Stable, Circulation-Preserving, Simplicial Fluids

Figure 1: *Discrete Fluids: we present a novel geometric integration scheme for fluids applicable to tetrahedral meshes of arbitrary domains. Aside from resolving the boundaries precisely, our approach also provides an accurate treatment of vorticity through a discrete preservation of Kelvin's circulation theorem. Here, a hot smoke cloud rises inside a bunny shaped domain of* $7K$ *vertices (*$32K$ *tets—the equivalent complexity of a regular* 19^3 *grid), significantly reducing the computational cost of the simulation for such an intricate boundary compared to regular grid-based techniques (*0.47*s/frame on a Pentium 4, 3GHz).*

Abstract

Visual quality, low computational cost, and numerical stability are foremost goals in computer animation. An important ingredient in achieving these goals is the conservation of fundamental motion invariants. For example, rigid and deformable body simulation benefits greatly from conservation of linear and angular momenta. In the case of fluids, however, none of the current techniques focuses on conserving invariants, and consequently, they often introduce a visually disturbing numerical diffusion of *vorticity*. Visually just as important is the resolution of complex simulation domains. Doing so with regular (even if adaptive) grid techniques can be computationally delicate.

In this paper, we propose a novel technique for the simulation of fluid flows. It is designed to respect the defining differential properties, *i.e.*, the *conservation of circulation* along arbitrary loops as they are transported by the flow. Consequently, our method offers several new and desirable properties: (1) arbitrary simplicial meshes (triangles in 2D, tetrahedra in 3D) can be used to define the fluid domain; (2) the computations are efficient due to discrete operators with small support; (3) the method is stable for arbitrarily large time steps; (4) it preserves *discrete circulation* avoiding numerical diffusion of vorticity; and (5) its implementation is straightforward.

1 Introduction

Conservation of motion invariants at the discrete computational level, *e.g.*, linear and angular momenta in solid mechanics, is now widely recognized as being an important ingredient in the construction of numerically stable simulations with a high degree of visual realism [Marsden and West 2001]. Much of the progress in this direction has been enabled by a deeper understanding of the underlying geometric structures and how they can be preserved as we go from continuous models to discrete computational realizations. So far, advances of this type have yet to deeply impact fluid flow simulations. Current methods in fluid simulation are rarely designed to conserve *defining physical properties*. Consider, for example, the need in many methods to continually project the numerically updated velocity field onto the set of divergence free velocity fields; or the need to continually reinject vorticity lost due to numerical dissipation as a simulation progresses. In this paper we introduce a novel, geometry-based algorithm for fluid simulation which, among other desirable properties, exactly preserves vorticity at a discrete level.

1.1 Previous Work

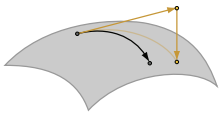
Fluid Mechanics has been studied extensively in the scientific community both mathematically and computationally. The physical behavior of incompressible fluids is usually modeled by the Navier Stokes (NS) equations for viscous fluids and by the Euler equations for inviscid (non-viscous) fluids. Numerical approaches in computational fluid dynamics typically discretize the governing equations through Finite Volumes (FV), Finite Elements (FE) or Finite Differences (FD) methods. We will not attempt to review the many methods proposed (an excellent survey can be found in [Langtangen et al. 2002]) and instead focus on approaches used for fluids in computer graphics. Some of the first fluid simulation techniques used in the movie industry were based on Vortex Blobs [Yaeger et al. 1986] and Finite Differences [Foster and Metaxas 1997]. To circumvent the ill-conditioning of these iterative approaches for large time steps and achieve unconditional stability, Jos Stam [1999; 2001] introduced to the graphics community the *method of characteristics* for fluid advection and the Helmholtz-Hodge decomposition to ensure the divergence-free nature of the fluid motion [Chorin and Marsden 1979]. The resulting algorithm, called *Stable Fluids*, is an extremely successful semi-Lagrangian approach based on a regular grid Eulerian space discretization, that led to many refinements and extensions which have contributed to the enhanced visual impact of fluid animations. Among others, these include the use of staggered grids and monotonic cubic interpolation [Fedkiw et al. 2001]; improvements in the handling of interfaces [Foster and Fedkiw 2001]; extensions to curved surfaces [Stam 2003; Tong et al. 2003; Shi and Yu 2004] and visco-elastic objects [Goktekin et al. 2004]; and goal oriented control of fluid motion [Treuille et al. 2003; McNamara et al. 2004; Pighin et al. 2004; Shi and Yu 2005].

1.2 Shortcomings of Stable Fluids

However, the Stable Fluids technique is not without drawbacks. Chief among them is the difficulty of accomodating complex domain boundaries with regular grids, as addressed recently with hybrid meshes [Feldman et al. 2005]. Local adaptivity [Losasso et al. 2004] can greatly help, but the associated octree structures require significant overhead. Note that regular partitions of space (adaptive or not) can suffer from preferred direction sampling, leading to visual artifacts similar to aliasing in rendering.

Additionally, the different variants of the original Stable Fluids algorithm [Stam 1999] are all based on a class of discretization approaches known in Computational Fluid Dynamics as fractional

step methods. In order to numerically solve the Euler equations over a time step, they proceed in two stages. They first update the velocity field assuming the fluid is inviscid and disregard the divergence-free constraint. Then, the resulting velocity is projected onto the closest divergence-free flow (in the $\mathcal{L}^2$ sense) through an exact Helmholtz-Hodge decomposition. Despite the simplicity of this fractional integration, one of its consequences is excessive numerical diffusion: advecting velocity before reprojecting onto a divergence-free field creates major energy loss. While this shortcoming is not a major issue in graphics as the visual impact of such a loss is not always significant, another consequence of the fractional integration is an *excessive diffusion of vorticity*. This last issue affects the motion significantly, since the presence of vortices in liquids and volutes in smoke is one of the most important visual cues of our perception of fluidity. Vorticity confinement [Steinhoff and Underhill 1994; Fedkiw et al. 2001] was designed to counteract this diffusion through local reinjection of vorticity. Unfortunately, it is hard to control how much can safely be added back without affecting stability or plausibility of the results. Adding vortex particles can further reduce this loss [Guendelman et al. 2005], but adds computational cost to the Stable Fluids method without suppressing the loss completely. One can understand this seemingly inevitable numerical diffusion through the following geometric argument: the solutions of Euler equations are *geodesic* (*i.e.*, shortest) paths *on* the manifold of all possible divergence-free flows; thus, advecting the fluid *out* of the manifold is not a proper substitute to this intrinsic constrained minimization, even if the post re-projection is, in itself, exact. For a more detailed numerical analysis of this flaw, see [Chang et al. 2002].



1.3 Contributions

In this paper we show that a careful setup of *discrete differential quantities*, designed to respect structural relationships such as vector calculus identities, leads in a direct manner to a numerical simulation method which respects the defining *geometric structure* of the fluid equations. A key ingredient in this approach is the location of physical quantities on the appropriate geometric structures (*i.e.*, vertices, edges, faces, or cells). This greatly simplifies the implementation as all variables are *intrinsic*. It also ensures that the approach works for curved manifolds without any changes. With the help of a *discrete calculus on simplicial complexes* we construct a novel integration scheme which employs intrinsically divergence-free variables. This removes the need to enforce the usual divergence-free constraint through a numerically lossy projection step. Our time integration method *by construction* preserves circulation and consequently vorticity. It accomplishes this while being simple, numerically efficient, and unconditionally stable, achieving high visual quality even for large time steps. Although our approach shares the same algorithmic structure as Stable Fluids based methods (use of backward path tracing and sparse linear systems), it fundamentally differs from previous techniques on the following points:

- **our technique is based on a classical *vorticity* formulation of the Navier-Stokes and Euler equations**; unlike most vorticity-based methods in CFD and CG, our approach is *Eulerian* as we work only with a fixed mesh and *not* a Lagrangian representation involving vorticity particles (or similar devices);
- we use an unconditionally-stable, semi-Lagrangian backward advection strategy for vertices just like Stable Fluids; **in contrast to Stable Fluids however we do not point-sample velocity, but rather compute *integrals of vorticity***; this simple change removes the need to enforce incompressibility of the velocity field through projection;
- **our strategy exactly preserves circulation along discrete loops in the mesh**; capturing this *geometric property* of fluid dynamics guarantees that vorticity does not get dissipated as is typically the case in Stable Fluids; consequently no vorticity confinement (or other post processes) are required to maintain this important element of visual realism;
- **our method has multiple advantages from an implementation point of view**: it handles arbitrary meshes (regular grids, hybrid meshes [Feldman et al. 2005], and even cell complexes) with non-trivial topology; the operators involved have very small support leading to very sparse linear systems; all quantities are stored intrinsically (scalars on edges and faces) without reference to global or local coordinate frames; the computational cost is comparable to previous approaches.

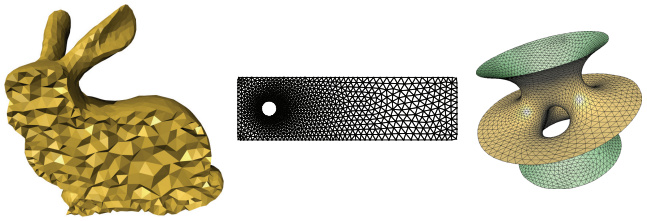


Figure 2: *Domain Mesh: our fluid simulator uses a simplicial mesh to discretize the equations of motion; (left) the domain mesh (shown as a cutaway view) used in Fig. 1; (middle) a coarser version of the flat 2D mesh used in Fig. 8; (right) the curved triangle mesh used in Fig. 10.*

The organization of this paper is as follows. In Section 2, we motivate our approach through a brief overview of the theory and computational algorithms for Fluid Mechanics. We present a novel discretization of fluid mechanics and a circulation-preserving integration algorithm in Section 2.2. The computational machinery required by our approach is developed in Section 3, while the algorithmic details are given in Section 4. Several numerical examples are shown and discussed in Section 5.

2 Of Motion & Discrete Flow Representation

Before going into the details of our algorithm we discuss the underlying fluid equations with their relevant properties and how these can be captured over discretized domains.

2.1 Geometry of Fluid Motion

Euler Fluids Consider an inviscid, incompressible, and homogeneous fluid on a domain $\mathcal{D}$ in 2 or 3D. The *Euler equations*, governing the motion of this fluid (with no external forces for now), can be written as:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= -\nabla p \,, \\ div(\mathbf{u}) &= 0 \,, \quad \mathbf{u} \parallel \partial\mathcal{D} \,. \end{aligned} \tag{1}$$

We assume unit density ($\rho = 1$) and use $\mathbf{u}$ to denote the fluid velocity, p the pressure, and $\partial\mathcal{D}$ the boundary of the fluid region $\mathcal{D}$. The pressure term in Eq. (1) can be dropped easily by rewriting the Euler equations in terms of *vorticity*. Recall that traditional vector calculus defines vorticity as the curl of the velocity field, $\boldsymbol{\omega} = \nabla \times \mathbf{u}$. Taking the curl ($\nabla\times$) of Eq.(1), we obtain

$$\begin{aligned} \frac{\partial \boldsymbol{\omega}}{\partial t} + \mathcal{L}_{\mathbf{u}}\boldsymbol{\omega} &= \mathbf{0} \,, \\ \boldsymbol{\omega} = \nabla \times \mathbf{u} \,, \quad div(\mathbf{u}) &= 0 \,, \quad \mathbf{u} \parallel \partial\mathcal{D} \,. \end{aligned} \tag{2}$$

where $\mathcal{L}_{\mathbf{u}}\boldsymbol{\omega}$ is the Lie derivative, equal in our case to $\mathbf{u} \cdot \nabla\boldsymbol{\omega} - \boldsymbol{\omega} \cdot \nabla\mathbf{u}$. In this form, this vorticity-based equation states that *vorticity*

is simply advected along the fluid flow. [1] Note that Equation (2) is equivalent to the more familiar $\frac{D\boldsymbol{\omega}}{Dt} = \boldsymbol{\omega} \cdot \nabla \mathbf{u}$, and therefore already includes the vortex stretching term appearing in *Lagrangian* approaches. Roughly speaking, vorticity measures the local spin of a fluid parcel. Therefore, vorticity advection means that this local spin moves along with the flow.

Since the integral of vorticity on a given bounded surface equals (by Stokes' theorem) the *circulation* around the bounding loop of the surface, one can explain the geometric nature of an ideal fluid flow in particularly simple terms: **the circulation around any closed loop $\mathcal{C}$ is conserved throughout the motion of this loop in the fluid**. This key result is known as Kelvin's circulation theorem, and is usually written as:

$$\Gamma(t) \;=\; \oint_{\mathcal{C}(t)} \mathbf{u} \cdot \mathbf{dl} \;=\; \text{constant}\,, \tag{3}$$

where $\Gamma(t)$ is the circulation of the velocity on the loop $\mathcal{C}$ at time t as it gets advected in the fluid. This will be the key to our time integration algorithm.

Viscous Fluids In contrast to ideal fluids, incompressible *viscous* fluids generate very different fluid behaviors. They can be modelled by the *Navier-Stokes* equations (compare with Eq. (1)):

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \;&=\; -\nabla p + \nu \Delta \mathbf{u}\,, \\ div(\mathbf{u}) \;&=\; 0\,, \quad \mathbf{u}|_{\partial \mathcal{D}} = \mathbf{0}\,. \end{aligned} \tag{4}$$

where Δ denotes the Laplace operator, and ν is the *kinematic viscosity*. Note that different types of boundary conditions can be added depending on the chosen model. Despite the apparent similarity between these two models for fluid flows, the added diffusion term dampens the motion, resulting in a slow decay of circulation. This diffusion also implies that the velocity of a viscous fluid at the boundary of a domain must be null, whereas an inviscid fluid could have a non-zero tangential component on the boundary. Here again, one can avoid the pressure term by taking the curl of the equations (compare with Eq. (2)):

$$\begin{aligned} \frac{\partial \boldsymbol{\omega}}{\partial t} + \mathcal{L}_{\mathbf{u}} \boldsymbol{\omega} \;&=\; \nu \Delta \boldsymbol{\omega}\,, \\ \boldsymbol{\omega} \;=\; \nabla \times \mathbf{u}\,, \quad div(\mathbf{u}) \;&=\; 0\,, \quad \mathbf{u}|_{\partial \mathcal{D}} = \mathbf{0}\,. \end{aligned} \tag{5}$$

2.2 Discrete Setup and Time Integration

For a discrete time and space numerical simulation of Eqs. (2) and (5) we need a discretized geometry of the domain (given as a simplicial mesh for instance), appropriate discrete analogs of velocity $\mathbf{u}$ and vorticity fields $\boldsymbol{\omega}$, along with the operators which act on them. We will present our choices before describing the geometry-based time integration approach.

2.2.1 Space Discretization

We discretize the spatial domain in which the flow takes place using a locally oriented simplicial complex, *i.e.*, either a tet mesh for 3D domains or a triangle mesh for 2D domains, and refer to this discrete domain as $\mathcal{M}$ (see Figure 2). The domain may have non-trivial topology, *e.g.*, it can contain tunnels and voids (3D) or holes (2D), but is assumed to be compact. To ensure good numerical properties in the subsequent simulation we require the simplices of $\mathcal{M}$ to be well shaped (aspect ratios bounded away from zero). This assumption is quite common since many numerical error estimates depend heavily on the element quality. We use meshes generated with the method of [Alliez et al. 2005]. Collectively we refer to the sets of vertices, edges, triangles, and tets as V, E, F, and T.

We will also need a *dual mesh*. It associates with each original simplex (vertex, edge, triangle, tet, respectively) its dual (dual cell, dual face, dual edge, and dual vertex, respectively) (see Fig. 3). The geometric realization of the dual mesh uses tet circumcenters as dual vertices and Voronoi cells as dual cells; dual edges are line segments connecting dual vertices across shared tet faces and dual faces are the faces of the Voronoi cells. Notice that storing values on primal simplices or on their associated dual cells is conceptually equivalent, since both sets have the same cardinality. We will see in Section 3 that corresponding primal and dual quantities are related through a simple (diagonal) linear operator.

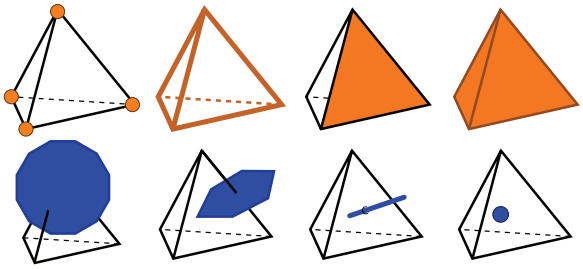


Figure 3: *Primal and Dual Cells: the simplices of our mesh are vertices, edges, triangles and tets (up); their circumcentric duals are dual cells, dual faces, dual edges and dual vertices (bottom).*

2.2.2 Intrinsic Discretization of Physical Quantities

In order to faithfully capture the geometric structure of fluid mechanics on the discrete mesh, we define the usual physical quantities, such as velocity and vorticity, through **integral values over the elements of the mesh $\mathcal{M}$**. This is the sharpest departure from traditional numerical techniques in CFD: we not only use values at nodes and tets (as in FEM and FVM), but also allow association (and storage) of field values at *any* appropriate simplex. Depending on whether a given quantity is defined pointwise, or as a line, area or volume density, the corresponding discrete representation will "live" at the associated 0, 1, 2, and 3 dimensional mesh elements. With this in mind we use a simple discretization of the physical quantities of fluid mechanics based on fluxes associated to faces.

Velocity as Discrete Flux To encode a coordinate free (intrinsic) representation of velocity on the mesh we use *flux*, *i.e.*, the mass of fluid transported across a given surface area per unit time. Note that this makes flux an *integrated*, not pointwise, quantity. On the discrete mesh, fluxes are associated with the triangles of the tet mesh. Thus fluid velocity $\mathbf{u}$ is treated as a 2-form and represented as a vector U of values on faces (size $|F|$). This coordinate-free point of view, also used in [Feldman et al. 2005], is reminiscent of the staggered grid method used in [Fedkiw et al. 2001] and other non-collocated grid techniques (see [Goktekin et al. 2004]). In the staggered grid approach one does not store the x, y, z components of a vector at nodes but rather associates them with the corresponding grid faces. We may therefore think of the idea of storing fluxes on the triangles of our tet mesh as a way of *extending* the idea of staggered grids to the more general simplical mesh setting. This was previously exploited in [Bossavit and Kettunen 1999] in the context of E&M computations. It also makes the usual no-transfer boundary conditions easy to encode: boundary faces experience no flux

[1] Note that this is a more canonical characterization of the motion than the velocity-based one: the latter was also an advection but under the additional *constraint* of keeping the velocity field divergence-free, which is the reason for the gradient of pressure.

across them. Encoding this boundary condition for velocity vectors stored at vertices is far more cumbersome.

Divergence as Net Flux on Tets Given the incompressibility of the fluid, the velocity field must be divergence-free ($\nabla \cdot \mathbf{u} = 0$). In the discrete setting, the integral of the divergence over a tet becomes particularly simple. According to the generalized Stokes' theorem this integral equals the sum of the fluxes on all four faces: the discrete notion of divergence is therefore simply the net flux for each tet. Divergence can therefore be stored as a 3-form, *i.e.*, as a value associated to each tet (a vector of cardinality $|T|$). The notion of incompressibility becomes particularly intuitive: whatever gets in each tet must be compensated by whatever comes out (see Fig. 4).

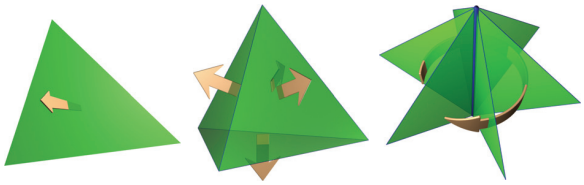


Figure 4: *Discrete Physical Quantities: in our geometric discretization, fluid flux lives on faces (left), divergence lives on tets (middle), and vorticity lives on edges (right).*

Vorticity as Flux Spin Finally we need to define vorticity on the mesh. To see the physical intuition behind our definition, consider an edge in the mesh. It has a number of faces incident on it, akin to a paddle wheel (see Figure 4). The flux on each face contributes a torque to the edge. The sum of all these, when going around an edge, is the net torque that would "spin" the edge. We can thus give a physical definition of vorticity as a weighted sum of fluxes on all faces incident to a given edge. This quantity is now associated with primal edges—or, equivalently, dual faces—and is thus represented by a vector Ω of size $|E|$.

In Section 3, we will see that these physical intuitions can be derived formally from simple algebraic relationships.

2.3 Geometric Integration of Fluid Motion

Since we are using the vorticity formulation of the fluid equations (Eqs. (2) or (5)) the time integration algorithm must update the discrete vorticity variables which are stored on each primal edge. We have seen that the fluid equations state that vorticity is advected by the velocity field. The fundamental idea of our geometric integration algorithm is thus to ensure that Kelvin's theorem holds in the discrete setting: the circulation around any loop in the fluid remains constant as the loop is advected. This can be achieved by backtracking loops: for any given loop at the current time, determine its backtracked image in the velocity field ("where did it come from?") and compute the circulation around the backtracked loop. This value is then assigned as the circulation around the original loop at the present time, *i.e.*, circulation is properly advected *by construction* (see Figure 5 for a depiction of this loop advection idea).

Since we store vorticity on primal edges, a natural choice for these loops are the bounding loops of the dual faces associated to each primal edge (see Figure 3). Notice that these loops are polylines formed by sequences of dual vertices around a given primal edge. Consequently an efficient implementation of this idea requires only that we backtrack *dual vertices* in the velocity field. Once these positions are known *all* backtracked dual loops associated to *all* primal edges are known. These Voronoi loops can indeed generate any discrete, dual loop: the sum of adjacent loops is a larger, outer loop as the interior edges cancel out due to opposite orientation as sketched in Fig. 5(right). The evaluation of circulation around these backtracked loops will be quite straightforward. Invoking Stokes' theorem, the integral of vorticity over a dual face equals the circulation around its boundary. With this observation we have achieved our goal of updating vorticities and, *by design*, ensured a discrete version of Kelvin's theorem. The algorithmic details of this simple geometric approach to time integration of the equations of motion for fluids will be given in Section 4.

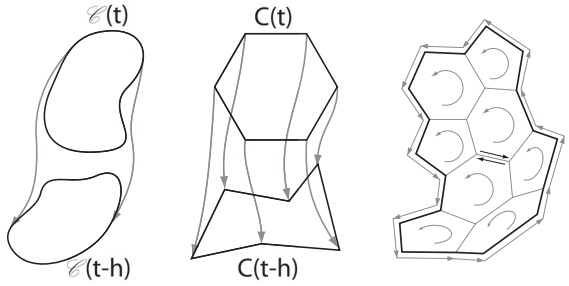


Figure 5: *Kelvin's Theorem: (left) in the continuous setting, the circulation on any loop being advected by the flow is constant. (middle) our discrete integration scheme enforces this property on each Voronoi loop, (right) thus on any discrete loop.*

3 Computational Machinery

Now that we have described our choices of spatial and physical discretizations, along with a way to use them to integrate the fluid's motion, we must manipulate the numerics involved in a principled manner to guarantee proper physical behavior. In this section, we point out that the intuitive definition of our physical quantities living at the different simplices of a mesh can be made precise through the definition of a *discrete differential structure*. Consequently, the basic operators to go from fluxes to the divergence, curl, or Laplacian of the velocity field can be *formally* defined through the use of discrete differential forms. We will mostly focus on presenting the practical implementation of the few operators we need; more importantly, we will show that this implementation reduces to *simple linear algebra* with very sparse matrices.

3.1 Discrete Differential Structure

Integrals and Forms In Section 2.2, we have opted for manipulating the physical quantities in the form of a line, surface, and volume integral computed directly on our meshed domain to render the setup entirely intrinsic, *i.e.*, with no need for vector fields to be stored with respect to arbitrary coordinate frames. Such an integral represents the evaluation of a mathematical entity called a *differential form* . In the continuous three-dimensional setting, a 0-form is simply a function on that 3D space. A 1-form, or line-form, is a quantity that can be *evaluated through integration over a curve*. Thus a 1-form can be thought of as a proxy for a vector field, and its integral over a curve becomes the circulation of this vector field. A 2-form, or area-form, is to be *integrated over a surface*, that is, it can be viewed as a proxy for a vector perpendicular to that surface (and its evaluation becomes the flux of that vector field through the surface); finally, a 3-form, or volume-form, is to be *integrated over a volume* and can be viewed as a proxy for a function. Classic calculus and vector calculus can then be substituted with a special calculus involving only differential forms, called *exterior calculus*—the basis of Hodge theory and modern differential geometry (for a comprehensive discussion, see, for example, [Abraham et al. 1988]).

Discrete Forms and Their Representation However, in our framework, the continuous domain is replaced (or approximated) by a mesh, the only structure we can work with. Therefore, the integrated physical values we store on the mesh corresponds to *discrete differential k-forms* [Desbrun et al. 2005]. A discrete differential k-form, $k = 0, 1, 2$, or 3, is the evaluation (*i.e.*, the integral) of the differential k-form on all k-cells, or k-simplices. In practice, discrete k-forms can simply be considered as *vectors of numbers*

according to the simplices they live on: 0-forms live on vertices, and are expressed as a vector of length $|V|$; correspondingly, 1-forms live on edges (length $|E|$), 2-forms live on faces (length $|F|$), and 3-forms live on tets (length $|T|$). Dual forms, *i.e.*, forms that we will evaluate on the dual mesh, are treated similarly. The reader should now realize that in our discretization of physical quantities, the notion of flux that we described is thus a primal 2-form (integrated over faces), while its vorticity is a dual 2-form (integrated over dual faces), and its divergence becomes a primal 3-form (integrated over tets).

Discrete Differential Calculus on Simplicial Meshes These discrete forms can now be used to build the tools of calculus through Discrete Exterior Calculus (DEC), a coherent calculus mimicking the continuous setting that only uses discrete combinatorial and geometric operations [Munkres 1984; Hirani 2003; Desbrun et al. 2005]. At the core of its construction is the definition of a discrete d operator (analog of the continuous exterior derivative), and a discrete Hodge star, which will allow us to move values from the primal mesh to the dual mesh and vice-versa. For a more comprehensive introduction to DEC and the use of discrete differential forms, we refer the interested readers to [Desbrun et al. 2005].

3.2 Two Basic Operators

The computations involved in our approach only require the definition of two basic operators: one is the exterior derivative d, necessary to compute derivatives, like gradients, divergences, or curls; the other is the Hodge star, to transfer values from primal simplices to dual simplices.

Exterior Derivative d Given an oriented mesh, we implement our first operator by simply assembling the *incidence matrices* of the mesh. These will act on the vectors of our discrete forms and implement the discrete exterior derivative operator d as explained in more details in Appendix A. For our 3D implementation, there are three sparse matrices involved, which contain only entries of type 0, $+1$, and -1. Care is required in assembling these incidence matrices, as the orientation must be taken into account in a consistent manner [Elcott and Schröder 2005]. The first one is d_0, the incidence matrix of vertices and edges ($|E|$ rows and $|V|$ columns). Each row contains a single $+1$ and -1 for the end points of the given edge (and zero otherwise). The sign is determined from the orientation of the edge. The second matrix similarly encodes the incidence relations of edges and faces ($|F|$ rows and $|E|$ columns), with appropriate $+1$ and -1 entries according to the orientation of edges as one moves around a face. More generally d_k is the incidence matrix of k-cells on $k+1$-cells.

A simple debugging sanity check (necessary but not sufficient) is to compute consecutive products: d_0 followed by d_1 must be a matrix of zeros; d_1 followed by d_2 must similarly give a zero matrix. This reflects the fact that the boundary of any boundary is the empty set. It also corresponds to the calculus fact that curl of grad is zero as is divergence of curl (see [Desbrun et al. 2005]).

Hodge Star The second operator we need will allow us to transfer quantities back and forth between the primal and dual mesh. We can project a primal k-form to a conceptually-equivalent dual $(3-k)$-form with the *Hodge star*. We will denote $\star_0$ (resp., $\star_1, \star_2, \star_3$) the Hodge star taking a 0-form (resp., 1-form, 2-form, and 3-form) to a dual 3-form (resp., dual 2-form, dual 1-form, dual 0-form). In this paper we will use what is known as the *diagonal Hodge star* [Bossavit 1998]. This operator simply scales whatever quantity that is stored on mesh cells by the volumes of the corresponding dual and primal cells: let $\mathrm{vol}(.)$ denote the volume of a cell (*i.e.*, 1 for vertices, length for edges, area for triangles, and volume for tets), then

$$(\star_k)_{ii} = \mathrm{vol}(\tilde{\sigma}_i)/\,\mathrm{vol}(\sigma_i)$$

where σ_i is any primal k-simplex, and $\tilde{\sigma}_i$ is its dual. These linear operators, describing the local metric, are diagonal and can be stored as vectors. Conveniently, the inverse matrices going from dual to primal quantities are trivial to compute for this diagonal Hodge star.

Overloading Operators Note that both the d_k and the $\star_k$ operators are *typed*: the subscript k is *implicitly determined* by the dimension of the argument. For example, the velocity field $\mathbf{u}$ is a 2-form stored as a vector U of cardinality $|F|$. Consequently the expression dU implies use of the $|T| \times |F|$-sized matrix d_2. In the implementation this is accomplished with operator overloading (in the sense of `C++`). We will take advantage of this in the paper from now on and drop the dimension subscripts.

3.3 Offline Matrix Setup

With these overloads of d and $\star$ in place, we can now set up the only two matrices (C and L) that will be used during simulation. They respectively represent the discrete analogs of the curl and Laplace operators [Desbrun et al. 2005].

Curl Since we store fluxes on faces and gather them in a vector U, the *circulation* of the vector field $\mathbf{u}$ can be derived as values on dual edges through $\star U$. *Vorticity*, typically a 2-form in fluid mechanics [Marsden and Wenstein 1983], is easily computed by then summing this circulation along the dual edges that form the boundary of a dual face. In other words, $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ becomes, in terms of our discrete operators, simply $\Omega = d^T \star U$. We therefore create a matrix C offline as $d^T\star$, *i.e.*, the composition of an incidence matrix with a diagonal matrix.

Laplacian The last matrix we need to define is the discrete Laplacian. The discrete analog of $\Delta\phi = (\nabla\nabla\cdot - \nabla\times\nabla\times)\phi = \boldsymbol{\omega}$ is simply $(\star d \star^{-1} d^T \star + d^T \star d)\ \Phi = \Omega$ as explained in Appendix B. This last matrix, a simple composition of incidence and diagonal matrices, is precomputed and stored as L for later use.

4 Implementation

To facilitate a direct implementation of our integration scheme, we provide pseudocode (Figure 6) along with implementation notes which provide details for specific steps and how these relate to the machinery developed in earlier sections.

```
//Load mesh and build incidence matrices
C ← d^T ⋆
L ← ⋆d⋆^{-1} d^T ⋆ + d^T ⋆ d

//Time stepping h
loop
    //Advect Vorticities
    for each dual vertex (tet circumcenter) c_i
        ĉ_i ← PathTraceBackwards(c_i);
        v_i ← InterpolateVelocityField(ĉ_i);
    for each dual face f
        Ω_f ← 0
        for each dual edge (i, j) on the boundary of f
            Ω_f ← Ω_f + ½(v_i + v_j) · (ĉ_i − ĉ_j);

    //Add forces
    Ω ← Ω + h C F

    //Add diffusion for Navier-Stokes
    Ψ ← SolveCG( (⋆ − ν h L)Ψ = Ω );
    Ω ← ⋆ Ψ

    //Convert vorticities back to fluxes
    Φ ← SolveCG( L Φ = Ω );
    U ← dΦ;
```

Figure 6: *Pseudocode of our fluid motion integrator.*

4.1 External Body Forces

The use of external body forces, like buoyancy, gravity, or stirring, is common practice to create interesting motions. Incorporating external forces into Eq. (4) is straightforward, resulting in:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \;=\; -\nabla p + \nu \Delta \mathbf{u} + \mathbf{f} \; .$$

Again, taking the curl of this equation allows us to recast this equation in terms of vorticity:

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \mathcal{L}_{\mathbf{u}} \boldsymbol{\omega} \;=\; \nu \Delta \boldsymbol{\omega} + \nabla \times \mathbf{f} \; . \tag{6}$$

Thus, we note that an external force influences the vorticity only through the force's curl (the $\nabla \cdot \mathbf{f}$ term is compensated for by the pressure term keeping the fluid divergence-free). Thus, if we express our forces through the vector F of their resulting fluxes in each face, we can directly add the forces to the domain by incrementing Ω by the circulation of F over the time step h, *i.e.*:

$$\Omega \leftarrow \Omega + h \; C \; F.$$

4.2 Adding Diffusion

If we desire to simulate a viscous fluid, we must add the diffusion term present in Eq. (5). Note that previous methods were sometimes omitting this term because their numerical dissipation was already creating (uncontrolled) diffusion. In our case, however, this diffusion needs to be properly handled if viscosity is desired. This is easily done through an unconditionally-stable implicit integration as done in Stable Fluids (*i.e.*, we also use a fractional step approach). Using the discrete Laplacian in Eq. (8) and the current vorticity Ω, we simply solve for the diffused vorticity Ω' using the following linear system:

$$(\star - \nu h L) \;\star^{-1} \Omega' = \Omega.$$

4.3 Interpolation of Velocity

In order to perform the backtracking of dual vertices we must first define a velocity field over the entire domain using the data we have on primal faces (fluxes). This can be done by computing a unique velocity vector for each dual vertex and then using barycentric interpolation of these vectors over each dual Voronoi cell [Warren et al. 2004], defining a continuous velocity field over the entire domain. This velocity field can be used to backtrack dual vertices as well as transport particles or dyes (*e.g.*, for visualization purposes) with standard methods.

To see that such a vector, one for each dual vertex, is well defined consider the following argument. The flux on a face corresponds under duality (via the Hodge star) to a circulation along the dual edge of this face. Now, there is a linear relation between fluxes per tet due to the incompressibility condition (fluxes must sum to zero). This translates directly to a linear condition on the four circulations at each tet too. Thus, there is a unique vector (with three components) at the dual vertex whose projection along the dual edges is consistent with the observed circulations.

Relation to k-form Basis Functions The standard method to interpolate k-form data in a piecewise linear fashion over simplicial complexes is based on Whitney forms [Bossavit 1998]. In the case of primal 2-forms (fluxes) this results in a piecewise constant (per tet) velocity field. Our argument above, using a Voronoi cell based generalized barycentric interpolation of dual 1-forms (circulation), in fact *extends* the Whitney form machinery to the dual setting. This is a novel contribution which may be useful in other computational applications of discrete forms. We note that the generalized barycentric coordinates have linear accuracy [Warren et al. 2004], an important requirement in many settings.

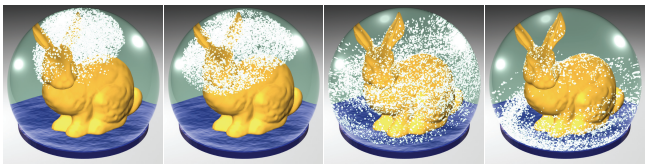


Figure 7: *Bunny Snow Globe: the snow in the globe is advected by the inner fluid, initially stirred by a vortex to simulate a spin of the globe.*

4.4 Handling Boundaries

The algorithm as described above does not constrain the boundaries, thus achieving "open" boundary conditions. No-transfer boundary conditions are easily imposed by setting the fluxes through the boundary triangles to zero. Non-zero flux boundary conditions (*i.e.*, forced fluxes through the boundary as in the case of Fig. 8) are more subtle. First, remark that all these boundary fluxes *must* sum to zero; otherwise, we would have little chance of getting a divergence-free fluid in the domain. Since the total divergence is zero, there exists a harmonic velocity field satisfying exactly these conditions. This is a consequence of the Helmholtz-Hodge decomposition theorem with normal boundary conditions [Chorin and Marsden 1979]. Thus, this harmonic part $\mathbf{h}$ can be computed *once and for all* through a Poisson equation using the same setup as described in Appendix B. This precomputed velocity field allows us to deal very elegantly with these boundary conditions: we simply perform the same algorithm as we described by setting all boundary conditions to zero (with the exception of backtracking which takes the precomputed velocity into account), and *reinject* the harmonic part at the end of each time step (*i.e.*, add $\mathbf{h}$ to the current velocity field).

Viscous Fluids near Boundaries The Voronoi cells at the boundaries are slightly different from the usual, interior ones, since boundary vertices do not have a full 1-ring of tets. In the case of NS equations, this has no significant consequence: we set the velocity on the boundary to zero, resulting also in a zero circulation on the dual edges on the boundary. The rest of the algorithm can be used as is.

Inviscid Fluids near Boundaries For Euler equations, however, the tangential velocity at the boundary is not explicitly stored anywhere. Consequently, the boundary Voronoi faces need an additional variable to remedy this lack of information. We store in these dual faces the current integral vorticity. From this additional information given at time $t = 0$, we can *deduce* at each later time step the missing circulation on the boundary: since the circulation over the inside dual edges is known, and since the total integral must sum to the vorticity (Stokes' theorem), a simple substraction is all that is needed to update this missing circulation.

4.5 Handling Arbitrary Topology

Although the problem of arbitrary domain topology (*e.g.*, when the first Betti number is not zero) is rarely discussed in CFD or in our field, it is important nonetheless. In the absence of external forces, the circulation along each loop (of winding number 1) around a tunnel is constant in time. So once again, we precalculate a constant harmonic field based on the initial circulation around each tunnel, and simply *add it* to the current velocity field for advection purposes. This procedure serves two purposes: first, notice that we now automatically enforce the discrete equivalent of Kelvin's theorem on *any* (shrinkable or non-shrinkable) loop; second, arbitrary topologies are accurately handled very efficiently.

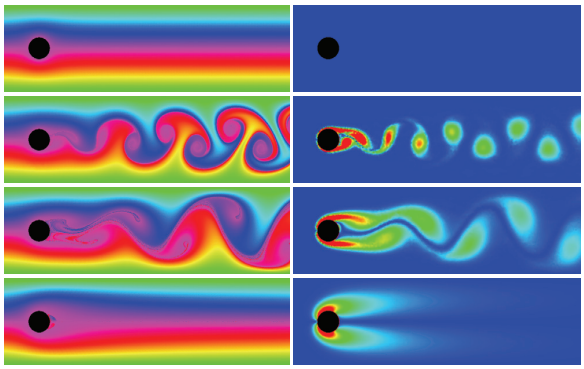
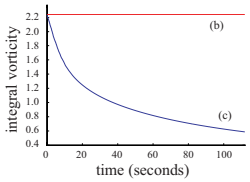


Figure 8: *Obstacle Course: in the usual experiment of a flow passing around a disk, the viscosity as well as the velocity can significantly affect the flow appearance; (left) our simulation results for increasing viscosity and same left boundary flux; (right) the vorticity magnitude (shown in false colors) of the same frame. Notice how the usual irrotational flow is obtained (top) for zero viscosity, while the von Karman vortex street appears as viscosity is introduced.*

5 Results and Discussion

We have tested our method on some of the usual "obstacle courses" in CFD. We start with the widely studied example of a flow past a disk (see Fig. 9). Starting with zero vorticity, it is well known that in the case of an inviscid fluid, the flow remains irrotational for all time. By construction, our method does respect this physical behavior since circulation is preserved for Euler equations. We then increase the viscosity of the fluid incrementally, and observe the formation of a vortex wake behind the obstacle, in agreement with physical experiments. As evidenced by the vorticity plots, vortices are shed from the boundary layer as a result of the adherence of the fluid to the obstacle, thanks to our proper treatment of the boundary conditions.

The behavior of vortex interactions observed in existing experimental results was compared to numerical results based on our novel model and those obtained from the semi-Lagrangian advection method. It is known from theory that two like-signed vortices with a finite vorticity core will merge when their distance of separation is smaller than some critical value. This behavior is captured by the experimental data and shown in the first series of snapshots of Fig. 9. As the next row of snapshots indicates, the numerical results that our model generated present striking similarities to the experimental data. In the last row, we see that a traditional semi-Lagrangian advection followed by re-projection misses most of the fine structures of this phenomenon. This can be attributed to the loss of total integral vorticity as evidenced in this inset; in comparison our technique preserves this integral exactly.



We have also considered the flow on curved surfaces in 3D with complex topology, as depicted in Fig. 10. We were able to easily extend our implementation of two-dimensional flows to this curved case thanks to the intrinsic nature of our approach.

We consider a smoke cloud surrounded by air, filling the body of a bunny as an example of flow in a domain with complex boundary. The buoyancy drives the air flow which, in turn, advects the smoke cloud in the three-dimensional domain bounded by the bunny mesh as shown in Fig. 1. In the last simulation, we show a snow globe with a bunny inside (see Fig. 7). We emulate the flow due to an initial spin of the globe using a swirl described as a vorticity field.

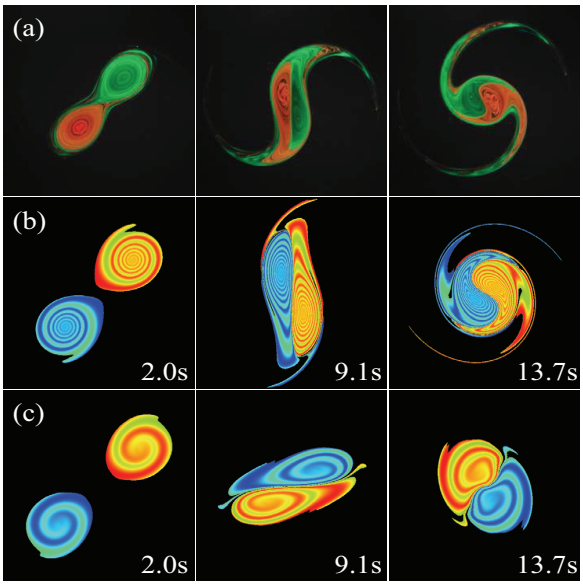


Figure 9: *Two Merging Vortices: discrete fluid simulations are compared with a real life experiment (courtesy of Dr. Trieling, Eindhoven University; see* `http://www.fluid.tue.nl/WDY/vort/index.html`*) where two vortices (colored in red and green) merge slowly due to their interaction (a); while our method faithfully captures the merging phenomenon (b), a traditional semi-lagrangian scheme does not capture the correct motion because of vorticity damping (c).*

The snow particles are transported by the flow as they fall down under the effect of gravity. Both examples took *less than half a second per frame* to compute, exemplifying the advantage of using tet meshes to resolve fine boundaries.

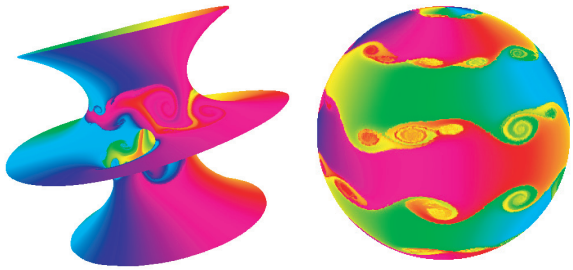


Figure 10: *Weather System on Planet Funky: the intrinsic nature of the variables used in our algorithm makes it amenable to the simulation of flows on arbitrary curved surfaces.*

6 Conclusion

In this paper, we have introduced a novel theoretical approach to fluid dynamics, along with its practical implementation and various simulation results. We have carefully discretized the physics of flows to respect the most fundamental geometric structures that characterize their behavior. Among the several specific benefits that we demonstrated, the most important is the circulation preservation property of the integration scheme, as evidenced by our numerical examples. The discrete quantities we used are intrinsic, allowing us to go to curved manifolds with no additional complication. Finally, the machinery employed in our approach can be used on any simplicial complex. However, the same methodology also applies directly to more general spatial partitionings, and in particular, to regular grids and hybrid meshes [Feldman et al. 2005]—rendering our approach widely applicable to existing fluid simulators.

For future work, a rigorous comparison of the current method with standard approaches should be undertaken. Using Bjerknes' circulation theorem for compressible flows may also be an interesting

avenue. Additionally, we limited ourselves to the investigation of our scheme without focusing on the separate issue of order of accuracy. Coming up with an integration scheme that is higher-order accurate will be the object of further investigation, as it requires a better (denser) Hodge star. Finally, we note that our geometric approach bears interesting similarities with the work of Chang *et al.* [2002], in which they propose a purely algebraic approach to remedy the shortcomings of the traditional fractional step approach. Using their numerical analysis on our approach may provide a simple way to study the accuracy of our scheme.

References

ABRAHAM, R., MARSDEN, J., AND RATIU, T., Eds. 1988. *Manifolds, Tensor Analysis, and Applications*, vol. 75 of *Applied Mathematical Sciences*. Springer.

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Transactions on Graphics 24*, 3, 617–625.

BOSSAVIT, A., AND KETTUNEN, L. 1999. Yee-like schemes on a tetrahedral mesh. *Int. J. Num. Modelling: Electr. Networks, Dev. and Fields 12* (July), 129–142.

BOSSAVIT, A. 1998. *Computational Electromagnetism*. Academic Press, Boston.

CHANG, W., GIRALDO, F., AND PEROT, B. 2002. Analysis of an exact fractional step method. *Journal of Computational Physics 180*, 3 (Nov.), 183–199.

CHORIN, A., AND MARSDEN, J. 1979. *A Mathematical Introduction to Fluid Mechanics*, 3rd edition ed. Springer-Verlag.

DESBRUN, M., KANSO, E., AND TONG, Y. 2005. Discrete differential forms for computational sciences. In *Discrete Differential Geometry*, E. Grinspun, P. Schröder, and M. Desbrun, Eds., Course Notes. ACM SIGGRAPH, ch. 7.

ELCOTT, S., AND SCHRÖDER, P. 2005. Building your own dec at home. In *Discrete Differential Geometry*, E. Grinspun, P. Schröder, and M. Desbrun, Eds., Course Notes. ACM SIGGRAPH, ch. 8.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, 15–22.

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. *ACM Transactions on Graphics 24*, 3, 904–909.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, 23–30.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Proceedings of SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, 181–188.

GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. 2004. A method for animating viscoelastic fluids. *ACM Transactions on Graphics 23*, 3 (Aug.), 463–468.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shell. *ACM Transactions on Graphics 24*, 3, 973–981.

HIRANI, A. 2003. *Discrete Exterior Calculus*. PhD thesis, California Institute of Technology.

LANGTANGEN, H.-P., MARDAL, K.-A., AND WINTER, R. 2002. Numerical methods for incompressible viscous flow. *Advances in Water Resources 25*, 8-12 (Aug-Dec), 1125–1146.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics 23*, 3 (Aug.), 457–462.

MARSDEN, J. E., AND WENSTEIN, A. 1983. Coadjoint orbits, vortices and Clebsch variables for incompressible fluids. *Physica D 7*, 305–323.

MARSDEN, J. E., AND WEST, M. 2001. Discrete mechanics and variational integrators. *Acta Numerica 10*, 357–515.

MCNAMARA, A., TREUILLE, A., POPOVIC, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Transactions on Graphics 23*, 3 (Aug.), 449–456.

MUNKRES, J. R. 1984. *Elements of Algebraic Topology*. Addison-Wesley.

PIGHIN, F., COHEN, J. M., AND SHAH, M. 2004. Modeling and Editing Flows using Advected Radial Basis Functions. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 223–232.

SHI, L., AND YU, Y. 2004. Inviscid and Incompressible Fluid Simulation on Triangle Meshes. *Journal of Computer Animation and Virtual Worlds 15*, 3-4 (June), 173–181.

SHI, L., AND YU, Y. 2005. Controllable smoke animation with guiding objects. *ACM Transactions on Graphics 24*, 1, 140–164.

STAM, J. 1999. Stable fluids. In *Proceedings of ACM SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, 121–128.

STAM, J. 2001. A simple fluid solver based on the fft. *Journal of Graphics Tools 6*, 2, 43–52.

STAM, J. 2003. Flows on surfaces of arbitrary topology. *ACM Transactions on Graphics 22*, 3 (July), 724–731.

STEINHOFF, J., AND UNDERHILL, D. 1994. Modification of the euler equations for *Vorticity Confinement*: Applications to the computation of interacting vortex rings. *Physics of Fluids 6*, 8 (Aug.), 2738–2744.

TONG, Y., LOMBEYDA, S., HIRANI, A. N., AND DESBRUN, M. 2003. Discrete multiscale vector field decomposition. *ACM Transactions on Graphics 22*, 3, 445–452.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Transactions on Graphics 22*, 3 (July), 716–723.

WARREN, J., SCHAEFER, S., HIRANI, A., AND DESBRUN, M., 2004. Barycentric coordinates for convex sets. Preprint.

YAEGER, L., UPSON, C., AND MYERS, R. 1986. Combining physical and visual simulation - creation of the planet jupiter for the film 2010. *Computer Graphics (Proceedings of ACM SIGGRAPH) 20*, 4, 85–93.

A Discrete Exterior Derivative

A thorough explanation of the discrete exterior derivative of discrete forms is out of the scope of this paper, and we refer the reader to existing tutorials [Desbrun et al. 2005]. However, the reader should be aware that this operator is simply implemented via the

use of *incidence matrices*. Indeed, a key ingredient to defining the discrete version of the exterior derivative d is Stokes' theorem:

$$\int_\sigma d\alpha = \int_{\partial\sigma} \alpha,$$

where σ denotes a $(k+1)$-cell and α is a k-form. Stokes' theorem states that the integral of $d\alpha$ (a $(k+1)$-form) over a $(k+1)$-cell equals the integral of the k-form α over the *boundary* of the $(k+1)$-cell (*i.e.*, a k-cell). Stokes' theorem can thus be used as a way to *define* the d operator in terms of the boundary operator ∂. Or, said differently, once we have the boundary operator, the operator d follows immediately if we wish Stokes' theorem to hold on the simplicial complex.

To use a very simple example, consider a 0-form f, *i.e.*, a function giving values at vertices. With that, df is a 1-form which can be integrated along an edge (say with end points denoted a and b) and Stokes' theorem states the well known fact:

$$\int_{[a,b]} df = f(b) - f(a).$$

The right hand side is simply the evaluation of the 0-form f on the boundary of the edge, *i.e.*, its endpoints (with appropriate signs indicating the orientation of the edge). Actually, one can define a hierarchy of these operators that mimic the operators given in the continuous setting (up to an application of the Hodge star) by the gradient (∇), curl ($\nabla\times$), and divergence ($\nabla\cdot$), namely,

- d_0: maps 0-forms to 1-forms and corresponds to the **Gradient**;
- d_1: maps 1-forms (values on edges) to 2-forms (values on faces). The value on a given face is simply the sum (by linearity of the integral) of the 1-form values on the boundary (edges) of the face with the signs chosen according to the local orientation. d_1 corresponds to the **Curl**;
- d_2: maps 2-forms to 3-forms and corresponds to the **Divergence**.

Since the boundary of any mesh element can be directly read from the incidence matrices of the mesh, the exterior derivative is trivial to implement once the mesh is known as it depends only on its connectivity [Elcott and Schröder 2005].

B Recovery of Velocity

We have seen in Section 4 how the vorticity ω can be derived directly from the set of all face fluxes as $d^T \star U = \omega$. However, during the simulation, we will also need to recover flux *from* vorticity. For this we employ the Helmholtz-Hodge decomposition theorem, stating that any vector field $\mathbf{u}$ can be decomposed into three components (given appropriate boundary conditions)

$$\mathbf{u} = \nabla \times \phi + \nabla\boldsymbol{\psi} + \mathbf{h}.$$

When represented in terms of discrete forms this reads as:

$$U = d\Phi + \star^{-1} d^T \star \Psi + H \tag{7}$$

For the case of incompressible fluids (*i.e.*, with zero divergence), two of the three components are sufficient to describe the velocity field: the curl of a vector potential and a harmonic field. To see this apply d to both sides of Eq. 7:

$$dU = 0 = dd\Phi + d \star^{-1} d^T \star \Psi + dH.$$

Since $dd = 0$ and d of a harmonic form always vanishes, we find that $d \star^{-1} d^T \star \Psi = 0$ to begin with. Since Ψ is a 3-form $d \star^{-1} d^T \star \Psi = \Delta\Psi = 0$, *i.e.*, Ψ is harmonic which implies in particular that $\star^{-1} d^T \star \Psi = 0$, proving our claim that

$$U = d\Phi + H.$$

If the topology of the domain is trivial, we can furthermore ignore the harmonic part H (we discuss a full treatment of arbitrary topology in Section 4.5), leaving us with $U = d\Phi$.

Since our algorithm computes an updated Ω which is related to U as $d^T \star U$, we need to find a solution to

$$\Omega = d^T \star d\Phi,$$

where Ω is the known quantity, and $d\Phi$ the unknown. Unfortunately the kernel of $d^T \star d$ is not empty so we can not determine Φ directly from this equation. To pick a unique solution for Φ, we require additionally that $d^T \star \Phi = 0$. By doing so we pick a *particular* solution from the kernel of d^T. But if $d^T \star \Phi = 0$ then certainly $(\star d \star^{-1} d^T \star)\Phi = 0$ and we can add it to our equation for Ω arriving at

$$\Omega = (d^T \star d + \star d \star^{-1} d^T \star)\Phi. \tag{8}$$

This latter equation is simply a Poisson equation for Φ since

$$\star^{-1}\Omega = \Delta\Phi$$

which has a unique solution. Let $U = d\Phi$, and we have recovered U as desired.

The fact that Eq. 8 is indeed a Poisson problem follows from the definition of the Laplacian Δ in *differential calculus* as $d \star^{-1} d^T \star + \star^{-1} d^T \star d$. In the language of vector calculus this translates to $\Delta\phi = (\nabla\nabla\cdot - \nabla\times\nabla\times)\phi = \nabla\times\mathbf{u}$. Notice that the left-side matrix (that we will denote L) is *symmetric and sparse*, thus ideally suited for fast numerical solvers. Our linear operators (and, in particular, the discrete Laplacian) differ from another discrete Poisson setup on simplicial complexes proposed in [Tong et al. 2003]: the ones we use have smaller support, which results in sparser and better conditioned linear systems [Bossavit 1998]—an attractive feature in the context of numerical simulation.