Real-Time Fluid Simulation on the Surface of a Sphere

BOWEN YANG*, WILLIAM CORSE*, JIECONG LU, JOSHUAH WOLPER, and CHENFANFU JIANG, University of Pennsylvania

We present a novel approach for animating incompressible fluids with Eulerian advection-projection solvers on the surface of a sphere by extending the recent work by Hill and Henderson [2016] with a staggered spherical grid discretization. By doing so, we avoid the infamous checkerboard null modes. We additionally introduce new, straightforward polar singularity treatments that avoid the previous need for any spectral filtering of high-frequency noise at the poles. Lastly, we enforce incompressibility with a fast Fourier solution to Poisson's equation for pressure in spherical coordinates. Our high-performance GPU-based framework combines scalability, art-directability, and ease of implementation, and reaches real-time speeds for various practical scenarios.

CCS Concepts: • **Computing methodologies → Physical simulation**.

Additional Key Words and Phrases: Fluid Simulation, Coordinate Singularity, Parallel Computing, Art-Directable Simulation

ACM Reference Format:
Bowen Yang, William Corse, Jiecong Lu, Joshuah Wolper, and Chenfanfu Jiang. 2019. Real-Time Fluid Simulation on the Surface of a Sphere. *Proc. ACM Comput. Graph. Interact. Tech.* 2, 1, Article 4 (May 2019), 17 pages. https://doi.org/10.1145/3320285

1 INTRODUCTION

Look closely at a soap bubble and enter a world often missed where swirling iridescent vortexes collide with glimmering bands of neon weaving in every direction. The physics behind this natural beauty is anything but simple. The difficulty associated with spherical flow is demonstrated by the dearth of spherical fluid animations in graphics, despite the myriad potential applications. This, given the ubiquity of Cartesian fluid simulation techniques, implies that their spherical cousins lack the requisite speed, stability, and simplicity that the graphics community demands. Regardless, we posit that this is a problem worth solving due to the vast visual potential—spanning everything from shimmering soap bubbles to whirling ocean currents.

While modeling incompressible flow over a sphere has been of interest to the scientific computing and meteorological communities for some time, it is still a relatively nascent area of research in graphics due largely to the demand for speed and efficiency. Despite this, there are already many strong formulations for incompressible flow over general complex topologies such as those of Stam [2003], Hegeman et al. [2009], and Carvalho et al. [2012]. Recently, however, Hill and Henderson [2016] noted that these methods omit important geometric terms in their formulation. They also rigorously proved these terms to be physically important, demonstrating that they are a

*Both authors contributed equally to this research.

Authors' address: Bowen Yang, bwyang@seas.upenn.edu; William Corse, wcorse@seas.upenn.edu; Jiecong Lu, jiecong@seas.upenn.edu; Joshuah Wolper; Chenfanfu Jiang, University of Pennsylvania.

2577-6193/2019/5-ART4 $15.00
https://doi.org/10.1145/3320285

Fig. 1. **Lollipop.** Fluid with a curl noise-initialized velocity field on a grid resolution of 2048×4096 solved at an average of 72 $ms/step$ on the GPU.

consequence of the coordinate changes associated with mapping from Cartesian space to other bases. Additionally, Hill and Henderson [2016] boast visually impressive results while attempting to resolve the long standing issue of mathematical singularities at the spherical poles.

Given the success of Hill and Henderson's approach [2016], we sought out to extend it through the use of a staggered discretization on a spherical grid. We also present a new method for avoiding the singularity problem at the poles and quickly solving the governing Poisson equation for pressure using Fourier solutions [Lai and Wang 2002]. Furthermore, we outline a GPU optimization with a simplified pole treatment, allowing our method to achieve real-time performance. Finally, our method bares some essential parameters to an artist for a variety of rich, art-directable features. We summarize our contributions below:

- A staggered spherical grid discretization of the spherical Navier-Stokes equation that circumvents the classic checkerboard instability and thus eliminates the need for null space smoothing.
- A novel and straightforward treatment of the pole singularities that enables ease of implementation with no visual disturbances at the poles.
- An efficient and memory-friendly GPU-optimized spherical fluid simulation framework that achieves real-time performance on many practical high-resolution examples.

Ultimately, in this work, we present a framework that is not only easy to implement and art-directable, but also reaches real time speeds and gracefully improves upon the work of Hill and Henderson [2016] without requiring spectral filtering at the poles or momentum diffusion for null modes. Note that our method also enables an easy treatment of solid boundary obstacles on the sphere, as demonstrated in Fig. 9.

2 BACKGROUND

2.1 Related Work

Given the commonplace nature of the geographic coordinate system (latitude and longitude), it is natural to use a similar approach when discretizing a sphere for fluid simulation. However, there exists a well documented problem that occurs at the poles in a spherical coordinate system. More specifically, at the poles, the spherical coordinate discretization of the governing incompressible fluid equations becomes divergent, causing instabilities [Randall 2011]. Due to these singularities, many works have explored ways to formulate alternative discretizations and meshes for spherical flow. One popular method focuses on using cubed-sphere grids to discretize a sphere into 6 distinct regions, each mapped from a unique face of a circumscribed cube [Nair et al. 2005; Putman and Lin 2007; Ronchi et al. 1996; White and Dongarra 2011]. This formulation is singularity-free but

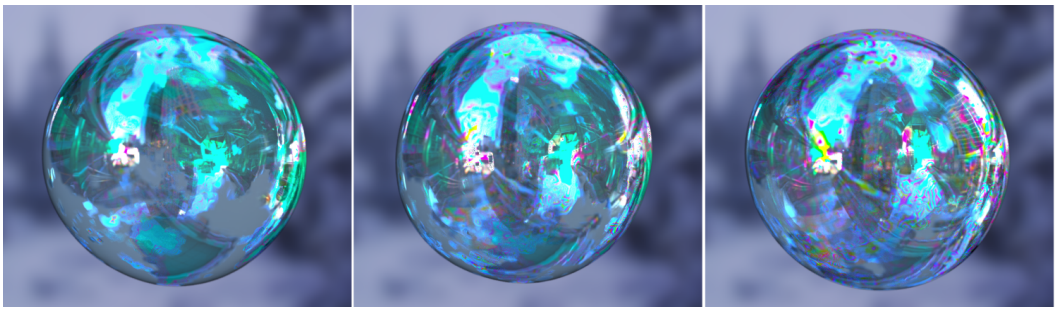

Fig. 2. **Soap bubble.** Our simulation captures intricate fluid flows on a soap bubble with grid resolution 2048×4096 at an average of 72 $ms/step$ on the GPU. Enhanced with iridescent color mapping and procedural sphere deformation.

unfortunately requires six different coordinate systems and, as such, introduces a nontrivial amount of bookkeeping and complexity. Another common alternative meshing approach is to use a geodesic grid discretization, as this is both singularity-free and has cells of equal area [Carfora 2007; Randall et al. 2002]. However, geodesic meshing suffers both from the aforementioned overhead associated with cubed-sphere methods and also from the the complexity posed by the Poisson equation that governs pressure projection in this formulation.

Although much work has focused on addressing the pole problem associated with spherical flow, others have explored using triangle mesh surfaces to both avoid the pole singularity entirely and to construct more general topologies. Shi and Yu [2004] introduce a triangle mesh local flattening technique for interpolating velocity and pair this with a novel, albeit engineered, way to treat the geometric terms noted by Hill and Henderson [2016]. Similarly, Elcott et al. [2007] focus on flow over general triangle meshes, but they develop their own vorticity formulation of the classical Navier-Stokes equations. This approach obviates the need to enforce the incompressibility of the velocity field and preserves circulation around mesh loops but introduces additional Helmholtz equations that must be solved to calculate velocity from vorticity. Azencot et al. [2014] improve upon Elcott et al.'s method [2007] by preserving vorticity instead of circulation, in turn removing the need for the computation of velocity flow lines.

Though many have had success with these alternative approaches to surface flow, the pole problem remains when we simply want to simulate flow over a sphere using a spherical grid. The main difficulty with poles is that the CFL condition changes with latitude and in turn requires increasingly smaller time steps as the poles are approached in either direction [Randall 2011]. A common way to deal with this is to use a uniform time step and concurrently remove high-frequency noise that develops in the vicinity of the poles by using a spectral filter as in Hill and Henderson [2016]. However, this filtering is not mass-conserving [Randall 2011], and, consequently, artifacts can develop at the poles.

To solve Poisson's equation for pressure, we use a method presented by Lai and Wang [2002] that accommodates a staggered spherical grid discretization. Their approach distributes grid nodes about but not on the poles of the sphere, which necessitates a special treatment of pressure computation that leverages a Fourier coefficient symmetry constraint. Mei et al. [2016] expand upon this method by leveraging Legendre polynomials in the longitudinal axis and by choosing Legendre-Gauss-Radau points at the poles to avoid the pole problem. This method proved to have higher convergence accuracy, but we instead choose to adopt Lai and Wang's method [2002] due to its ease of implementation and straightforward avoidance of the pole problem.

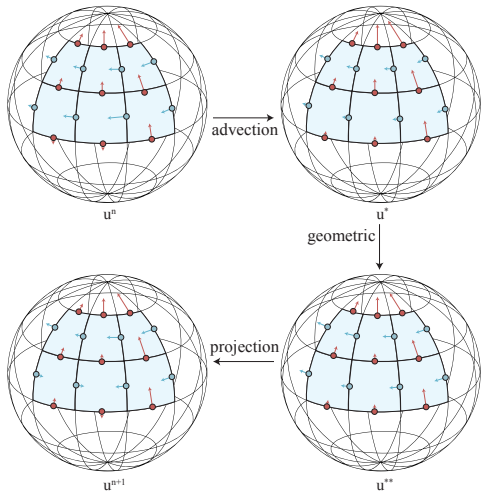

Fig. 3. **Split-wise integration.** Integrating the Navier-Stokes equations on a sphere over a single time step Δt.

2.2 Equations on the Sphere

We start with the standard Euler equations for a uniform incompressible fluid:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{f}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

where $\mathbf{u}$ is the fluid velocity, ρ is density, p is pressure, and $\mathbf{f}$ describes body forces. We express $\mathbf{u}$ in spherical components as $\mathbf{u}_s = (u_r, u_\theta, u_\phi)$ where u_r is the radial velocity, u_θ is the velocity along polar angles, and u_ϕ is the velocity along azimuthal angles. Because we restrict our analysis to a two-dimensional flow on the surface of a sphere, we can assume radial velocity is negligible. Written explicitly in spherical coordinates, the remaining equations are presented in Batchelor [2000]:

$$\frac{Du_\theta}{Dt} - \frac{u_\phi^2 \cot\theta}{R} = -\frac{1}{\rho R}\frac{\partial p}{\partial \theta} + f_\theta, \tag{3}$$

$$\frac{Du_\phi}{Dt} + \frac{u_\theta u_\phi \cot\theta}{R} = -\frac{1}{\rho R \sin\theta}\frac{\partial p}{\partial \phi} + f_\phi, \tag{4}$$

where D/Dt denotes the material derivative and R is sphere radius. Note the inclusion of the geometric terms proportional to $u_\phi \cot\theta/R$ highlighted in Hill and Henderson's paper [2016], which account for the rotation of the local coordinate system $\mathbf{u}_s$ as a function of position on the sphere's surface. For convenience, we list the material derivative in spherical coordinates as well as the incompressibility condition:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla = \frac{\partial}{\partial t} + \frac{u_\theta}{R}\frac{\partial}{\partial \theta} + \frac{u_\phi}{R \sin\theta}\frac{\partial}{\partial \phi}, \tag{5}$$

$$\nabla \cdot \mathbf{u} = \frac{1}{R}\frac{\partial u_\theta}{\partial \theta} + \frac{1}{R \sin\theta}\frac{\partial u_\phi}{\partial \phi} + \frac{\cot\theta}{R}u_\theta = 0. \tag{6}$$

3 OUR METHOD

We approach solving the Navier-Stokes equations on the sphere using the same general split-wise integration scheme outlined in previous works [Bridson 2015; Hill and Henderson 2016; Stam 1999], with the inclusion of the implicit integration of the geometric terms in Eq. (3) and (4) noted above. We outline the entire procedure in Fig. 3. Our main difference from the method in Hill and Henderson [2016] is the adoption of a staggered spherical grid with novel, robust, and easy-to-implement pole treatments. We discuss our grid definition and its implications in detail in this section.

3.1 Spherical Discretization

The staggered grid enables the calculation of derivatives using an unbiased, second-order accurate central difference method. It also removes the need to handle "checkerboard" null modes. Although commonly used in Eulerian fluid simulations on a Cartesian grid (see e.g., [Fedkiw et al. 2001]), staggered discretization has not been applied to spherical fluid simulations in computer graphics. Doing so demands care in the vicinity of the sphere poles ($\theta = 0, \pi$) due to the singularities seen in Eq. (3) and (4). We show that the staggered grid definition determines a unique approach to semi-Lagrangian advection (§3.3) on the sphere, as well as a specific Fourier series solution to Poisson's equation solved on a discretized sphere with grid points distributed about, but not on, the poles (§3.5).

We define the staggered grid in (θ, ϕ)-space by an equally-spaced non-collocated grid consisting of $N_\theta \times N_\phi$ cells, where $N_\phi = 2N_\theta$, $\Delta\theta = \pi/N_\theta$, and $\Delta\phi = 2\pi/N_\phi$. Pressure values are stored at

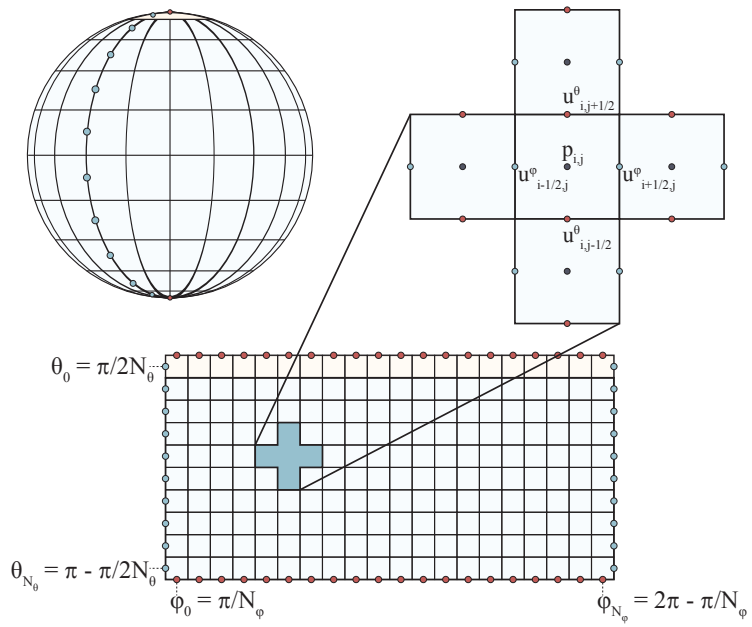

Fig. 4. **The staggered spherical grid.** (Bottom) The staggered grid in (θ, ϕ)-space. Cells connected to the North pole are indicated in tan. (Top Right) u_θ and u_ϕ velocities are stored along cell edges (red and blue dots, respectively) and pressure values $p_{i,j}$ are stored at cell centers (grey dots). (Top Left) Grid locations storing values of u_ϕ at $\phi = 0, 2\pi$ are cylindrically mapped in $\mathbb{R}^3$. Grid locations storing values of $u_\theta(0, \phi)$ and $u_\theta(\pi, \phi)$ are mapped to the North and South poles, respectively.

cell centers and u_θ and u_ϕ are stored on cell faces (Fig. 4). u_ϕ must satisfy the cylindrical boundary condition:

$$u_\phi(\theta, 0) = u_\phi(\theta, 2\pi)$$

and so the array storing u_ϕ values on the grid is of dimension $N_\theta \times N_\phi$. Note that we use a dense array in this work, though incorporating a sparse grid data structure such as OpenVDB in the same space would be straightforward [Museth et al. 2013]. The $\theta = 0$ and $\theta = \pi$ lines on the staggered grid consist of N_ϕ u_θ grid points each, but on the sphere, these map to the North and South poles, respectively (Fig. 4). The velocities defined at the poles in $\mathbb{R}^3$ must be uniquely defined, but the u_θ values at the poles in (θ, ϕ)-space vary as a function of ϕ. We discuss this more in §3.2.

We align the upper left corner of our staggered grid in (θ, ϕ)-space with the origin, with θ increasing downwards and ϕ increasing to the right (Fig. 4). This choice departs from the typical placement of a cell center at the origin of a staggered grid in Cartesian coordinates, but it is deliberate – it moves face centers off the poles, enabling an effective discretization scheme (§3.5) for solving Poisson's equation for pressure, and leaves polar singularity considerations to operations involving u_θ values that are stored along the polar edges in (θ, ϕ)-space at $\theta = 0$ and $\theta = \pi$.

3.2 Boundary Conditions at the Poles

We solve the material derivatives in Eq. (3) and (4) using a spherical staggered-grid semi-Lagrangian advection scheme to evaluate the intermediate velocity components u_θ^* and u_ϕ^*. Apart from boundary grid points, physical quantities in (θ, ϕ)-space are advected over a time Δt with speed $1/R\left(u_\theta, u_\phi/\sin\theta\right)$. A quantity q (velocity, temperature, smoke concentration) is simply carried across the seam at $\phi = 0$ according to the cylindrical boundary condition (§3.1), but the poles present a more difficult challenge. At the poles, the advection speed is singular, so traditional semi-Lagrangian advection on a grid is infeasible.

This problem is solved by introducing boundary conditions at the poles. Here, the otherwise independent orthogonal components of the velocity vector are related to one another and are a function of ϕ. Hill and Henderson [2016] choose the following boundary condition at the North Pole

$$u_\theta(0, \phi) = u_x \cos\phi + u_y \sin\phi, \tag{7}$$

$$u_\phi(0, \phi) = u_\theta(0, \phi + \pi/2) \tag{8}$$

and describe a similar boundary condition at the South pole in which only the sign of u_θ is changed.

By enforcing this boundary condition in Eq. (3) at the poles, the singularity in the advection term exactly cancels the singularity in the geometric term, such that when updating u_θ at the poles, they use the equation

$$\frac{\partial u_\theta}{\partial t} + \frac{u_\theta}{R}\frac{\partial u_\theta}{\partial \theta} = -\frac{1}{\rho R}\frac{\partial P}{\partial \theta} \tag{9}$$

Hill and Henderson include an additional spectral filter to enforce the consistency boundary condition for the velocity at the poles [2016]. After computing Eq. (9), they calculate

$$u_x = \frac{2}{N_\theta}\sum_{k=0}^{N_\theta - 1} u_\theta(0, k\Delta\theta)\cos(k\Delta\theta), \tag{10}$$

$$u_y = \frac{2}{N_\theta}\sum_{k=0}^{N_\theta - 1} u_\theta(0, k\Delta\theta)\sin(k\Delta\theta), \tag{11}$$

which are consistent with the first mode in a Fourier series relating velocity components in $\mathbb{R}^3$ and u_θ components at the poles. These values are then used in the boundary conditions Eq. (7) and (8) to evaluate the final velocities at each pole per time step.

3.3 Inexpensive Pole-Aware Advection

While Hill and Henderson's method is analytically precise, the numerical evaluation of Eq. (9) using standard differencing schemes on a grid generates erroneous high-frequency noise [2016]. The inclusion of the spectral filter corrects some of this noise, but comes with additional computational costs. Consequently, we use a simple averaging scheme instead that works as follows:

For the sake of brevity, we present only the analysis at the North pole (see Fig. 5). First, all grid points but the poles are advected as described above. We then consider the u_ϕ grid points that encircle the pole. The speeds stored here, drawn as tangent vectors in $\mathbb{R}^3$, lie in a plane that is parallel to the tangent plane at the North pole. We argue that the average of these vectors is the tangent vector orthogonal to the principal direction of flow over the North pole. We then use the boundary condition in Eq. (8) to rotate this vector by $-\pi/2$ to estimate the tangent velocity vector (u_x, u_y) that points in the principal direction of flow (in the θ direction). The projection of this vector onto the u_θ grid coordinates described by $\hat{r} = (\cos\phi, \sin\phi)$ then reproduces the boundary condition in Eq. (7).

Note that as the grid resolution on the sphere is increased, the u_ϕ grid points adjacent to either pole converge to the pole, such that in the infinitesimal vicinity of the pole, the averaged velocity there should be equal to the velocity at the pole itself. In this sense, the accuracy of our polar treatment increases with grid resolution.

In our framework, we use a second-order Runge-Kutta (RK2) path tracer as well as bilinear interpolation for sampling quantities quickly on the grid. This efficient solving scheme offsets some of the additional computational cost of handling the intricacies associated with the sphere.

3.4 Geometric Term

The cotangent factor in the geometric terms makes it inadvisable to apply explicit integration to the velocities because instabilities would develop near the poles. Hill and Henderson [2016] apply

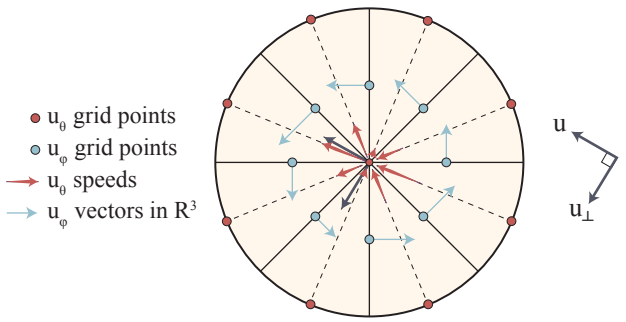

Fig. 5. **Robust advection near the North pole.** To solve for u_θ values at the poles, we average the surrounding u_ϕ vectors expressed in $\mathbb{R}^3$, rotate the resulting vector by $-\pi/2$ to get the principal direction of flow $\mathbf{u}$, then project this vector along the ϕ directions where values of u_θ are stored.

a backward Euler integration for Eq. (3) and (4) as follows:

$$\frac{u_\theta^{**} - u_\theta^*}{\Delta t} = \frac{\left(u_\phi^{**}\right)^2 \cot\theta}{R}, \tag{12}$$

$$\frac{u_\phi^{**} - u_\phi^*}{\Delta t} = -\frac{u_\theta^{**} u_\phi^{**} \cot\theta}{R}. \tag{13}$$

The solution in their case is analytical – the real root to a cubic equation. A simple backward substitution yields the orthogonal u components.

In our case, however, u_θ and u_ϕ are not collocated. Correspondingly, we resample u_θ and u_ϕ at cell centers before executing implicit integration. The updated u_θ and u_ϕ values at cell centers are then interpolated at cell edges as staggered solutions.

3.5 Pressure Projection

Pressure projection in spherical coordinates amounts to updating grid velocities using the following equations:

$$u_\theta^{n+1} = u_\theta^{**} - \frac{\Delta t}{\rho R}\left(p_{i,j+1/2} - p_{i,j-1/2}\right), \tag{14}$$

$$u_\phi^{n+1} = u_\phi^{**} - \frac{\Delta t}{\rho R \sin\theta}\left(p_{i+1/2,j} - p_{i-1/2,j}\right), \tag{15}$$

where the pressure is a solution of the Poisson equation

$$\nabla^2 p = \frac{\rho}{\Delta t}\nabla \cdot \mathbf{u}^{**} \tag{16}$$

with proper boundary conditions.

Hill and Henderson opted to solve Eq. (16) using Intel's MKL [2016]. To accommodate a GPU implementation, we instead select a Fourier solution uniquely suited to our staggered grid definition. Following Lai and Wang [2002], we discretize the equation on a sphere that is subdivided along longitudinal lines $\phi_i = i\Delta\phi$ by

$$\theta_j = (j - 1/2)\,\Delta\theta, \tag{17}$$

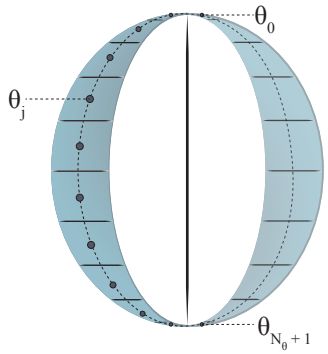

Fig. 6. Pressures Stored Along a Longitudinal Strip. The staggered spherical grid discretization distributes cell centers about but not on the poles. $p_n\,(\theta_0)$ and $p_n\,(\theta_{N_\theta+1})$ are evaluated using symmetry conditions 25 and 26.

where $i = 1, \ldots, N_\theta$ and $j = 1, \ldots, N_\phi$. As shown in figure 6, this grid definition avoids selecting poles as grid points and exactly matches cell center locations in our staggered grid where pressure values are stored.

Eq. (16) in spherical coordinates is given by

$$\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial p}{\partial\theta}\right) + \frac{1}{\sin^2\theta}\frac{\partial^2 p}{\partial\phi^2} = \frac{\rho}{\Delta t}\nabla\cdot\mathbf{u}^{**} = f(\theta,\phi). \tag{18}$$

Note that care should be taken when numerically evaluating the divergence $\nabla\cdot\mathbf{u}^{**}$ of a grid cell. The edge lengths of grid cells in the longitudinal direction are nonuniformly scaled by $\sin\theta$:

$$\begin{aligned}\nabla\cdot\mathbf{u} &= \frac{1}{R}\frac{\partial u_\theta}{\partial\theta} + \frac{1}{R\sin\theta}\frac{\partial u_\phi}{\partial\phi} + \frac{\cot\theta}{R}u_\theta \\ &= \frac{1}{R\sin\theta}\left(\frac{\partial u_\phi}{\partial\phi} + \frac{\partial}{\partial\theta}(u_\theta\sin\theta)\right), \end{aligned}\tag{19}$$

$$\begin{aligned}(\nabla\cdot\mathbf{u})_{i,j} = \frac{1}{R\sin\theta_{i,j}}\Bigg(&\frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta\phi} + \\ &\frac{u_{i,j+1/2}\sin\theta_{j+1/2} - u_{i,j-1/2}\sin\theta_{j-1/2}}{\Delta\theta}\Bigg). \end{aligned}\tag{20}$$

We approximate the solution $p(\theta,\phi)$ using a truncated Fourier series:

$$p(\theta,\phi) = \sum_{n=-N_\phi/2}^{N_\phi/2-1} p_n(\theta)\, e^{in\phi}, \tag{21}$$

with Fourier coefficients given by

$$p_n(\theta) = \frac{1}{N_\phi}\sum_{i=0}^{N_\phi-1} p(\theta,\phi_i)\, e^{-in\phi_i}, \tag{22}$$

where $\phi_i = i\Delta\phi$. $f(\theta,\phi)$ can be similarly expanded. Associating left and right-hand side Fourier coefficients in Eq. (18), we derive the following expression for $p_n(\theta)$:

$$\frac{d^2 p_n}{d\theta^2} + \cot\theta\frac{dp_n}{d\theta} - \frac{n^2}{\sin^2\theta}p_n = f_n(\theta) \tag{23}$$

Re-written using the central difference difference method and making the discrete substitutions $U_j = p_n(\theta_j)$ and $F_j = f_n(\theta_j)$, we derive a tridiagonal linear system for U_j:

$$\frac{U_{j+1} - 2U_j + U_{j-1}}{(\Delta\theta)^2} + (\cot\theta_j)\frac{U_{j+1} - U_{j-1}}{2\Delta\theta} - \frac{n^2}{\sin^2\theta_j}U_j = F_j \tag{24}$$

which can be evaluated using a variety of different tridiagonal matrix solving techniques. Notice, however, that we do not have values for U_0 and $U_{N_\theta+1}$. We can calculate these by using the symmetry condition $f(-\theta,\phi) = f(\theta,\phi+\pi)$ on the sphere to prove that the nth Fourier coefficient of a function defined on the sphere satisfies

$$a_n(-\theta) = (-1)^n a_n(\theta), \tag{25}$$

$$a_n(\pi+\theta) = (-1)^n a_n(\pi-\theta). \tag{26}$$

These symmetry conditions can be used to show that $U_0 = (-1)^n U_1$ and $U_{N_\theta+1} = (-1)^n N_\theta$ [Lai and Wang 2002].

Values for $p\left(\theta_j, \phi_i\right)$ determined using this technique are then used in Eq. (14) and (15) to update u_θ and u_ϕ on cell edges.

4 GPU EXTENSION AND PERFORMANCE

We harness the power of the modern GPU platform to make our solver real-time at exceedingly high resolution, allowing artists to do their work on-the-fly. Although our solver may superficially appear to be trivial to parallelize, two potential bottlenecks would emerge if we directly ported our implementation to the GPU without any optimization. To get rid of these bottlenecks and obtain better performance, we make the following adjustments to our solver.

4.1 Adjustment to Advection Sampling

Solving velocities at polar points is quite trivial for CPUs, but can be extremely expensive to GPUs, since it requires averaging speed projections along the belt at the poles. However, the velocity at a pole is used only when we need to sample velocities near the pole during advection. Since the velocity does not contribute to the divergence here, it does not affect projection calculations.

Consequently, we can use a different sampling scheme at the poles without having to average speed projections. During the advection phase, when we trace a hypothetical particle back to some location near the pole, we simply sample the velocity there by a distance-weighted interpolation between the two wedges on opposite sides of the pole (Fig. 7). This simplified sampling is less accurate. However, in practice, we have not observed any visual artifacts when applying this adjustment.

4.2 Memory Arrangement

It is impossible to predict where a hypothetical particle in semi-Lagrangian advection will back-trace to beforehand. Consequently, if we use trivial linear or multi-dimensional memory to store our attributes, cache misses would frequently happen because of poor locality. The situation is even worse on GPUs, as CUDA uses a bank mechanism for memory arrangement.

To avoid bank conflicts and achieve better locality, we allocate the attributes in pitched linear memory with appropriate pitch size. Elements on the same latitudinal belt along the ϕ direction are stored in a row in the pitched memory, with its end padded with zeros to fill the remainder of the pitch.

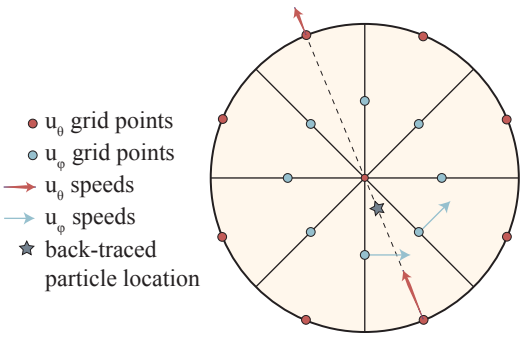

Fig. 7. Simplified advection scheme on the GPU. The velocity of the back-traced particle is calculated by a distance-weighted interpolation across opposing wedges.

4.3 Solving Fourier Coefficients in Projection

We showed in §3.5 that Fourier coefficients for pressure can be evaluated by solving tridiagonal linear equations. There are several algorithms that only require $O(n)$ time to complete and require far less iterations compared to ordinary sparse matrix problems. However, most of these algorithms are sequential by nature and are not easily ported to the GPU.

Consequently, to boost our GPU performance, we applied the algorithm proposed in a paper by Zhang et al. [2010] based on cyclic-reduction to solve Eq. 24. The kernel for the algorithm was organized in N blocks by M threads, with N being the total set of problems, and M being the matrix dimension divided by 2. In our case, $N = N_\theta$, and $M = \frac{N_\phi}{2}$.

5 ART DIRECTABILITY AND WORKFLOW

The framework we have described above is sufficiently general to accommodate a variety of initial velocity distributions, body forces, and rendering techniques. We detail those we have experimented with here, as well as several features built into an art-directable tool that runs the split-wise integration scheme. Together, these features demonstrate the breadth of applications for our method as well as its flexibility.

5.1 Velocity Field Initialization

We experimented with two initial velocity distributions. The first uses curl-noise [Bridson et al. 2007] written in spherical coordinates, which is divergence-free by design:

$$u^0 = (u_\theta, u_\phi) = \left(\frac{1}{R \sin\theta} \frac{\partial \psi}{\partial \phi}, -\frac{1}{R} \frac{\partial \psi}{\partial \theta} \right). \tag{27}$$

We use Perlin noise to store ψ values at grid cell faces. Spatial derivatives of ψ are evaluated across cell faces and stored at face centers using the central difference method:

$$\frac{\partial \psi}{\partial \theta} = \frac{\psi_{i,j+1/2} - \psi_{i,j-1/2}}{\Delta\theta}, \tag{28}$$

$$\frac{\partial \psi}{\partial \phi} = \frac{\psi_{i+1/2,j} - \psi_{i-1/2,j}}{\Delta\phi}. \tag{29}$$

Velocity values at face centers are then set by averaging across cell centers. We also use two dimensional Fourier sums over θ and ϕ to initialize u_θ and u_ϕ:

$$u_\theta^0 = \sum_m \sum_n c_{mn} \sin(m\theta) \sin(n\phi), \tag{30}$$

$$u_\phi^0 = \sum_p \sum_q d_{pq} \sin(p\theta) \sin(q\phi), \tag{31}$$

where $0 \le \theta \le \pi$ and $0 \le \phi < 2\pi$. These are highly tunable and construct a velocity distribution with a net angular velocity of zero, which we find contributes to the plausibility of the simulation.

5.2 Body Forces

We evaluate body forces defined by the user using forward Euler integration after solving for the geometric terms. Forces in this case are defined tangent to the sphere. For example, the gravitational force along the surface of the sphere is $\mathbf{f}_g = -m\mathbf{g} \sin\theta$. The user may also define a time-dependent force $f(\theta, \phi, t)$ to significantly alter fluid behavior over specified intervals for artistic control.

5.3 Artist Workflow

Our method is well suited for rendering a variety of different fluids on a sphere, but perhaps the most visually-compelling of these is soap. We discuss rendering soap bubbles below. Note that we do not model surface tension but achieve realistic-looking bubble deformation through procedural animation (see Fig. 2).

- **Procedural Specular Shading**. In order to texture the soap bubble's surface with iridescent colors, we created a custom procedural shader inspired by Andrew Glassner's work [1999], which is based on the characteristics of soap film. We map normalized density values on the sphere to an iridescent color spectrum that mimics the spectrum of light refracted through a thin soap film with varying thickness over its surface and in time.
- **Procedural Deformation**. We also deform the sphere in time using procedural simplex noise to mimic the aerodynamic forces that shape bubbles. We tune the lacunarity, roughness, and contributing frequencies of this noise to craft detailed and realistic bubble deformations.
- **Color Mapping**. Our tool includes multiple features that leverage the OpenCV library [Bradski 2000]. Namely, users can load a greyscale image that maps to a scalar density field defined on the sphere, an RGB image that maps to fluid particle color, and a binary black and white image that defines grid cells to be fluid or solid (fluid is not initialized in solid cells and cannot subsequently flow into solid cells).
- **Solid Boundaries**. We account for solid cells in our core algorithm when calculating the velocity divergence of a particular cell in Eq. (18) – fluid cells adjacent to solid cells see zero flux through the adjoining face. Velocity values at solid boundaries are enforced to be zero in the pressure projection step. As in Hill and Henderson's implementation [2016], this simple treatment is surprisingly visually effective despite not enforcing an accurate boundary condition.

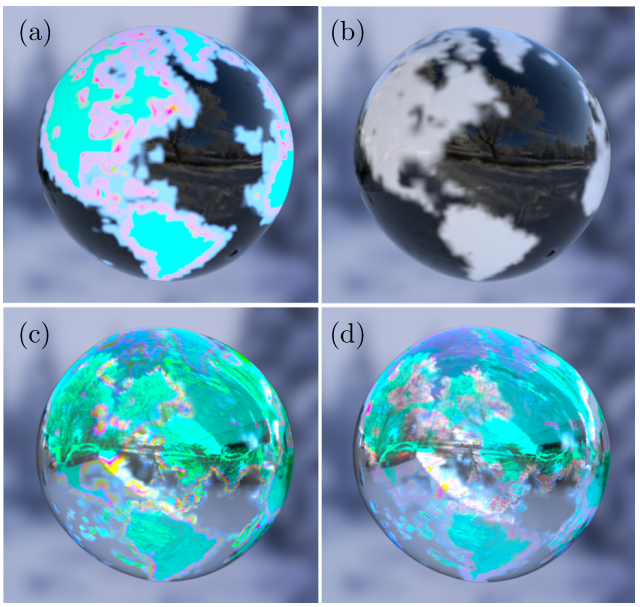

Fig. 8. **An artist's typical workflow.** (a) rgb color map; (b) greyscale density map; (c) synthesized texture at $t = 0$; (d) time-evolved render.

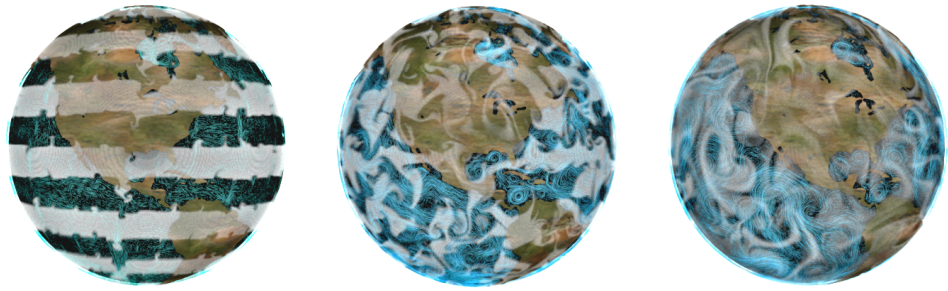

Fig. 9. **Planet.** We simulate ocean currents and atmospheric circulation on an imaginary planet with resolution 2048×4096.

After specifying the initial velocity distribution on the sphere, as well as any body forces and boundary conditions, our solver generates the fluid evolution sequence of a texture/density field. We demonstrate a typical workflow in Fig. 8.

6 RESULTS

We implemented our framework as a SideFX Houdini plugin, which we provide as a supplementary document (together with our CPU and GPU source code). These will become open-sourced after the publication of this work. All examples and benchmark data presented in this paper were performed on an Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz CPU with NVidia Quadro P6000 GPU.

6.1 Performance

With CUDA acceleration on the GPU, our performance reached 222.9 steps per second at a grid resolution of 512×1024, a resolution sufficiently high for movie-grade rendering. This is equivalent to 47 frames per second assuming a time step of 0.005 seconds and a frame duration of 1/24 second. Our solver runs approximately 150 times faster on the GPU than on the CPU. Fig. 10 shows the scalability of our solver on the GPU.

Fig. 11 hints at the different challenges inherent in optimizing CPU and GPU implementations. On the CPU side, the bottleneck lies in projection, which is expected since it has a complexity of

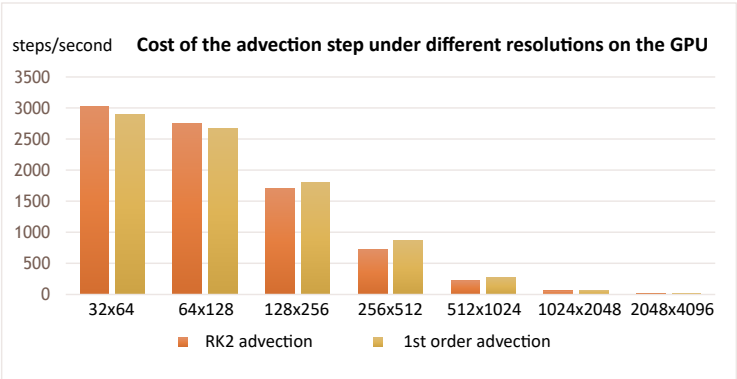

Fig. 10. Timing comparison for the GPU advection step in a typical simulation under different resolutions.

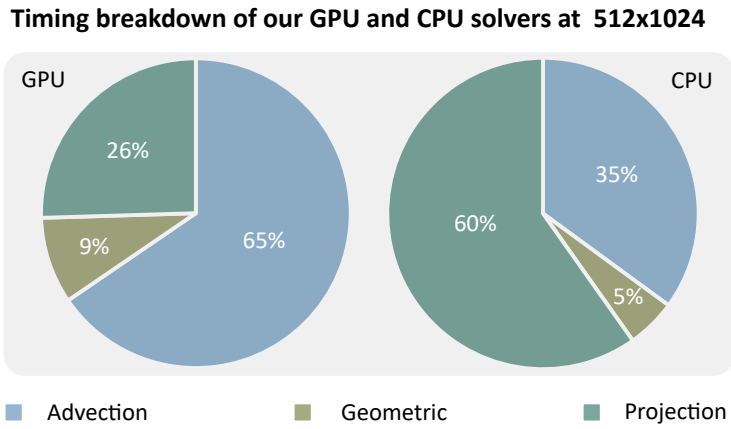

Fig. 11. Timing breakdown of GPU and CPU solvers. Note the difference in the dominating bottleneck.

$O(N_\theta N_\phi log N_\phi)$. On the GPU side however, the advection phase becomes the bottleneck despite its lower complexity $O(N_\theta N_\phi)$. This is caused by both the relatively low locality in advection mentioned in §4.1 as well as the branching mechanism of GPU parallelization. Although we simplified the sampling process for our GPU implementation and reduced the kernel branching to a minimum, threads from the same warp are still most likely to pick different branches during runtime, introducing warp divergence. Since the scale of the problem and the accessing sequence is known beforehand for projection, we can schedule a batch of FFT operations prior to execution to achieve maximum performance.

6.2 Singularity

As shown in Fig. 12, our method circumvents the singularity problem at the poles without revealing any visible artifacts.

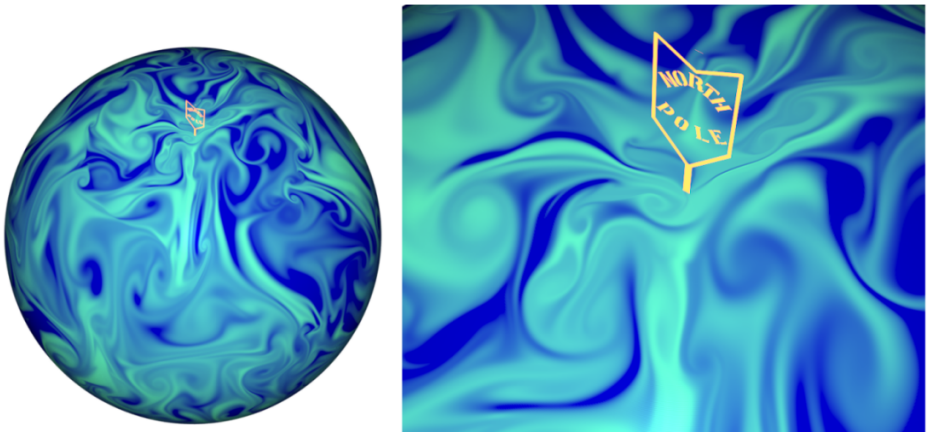

Fig. 12. Density flow over the North pole without visual artifacts.

Fig. 13. Identical velocity and density field initializations simulated on two different polar axis orientations. The right two images show flow on a sphere with its polar axis rotated 30 degrees with respect to the polar axis in the left two images. The top images are at frame 25 and the bottom images are at frame 75.

6.3 Pole Orientation

Fig. 13 demonstrates that identical fluid initializations are not significantly affected by the orientation of the underlying spherical coordinate system used to evaluate the simulation. In Fig. 13, the right column of images show fluid flow over time on a sphere that has a polar axis that is rotated with respect to the polar axis of the sphere in the left column of images. The simulations appear nearly identical in early frames, and only diverge in appearance over a long period of time. This is to be expected, as the initial velocity and density distributions, though identical, are evaluated along grids with rotated latitude and longitude lines with respect to one another.

7 LIMITATIONS AND FUTURE WORK

Using a staggered spherical grid enabled the real-time split-wise integration of the Navier-Stokes equations in spherical coordinates while circumventing issues at the poles. However, our work still has certain limitations.

Our approach uses a semi-Lagrangian method to solve for advection and therefore also suffers from numerical dissipation. We can partially correct these losses by adding vorticity confinement as described by Fedkiw et al. [2001], or by following any other method established in this direction [Huang et al. 2015]. Most notably, perhaps, would be the advection-reflection method recently proposed by Zehnder et al. [2018], which can be directly applied to our solver because of how we have defined our advection-projection scheme. Additionally, to implicitly solve for the geometric terms introduced by spherical coordinates, we applied an edge-cell center-edge interpolation loop. This would theoretically introduce a small amount of dissipation. Since it is not the dominant source of dissipation compared to that introduced in advection and projection, we do not consider this a significant issue.

It is also worth noting that while our GPU and CPU implementations both generate well-behaved fluid simulations on the sphere, they will in general give different results, primarily because the GPU implementation uses a simplified version of our CPU advection scheme at the poles. It is worth exploring the numerical differences between these two implementations while independently evaluating their respective architecture precision dependencies.

We can further extend the art-directability of our solver with a few additional features. For example, the tools in Hill and Henderson's implementation [2016] support the definition of local velocity sinks and sources, which can be easily ported to our solver as well. The initialization of velocities and other scalar fields in (θ, ϕ) space should also be made more intuitive to artists. The ability to 'paint' these fields onto the surface of a sphere might accomplish this.

More challenging extensions would include new forces and types of fluids. We could, for example, extend our solution model to include viscosity and other nonlinear fluid forces. Lastly, by using a staggered grid on a discretized surface, we position our work within the broader community of hybrid particle-grid solvers. This means that there are ample opportunities and works to draw from to refine our method and extend its potential applications. For example, we could model other types of free surface fluids [Zhu and Bridson 2005] and granular media [Klár et al. 2016; Stomakhin et al. 2013] on a sphere.

ACKNOWLEDGMENTS

We would like to thank Professor Stephen Lane for useful discussions. This work was supported in part by NSF Grants IIS-1755544 and CCF-1813624, a gift from Harlan Stone, a gift from Adobe Inc., NVIDIA GPU grants, and Houdini licenses from SideFX.

REFERENCES

G. K. Batchelor. 2000. *An introduction to fluid dynamics.* Cambridge University Press.

G. Bradski. 2000. The OpenCV library. *J Soft Tools* (2000).

Robert Bridson. 2015. *Fluid simulation for computer graphics.* CRC Press.

Robert Bridson, Jim Houriham, and Marcus Nordenstam. 2007. Curl-noise for procedural fluid flow. *ACM Trans Graph* 26, 3 (2007).

Maria Francesca Carfora. 2007. Semi-Lagrangian advection on a spherical geodesic grid. 55 (09 2007), 127.

Leonardo Carvalho, Maria Andrade, and Luiz Velho. 2012. Fluid simulation on surfaces in the GPU. In *SIBGRAPI Conf Graph Patt Images*. 205–212.

Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. 2007. Stable, circulation-preserving, simplicial fluids. *ACM Trans Graph* 26, 1 (2007).

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Comp Graph Inter Tech*. 15–22.

Andrew S Glassner. 1999. *Andrew Glassner's notebook: Recreational computer graphics.*

Kyle Hegeman, Michael Ashikhmin, Hongyu Wang, and Hong Qin. 2009. GPU-based conformal flow on surfaces.

David J. Hill and Ronald D. Henderson. 2016. Efficient fluid simulation on the surface of a sphere. *ACM Trans Graph* 35, 2 (2016), 16:1–16:9.

Zhanpeng Huang, Ladislav Kavan, Weikai Li, Pan Hui, and Guanghong Gong. 2015. Reducing numerical dissipation in smoke simulation. *Graph. Models* 78, C (2015), 10–25.

Gergely Klár, Theodore Gast, Andre Pradhana, Chuyuan Fu, Craig Schroeder, Chenfanfu Jiang, and Joseph Teran. 2016. Drucker-prager elastoplasticity for sand animation. *ACM Trans Graph* 35, 4 (2016), 103.

Ming-Chih Lai and Wei-Cheng Wang. 2002. Fast direct solvers for Poisson equation on 2D polar and spherical geometries. *Num Meth Part Diff Eq* 18, 1 (2002), 18:56–18:68.

Huan Mei, Faming Wang, Zhong Zeng, Zhouhua Qiu, Linmao Yin, and Liang Li. 2016. A global spectral element model for Poisson equations and advective flow over a sphere. *Adv Atom Sci* 33, 3 (2016), 377–390.

Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. OpenVDB: an open-source data structure and toolkit for high-resolution volumes. In *SIGGRAPH Courses*. 19.

Ramachandran D. Nair, Stephen J. Thomas, and Richard D. Loft. 2005. A discontinuous Galerkin transport scheme on the cubed sphere. *Mon Weather Rev* 133, 4 (2005), 814–828.

Azencot Omri, WeiÃ§mann Steffen, Ovsjanikov Maks, Wardetzky Max, and BenâĂŘChen Mirela. 2014. Functional fluids on surfaces. *Comp Graph Forum* 33, 5 (2014), 237–246.

William M. Putman and Shian-Jiann Lin. 2007. Finite-volume transport on various cubed-sphere grids. *J Comp Phys* 227, 1 (2007), 55–78.

Dave Randall. 2011. An introduction to numerical modeling of the atmosphere.

David A. Randall, Todd D. Ringler, Ross P. Heikes, Phil Jones, and John Baumgardner. 2002. Climate modeling with spherical geodesic grids. *Comp Sci Eng* 4, 5 (2002), 32–41.

C. Ronchi, R. Iacono, and P.S. Paolucci. 1996. The "cubed sphere". *J Comp Phys* 124, 1 (1996), 93–114.

Lin Shi and Yizhou Yu. 2004. Inviscid and incompressible fluid simulation on triangle meshes: Research articles. *Comp Anim Virt Worlds* 15, 3-4 (2004), 173–181.

Jos Stam. 1999. Stable fluids. In *Comp Graph Inter Tech*. 121–128.

Jos Stam. 2003. Flows on surfaces of arbitrary topology. *ACM Trans Graph* 22, 3 (2003), 724–731.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A material point method for snow simulation. *ACM Trans Graph* 32, 4 (2013), 102.

J. B. White, III and J. J. Dongarra. 2011. High-performance high-resolution semi-Lagrangian tracer transport on a sphere. *J Comp Phys* 230, 17 (2011), 6778–6799.

J. Zehnder, R. Narain, and B. Thomaszewski. 2018. An advection-reflection solver for detail-preserving fluid simulation. *ACM Trans Graph* 37, 4 (2018).

Yao Zhang, Jonathan Cohen, and John D. Owens. 2010. Fast tridiagonal solvers on the GPU. *SIGPLAN Not* 45, 5 (2010), 127–136.

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Trans Graph* 24, 3 (2005), 965–972.