k} - p_{i-1,j,k} \\ p_{i,j,k} - p_{i,j-1,k} \\ p_{i,j,k} - p_{i,j,k-1} \end{pmatrix}, \tag{2.9}$$

where each component of the vector is defined at the appropriate face of the cell (i,j,k). This approximation is a first-order centered difference operator, *i.e.*, errors grow with $\mathcal{O}(h^2)$ measured in the number of terms the Taylor series expands to. Next, the finite difference approximation of the divergence operator $\nabla\cdot$ is defined by taking the net flux of the velocity values adjacent to the cell (i,j,k), or in other words

$$(\nabla \cdot \vec{u})_{i,j,k} = \frac{1}{h}\left(u_{i+1,j,k} - u_{i,j,k} + v_{i,j+1,k} - v_{i,j,k} + w_{i,j,k+1} - w_{i,j,k}\right), \tag{2.10}$$

and is stored at the center of the cell. Finally, we require a discretization of the Laplacian operator ∇^2. A second-order approximation of the Laplacian operator, applied to a velocity component $u_{i,j,k}$, is given by

$$\nabla^2 u_{i,j,k} = \frac{1}{h^2}\left(u_{i-1,j,k} + u_{i,j-1,k} + u_{i,j,k-1} - 6u_{i,j,k} + u_{i+1,j,k} + u_{i,j+1,k} + u_{i,j,k+1}\right). \qquad (2.11)$$

In the upcoming Section 2.4.2, the Laplace operator is needed in the context of a Poisson equation we must solve to determine the pressure field of the fluid at a given time step, in addition to an implicit formulation of the viscosity term.

2.4 Updating the Velocity Field

We update the velocity field using Equations 2.6 and 2.7 at all grid cells that are full of fluid. We determine if a cell is included in the Navier-Stokes solve (if it is 'full' of fluid) by evaluating the mass density ρ, specifically if $\rho_i > 1 - \epsilon$, the cell i is included in the NS update, thus allowing pressure projection to a divergence-free velocity field to be computed in the usual fashion. Earlier, it was hinted at that the NS Equations are solved in a term-by-term fashion, temporarily relaxing the divergence free constraint in between time steps. The operator splitting technique used by Stam [58] is applied to solve Equation 2.7 in a fractional step integration, from step n to $n+1$ via a forward Euler time discretization

$$\vec{u}_1 = \vec{u}_n - \Delta t[(\vec{u}_n \cdot \nabla)\vec{u}_n - \vec{g}], \qquad (2.12)$$

$$\vec{u}_2 = \vec{u}_1 + \Delta t\ \nu \nabla^2 \vec{u}_1, \qquad (2.13)$$

$$\vec{u}_{n+1} = \vec{u}_2 - \frac{\Delta t}{\rho}\nabla p. \qquad (2.14)$$

In the following subsections we will discuss how to evaluate both the viscosity term $\nu\nabla^2\vec{u}$ and the advection term $(\vec{u}\cdot\nabla)\vec{u})$ through an unconditional stable approach. In section 2.4.3 we will see how our volumetric representation of the fluid, as it moves through the domain $\mathscr{D}$, interacts with the solution of the NS Equations through Dirichlet boundary conditions on the pressure.

2.4.1 Solving for Self-Advection and Body Forces

Self-advection is the phenomenon whereby disturbances in a vector field are propagated through the field over time. Alternatively, you can think of this as infinitesimally small pockets of fluid pushing and pulling on other pockets of fluids. A river, for example, is a flow dominated by self-advection (and gravity, obviously). To solve the self-advection term, we invoke the semi-Lagrangian method of characteristics employed by Stam in [58]. It is highly desirable to use a time integration method that is Total Variation Diminishing (TVD), in other words a scheme which does not produce any spurious oscillations. We will apply a fourth-order TVD Runge-Kutta routine for update of velocity. For a particle at $\vec{x}_n$, its position is updated from step n to $n+1$ via the equations

$$k_1 = \Delta t\, \vec{u}(\vec{x}_n), \tag{2.15}$$

$$k_2 = \Delta t\, \vec{u}(\vec{x}_n + \frac{k_1}{2}), \tag{2.16}$$

$$k_3 = \Delta t\, \vec{u}(\vec{x}_n + \frac{k_2}{2}), \tag{2.17}$$

$$k_4 = \Delta t\, \vec{u}(\vec{x}_n + k_3), \tag{2.18}$$

$$\vec{x}_{n+1} = \vec{x}_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathcal{O}(\Delta t^5). \tag{2.19}$$

The setup above give the fourth-order Runge-Kutta scheme applied to the ordinary differenatial equation

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{u}(\vec{x}), \tag{2.20}$$

which governs the motion of a particle in a velocity field. We treat each of the velocity components $u_{i,j,k}$, $v_{i,j,k}$, and $w_{i,j,k}$ at step n as particles originating at their respective face on the cell (i,j,k). We then ask the question “where was this particle at time step n-1?”. Once we have computed the solution to this question through the Runge-Kutta technique, we now know the position of the particle at step n that will be exactly on the face at step n+1. So for a particle positioned at $u_{i,j,k}$ we follow its movement back through time, evaluate the velocity at that location, and update $u_{i,j,k}$ with this freshly evaluated velocity. This Runge-Kutta scheme results in an implicit and unconditionally stable update through backtracking, as discussed previously.

We are now free to add in body forces. In recent history, inclusion of complex body forces have allowed for key-frame animation of fluid [63], while here it is simply included to account for the force of gravity. We specify $g = -9.81ms^{-2}$ and apply the formula

$$v_{i,j,k} = v_{i,j,k} + \Delta t\ g \tag{2.21}$$

for all v components in the velocity that should be updated. In particular, we add gravity to every y-component of velocity $v_{i,j,k}$ if and only if the adjacent cell above the face on which $v_{i,j,k}$ lives, *i.e.*, the cell (i, j, k), has some relevant amount of fluid that should feel the force of gravity. We measure this amount of fluid by looking at the mass density function ρ. For each $v_{i,j,k}$ we add gravity iff $\rho_{i,j,k} > \epsilon$, where ϵ is a tiny numerical cutoff threshold (*e.g.*, $\epsilon = 10^{-12}$).

This completes the computation of the first step in the operator splitting (Equation 2.12), leaving us with the resultant intermediary velocity field $\vec{u}_1$ which is a *best guess* velocity to the final solution. Now, let us adjust this field to account for pressure forces, a crucially important procedure in producing a visually believable liquid simulation. Note that in doing so, we skip the viscosity update in Equation 2.13. Due to numerical diffusion inherent with the implicit methods presented here, omitting the viscosity term is very common in practice. In the next section, we denote $\vec{u}_{temp}$ the intermediate velocity field $\vec{u}_1$ *or* the field $\vec{u}_2$ if viscosity is desired. We will discuss the viscosity update in Section 2.4.5, after presenting the pressure term, as our implicit viscosity formulation builds upon machinery (solving linear systems) that is discussed in the context of the pressure update. The pressure projection will yield the final velocity field $\vec{u}_{n+1}$ for step $n + 1$.

2.4.2 Pressure Projection through Helmholtz-Hodge Decomposition

Following Chorin and Marsden's treatment of the subject in [17], we introduce for the purpose of pressure projection, a well known mathematical result called the *Helmholtz-Hodge decomposition*. The Hodge decomposition states that any vector field $\vec{w}$ with appropriate boundary constraints can be uniquely decomposed into the following useful components,

$$\vec{w} = \nabla \times \vec{a} + \nabla\phi + \vec{b} \tag{2.22}$$

where $\vec{a}$ is a vector valued potential field, ϕ a scalar potential field and $\vec{b}$ is a "harmonic" vector field. Note that the divergence of $\nabla \times \vec{a}$ is zero ($\nabla \cdot (\nabla \times \vec{a}) = 0$), the rotational of $\nabla\phi$ is zero

$(\nabla \times (\nabla \phi) = 0)$ and that the harmonic field $\vec{b}$ is both divergence-free $(\nabla \cdot \vec{b} = 0)$ and curl-free $(\nabla \times \vec{b} = 0)$. It is simple to see how our fluid equations fit this form; the vector field $\nabla \times a$ is a divergence-free, mass-conserving vector field with the same properties we wish to enforce on $\vec{u}$ at the end of each time step. We hence set $\vec{w}$ to our best guess velocity field $\vec{u}_{temp}$ and ϕ to be our pressure field p, and see that a divergence-free field $\vec{u}$ can be created by subtracting from $\vec{u}_{temp}$ the gradient of the pressure field, ∇p. But how do we know what is the pressure p? We solve for a pressure such that the L_2 norm is minimized, *i.e.*, we search for a pressure such that subtracting its gradient causes minimum distortion between the initial velocity $\vec{u}_{temp}$ and the final velocity $\vec{u}$, while still ensuring that $\nabla \cdot \vec{u} = 0$. In order to achieve this, Equation 2.6 is applied to both sides of Equation 2.14, giving us

$$\nabla \cdot \vec{u}_{n+1} = \nabla \cdot \vec{u}_{temp} - \frac{\Delta t}{\rho} \nabla \cdot (\nabla p) = 0. \tag{2.23}$$

By rearranging Equation 2.23 we obtain a somewhat disguised Poisson equation,

$$\Delta t \nabla^2 p = \rho \nabla \cdot \vec{u}_{temp}, \tag{2.24}$$

which can be solved for the pressure field p. Once we have obtained the field p, it can be substituted back into Equation 2.14 to provide the final velocity $\vec{u}_{n+1}$. Let us analyze the form of Equation 2.24; the right hand side is known and can easily be computed explicitly, while the left hand side contains a Laplacian operator (refer in this case to the finite difference approximation in 2.11). Indeed, we are presented with one linear equation and one unknown variable for each of the N number of fluid cells, forming what is known as a *linear system* that must be solved for p. Linear systems are very frequently derived in applied mathematics and engineering, giving way to a wealth of literature and numerous numerical methods for solving such systems. While early fluid animation systems such as [29] used a Successive Over Relaxation iterate, in this text we will overview the preconditioned conjugate gradient algorithm (PCG) that provides faster convergence (*i.e.*, speed) through superior numerical conditioning.

2.4.3 Solving a Linear System via Preconditioned Conjugate Gradient

In order to use the PCG method, it is essential to create, from the fluid cells of the MAC grid, the proper matrices that are fed as input to the numerical system solver. A typical linear system is written in the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.25}$$

in which **A** is an $N \times N$ matrix for which each of the N rows represents an equation, **x** is a column vector of N unknown variables, and **b** is a column vector of N specified (known) values. To solve the system, it is necessary to compute $\mathbf{A}^{-1}$; a surprisingly non-trivial proposition, especially if **A** is large and/or nearly singular. Relating the Equation 2.24 with Equation 2.25, we set

$$\mathbf{A} \equiv -h^2\nabla^2, \tag{2.26}$$

scaling by a factor of $h^2/\Delta t$ to make the matrix entries of **A** integer (useful for preserving memory), and multiplying through by negative one (-1) to form a *positive definite* rather than negative definite—a prerequisite for convergence of the PCG algorithm we will apply shortly. We relate the **x** vector by

$$\mathbf{x} \equiv p, \tag{2.27}$$

as expected, and finally the vector **b** is set through the Equation

$$\mathbf{b} \equiv \frac{-h^2\rho}{\Delta t}\nabla \cdot \vec{u}_{temp}, \tag{2.28}$$

where the scaling by $-h^2/\Delta t$ is a consequence of earlier applying the same scaling to the left hand size. The divergence, $\nabla \cdot \vec{u}_{temp}$, is calculated via the finite difference Equation 2.10; note that it has a factor h in the denominator, effectively canceling out a h in the Equation above.

Neumann Boundary Conditions In creating the matrix **A** from the N fluid cells, it becomes apparent that the correct boundary conditions will be required to deal with two different kinds of situations. For fluid cells with one or more solid wall neighbors, *Neumann* boundary conditions are required because solid wall cells do not have an entry in **A**. We set a Neumann boundary condition of zero, implying that there is no pressure gradient between the solid wall and neighboring fluid. This is correct in our model because we explicitly enforce a zero velocity component perpendicular to the wall and therefore as we subtract off the pressure gradient, we expect this zero velocity to be maintained. Consider the left hand side of Equation 2.25, plugging in Equations 2.26 and 2.27. A line from this matrix representing an equation with unknown pressure values might look as follows,

$$-p_{i-1} - p_{j-1} - p_{k-1} + 6p - p_{i+1} - p_{j+1} - p_{k+1}, \tag{2.29}$$

where we have suppressed the irrelevant subscripts. This can be reformulated as

$$(p - p_{i-1}) + (p - p_{j-1}) + (p - p_{k-1}) + (p - p_{i+1}) + (p - p_{j+1}) + (p - p_{k+1}). \tag{2.30}$$

The Equation above clarifies how to appropriately deal with solid boundaries. Since we want the pressure gradient to be zero, the appropriate difference term simply falls out of the Equation above. For example, if cell $(i-1, j, k)$ is a solid wall, we remove the $(p - p_{i-1})$ term. Extensive discussion and further clarification can be found in the doctoral thesis of Mark Carlson [13].

Dirichlet Boundary Conditions In the alternate case, *Dirichlet* boundary conditions are needed for when a fluid cell is neighboring an empty cell that has no entry in matrix **A**. Because we assume a zero pressure in empty cells, this is easily deal with by setting corresponding entry in the matrix **A** to zero as well. A final word over the boundary conditions, is that there must be *at least one* Dirichlet boundary condition contained in the matrix **A**, in order to ensure **A** is not singular and remains uniquely invertible, *i.e.*, positive definite rather than positive semi-definite.

A more accurate pressure discretization at the free surface can be obtained by taking into account information from the mass density function ρ. We aim to enforce a pressure $p = 0$ at the immiscible interface. By setting Dirichlet boundary conditions of zero on the pressure at neighboring cells, we implicitly assume the surface passes through the center of such a neighboring cell (recall that pressure is cell-centered). Using the mass density, we can obtain a more accurate location of the interface $\rho = 1/2$ and use this to set Dirichlet conditions. We specify these more complex Dirichlet boundary conditions on the Poisson equation at the fluid/vacuum interface by setting the weight of the dual edge between two cells on opposite sides of the interface to $1/(\frac{1}{2} + \rho_i)$, where i is the cell on the 'vacuum' side of the interface. This is a heuristic similar in spirit to the level set fluid interface alternative described by Bridson [9], as we are ensuring that pressure is zero at the surface estimated at the distance of $\frac{1}{2} + \rho_i$ from the center of the fluid cell.

After creating the required matrices, we apply a conjugate gradient algorithm. For a good reference on learning about the theory of conjugate gradients and the fundamental principles they exploit, see [53] or [5], as such discussion is omitted here for brevity.

2.4.4 PCG Pseudo Code

Please refer to Algorithm 1. We solve $\mathbf{Ax} = \mathbf{b}$ using a preconditioner matrix $\mathbf{M}$. The $N \times N$ matrices $\mathbf{A}$ and $\mathbf{M}$ are known, as is the column vector $\mathbf{b}$. The solution is computed through an iterative process and stored in $\mathbf{x}$. As the method converges, $\mathbf{r}$ holds the residual of the solution. The vectors $\mathbf{p}$ and $\mathbf{q}$, and scalars α, β, φ, φ^{old} and b^{norm} store temporary values. The algorithm terminates when the maximum number of iterations, $iter^{max}$, is reached or when the solution is found up to a tolerance threshold ϵ. For a typical simulations, we set $iter^{max} = 1000$ and $\epsilon = 10^{-12}$ when computing with 32-bit floating point (doubles). In general, larger systems will have increasingly ill-conditioned matrices $\mathbf{A}$, so it may be necessary to relax these conditions by increasing $iter^{max}$ and/or ϵ. The preconditioner $\mathbf{M}$ can be used to improve the numerical

Algorithm 1: Preconditioned Conjugate Gradient

```
$\varphi = 0$
$b^{norm} = \sqrt{\mathbf{b} \cdot \mathbf{b}}$
$\mathbf{r} = \mathbf{b} - \mathbf{Ax}$

for $i = 0$ to $iter^{max}$ do
    solve $\mathbf{Mz} = \mathbf{r}$
    $\varphi^{old} = \varphi$
    $\varphi = \mathbf{r} \cdot \mathbf{z}$
    if $\varphi == 0$ or $\sqrt{\varphi} \leq \epsilon b^{norm}$ then
        break
    if $i == 0$ then
        $\mathbf{p} = \mathbf{z}$
    else
        $\beta = \varphi / \varphi^{old}$
        $\mathbf{p} = \mathbf{z} + \beta \mathbf{p}$
    $\mathbf{q} = \mathbf{Ap}$
    $\alpha = \varphi / (\mathbf{p} \cdot \mathbf{q})$
    $\mathbf{x} = \mathbf{x} + \alpha \mathbf{p}$
    $\mathbf{r} = \mathbf{r} - \alpha \mathbf{q}$
```

conditioning of the solution, allowing for improved convergence. Many preconditioners can be used here, such as Gauss-Siedel, Incomplete LU, Incomplete Cholesky, and others. A trivial preconditioner to implement is the *Jacobi* preconditioner, where $\mathbf{M} = \operatorname{diag} \mathbf{A}$, hence making the inversion required by line 5 of Algorithm 1 very easy; recall in this special case that $\mathbf{M}^{-1}$ is given by the element-wise reciprocal of the diagonal terms of $\mathbf{M}$. Alternate preconditioners are also possible. In particular, a popular choice in fluid animation literature is the *Incomplete Cholesky* (IC) which is generally considered to have improved numerical properties, at the cost of a delicate implementation. In this case we therefore recommend the reader to TAUCS [61], a packaged and freely available library for efficiently solving sparse linear systems, which includes many possible preconditioners, including IC, for users to experiment with.

2.4.5 Implicit Solution to Viscosity

The most straightforward way to solve the viscosity term $\nu\nabla^2\vec{u}$ is to discretize the Laplacian as in Equation 2.11. We can then express this viscosity term as a matrix $\mathbf{V}$, and use a forward Euler step of viscosity from the temporary operator splitting velocity field $\vec{u}_1$ to find $\vec{u}_2$ via

$$\vec{u}_2 = \vec{u}_1 + \Delta t \;\; \mathbf{V}\vec{u}_1. \tag{2.31}$$

For a particular velocity, for example the x-component at cell (i, j, k), the line in the matrix $\mathbf{V}$ for this particular equation may look like

$$-u_{i-1} - u_{j-1} - u_{k-1} + 6u - u_{i+1} - u_{j+1} - u_{k+1}, \tag{2.32}$$

where we have suppressed the obvious subscripts. Note that we apply such a matrix $\mathbf{V}$ to x, y, and z-components individually. Unfortunately the explicit forward Euler discretization above leads to severe constraints on the time step Δt. When the kinematic viscosity ν is large, the time integration suffers from stiffness, and through stability analysis we determine the time constraint is given by

$$\Delta t < \frac{h^2}{6\nu}. \tag{2.33}$$

This condition is particularly severe; for advection we saw that the CFL condition is proportional to the cell size h, $\Delta t \sim h$, while here for viscosity we have found that $\Delta t \sim h^2$. This quickly becomes computationally untractable as we refine the mesh, *i.e.*, when reducing h.

In order to achieve stability, we convert the explicit matrix formulation into an implicit backward Euler technique. This can handle arbitrarily large time steps, and although it requires the solution to a costly linear system, it circumvents the need to adhere to the small time step imposed to enforce stability in explicit schemes. The net is result is that this implicit scheme is actually faster, in practice. The backward Euler formulation, first presented by Stam in [58], is given by

$$(\mathbf{I} - \Delta t \; \mathbf{V})\, \vec{u}_2 = \vec{u}_1, \tag{2.34}$$

where $\mathbf{I}$ is the identity matrix. We therefore must solve for $(\mathbf{I} - \Delta t \; \mathbf{V})^{-1}$ and apply it to our velocity field $\vec{u}_n$ to account for velocity. We use the PCG method presented in Section 2.4.3 as a means to solve the linear system in Equation 2.34. This is quite simple, since the pressure projection machinery (PCG solver) is already in place.

2.5 Computational Cycle

In this section we present the computational cycle of the complete fluid solver to perform a time update through a brief explanation of each of the algorithmic steps not yet discussed. An excellent supplement to this discussion can again be found in [13]. Let us examine the remaining steps required to complete the picture—dynamically determining a time step during simulation, enforcing physical boundaries, and velocity extrapolation.

2.5.1 Selecting a Time Step Δt

As a predefined user input to the system, a *frame rate* for the simulation should be specified. Typically this might be set to $1/24^{th}$ of a second to parallel the speed of film, $1/30^{th}$ for video, and perhaps even a faster speed if slow motion footage or motion bluring effects are desired. In the case of video, we are thus required to save and store the state of the liquid/air interface thirty times per second of simulation time for further post-processing; isosurface extraction, rendering and so forth. As we have seen previously, the Courant-Friedreichs-Lewy (CFL) condition will limit explicit simulation time steps for numerical stability, and although our simulator is unconditionally stable for large time steps through use of implicit formulations, we choose to moderate this by marching forward in time by a CFL factor of $1/2$ in order to obtain a visually acceptable accuracy, rather than an arbitrarily large time step, for example $\Delta t = 1/30$. Because an output frames is desired at the specified frame rate, we introduce the notion of time subcycling. At each iteration of the algorithm we select the upper threshold on the time step,

$$\Delta t = \frac{1}{2}\left(\frac{h}{\sqrt{3}\max|\vec{u}|}\right), \tag{2.35}$$

unless the step required to progress the absolute simulation time t to the next frame output point is less than the Δt defined above. Intuitively, from Equation 2.35 it follows that very turbulent fluid flows will require more iterations per frame than calm flows, since the CFL condition is inversely proportional to the maximum velocity in the grid. The time step Δt is maintained throughout the iteration, and is used to guide both the update in velocity field $\vec{u}$ as well as the evolution of the fluid's mass density.

2.5.2 Enforcing Physical Domain Boundaries

Recall that the second component of Equation 2.6, $\vec{u}|_{\partial\mathscr{D}} = 0$ ensures that fluid cannot travel through the solid wall interface that encases the valid fluid domain $\mathscr{D}$. Because the perpendicular velocity components of $\vec{u}$ are defined precisely on the wall interface $\partial\mathscr{D}$, and because we said earlier that perpendicular velocity must be eliminated, we simply set to zero all the velocity components of cell faces that constitute a portion of the wall. In other words, $u_{1,j,k} = u_{I-1,j,k} = 0\ \forall j,k$, and similarly in the other dimensions. While we have accounted for the velocities at the interface $\partial\mathscr{D}$, it is necessary to correctly set the velocities tangent to the solid wall as well, *i.e.*, the velocities *inside* the wall. This is true because we are using an interpolation scheme to obtain valid velocity values, so particles near the wall will be effected by such tangential values, even although they remain inside $\mathscr{D}$. Three options exist for specifying these velocities. Slip conditions simply try to facilitate the motion of the flow by specifying that $v_{0,j,k} = v_{1,j,k}\ \forall j,k$, anti-slip conditions purposely hinder the flow by setting $v_{0,j,k} = -v_{1,j,k}\ \forall j,k$, while 'do nothing' conditions — $v_{0,j,k} = 0\ \forall j,k$ — admittedly do nothing. The other dimensions are set similarly. Under grid refinement, as $h \to 0$, anti-slip conditions are the physically correct choice because it is reasonable to expect that surface friction will hinder fluid motion. Over a discrete lattice, however, we have frequently chosen slip conditions because they help to keep the liquid motion lively and interesting for longer, in accordance with our goals as a computer graphics simulation.

2.5.3 Velocity Extrapolation

We perform a velocity extrapolation similar in spirit to Adalsteinsson and Sethian [2]. Velocity extrapolation is a heuristic used to counteract the velocity dissipation caused by trilinear interpolation of the velocity during the semi-Lagrangian self advection step. By extrapolating the velocity field to regions outside of the fluid (not accounted for by the Navier-Stokes update), we create the effect of an "upwind" advection scheme to counteract the dissipation, as the velocities unaccounted for by the Navier-Stokes update will not be used during any interpolation. The procedure exploits the use of a queue to flood the domain, growing velocities from the fluid into the empty neighbors.

To initialize, we cycle through all cells in the mesh and add to the queue all cells along the vacuum / liquid interface, specifically all cells 'i' where $\rho_i \leq 1/2$, and where at least one adjacent cell 'j' has $\rho_j > 1/2$. We also cycle through all faces, tagging velocities as 'valid' if at least one cell adjacent to the face is on the liquid side of the interface, *i.e.*, if $\rho_i > 1/2$ and / or $\rho_j > 1/2$. We then begin to process the queue: for each cell in the queue, we cycle through all velocities that are not tagged as 'valid'. We evaluate the neighboring velocities of the same orientation (*e.g.* $x-$, y- or z-components) using a six-point stencil, and assign to the face the average velocity of all the neighbors tagged with a 'valid' velocity. This updated face is then itself tagged as a 'valid' velocity. Once we have performed this computation on each of the faces of the cell, all faces will have 'valid' velocities. We continue the extrapolation flooding by adding any adjacent empty cells to the queue. Once the queue is empty, the entire domain has been marked with valid velocities.

2.6 Conclusions

In this chapter we have detailed a MAC based algorithm for discretization of the Navier-Stokes equations, and presented a novel mass-density representation of fluid with conservative advection using a forward Euler HOLA scheme. This yields robust free surface simulations, as mass is preserved in the computational domain for all time. The mass-density representation also accommodates volumetric interaction of miscible fluids through use of multiple density fields.

Bibliography

[1] R. Abraham, J. Marsden, and T. Ratiu, editors. *Manifolds, Tensor Analysis, and Applications*. Applied Mathematical Sciences Vol. 75, Springer, 1988.

[2] D. Adalsteinsson and J. Sethian. The Fast Construction of Extension Velocities in Level set Methods. *J. Comput. Physics*, 148:2–22, 1999.

[3] D. Arnold, P. Bochev, R. Lehoucq, R. Nicolaides, and M. Shashkov, editors. *Compatible Spatial Discretizations*, volume 142 of *The IMA Volumes in Mathematics and its Applications*. Springer, 2006.

[4] D. N. Arnold, R. S. Falk, and R. Winther. Finite Element Exterior Calculus, Homological Techniques, and Applications. *Acta Numerica*, 15:1–155, 2006.

[5] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.

[6] P. B. Bochev and J. M. Hyman. Principles of Mimetic Discretizations of Differential Operators. *IMA Volumes In Mathematics and its Applications*, 142:89–119, 2006.

[7] A. Bossavit. *Computational Electromagnetism*. Academic Press, Boston, 1998.

[8] A. Bossavit. Extrusion, Contraction : their Discretization via Whitney Forms. *COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, 22(3):470–480, 2003.

[9] R. Bridson, R. Fedkiw, and M. Müller-Fischer. Fluid Simulation. In *ACM SIGGRAPH Course Notes*, 2006.

[10] S. Bryson and D. Levy. Mapped WENO and Weighted Power ENO Reconstructions in Semi-Discrete Central Schemes for Hamilton-Jacobi Equations. *Applied Numerical Mathematics*, 56:1211–1224, 2006.

[11] W. L. Burke. *Applied Differential Geometry*. Cambridge University Press, 1985.

[12] C. Canuto, M. Yousuff-Hussaini, A. Quarteroni, and T. Zang. *Spectral Methods in Fluid Dynamics*. Series in Computational Physics. Springer-Verlag, 1987.

[13] M. Carlson. *Rigid, Melting, and Flowing Fluids*. PhD thesis, Georgia Institute of Technology, 2004.

[14] S. Carroll. *Spacetime and Geometry: An Introduction to General Relativity*. Pearson Education, 2003.

[15] É. Cartan. *Les Systèmes Differentiels Exterieurs et leurs Applications Géometriques*. Hermann, Paris, 1945.

[16] W. Chang, F. Giraldo, and B. Perot. Analysis of an Exact Fractional Step Method. *Journal of Computational Physics*, 180(3):183–199, Nov. 2002.

[17] A. J. Chorin and J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Springer, New York, 1993.

[18] R. Courant, E. Issacson, and M. Rees. On the Solution of Nonlinear Hyperbolic Differntial Equations by Finite Differences. *Comm. Pure and Applied Math*, 5:243–255, 1952.

[19] K. Crane. Free Surface Flows Implemented on the GPGPU. Promotional demonstration nVidia available at http://www.acm.uiuc.edu/ kcrane/www/, 2006.

[20] M. Desbrun, E. Kanso, and Y. Tong. Discrete Differential Forms for Computational Sciences. In E. Grinspun, P. Schröder, and M. Desbrun, editors, *Discrete Differential Geometry*, Course Notes. ACM SIGGRAPH, 2006.

[21] T. Dupont and Y. Liu. Back-and-Forth Error Compensation and Correction Methods for Removing Errors Induced by Uneven Gradients of the Level Set Function. *Journal of Computational Physics*, 190(1):311Ű–324, 2003.

[22] V. Dyadechko and M. Shashkov. Moment-of-Fluid Interface Reconstruction. LANL Technical Report LA-UR-05-7571, 2006.

[23] S. Elcott, Y. Tong, E. Kanso, P. Schröder, and M. Desbrun. Stable, Circulation-Preserving, Simplicial Fluids. *ACM Trans. Graph.*, 26(1):4, 2007.

[24] B. Engquist and S. Osher. One-Sided Difference Schemes and Transonic Flow. *PNAS*, 77(6):3071–3074, 1980.

[25] D. Enright, S. Marschner, and R. Fedkiw. Animation and Rendering of Complex Water Surfaces. *ACM Trans. Graph.*, 21(3):736–744, 2002.

[26] R. Fedkiw, J. Stam, and H. W. Jensen. Visual Simulation of Smoke. *ACM Trans. Graph.*, 20(3):15–22, 2001.

[27] H. Flanders. *Differential Forms and Applications to Physical Sciences*. Dover Publications, 1990.

[28] N. Foster and R. Fedkiw. Practical Animation of Liquids. *ACM Trans. Graph.*, 20(3):23–30, 2001.

[29] N. Foster and D. Metaxas. Realistic Animation of Liquids. *Graphics Models and Image Processing*, 58:471–483, 1996.

[30] T. Frankel. *The Geometry of Physics*. Second Edition. Cambridge University Press, United Kingdom, 2004.

[31] P. Frolkovic and K. Mikula. High-resolution Flux-based Level Set Method. Preprint 2005-12, 2005. Department of Mathematics and Descriptive Geometry, Slovak University of Technology, Bratislava.

[32] X. Gu and S.-T. Yau. Global Conformal Surface Parameterization. In *Proc. Symp. Geometry Processing*, pages 127–137, 2003.

[33] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for ODEs*. Springer, 2002.

[34] F. H. Harlow and J. E. Welch. Numerical Calculation of Time-dependent Viscous Incompressible Flow of Fluid with Free Surfaces. *Phys. Fluids*, 8:2182Ű–2189, 1965.

[35] D. J. Hill and D. I. Pullin. Hybrid Tuned Center-Difference-WENO Method for Large Eddy Simulations in the Presence of Strong Shocks. *J. Comput. Phys.*, 194(2):435–450, 2004.

[36] A. N. Hirani. *Discrete Exterior Calculus*. PhD thesis, Caltech, May 2003.

[37] A. Iske and M. Käser. Conservative Semi-Lagrangian Advection on Adaptive Unstructured Meshes. *Numerical Methods for Partial Differential Equations*, 20(3):388–411, 2004.

[38] H.-P. Langtangen, K.-A. Mardal, and R. Winter. Numerical Methods for Incompressible Viscous Flow. *Advances in Water Resources*, 25:1125–1146, 2002.

[39] D. Levy, S. Nayak, C. Shu, and Y. Zhang. Central WENO Schemes for Hamilton-Jacobi Equations on Triangular Meshes. *J. Sci. Comput.*, 27:532–552, 2005.

[40] X. Liu, S. Osher, and T. Chan. Weighted Essentially Non-oscillatory Schemes. *J. Sci. Comput.*, 126:202–212, 1996.

[41] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Proceedings of SIGGRAPH*, 21(4):163–169, 1987.

[42] D. Lovelock and H. Rund. *Tensors, Differential Forms, and Variational Principles*. Dover Publications, 1993.

[43] J. E. Marsden and M. West. Discrete Mechanics and Variational Integrators. *Acta Numerica*, 2001.

[44] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid Control using the Adjoint Method. *ACM Trans. Graph.*, 23(3), 2004.

[45] S. Morita. *Geometry of Differential Forms*. Translations of Mathematical Monographs, Vol. 201. Am. Math. Soc., 2001.

[46] P. Mullen, A. McKenzie, Y. Tong, and M. Desbrun. A Variational Approach to Eulerian Geometry Processing. *ACM Trans. on Graphics (SIGGRAPH)*, Aug. 2007.

[47] J. R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, Menlo Park, CA, 1984.

[48] J. Nédélec. Mixed Finite Elements in 3D in H(div) and H(curl). 1192, 1986.

[49] R. A. Nicolaides and X. Wu. Covolume Solutions of Three Dimensional DIV-CURL Equations. *SIAM J. Numer. Anal.*, 34:2195, 1997.

[50] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2003.

[51] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable Photorealistic Liquids. *Symp. on Computer Animation*, pages 193–202, 2004.

[52] J. A. Sethian. *Level Set Methods and Fast Marching Methods*, volume 3 of *Monographs on Appl. Comput. Math.* Cambridge University Press, Cambridge, 2nd edition, 1999.

[53] J. Shewchuk. An Introduction to the Conjugate Gradient Method without the Agonizing Pain, 1994.

[54] J. Shi, C. Hu, and C. Shu. A Technique for treating Negative Weights in WENO Schemes. *J. Comput. Phys.*, 175:108–127, 2002.

[55] L. Shi and Y. Yu. Taming Liquids for Rapidly Changing Targets. In *Symp. on Computer Animation*, pages 229–236, 2005.

[56] C. Shu and S. Osher. Efficient Implementation of Essentially non-Oscillatory Shock Capturing Schemes. *J. Sci. Comput.*, 77:439–471, 1988.

[57] C.-W. Shu. *Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws*, volume 1697 of *Lecture Notes in Mathematics*, pages 325–432. Springer, 1998.

[58] J. Stam. Stable Fluids. *ACM Trans. Graph.*, 18(3):121–128, 1999.

[59] A. Staniforth and J. Cote. Semi-Lagrangian Integration Schemes for Atmospheric Models—a Review. *Monthly Weather Review*, 119:2206–2223, 1952.

[60] V. A. Titarev and E. F. Toro. Finite-Volume WENO Schemes for Three-Dimensional Conservation Laws. *J. Comput. Phys.*, 201(1):238–260, 2004.

[61] S. Toledo. Taucs. Software at http://www.tau.ac.il/ stoledo/taucs/, 2003.

[62] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing Quadrangulations with Discrete Harmonic Forms. In *Proc. Symp. Geometry Processing*, pages 201–210, 2006.

[63] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe Control of Smoke Simulations. *ACM Trans. Graph.*, 22(3):716–723, 2003.

[64] J. Welch, F. Harlow, J. Shannon, and B. Daly. The MAC Method: A Computing Technique for solving Viscous, Incompressible, Transient Fluid-Flow Problems involving Free Surfaces. *Report LA-3424*, 1965.

[65] H. Whitney. *Geometric Integration Theory*. Princeton Press, Princeton, 1957.

[66] M. Wiebe and B. Houston. The Tar Monster: Creating a Character with Fluid Simulation. In *ACM SIGGRAPH 2004 Sketches*, 2004.

[67] Y. Zhang and C. Shu. High-Order WENO Schemes for Hamilton-Jacobi Equations on Triangular Meshes. *J. Sci. Comput.*, 24:1005–1030, 2003.