Scalable Laplacian Eigenfluids

QIAODONG CUI, PRADEEP SEN, University of California, Santa Barbara
THEODORE KIM, Pixar Animation Studios

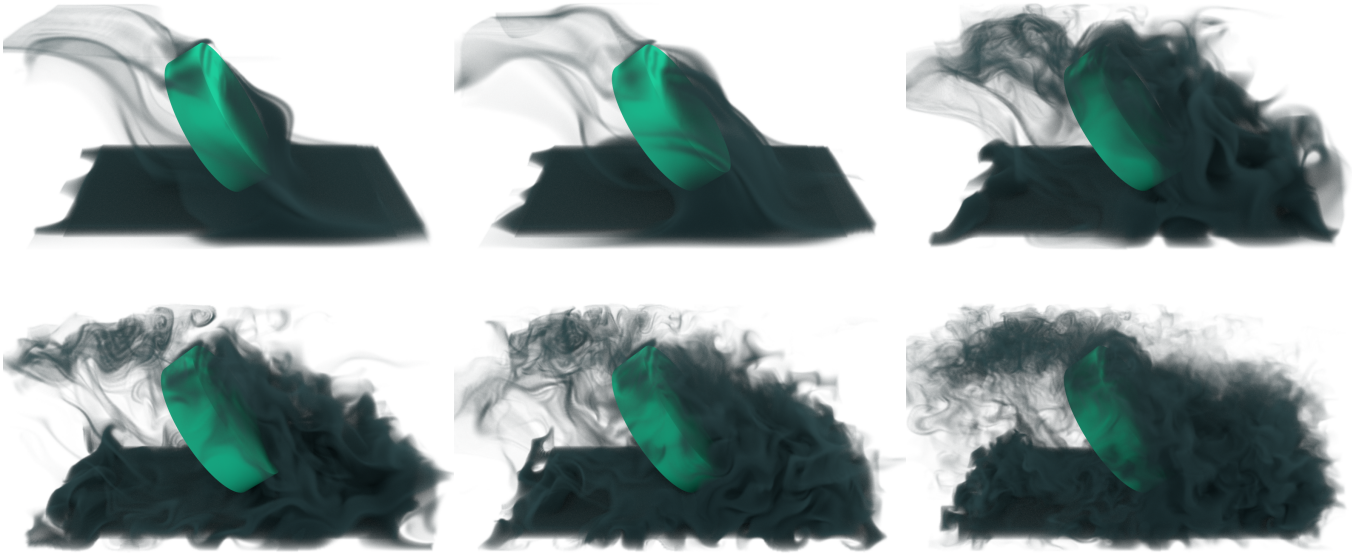


Fig. 1. Paddle wheel scene: Top row: $r = 100$, $r = 200$, $r = 1000$ basis functions are used. Bottom row: $r = 2000$, $r = 4000$, $r = 12600$ basis functions are used. Previous approaches have only been able to achieve $r \approx 500$, and would have needed 2.25 TB of memory to simulate the $r = 12600$ case.

The Laplacian Eigenfunction method for fluid simulation, which we refer to as *Eigenfluids*, introduced an elegant new way to capture intricate fluid flows with near-zero viscosity. However, the approach does not scale well, as the memory cost grows prohibitively with the number of eigenfunctions. The method also lacks generality, because the dynamics are constrained to a closed box with Dirichlet boundaries, while open, Neumann boundaries are also needed in most practical scenarios. To address these limitations, we present a set of analytic eigenfunctions that supports uniform Neumann and Dirichlet conditions along each domain boundary, and show that by carefully applying the discrete sine and cosine transforms, the storage costs of the eigenfunctions can be made completely negligible. The resulting algorithm is both faster and more memory-efficient than previous approaches, and able to achieve lower viscosities than similar pseudo-spectral methods. We are able to surpass the scalability of the original Laplacian Eigenfunction approach by over two orders of magnitude when simulating rectangular domains. Finally, we show that the formulation allows forward scattering to be directed in a way that is not possible with any other method.

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

Additional Key Words and Phrases: fluid simulation, physically based animation

Authors' addresses: Qiaodong Cui, Pradeep Sen, University of California, Santa Barbara; Theodore Kim, Pixar Animation Studios.

ACM Reference Format:
Qiaodong Cui, Pradeep Sen and Theodore Kim. 2018. Scalable Laplacian Eigenfluids. *ACM Trans. Graph.* 37, 4, Article 87 (August 2018), 12 pages. https://doi.org/10.1145/3197517.3201352

1 INTRODUCTION

While much progress has been made in fluid simulation over the last decade, efficiently simulating inviscid fluids that do not suffer from excessive numerical dissipation remains an ongoing challenge. Even in simple scenarios, the results can look unconvincingly viscous if any extraneous numerical diffusion is present.

Methods based on the traditional method of Stam [1999] usually compensate for the dissipated energy using techniques such as vorticity confinement [Fedkiw et al. 2001] or IVOCK [Zhang et al. 2015]. Alternatively, methods can be devised that are non-dissipative by construction, though these can involve asymmetric linear solves and non-linear Newton iterations [Mullen et al. 2009].

The method of Laplacian Eigenfunctions [De Witt et al. 2012], which we shall refer to as the *Eigenfluids* method, proposes an alternative approach for simulating inviscid flows. The simulation occurs over a set of eigenfunctions, and the primary variables are the coefficients of these functions. The functions are inherently divergence-free, so the dissipation introduced by a pressure projection is avoided entirely. The advection operator is formulated in terms of the eigenfunctions, so the numerical smearing that occurs

during semi-Lagrangian backtraces is also eliminated. The functions have global support, so interesting results appear with even a handful of coefficients. The simulations are fast and lively, and possess natural connections to model reduction [Liu et al. 2015].

With these advantages come several significant drawbacks. Like most model reduction methods, Eigenfluids require a large basis matrix to be present in memory at runtime. Each column of the matrix represents an eigenfunction sampled over some spatial partitioning, e.g., a regular grid or a tetrahedral mesh. When a high spatial resolution is needed, only a limited number of eigenfunctions can be used, so the 3D simulations in previous work were limited to several hundred basis functions (540 for [De Witt et al. 2012] and 230 for [Liu et al. 2015]). However, compelling, fine-scale dynamics continue to appear when more basis functions are added, so a practical method for improving the scalability is needed.

If the domain is rectangular–which is an extremely common production scenario [Angelidis 2017]–a potential solution was proposed in De Witt et al. [2012]: the eigenfunctions can be written in closed form. In lieu of storing a large matrix, entries can be recomputed on-the-fly at runtime. However, evaluating these functions dominates the running time and makes the algorithm unacceptably slow.

The Eigenfluids method thus appears to possess the classic zero-sum tradeoff of time vs. memory, but we show that this is not in fact the case. By carefully applying discrete sine (DST) and cosine (DCT) transforms over a rectangular domain, it is in fact possible to arrive at an algorithm that is *both faster and more memory efficient* than previous approaches. This is achieved without introducing any numerical approximations. Using our approach, we are able to scale the original Eigenfluids algorithm by an additional two orders of magnitude. We are able to efficiently simulate scenarios that would otherwise have taken terabytes of memory or more than an hour of per-frame computation.

The second major limitation of the Eigenfluids method is that it is constrained to a closed box, because analytic eigenfunctions have only been presented for boundary conditions that are Dirichlet on the velocity. However, many practical scenarios require Neumann velocity boundaries that allow mass to flow out of the domain. To remove this limitation, we derive the analytic eigenfunctions that arise when 1 to 6 walls of a 3D domain are set uniformly to Neumann boundaries. Fortunately, the DCT and DST accelerations from the Dirichlet case carry over to the Neumann case as well.

As the algorithm scales, a CFL-like stability condition emerges that makes it necessary to use an implicit solver. While the underlying advection tensor is inherently anti-symmetric, we find that symmetric solvers can still be used, because the resulting systems are extremely well-conditioned. We additionally find that as Eigenfluids is scaled up to thousands of basis functions, the bottleneck becomes the storage of the sparse, 3^{rd}-order tensor for advection. However, most of these entries are near-zero, and we show that ~90% of the entries can be discarded while still retaining the overall character of the flow. All of these advantages can be leveraged to obtain a real-time version of the algorithm.

Our method shares many similarities with the wide family of spectral methods that also employ fast transformations [Boyd 2001; Canuto et al. 2007; Gottlieb and Orszag 1977; Trefethen 2000], so we perform an in-depth comparison in §6 that shows that our algorithm is capable of achieving significantly lower viscosities.

Finally, the Eigenfluids advection formulation offers precise and direct control of the phenomenon of *forward scattering*, i.e., the rate at which energy cascades from low to high frequencies. Using this mechanism, we can stably produce aesthetically interesting flows that are not possible with any other method.

In summary, our work makes the specific contributions:

- Use of DCT and DST to remove the memory limitations imposed by the eigenfunction basis matrix.
- Generalization of the analytic eigenfunctions of Eigenfluids to support Neumann velocity boundaries.
- Directable forward scattering through tensor reweighting.
- Demonstration of effective lossy compression on the 3^{rd}-order advection tensor.
- Demonstration of support for real-time interaction.
- Lower viscosity flows than equivalent spectral methods.

2 RELATED WORK

Stam [1999] introduced the now-standard method for fluid simulation for computer graphics in the form of an advection-projection solver. However, both the pressure projection and semi-Lagrangian advection schemes introduce numerical dissipation that many subsequent works have sought to mitigate. The most direct method to do this is to increase the underlying grid resolution, so simulations have been performed on large grids such as octrees [Losasso et al. 2004], multigrid hierarchies [Ferstl et al. 2014; McAdams et al. 2010], and sparsely paged grids [Setaluri et al. 2014].

Methods for re-injecting dissipated energy have also been explored, such as vorticity confinement [Fedkiw et al. 2001], vortex particles [Selle et al. 2005], IVOCK [Zhang et al. 2015], and turbulence methods that are applied as a post-process [Kim et al. 2008; Narain et al. 2008; Schechter and Bridson 2008]. The structure-preserving properties of Lagrangian methods have also been leveraged in the form of vortex filament [Weissmann and Pinkall 2010] and vortex sheet methods [Brochu et al. 2012; Pfaff et al. 2012]. Leveraging the respective advantages of both the Eulerian and Lagrangian frames, FLIP [Zhu and Bridson 2005], APIC [Jiang et al. 2015], and PPIC [Fu et al. 2017] methods have also been developed.

Mullen et al. [2009] introduced the first method in computer graphics for simulating totally inviscid flows. While the dynamics of a zero-viscosity "super-fluid" can be somewhat unintuitive, being able to achieve this regime then allows the user to gradually dial in the desired level of viscosity. However, the method can be computationally expensive, as it involves asymmetric linear solves and non-linear Newton iterations. The Schödinger's Smoke [Chern et al. 2016] algorithm also exhibits inviscid behavior, but it does not contain a viscosity parameter, so direct comparisons are difficult.

The method of Laplacian Eigenfunctions, which we refer to as *Eigenfluids*, was developed by De Witt et al. [2012], and further stabilized using variational methods by Liu et al. [2015]. While we will describe the method in detail later (§3), we will position it within the literature here. Because the method begins to produce non-trivial results even with a very small number of degrees of freedom, it can be seen as a *model reduction* method, albeit one that does not need

the example snapshots required by previous approaches [Kim and Delaney 2013; Stanton et al. 2013; Treuille et al. 2006]. Instead of discovering a basis from example data, the simulation is performed in the space of Laplacian eigenvectors defined over the simulation mesh. Due to the correspondence of these eigenvectors to the intrinsic frequencies of the mesh, numerical dissipation can be eliminated entirely. Interestingly, the work of Gupta and Narasimhan [2007] also performed reduced fluid simulations in an analytic (Legendre) space, but traded spatial resolution for rendering efficiency.

Model reduction methods suffer from the problem of basis matrix storage. If a simulation on a very high resolution $O(N^3)$ velocity field is desired, a matrix is required where each column is effectively a copy of the high resolution field. A matrix with r columns takes up $O(rN^3)$ memory, so the system capacity is quickly exhausted for $r \approx 500$. Several approaches have tried to address this issue through modularization [Wicke et al. 2009] and JPEG-like compression [Jones et al. 2016], but the issue is far from resolved. We will show that it is possible to store the analytic Eigenfluid basis in $O(r)$ memory, which removes the basis storage problem entirely.

Our method makes extensive use of discrete sine (DST) and cosine (DCT) transforms, and thus shares connections with a variety of spectral fluid solvers. Stam [1999, 2002] first showed that by imposing periodic boundaries, the FFT could be used to accelerate the pressure projection stage of Stable Fluids. Later, Long and Reinhard [2009] showed that this approach could be extended to Dirichlet boundaries by using the DST and DCT, and Henderson [2012] showed that it scales favorably over multiple processors. We will show that by leveraging the correspondence between these spectral modes and the eigenfunctions of the Laplacian, and by performing the non-linear advection entirely within the spectral domain, we can efficiently compute inviscid flows.

The relationship to spectral solvers extends more widely to spectral methods in general. Spectral methods have a long history stretching back to Lanczos [1938], and gained wider attention in fluid mechanics during the 1970s due to the work of Orszag [1969, 1971]. Many excellent texts exist that describe these methods [Boyd 2001; Canuto et al. 1988; Trefethen 2000], but relative to our current method, the use of Chebyshev and Legendre polynomials to handle non-periodic boundary conditions is the most relevant feature [Canuto et al. 2007; Gottlieb and Orszag 1977]. We will perform an extensive comparison of our own method against a modern pseudo-spectral library [Burns et al. 2017] in §6, and show that our method can simulate flows with significantly lower viscosity.

3 THE METHOD OF LAPLACIAN EIGENFUNCTIONS

Notation: We will use unbolded lower case to denote scalars (k), bold lower case to denote vectors ($\mathbf{u}$), and bold upper case to denote matrices ($\mathbf{C}$). An overdot denotes a time derivative ($\dot{\mathbf{u}} = \frac{\partial \mathbf{u}}{\partial t}$), and superscripts denotes the timestep ($\mathbf{w}^t$ and $\mathbf{w}^{t+1}$). Angle brackets denote the inner product of two fields, i.e., $\langle \mathbf{u}, \mathbf{\Psi} \rangle = \int_\Omega \mathbf{u} \cdot \mathbf{\Psi} d\Omega$.

The 3rd-order advection tensor $\mathbf{\mathfrak{C}} \in \mathbb{R}^{r \times r \times r}$ appears throughout. Note, the order here refers to the tensor rank, not the Taylor truncation order. We denote contraction along the third index using the $\times_3$ notation [Golub and Van Loan 2012]. This yields a matrix, e.g., $\mathbf{\mathfrak{C}} \times_3 \mathbf{w} = \mathbf{C}$, where $\mathbf{C} \in \mathbb{R}^{r \times r}$. Products along the other two indices can be written using the usual matrix notation, but are then applied to all r matrices along the third index, i.e., $\mathbf{w}^T \mathbf{\mathfrak{C}} \mathbf{w} = \mathbf{x}$ where $\mathbf{x} \in \mathbb{R}^r$.

Algorithm: We will first summarize the Eigenfluids approach of De Witt et al. [2012] for the Navier-Stokes equations:

$$\begin{aligned} \dot{\mathbf{u}} &= -\mathbf{u} \cdot \nabla \mathbf{u} + \nu \nabla^2 \mathbf{u} - \nabla p + \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0. \end{aligned} \tag{1}$$

Here, ν denotes viscosity, p the pressure, and $\mathbf{f}$ external forces. The velocity field $\mathbf{u}$ is usually discretized on a uniform grid of size $N \times N \times N = N^3$.

In the Eigenfluids algorithm, $\mathbf{u}$ is encoded as a linear combination of vector eigenfunctions, $\mathbf{\Psi}$. These functions are defined according to a vector wave index, $\mathbf{k} = (k_x, k_y, k_z)$. Each $\mathbf{\Psi}$ has three associated scalar eigenfunctions, $\Phi_x(\mathbf{k})$, $\Phi_y(\mathbf{k})$, and $\Phi_z(\mathbf{k})$, which respectively specify the x, y, and z velocity components for that index, i.e., $\mathbf{\Psi} = \{\Phi_x(\mathbf{k}), \Phi_y(\mathbf{k}), \Phi_z(\mathbf{k})\}$. The total number of eigenfunctions being simulated is denoted r, which is the reduced simulation rank.

We use the notation $\mathbf{\Psi}_i$ for cases where it is necessary to generically iterate over all r eigenfunctions. This allows us to write the velocity field $\mathbf{u}$ in terms of the eigenfunctions $\mathbf{\Psi}$ simply as:

$$\mathbf{u} = \sum_{i=1}^{r} w_i \mathbf{\Psi}_i. \tag{2}$$

All of the weights w_i are then concatenated into a vector $\mathbf{w} \in \mathbb{R}^r$. If the number of eigenfunctions r is much less than N^3, then a model reduction-like acceleration is realized, as we only have to solve a system of size $O(r^2)$ instead of $O(N^3)$ (i.e., an $N^3 \times N^3$ matrix with $O(N^3)$ sparsity). In practice, each $\mathbf{\Psi}_i$ is sampled onto the same grid as $\mathbf{u}$, and a large matrix $\mathbf{U} \in \mathbb{R}^{N^3 \times r}$ is used to transform between the two representations. Thus, Eqn. 2 can be written as $\mathbf{u} = \mathbf{U}\mathbf{w}$.

The basis vectors $\mathbf{\Psi}_i$ are found by solving for the eigenfunctions of the vector Laplacian, $\nabla^2 \mathbf{\Psi}_i = \lambda_i \mathbf{\Psi}_i$, with Dirichlet conditions imposed along the boundary Γ, i.e., $\mathbf{\Psi}_\Gamma = 0$. On a rectangular domain, these eigenfunctions have an analytic form that we will describe in §4. The velocity field formed by these eigenfunctions is intrinsically divergence-free, so if the simulation is performed in this coordinate system, no pressure projection step is needed. Similar to Stam [1999; 2002], damping becomes a point-wise exponential: $w_k^{t+1} = w_k^t e^{\nu \lambda_i \Delta t}$. External forces $\mathbf{f}$ can be projected onto the eigenfunctions using $\mathbf{U}^T \mathbf{f} = \hat{\mathbf{f}}$.

In order to formulate the advection operator, a vorticity basis function ϕ is constructed for each eigenfunctions by computing $\phi_i = \nabla \times \mathbf{\Psi}_i$. A 3rd-order advection tensor $\mathbf{\mathfrak{C}}$ is then computed with entries $\mathbf{\mathfrak{C}}(g, h, i) = [\nabla \times (\phi_h \times \mathbf{\Psi}_i)] \cdot \phi_g$. The contribution of the advection operator to the time derivative can then be written,

$$\dot{w}_g = \sum_{h=1}^{r} \sum_{i=1}^{r} w_h w_i \mathbf{\mathfrak{C}}(g, h, i), \tag{3}$$

which can be expressed in tensor form as $\dot{\mathbf{w}} = \mathbf{w}^T \mathbf{\mathfrak{C}} \mathbf{w}$. Following De Witt et al. [2012], the complete equations can now be integrated using forward Euler,

$$\mathbf{w}^{t+1} = \left(\mathbf{w}^t + \Delta t \left(\mathbf{w}^t \right)^T \mathbf{\mathfrak{C}} \mathbf{w}^t + \Delta t \hat{\mathbf{f}} \right) e^{\nu \Delta t \Lambda}, \tag{4}$$

where Λ denotes a diagonal matrix of all the vector Laplacian eigenvalues, and we assume the mass associated with the force term is

equal to one. Applying an alternate explicit method such as RK4 or exponential integration also becomes straightforward.

Discussion: Once $\mathbf{w}$ has been stepped forward in time, the velocity field $\mathbf{u}$ can be reconstructed via $\mathbf{u} = \mathbf{U}\mathbf{w}$ and used to advect particles or densities. Both the storage and application of $\mathbf{U}$ presents challenges. If a high-resolution velocity field is needed, e.g., $N = 256$, then the $\mathbb{R}^{N^3 \times r}$ matrix quickly consumes all available memory as r is increased. In addition to these memory issues, the computational cost of the $\mathbf{U}\mathbf{w}$ and $\mathbf{U}^T\mathbf{f}$ matrix-vector multiplies can dominate the overall running time. In De Witt et al. [2012], these multiplies can take up to 84% of the running time, and similar stages in other algorithms [Kim and Delaney 2013] take up to 99%.

Alternatively, De Witt et al. [2012] observe that if analytic eigenfunctions are available, then storage issues can be eliminated entirely by recomputing the entries of $\mathbf{U}$ on the fly. Our own measurements show that this increases the already considerable expense of the matrix-vector multiply by an additional 5× to 7× (Table 1), and makes the overall algorithm prohibitively slow.

4 FAST, GENERAL, ANALYTIC BASIS FUNCTIONS

In this section, we will first show how to use DCT and DST to improve both the memory complexity and runtime performance of the Eigenfluids algorithm. Specifically, the memory complexity will drop from $O(rN^3)$ to $O(r)$, effectively removing the basis storage problem. The running time will shift from $O(rN^3)$ to $O(N^3 \log N)$, which will yield an order of magnitude speedup in practice.

With these improvements in place, we will present a set of analytical eigenfunctions that support any combination of Neumann and Dirichlet velocity conditions along the boundaries of the simulation. These functions will be chosen so that the accelerations from DCT and DST can be applied with only minor modifications.

4.1 Fast Projection and Reconstruction

The DCT can be used to perform fast, memory-efficient projections and reconstructions. For simplicity, we will show this in 2D, but generalization to 3D is straightforward. De Witt et al. [2012] proposed eigenfunctions defined over $\Omega \in [0, \pi]^2$ that satisfy Dirichlet boundary conditions along its walls,

$$\begin{aligned} \Phi_x(\mathbf{k}) &= -\frac{1}{\eta_\mathbf{k}} k_y \sin(k_x x)\cos(k_y y) \\ \Phi_y(\mathbf{k}) &= \frac{1}{\eta_\mathbf{k}} k_x \cos(k_x x)\sin(k_y y), \end{aligned} \tag{5}$$

where $k_x, k_y \in \mathbb{Z}^+$. We use the normalization term $\eta_\mathbf{k}$ to denote the square root of $-\lambda_\mathbf{k}$, i.e., $\eta_\mathbf{k} = \sqrt{k_x^2 + k_y^2}$. It is straightforward to project a force field $\mathbf{f}$ onto these functions using a mix of sine and cosine transforms. For example, the projected x and y components of $\mathbf{f}$ correspond to:

$$\begin{aligned} \langle \mathbf{f}_x, \Phi_x(\mathbf{k}) \rangle &= -\frac{1}{\eta_\mathbf{k}} k_y \iint_\Omega \mathbf{f}_x \sin(k_x x)\cos(k_y y)\, dx\, dy \\ \left\langle \mathbf{f}_y, \Phi_y(\mathbf{k}) \right\rangle &= \frac{1}{\eta_\mathbf{k}} k_x \iint_\Omega \mathbf{f}_y \cos(k_x x)\sin(k_y y)\, dx\, dy. \end{aligned} \tag{6}$$

The first projection can be computed by performing a DST in the x direction and a DCT in the y direction, and the second by applying DCT in x and DST in y. The result is a delta function centered at $\mathbf{k}$ which is scaled by the projected quantity of interest:

$$\begin{aligned} \langle \mathbf{f}_x, \Phi_x(\mathbf{k}) \rangle &= -\frac{1}{\eta_\mathbf{k}} k_y \hat{\mathbf{f}}_x(\mathbf{k}) \\ \left\langle \mathbf{f}_y, \Phi_y(\mathbf{k}) \right\rangle &= \frac{1}{\eta_\mathbf{k}} k_x \hat{\mathbf{f}}_y(\mathbf{k}). \end{aligned} \tag{7}$$

Only the $\hat{\mathbf{f}}_x(\mathbf{k})$ and $\hat{\mathbf{f}}_y(\mathbf{k})$ coefficients need to be stored, which takes $O(r)$ memory; the basis matrix is implicitly encoded by the DCT/DST. Velocity reconstruction follows analogously: for example, the elements of $\mathbf{w}$ can be restated as $\hat{\mathbf{u}}_x(\mathbf{k})$ and mapped into 2D frequency space according to their wave index. An IDST in the x direction followed by an IDCT in y then recovers $\mathbf{u}_x$. In 3D, an additional trigonometric function appears in the product, which requires an additional DST or DCT to be performed in a third direction, but the approach is otherwise identical.

The fact that these bases take on a compact structure under these transforms was previously observed by Jones et al. [2016], but they did not use it to accelerate an Eigenfluids simulation. The transformation also has fundamental connections to the spectral methods of Stam [1999; 2002], Long and Reinhard [2009] and Henderson [2012], but they all used the transform to accelerate pressure projection, not velocity reconstruction.

4.2 Enabling Neumann Boundaries

The above transform only applies to analytic eigenfunctions corresponding to Dirichlet boundary conditions. We now present eigenfunctions that correspond to any number of walls being set to a Neumann condition, and show that the DCT-based accelerations can be applied to these functions as well. For simplicity, we will again present results in 2D, but the extension to 3D is straightforward. For completeness, the eigenfunctions for all the 3D cases are listed in the supplementary material.

Laplacian eigenfunctions can more generally be viewed as solutions to the homogeneous Helmholtz equation: $\nabla^2 g(x, y) = \lambda_\mathbf{k} g(x, y)$. In 2D, the function takes the form,

$$g(x, y) = \Big(a\cos(k_x x) + b\sin(k_x x)\Big)\Big(c\cos(k_y y) + d\sin(k_y y)\Big), \tag{8}$$

where (a, b, c, d) are undetermined constants. Each velocity eigenfunction then becomes an instance of this solution:

$$\begin{aligned} \Phi_x(\mathbf{k}) &= \Big(a_x\cos(k_x x) + b_x\sin(k_x x)\Big)\Big(c_x\cos(k_y y) + d_x\sin(k_y y)\Big) \\ \Phi_y(\mathbf{k}) &= \Big(a_y\cos(k_x x) + b_y\sin(k_x x)\Big)\Big(c_y\cos(k_y y) + d_y\sin(k_y y)\Big). \end{aligned}$$

The four-walled Dirichlet solution is retrieved for the special case where $a_x = d_x = b_y = c_y = 0$, $b_x c_x = -k_y/\eta_\mathbf{k}$, and $a_y d_y = k_x/\eta_\mathbf{k}$. We can solve for other solutions by coupling the two equations via the divergence-free constraint $\nabla \cdot \Psi = 0$, which expands to:

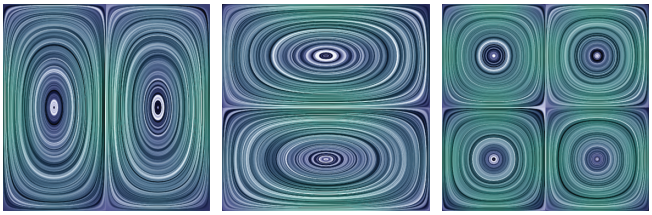
$$k_x b_x c_x + k_y a_y d_y = 0 \qquad k_x b_x d_x - k_y a_y c_y = 0 \tag{9}$$

$$-k_x a_x c_x + k_y b_y d_y = 0 \qquad k_x a_x d_x + k_y b_y c_y = 0. \tag{10}$$

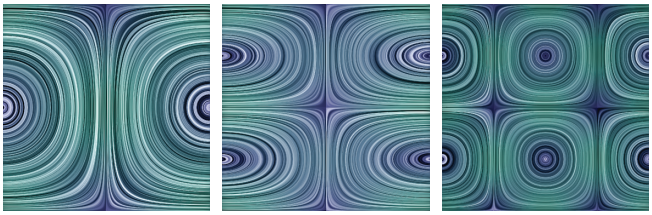
Additionally, we observe that the following conditions will minimize the number of DSTs and DCTs that are needed:

$$a_x b_x = 0 \qquad c_y d_y = 0 \tag{11}$$

$$a_y b_y = 0 \qquad c_x d_x = 0. \tag{12}$$



(a) Dirichlet boundaries along all walls



(b) Neumann boundaries on left and right walls, Dirichlet along top and bottom

Fig. 2. Visualization of the first three Dirichlet and Neumann bases in 2D using line integral convolution. Mass cannot flow through the walls in the Dirichlet case, but it can leave the domain in the Neumann case.

Sufficient conditions have now been specified to solve for Neumann eigenfunctions.

Two Neumann Walls: We first illustrate the case of two Neumann walls in the x direction and two Dirichlet walls along y:

$$\left.\frac{\partial \Phi_x}{\partial x}\right|_{x=0,\,\pi} = 0 \qquad\qquad \left.\Phi_y\right|_{y=0,\,\pi} = 0. \tag{13}$$

The values $b_x = c_y = 0$ select the trigonometric functions that satisfy these boundaries, as well as Eqn. 11. The solution now becomes,

$$\begin{aligned} \Phi_x(\mathbf{k}) &= a_x \cos(k_x x)\Big(c_x \cos(k_y y) + d_x \sin(k_y y)\Big) \\ \Phi_y(\mathbf{k}) &= d_y \sin(k_y y)\Big(a_y \cos(k_x x) + b_y \sin(k_x x)\Big), \end{aligned} \tag{14}$$

and two of the constraints from Eqn. 10 become:

$$k_x a_x d_x = 0 \qquad\qquad k_y a_y d_y = 0.$$

Setting $d_x = a_y = 0$ avoids a trivial solution and yields:

$$\begin{aligned} \Phi_x(\mathbf{k}) &= a_x c_x \cos(k_x x)\cos(k_y y) \\ \Phi_y(\mathbf{k}) &= b_y d_y \sin(k_y y)\sin(k_x x). \end{aligned} \tag{15}$$

The last constraint, $-k_x a_x c_x + k_y b_y d_y = 0$, can be satisfied using $a_x c_x = k_y/\eta_{\mathbf{k}}$ and $b_y d_y = k_x/\eta_{\mathbf{k}}$, where the $1/\eta_{\mathbf{k}}$ is added as a normalization. The final eigenfunctions are then:

$$\begin{aligned} \Phi_x(\mathbf{k}) &= \frac{1}{\eta_{\mathbf{k}}} k_y \cos(k_x x)\cos(k_y y) \\ \Phi_y(\mathbf{k}) &= \frac{1}{\eta_{\mathbf{k}}} k_x \sin(k_x x)\sin(k_y y). \end{aligned} \tag{16}$$

These eigenfunctions are clearly amenable to DCT and DST acceleration, as they are all products of trigonometric functions. They are visualized in Fig. 2. The eigenfunctions for Neumann walls along the y direction, as well as the case where all four walls are Neumann, can be obtained using a similar process.

One Neumann Wall: If Neumann conditions are only needed along one wall, Dirichlet conditions can be restored on the opposing wall. For example, if the x boundary conditions from Eqn. 13 are instead set to,

$$\left.\frac{\partial \Phi_x}{\partial x}\right|_{x=\pi} = 0 \qquad\qquad \left.\Phi_x\right|_{x=0} = 0, \tag{17}$$

then the same eigenfunctions from Eqn. 26 can be used. However, a half-period frequency shift is added so that k_x is a non-negative half-integer in lieu of an integer, i.e. $k_x \in (\mathbb{Z}^+ - 1/2)$. This is illustrated by the function $\sin((k - 1/2)x)$, where k is a positive integer. The function is zero at $x = 0$, which satisfies the Dirichlet condition, and its derivative, $\cos((k - 1/2)x)$, is zero at $x = \pi$, which satisfies the Neumann condition.

4.3 Computational Considerations

The velocity reconstruction method from §4.1 is already quite fast, but since $r \ll N^3$, it is also possible to perform a *pruned* DCT. In general, when $\mathbf{w}$ is transformed into the 3D frequency representation for $\hat{\mathbf{u}}_x(\mathbf{k})$, the non-zero entries are localized to a cube with length $\sqrt[3]{r}$ on each side, with one corner coincident with the zero-frequency, DC component. In lieu of performing three transforms of size $N^3 \log N$, we can use the knowledge that most of the coefficients are zero to skip many of the 1D transforms. The transforms along the first two dimensions can be pruned to $\sqrt[3]{r^2} N \log N$ and $\sqrt[3]{r} N^2 \log N$, and only the last dimension requires the full $N^3 \log N$. In practice, we found that this easy modification yields a 30% acceleration.

A slight modification is needed when there is a single Neumann boundary along a direction, because most FFT libraries do not support half-integer wave numbers. In this case, we double the resolution of the DST grid in the respective direction, and only keep the odd wave numbered coefficients. This extra factor of two is very modest compared to the $O(rN^3)$ memory complexity of the original Eigenfluids algorithm, so we found it acceptable.

5 STABLE EIGENFLUID DYNAMICS

With the basis functions in place, we will now describe the construction of the 3rd-order advection tensor $\mathfrak{C}$ and the time integration scheme. In particular, we will show that following the method of De Witt et al. [2012] results in an unstable simulation, and that the variational form from Liu et al. [2015] must be used instead. We will also comment on the sparsity of this tensor, and show how to perform the time integration using a symmetric solver.

5.1 Advection Tensor

The method of De Witt et al. [2012] computes each entry of the advection tensor as $\mathfrak{C}(g,h,i) = [\nabla \times (\phi_h \times \Psi_i)] \cdot \phi_g$, where Ψ_i is a velocity basis, and ϕ_* are vorticity bases. This formulation is effective for Dirichlet bases because the projection onto the vorticity basis is sparse, i.e., the cross product only produces a non-zero projection onto a small number of ϕ basis functions.

We have found that this property does not always hold for Neumann boundaries. For example, a single Neumann wall adds basis functions containing a half-period frequency shift, and the ϕ_g function can contain this shift while the cross product $[\nabla \times (\phi_h \times \Psi_i)]$ does not. The phase mismatch will cause $[\nabla \times (\phi_h \times \Psi_i)]$ to have non-zero projections onto an infinite series of ϕ_g. The error introduced by truncating the series will manifest as a blowup in energy

that occurs regardless of the timestep size. Energy renormalization [De Witt et al. 2012] can be used to coerce the simulation back to stability, but the resulting motion is clearly non-physical.

In order to avoid blowups, the tensor must be have an energy conserving anti-symmetry, $\mathfrak{C}(g,h,i) = -\mathfrak{C}(h,g,i)$. We found that even a simple post-process that forced the advection tensor to have this property significantly stabilized the simulation. However, Liu et al. [2015] showed a more principled way of enforcing this condition:

$$\mathfrak{C}(g,h,i) = \int_\Omega (\nabla \times \Psi_i) \cdot (\Psi_g \times \Psi_h) d\Omega. \tag{18}$$

By preferring to use this form, the anti-symmetry of the tensor is preserved by construction, energy blowups are avoided, and no ad-hoc post-processing of $\mathfrak{C}$ is needed. Details on computing this tensor in 2D are given in Appendix A.

Tensor Sparsity: Each entry in $\mathfrak{C}$ has three associated wave indices, which we will denote $\mathbf{k} = (k_x, k_y, k_z)$, $\mathbf{l} = (l_x, l_y, l_z)$, and $\mathbf{m} = (m_x, m_y, m_z)$, and can be expanded into sine and cosine integrals. As shown in §2.1 of supplementary material, with Dirichlet boundaries, the following conditions determine the sparsity:

$$m_x = l_x + k_x \qquad m_x = l_x - k_x \qquad m_x = k_x - l_x \tag{19}$$
$$m_y = l_y + k_y \qquad m_y = l_y - k_y \qquad m_y = k_y - l_y \tag{20}$$
$$m_z = l_z + k_z \qquad m_z = l_z - k_z \qquad m_z = k_z - l_z. \tag{21}$$

In order for an entry in $\mathfrak{C}$ to be non-zero, one relation in each row of Eqns. 19-21 must be satisfied. A single, fixed assignment of $\mathbf{l}$ and $\mathbf{k}$ can thus only generate 27 values for $\mathbf{m}$ that satisfy these relations. Since there are r^2 possible assignments for $\mathbf{l}$ and $\mathbf{k}$, there are $27r^2$ possible non-zero entries, or $O(r^2)$ sparsity in $\mathfrak{C}$.

When the boundary conditions in one direction is switched from Dirichlet to Neumann, one of the constraint rows in Eqns. 19-21 is dropped. For each fixed $\mathbf{l}$ and $\mathbf{k}$, the number of possible valid combinations relaxes from $3^3 = 27$ to $3^2\sqrt[3]{r} = 9\sqrt[3]{r}$. Over all r^2 combinations of $\mathbf{l}$ and $\mathbf{k}$, this then yields $(9\sqrt[3]{r})r^2$, or $O\left(r^{2+1/3}\right)$ sparsity in $\mathfrak{C}$. This trend continues as the number of Neumann boundaries is increased. For four Neumann walls, the sparsity becomes $O\left(r^{2+2/3}\right)$, and when all six walls, the tensor becomes a dense $O(r^3)$. For odd numbered walls, one Neumann boundary gives $O\left(r^{2+1/3}\right)$, three gives $O\left(r^{2+2/3}\right)$, and five yields $O\left(r^3\right)$.

The density of the Neumann advection tensor is initially counter-intuitive from a physical perspective, because it suggests that two low-frequency modes can combine to interact with an arbitrary high-frequency mode. This is in contrast to Fourier or Dirichlet modes, where two low-frequency modes can only scatter into a mode that is at most double the pair's maximum wavenumber. While longer-range frequency interactions are now possible, the advection coefficients are very nearly zero. The ability of low frequencies to activate arbitrary high frequencies is in fact severely limited.

The sparsity can vary from quadratic to cubic, so storing $\mathfrak{C}$ can become a scaling limitation on the Eigenfluids algorithm. Compression using sparse schemes [An et al. 2008; Hasan et al. 2008] is a direction for future work. However we will later show in §7 that the simplest lossy scheme, i.e., discarding small entries, can reduce the size of the tensor by an order of magnitude while still maintaining the overall character of the flow. This scheme will be particularly effective in the Neumann case, because as previously described, most of its $O(r^3)$ entries are near-zero.

Reweighting the tensor: One advantage of the Eigenfluids formulation is that energy cascades between different frequencies are directly encoded by the advection tensor. Therefore, *forward scattering*, which is usually characterized statistically over long time scales [Frisch 1995], can be observed with much higher temporal and spectral resolution, and even directly manipulated.

By reweighting the advection tensor, we observe that we can achieve a variety of stable fluid dynamics that are not possible using any other method. We state this modified advection tensor as:

$$\overline{\mathfrak{C}}(g,h,i) = b_g b_h b_i \cdot \mathfrak{C}(g,h,i). \tag{22}$$

We use a simple linear function $b_k = (1 + c|\mathbf{k}|^2)$ as our reweighting strategy, but many other choices are possible. Here, c is a tuning parameter that adjusts the speed of the energy cascade ($c = 0$ yields the original tensor). Intuitively, weights larger than one amplify scattering to specific frequencies, while weights smaller than one slow the rate of energy transfer. Since the weighted tensor $\overline{\mathfrak{C}}$ is still antisymmetric, $\overline{\mathfrak{C}}(g,h,i) = -\overline{\mathfrak{C}}(h,g,i)$, the new tensor will preserve energy. Different scattering behaviors will be shown in §7.

5.2 Implicit Time Integration

With our scalability improvements, we are able to perform simulations with much larger rank than previously possible. As a consequence, the stability of explicit timestepping becomes a concern. Deriving a CFL-like condition for the maximum stable Δt is not straightforward, as the usual "speed of sound" argument [Bridson 2015] is difficult to apply in the spectral domain, and the non-linear advection tensor interferes with spectral eigenanalysis approaches, which are inherently linear [Trefethen 2000]. In lieu of a direct expression, we have found empirically that the maximum stable Δt decreases quadratically with the basis rank. For $r = 1000$ this is already $\Delta t \approx 10^{-6}$, so an implicit treatment that allows for larger Δt is clearly needed.

We are again able to use machinery from Liu et al. [2015] in the form of their implicit trapezoidal update,

$$\mathbf{w}^{t+1} = \frac{\Delta t}{2}\mathbf{C}^{t+1}\mathbf{w}^{t+1} + \frac{\Delta t}{2}\mathbf{C}^{t}\mathbf{w}^{t} + \mathbf{w}^{t} + \hat{\mathbf{f}}, \tag{23}$$

where $\mathbf{C}^{t+1} = \mathfrak{C} \times_3 \mathbf{w}^{t+1}$ and $\mathbf{C}^{t} = \mathfrak{C} \times_3 \mathbf{w}^{t}$ denote contractions along the third mode of $\mathfrak{C}$. For brevity, we have written the equation in its inviscid form here, but an additional $e^{\nu \Delta t \Lambda}$ term should be multiplied on the right-hand side if viscosity is desired. While a full Newton solve could be performed to reconcile the $\mathbf{C}^{t+1}$ and $\mathbf{w}^{t+1}$ terms, we show in Fig. 3 that a semi-implicit solve (i.e., a single Newton iteration in the style of Baraff and Witkin [1998]) was sufficient to maintain stability and good energy behavior.

Symmetric Solvers: One drawback of implicit integration is that the anti-symmetric $\mathfrak{C}$ imposes anti-symmetry on its contraction $\mathbf{C}^{t+1}$, and necessitates the use of a non-symmetric solver such as BiCGSTAB. While these solvers can be effective, it is usually preferable to use a symmetric solver such as PCG whose convergence is both faster and better understood.

A classic method for applying a symmetric solver to an asymmetric matrix $\mathbf{A}$ is to apply conjugate gradient to its normal form, $\mathbf{A}^T\mathbf{A}$,

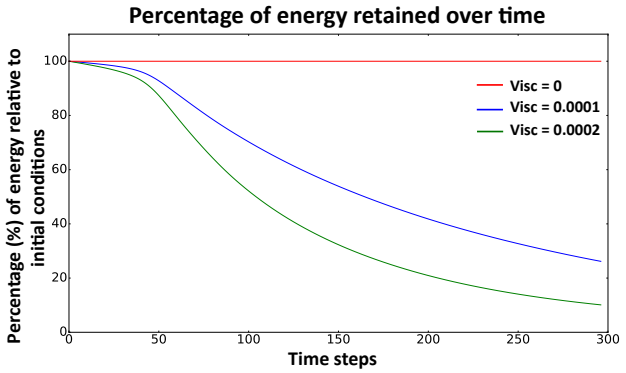


Fig. 3. Energy of the simulation in Fig. 5 over time, for multiple viscosities. For the zero-viscosity regime, energy is stably preserved even when only a single Newton iteration is used.

i.e., CGNR [Saad 2003]. The main caveat of CGNR is that the condition number of $\mathbf{A}$ is squared, which can make a badly-conditioned matrix even worse. However, we have found that Eqn. 23 is sufficiently well-conditioned that this caveat does not apply. We can rewrite the system as:

$$\left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C}^{t+1}\right)\mathbf{w}^{t+1} = \frac{\Delta t}{2}\mathbf{C}^{t}\mathbf{w}^{t} + \mathbf{w}^{t} + \hat{\mathbf{f}}. \tag{24}$$

Dropping the $t+1$ superscript from $\mathbf{C}^{t+1}$ for brevity, forming the normal matrix, and applying the identity $\mathbf{C} = -\mathbf{C}^{T}$ yields:

$$\left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C}\right)^{T}\left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C}\right) = \mathbf{I} + \frac{\Delta t^{2}}{4}\mathbf{C}^{T}\mathbf{C}. \tag{25}$$

The matrix $\mathbf{C}$ and its condition number are squared, but since $\Delta t \ll 1$, the squaring is offset by the $\Delta t^2/4$ term. The resulting normal matrix is very close to identity, and adding the viscosity term only pushes it closer. When $r = 8000$, frame-rate timesteps of $\Delta t \approx 1/30$ require 3 to 4 CGNR iterations to converge to a tolerance of 10^{-10}, and even large timesteps of $\Delta t = 0.2$ only need 6 iterations. The convergence is sufficiently fast that preconditioning is unnecessary.

6 COMPARISON WITH SPECTRAL METHODS

Since our method is closely related to spectral methods [Canuto et al. 1988], we discuss and compare the approaches here. Specifically, we compare our algorithm to the collocation methods from Dedalus [Burns et al. 2017], a recent spectral library that has been successfully used to advance understanding in both general [Davidovits and Fisch 2016] and computational [Vasil et al. 2016] physics.

Spectral collocation methods usually use Fourier bases to represent periodic boundaries, and either sine functions or Chebyshev polynomials to implement non-periodic Dirichlet boundaries. Thus, there are obvious similarities to our use of sine and cosine functions. When Neumann conditions are desired, Chebyshev polynomials are employed due to their non-periodicity. However, attempts to use trigonometric functions to perform an expansion of the Neumann boundary conditions suffer from the same problems with infinite non-zero projections we encountered with the method of De Witt et al. [2012] in §5.1. If Neumann conditions are desired in multiple directions, Chebyshev polynomials must be used in multiple directions as well. However, this introduces additional issues, because the derivatives of the polynomials become non-trivially coupled along multiple modes (e.g., Dedalus does not even allow the use of Chebyshev along more than one direction). In our method, using multiple Neumann conditions only requires a tweak to the trigonometric transform and the use of a different advection tensor.

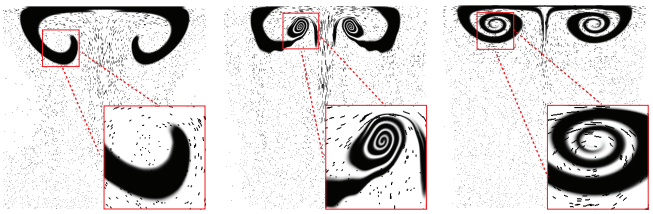


Fig. 4. **Left:** A spectral simulation using 60×60 basis functions in Dedalus. The top and bottom walls were set to Dirichlet velocity boundaries, while the left and right were set to Neumann. We used the smallest possible viscosity that did not destabilize the simulation. **Middle:** An equivalent 60×60 simulation using our method. We capture small-scale vortical features that the spectral simulation cannot. **Right:** A reference spectral simulation using 240×240 basis functions in Dedalus, with $\nu = 2 \times 10^{-5}$.

Furthermore, we can show that when Chebyshev polynomials are used, the resulting system will not be energy-preserving. Spectral collocation methods make extensive use of differentiation matrices [Canuto et al. 2007], as they are employed to obtain spatial derivatives at specific collocation points [Trefethen 2000]. While these differentiation matrices are never constructed explicitly, we can use them to determine the conservation properties of the underlying scheme. As shown in Canuto et al. [2007], in order for the semi-discrete Navier-Stokes equations to be energy-preserving, the differentiation matrix must be skew-symmetric. However, as we show in the supplemental material, the differentiation matrix that arises from Chebyshev polynomials does not fit this form.

In practice, this means that a viscosity term must be introduced into the spectral simulation or it will become unstable. Qualitatively, it also means that our Eigenfluid method will be able to capture lower viscosity flows. We verify this hypothesis by comparing our method to a 2D Dedalus simulation where both simulations use 60 basis functions along each axis (Fig. 4). The viscosity of the spectral simulation was set to $\nu = 10^{-4}$; further decrease destabilized the simulation. Our Eigenfluids simulation with $\nu = 2 \times 10^{-5}$ clearly captures non-trivial vortical structures which are not resolved in spectral simulation with the same resolution. A higher resolution spectral result with a more converged version of the same feature is also shown in Fig. 4. Overall, compared to spectral collocation methods, our method handles various boundary conditions more easily and captures a wider variety of low viscosity flows.

7 RESULTS

We implemented our Scalable Laplacian Eigenfluids algorithm in C++. The CGNR algorithm was implemented by modifying the CG implementation in Eigen [Guennebaud et al. 2010] to include an extra transposed matrix multiply. We used FFTW3 [Frigo and Johnson 2005] to perform DCT and DST. Multi-threading was enabled

Grid size	Basis dimension	Running Time					Total Memory Usage		
		on-the-fly basis (OTF)	cached basis	DCT (ours)	speedup vs. OTF	speedup vs. cached	cached basis	Ours	memory savings
128^3	200	8.660 secs	1.65 secs	0.10 secs	**87**×	**17**×	10.2 GB	185 MB	**55**×
	1000	44.10 secs	9.54 secs	0.10 secs	**440**×	**95**×	50.5 GB	223 MB	**226**×
220^3	200	45.56 secs	17.2 secs	0.78 secs	**58**×	**22**×	52.0 GB	938 MB	**55**×
	24000	6630 secs	-	0.78 secs	**8499**×	-	6.10 TB	26.0 GB	**235**×

Table 1. Running time and memory usage results of our DCT-based approach compared to caching a large matrix **U** of basis functions, or recomputing the entries of **U** on-the-fly. Double precision floating point was used, and multithreading is enabled for all three methods. Advection tensor size is reported using Dirichlet boundary conditions.

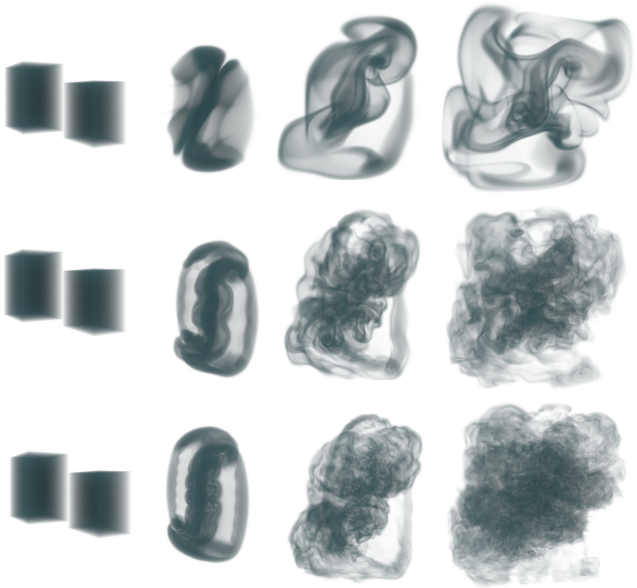
using OpenMP whenever possible, including during DCT and DST computations. We used a collocated grid for our velocity fields because FFTW3 computes the transformation at the center of the grid cell. Since there is no pressure projection performed in the spatial domain, the null space arguments for MAC grids do not apply.

The implementation of Long and Reinhard [2009] used the semi-Lagrangian advection from Zephyr [Kim 2013], but replaced the pressure projection with the DCT-based approach. All density advection was performed using a MacCormack scheme [Selle et al. 2008]. Similar to De Witt et al. [2012], Treuille et al. [2006], and Chern et al. [2016], explicit penalty forces are used to insert static and dynamic obstacles into scenes. All our results were run on a desktop with 96GB of memory and 12 cores running at 2.4 GHz.

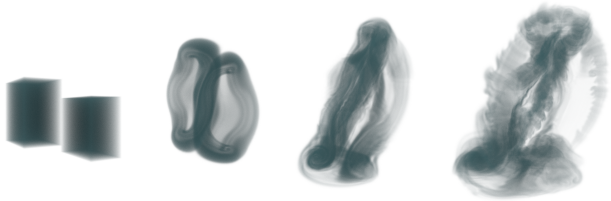
Colliding smoke jets. Our simplest example contains two blocks of smoke driven together by an initial impulse. As can be seen in Fig. 5, visually interesting details continue to appear as we increase the basis rank. The grid resolution is 220^3. As shown in Table 1, these scenes are infeasible with the original Eigenfluids algorithm. Either 6.10 *terabytes* of memory would be needed to store the basis, or 1.84 *hours* would be needed per frame to recompute the basis on-the-fly. Instead, we compute the velocity reconstruction **8499**× faster than the on-the-fly approach, and use **235**× less memory than the cached basis approach. The timings all use the unpruned DCT.

We also compare our method to Long and Reinhard [2009], as their use of DCT more closely matches our approach than the original Stam [1999] algorithm. Our results are clearly less viscous, as we do not perform the smearing-prone pressure projection or semi-Lagrangian advection. The complexity of the DCT-based projection is the same as our velocity reconstruction and force projection. If the pruned DCT from §4.3 is used, this stage of our method runs 30% faster. In Fig. 5, Long and Reinhard [2009] takes 2.50 secs per frame, while our method with 3000 basis functions takes 1.53 secs per frame. Our method preserves more detail and runs slightly faster.

Paddle wheel. We show a scene containing a moving Neumann obstacle in Fig. 1. This scene tests the scalability of our approach and shows its ability to accommodate Neumann boundary conditions. The scene contains two Neumann walls in the positive and negative x directions, and the basis rank is varied from $r = 100$ to 12600. The viscosity varies from 0.002 to zero, and smoke density is continually added along the bottom of the domain. The smoke also dissipates over time, so the entire box never becomes full. As shown in Fig. 1, more detail appears as we increase the basis rank. The timings are shown in Table 3. For this scene, at the maximum rank of $r = 12600$, each frame takes 6.6 secs.



(a) Top to bottom: $r = 200$, $r = 3000$, and $r = 24000$ with Dirichlet boundaries.



(b) Results of the semi-Lagrangian / DCT method of Long and Reinhard [2009]

Fig. 5. Colliding smoke scene: On top, the results of our method. As basis functions are added, more fine-scale detail emerges. On bottom, the results of the Stam-like DCT method of Long and Reinhard [2009]. Our results are clearly more inviscid.

Thin Dirichlet obstacles. We also test our algorithm using static, Dirichlet obstacles. As observed by De Witt et al. [2012],

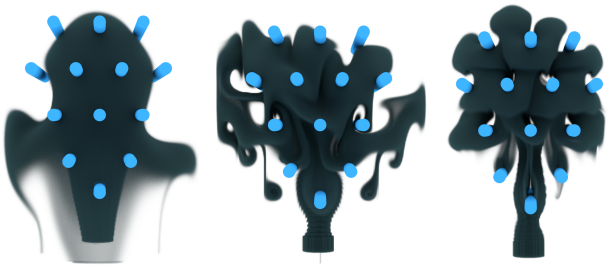


Fig. 6. Larger bases resolve thinner obstacles, as shown in the CYLINDERS scene. The basis rank from left to right is $r = 200$, $r = 1000$, and $r = 7000$.

the ability of the Eigenfluids algorithm to resolve obstacles is limited by the basis rank. Thus, as we add more bases, the fluid should be able to resolve finer obstacles. We test this by placing many thin cylinders into a scene, as shown in Fig. 6. We use a basis with two Neumann walls along the positive and negative y directions and vary the basis rank from $r = 200$ to 7000. The $r = 200$ basis completely fails to resolve the cylinders. When r is increased to 1000, the smoke interacts with the obstacles, but some of the smoke is pushed against the side of the box instead of flowing between the cylinders. When $r = 7000$, the velocity field is able to resolve each obstacle, so the smoke flows around them.

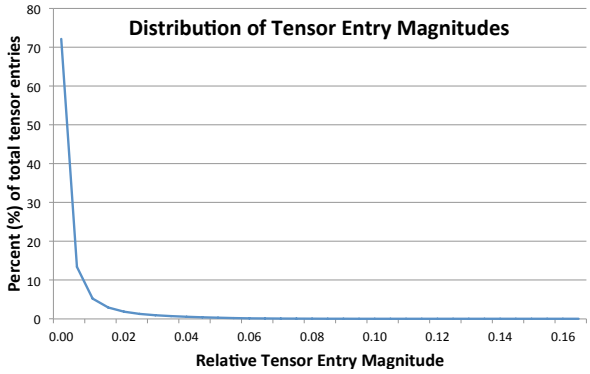


Fig. 7. Distribution of magnitudes in the advection tensor. Most of the entries have values that are near-zero, and can be discarded without significantly influencing the overall fluid motion.

Precomputing and compressing the tensor. While it should be possible to directly address the entries that satisfy Eqns. 19-21, we instead used the direct, $O(r^3)$ method for precomputing the advection tensor. The Dirichlet tensor for $r = 24000$ took 32 hours 4 minutes to precompute. When $r = 14000$, it took 5 hours 30 minutes. The two-wall Neumann tensor with $r = 12600$ took 6 hours 3 minutes to precompute. The advection tensor only depends on the wall boundaries, so it can be re-used across scenes.

As we have removed the storage issues surrounding the $O(rN^3)$ basis matrix, the advection tensor becomes the main memory bottleneck. When $r = 8000$, a Dirichlet tensor has $O(r^2)$ sparsity, and takes up 2.5 GB of memory. A two-Neumann wall case has $O(r^{2+1/3})$ sparsity, and consumes 25.7 GB of memory. However, even the simplest lossy compression scheme of dropping small entries is highly effective. As shown in Fig. 8, results that retain the lively character of Eigenfluids flows can be obtained even when 92% of the tensor entries have been dropped. As shown in Table 2, the time needed to compute the mode-3 product ($\times_3$) also decreases as entries are discarded. For flows that are dominated by external forces, even more entries can be dropped. More principled compression methods [An et al. 2008; Hasan et al. 2008] are a direction for future research.

% of entries discarded	0 %	60 %	80 %	90%
Tensor size	11 GB	4.4 GB	2.2 GB	1.1 GB
Contraction time	0.89s	0.45s	0.43s	0.22s

Table 2. Contraction timings for compressed tensors. As entries are discarded, the contraction time predictably decreases. Even after **10× lossy compression**, much of the overall fluid motion remains (Fig. 8).

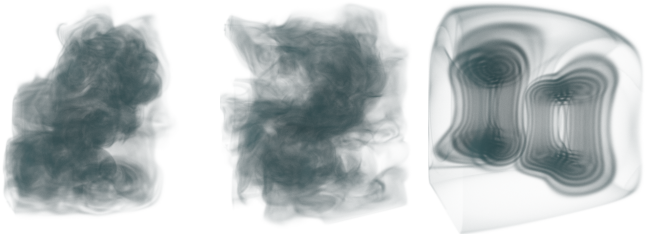


Fig. 8. From left to right, 0%, 92%, 100% of the smallest tensor entries are discarded in an $r = 7000$ simulation. At 100%, no energy is transferred between basis functions, which creates a static velocity field.

Real-time interaction. Even with a large basis rank, our method is fast enough to run interactively. Fig. 9 shows an interactive simulation with $r = 1000$ and two Neumann walls. The advection tensor is compressed by dropping 80% of the smallest entries, and the grid resolution is $120 \times 60 \times 60$. The example runs at 13 FPS.

Directable forward scattering. As described in §5.1, we can reweight the advection tensor to introduce directability into the phenomenon of forward scattering. In Fig. 10, we show the results of different forward scattering intensities using a various settings for c in the b_i function from Eqn. 22. When scattering is amplified, details emerge at higher frequencies much more quickly. In Fig. 11, we show that when a negative is introduced into the reweighting function, flows emerge that undergo ghostly reversals. The detailed motion can be viewed in the video.

8 DISCUSSION AND FUTURE WORK

We have described a version of the Eigenfluids algorithm that removes the memory limitations imposed by basis storage, generalizes to Neumann boundary conditions, and allows the use of symmetric solvers. We are able to improve the scalability of the original algorithm by over two orders of magnitude.

Since Eigenfluids can simulate totally inviscid flows, its energy characteristics can be visually distinct from more established methods. Particularly when the viscosity is low, the algorithm can exhibit

Scene	Grid Resolution	Boundary condition	Basis dimension	Tensor Contraction	Linear Solver	DCT/DST	Density Advection	Total
Paddle Wheel†	400 × 200 × 200	two Neumann	12600	4.2 secs	0.69 secs	1.2 secs	0.46 secs	6.6 secs
Colliding Smoke	220 × 220 × 220	two Neumann	7000	0.89 secs	0.42 secs	0.78 secs	0.48 secs	2.6 secs
		six Dirichlet	24000	8.9 secs	3.0 secs	0.78 secs	0.48 secs	13 secs
Cylinders	266 × 200 × 200	two Neumann	7000	0.92 secs	0.39 secs	0.85 secs	0.25 secs	2.3 secs
Interactive	120 × 60 × 60	two Neumann	1000	0.0070 secs	0.0050 secs	0.044 secs	0.020 secs	0.076 secs

Table 3. Timing breakdown of our algorithm across all the different examples. The tensor † uses single precision floating point, and the rest use double.

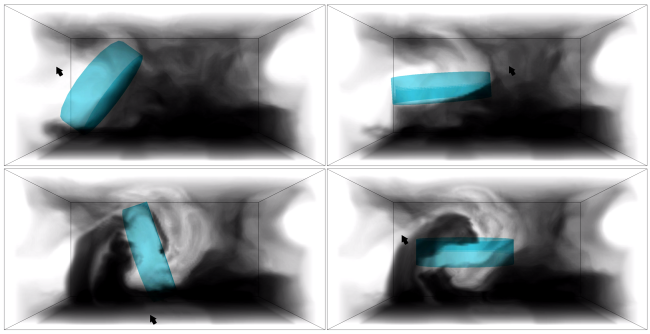


Fig. 9. Interactive example, with $r = 1000$ basis functions and two Neumann walls.

motions that can appear foreign to practitioners. While energy is conserved, its cascade is capped at a highest frequency. The energy dynamics of this case can be seen in the supplemental video, where in the totally inviscid regime, energy tends to spread evenly across the entire spectrum. The fact that we are able to capture these flows enables a look-development workflow where a user can start in the inviscid regime and gradually increases the viscosity until the desired look is achieved (see, e.g., Mullen et al. [2009])

At this point, a new memory bottleneck appears in the form of advection tensor storage. The most immediate direction for future work is to reduce the memory footprint of this tensor. This can be accomplished through brute-force compression methods [Hasan et al. 2008; Jones et al. 2016], or by discovering compact new structures in the tensor, such as Kronecker product formulations [Golub and Van Loan 2012].

The Eigenfluids method shares many similarities with spectral methods, so treating our results as a fast transform for a single element would allow the advection tensor to be re-used across multiple tiled domains. Mixed boundary conditions could then be achieved by varying the conditions across these tiles. Coupling methods [Wicke et al. 2009] still need to devised for such elements.

We have not yet explored the extension of Eigenfluids to include liquid surfaces. Although Long and Reinhard [2009] showed some preliminary results, the ability for the basis functions to resolve the velocity discontinuity across the interface is likely to be the limiting factor. In this respect, the pseudo-spectral approach of Heo and Ko [2010] offers interesting possibilities. For slip, perfectly-matched layer, or prescribed boundary conditions, additional constraints need to be considered when selecting basis functions. It remains to be seen if closed-form, FFT-friendly solutions continue to exist in the presence of these constraints.

Finally, our simulations are fast and memory-efficient because the simulation domain is limited to a rectangular box. This allows us to use DCT and DST libraries directly, but these operations cannot be directly applied to the unstructured tetrahedral meshes shown in other work [De Witt et al. 2012; Liu et al. 2015]. New transform methods will need to be devised before these irregular domains can achieve the same level of scalability. Wavelets and their associated transforms seem like a promising direction, as they would also allow degrees of freedom to be added to the regions that show the most spatial complexity.

ACKNOWLEDGEMENTS

TK would like to thank Aaron Demby-Jones, Fernando de Goes and Doug James for early discussions of this work. This work was supported by NSF CAREER award IIS-1253948, as well as IIS-1321168 and IIS-1619376. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We acknowledge support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-1720256).

A COMPUTING THE ADVECTION TENSOR

A.1 2D Dirichlet Basis

First we will show how to compute the advection tensor for 2D Dirichlet basis. Assume the simulation domain is defined as $\Omega = [0, \pi]^2$. Each entry in advection tensor $\mathfrak{C}$ has three associated wave indices, we will denote them as $\mathbf{k} = (k_x, k_y)$, $\mathbf{l} = (l_x, l_y)$, and $\mathbf{m} = (m_x, m_y)$. Basis functions with corresponding wave indices will be indexed with unbolded letters, i.e. $\Psi_k = \Psi(\mathbf{k}), \Psi_l = \Psi(\mathbf{l})$, $\Psi_m = \Psi(\mathbf{m})$. The normalized 2D Dirichlet basis functions ($\Psi(\mathbf{k}) = \{\Phi_x(\mathbf{k}), \Phi_y(\mathbf{k})\}$) are:

$$\begin{aligned} \Phi_x(\mathbf{k}) &= -\frac{2}{\pi}\frac{k_y}{\eta_\mathbf{k}} \sin(k_x x)\cos(k_y y) \\ \Phi_y(\mathbf{k}) &= \frac{2}{\pi}\frac{k_x}{\eta_\mathbf{k}} \cos(k_x x)\sin(k_y y) \end{aligned} \tag{26}$$

Where $\eta_\mathbf{k} = \sqrt{k_x^2 + k_y^2}$, and $\Psi(\mathbf{l})$ and $\Psi(\mathbf{m})$ have the same form as above but with $\mathbf{k}$ replaced with $\mathbf{l}$ and $\mathbf{m}$. We can compute $\nabla \times \Psi_m$ as:

$$\nabla \times \Psi_m = -\frac{2}{\pi}\eta_\mathbf{m} \sin(m_x x)\sin(m_y y) \tag{27}$$

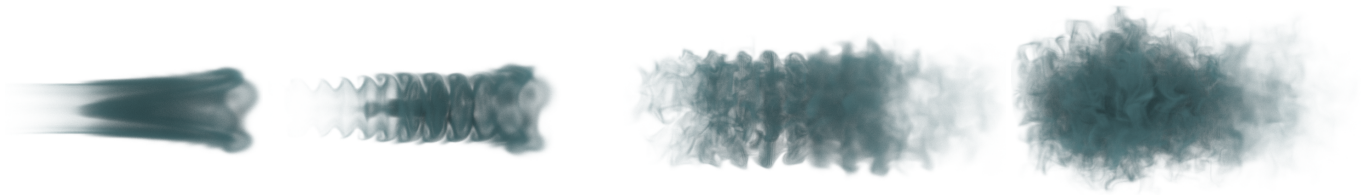


Fig. 10. We introduce a directability parameter c that reweights the advection tensor according to Eqn. 22 and uses the function $b_k = (1 + c|\mathbf{k}|^2)$. Each image in the sequence shows the same simulation timestep, but with a different setting for c. From left to right, the settings are $c = 0$ (i.e. the original, default tensor), $c = 0.0003$, $c = 0.0005$, and $c = 0.0017$. As c increases, energy cascades rapidly into high-frequency modes, and creates more turbulent flows.

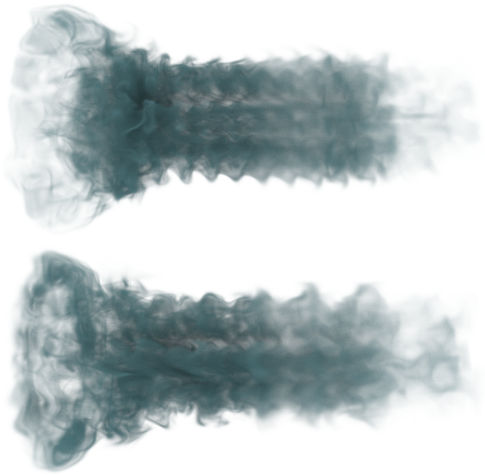


Fig. 11. Using different tensor reweighting schemes, e.g. $b_k = -(1 + c|\mathbf{k}|^2)$, a wider variety of flows are observed. Above, reweighted versions of the same simulation frame as Fig. 10 are shown.

The term $\mathbf{\Psi}_k \times \mathbf{\Psi}_l$ is equal to:

$$\begin{aligned}\mathbf{\Psi}_k \times \mathbf{\Psi}_l = &-\frac{1}{\pi^2 \eta_\mathbf{l} \eta_\mathbf{k}} l_x k_y (\sin((k_x + l_x)x) + sin((k_x - l_x)x)) \\ &(\sin((k_y + l_y)y) - \sin((k_y - l_y)y)) + \\ &\frac{1}{\pi^2 \eta_\mathbf{l} \eta_\mathbf{k}} l_y k_x (\sin((k_x + l_x)x) - \sin((k_x - l_x)x)) \\ &(\sin((k_y + l_y)y) + \sin((k_y - l_y)y))\end{aligned} \tag{28}$$

We then have

$$\begin{aligned}&\mathfrak{C}(k, l, m) = \frac{2\eta_\mathbf{m}}{\pi^3 \eta_\mathbf{l} \eta_\mathbf{k}} [\\ &l_x k_y \int_0^\pi \sin(m_x x)(\sin((k_x + l_x)x) + sin((k_x - l_x)x))dx \\ &\int_0^\pi \sin(m_y y)(\sin((k_y + l_y)y) - \sin((k_y - l_y)y))dy - \\ &l_y k_x \int_0^\pi \sin(m_x x)(\sin((k_x + l_x)x) - \sin((k_x - l_x)x))dx \\ &\int_0^\pi \sin(m_y y)(\sin((k_y + l_y)y) + \sin((k_y - l_y)y))dy]\end{aligned} \tag{29}$$

where the integral

$$\begin{aligned}&\int_0^\pi \sin(m_x x)(\sin((k_x + l_x)x)dx = \\ &\frac{1}{2}\int_0^\pi \cos((m_x - k_x - l_x)x) - \cos((m_x + k_x + l_x)x)dx\end{aligned} \tag{30}$$

is only non-zero when $m_x = k_x + l_x$. Similarly, $\int_0^\pi \sin(m_x x)(\sin((k_x - l_x)x)dx$ is non-zero only when $m_x = k_x - l_x$ or $m_x = l_x - k_x$. The same constraints can be derived for m_y, k_y and l_y. These integrals will determine the density of the advection tensor. Finally, as an example, when $m_x = k_x + l_x$ and $m_y = k_y + l_y$, the tensor entry becomes $\mathfrak{C}(k, l, m) = \frac{\eta_\mathbf{m}(l_x k_y - l_y k_x)}{2\pi \eta_\mathbf{l} \eta_\mathbf{k}}$

A.2 2D Neumann Basis

Next we show how to compute the advection tensor for a 2D Neumann basis. For the basis with two Neumann walls for Φ_x at $x = 0$ and $x = \pi$, the normalized basis is:

$$\begin{aligned}\Phi_x(\mathbf{k}) &= \frac{2}{\pi}\frac{k_y}{\eta_\mathbf{k}} \cos(k_x x) \cos(k_y y) \\ \Phi_y(\mathbf{k}) &= \frac{2}{\pi}\frac{k_x}{\eta_\mathbf{k}} \sin(k_x x) \sin(k_y y),\end{aligned} \tag{31}$$

We can compute $\nabla \times \mathbf{\Psi}_m$ as

$$\nabla \times \mathbf{\Psi}_m = \frac{2}{\pi}\eta_\mathbf{m} \cos(m_x x) \sin(m_y y) \tag{32}$$

The term $\mathbf{\Psi}_k \times \mathbf{\Psi}_l$ equals:

$$\begin{aligned}\mathbf{\Psi}_k \times \mathbf{\Psi}_l = &-\frac{1}{\pi^2 \eta_\mathbf{l} \eta_\mathbf{k}} l_y k_x (\sin((k_x + l_x)x) + sin((k_x - l_x)x)) \\ &(\sin((k_y + l_y)y) + \sin((k_y - l_y)y)) + \\ &\frac{1}{\pi^2 \eta_\mathbf{l} \eta_\mathbf{k}} l_x k_y (\sin((k_x + l_x)x) - \sin((k_x - l_x)x)) \\ &(\sin((k_y + l_y)y) - \sin((k_y - l_y)y))\end{aligned}$$

We then have

$$\mathbf{C}(k,l,m) = \frac{2\eta_{\mathbf{m}}}{\pi^3\eta_{\mathbf{l}}\eta_{\mathbf{k}}}[$$
$$- l_y k_x \int_0^\pi \cos(m_x x)(\sin((k_x + l_x)x) + sin((k_x - l_x)x))dx$$
$$\int_0^\pi \sin(m_y y)(\sin((k_y + l_y)y) + \sin((k_y - l_y)y))dy + \quad (33)$$
$$l_x k_y \int_0^\pi \cos(m_x x)(\sin((k_x + l_x)x) - \sin((k_x - l_x)x))dx$$
$$\int_0^\pi \sin(m_y y)(\sin((k_y + l_y)y) - \sin((k_y - l_y)y))dy]$$

The major difference between equation 33 and equation 29 is that the integrand along x direction in equation 33 is the product of cosine and sine functions. But the integrand along the y direction is still the product of sine and sine functions. Since the integral of sine functions over $[0, \pi]$ is non-zero for odd wavenumbers, the only necessary condition for equation 33 to be non-zero is $m_y = k_y + l_y, m_y = k_y - l_y$ or $m_y = l_y - k_y$. Thus, the Neumann basis tensor will be denser than the Dirichlet basis tensor.

REFERENCES

Steven S. An, Theodore Kim, and Doug L. James. 2008. Optimizing Cubature for Efficient Integration of Subspace Deformations. *ACM Trans. Graph.* 27, 5, Article 165 (Dec. 2008), 10 pages.

Alexis Angelidis. 2017. Personal Communication. (2017).

David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of SIGGRAPH*. 43–54.

John P Boyd. 2001. *Chebyshev and Fourier spectral methods*. Dover Publications.

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

Tyson Brochu, Todd Keeler, and Robert Bridson. 2012. Linear-time Smoke Animation with Vortex Sheet Meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 87–95.

K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, B. P. Brown, and E. Quataert. 2017. Dedalus project. http://dedalus-project.org. (2017).

CG Canuto, MY Hussaini, A Quarteroni, and TA Zang. 2007. *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*. Springer.

Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, and TA Zang. 1988. *Spectral methods in fluid dynamics*. Springer.

Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, and Steffen Weissmann. 2016. Schrödinger's Smoke. *ACM Trans. Graph.* 35, 4, Article 77 (July 2016), 13 pages.

Seth Davidovits and Nathaniel J Fisch. 2016. Sudden viscous dissipation of compressing turbulence. *Physical Review Letters* 116, 10 (2016).

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid Simulation Using Laplacian Eigenfunctions. *ACM Trans. Graph.* 31, 1, Article 10 (2012), 11 pages.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual Simulation of Smoke. In *Proceedings of SIGGRAPH*. 15–22.

Florian Ferstl, Rüdiger Westermann, and Christian Dick. 2014. Large-scale liquid simulation on adaptive hexahedral grids. *IEEE Transactions on Visualization and Computer Graphics* 20, 10 (2014), 1405–1417.

Matteo Frigo and Steven G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

Uriel Frisch. 1995. *Turbulence: The Legacy of AN Kolmogorov*. Cambridge University Press.

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

Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*. Vol. 3. JHU Press.

David Gottlieb and Steven A Orszag. 1977. *Numerical analysis of spectral methods: theory and applications*. SIAM.

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

Mohit Gupta and Srinivasa G. Narasimhan. 2007. Legendre Fluids: A Unified Framework for Analytic Reduced Space Modeling and Rendering of Participating Media. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 17–25.

Milos Hasan, Edgar Velazquez-Armendariz, Fabio Pellacini, and Kavita Bala. 2008. Tensor Clustering for Rendering Many-Light Animations. *Computer Graphics Forum* 27, 4 (2008), 1105–1114.

Ronald D. Henderson. 2012. Scalable Fluid Simulation in Linear Time on Shared Memory Multiprocessors. In *Proceedings of the Digital Production Symposium (DigiPro '12)*. 43–52.

Nambin Heo and Hyeong-Seok Ko. 2010. Detail-preserving fully-Eulerian Interface Tracking Framework. *ACM Trans. Graph.* 29, 6, Article 176 (Dec. 2010), 8 pages.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The Affine Particle-in-cell Method. *ACM Trans. Graph.* 34, 4, Article 51 (July 2015), 10 pages.

Aaron Demby Jones, Pradeep Sen, and Theodore Kim. 2016. Compressing Fluid Subspaces. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 77–84.

Theodore Kim. 2013. Zephyr. http://www.tkim.graphics/RESIM/source.html. (2013).

Theodore Kim and John Delaney. 2013. Subspace Fluid Re-simulation. *ACM Trans. Graph.* 32, 4, Article 62 (2013), 9 pages.

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 2008. Wavelet Turbulence for Fluid Simulation. *ACM Trans. Graph.* 27, 3, Article 50 (Aug. 2008), 6 pages.

Cornelius Lanczos. 1938. Trigonometric interpolation of empirical and analytical functions. *J. Math Phys.* 17, 1-4 (1938), 123–199.

Beibei Liu, Gemma Mason, Julian Hodgson, Yiying Tong, and Mathieu Desbrun. 2015. Model-reduced Variational Fluid Simulation. *ACM Trans. Graph.* 34, 6, Article 244 (2015), 12 pages.

Benjamin Long and Erik Reinhard. 2009. Real-time Fluid Simulation Using Discrete Sine/Cosine Transforms. In *Symposium on Interactive 3D Graphics and Games*. 99–106.

Frank Losasso, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating Water and Smoke with an Octree Data Structure. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 457–462.

A. McAdams, E. Sifakis, and J. Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 65–74.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. 2009. Energy-preserving Integrators for Fluid Animation. *ACM Trans. Graph.* 28, 3, Article 38 (2009), 8 pages.

Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. 2008. Fast Animation of Turbulence Using Energy Transport and Procedural Synthesis. *ACM Trans. Graph.* 27, 5, Article 166 (Dec. 2008), 8 pages.

Steven A Orszag. 1969. Numerical methods for the simulation of turbulence. *The Physics of Fluids* 12, 12 (1969), II–250.

Steven A Orszag. 1971. Accurate solution of the Orr–Sommerfeld stability equation. *Journal of Fluid Mechanics* 50, 4 (1971), 689–703.

Tobias Pfaff, Nils Thuerey, and Markus Gross. 2012. Lagrangian Vortex Sheets for Animating Fluids. *ACM Trans. Graph.* 31, 4, Article 112 (July 2012), 8 pages.

Yousef Saad. 2003. *Iterative methods for sparse linear systems*. SIAM.

H. Schechter and R. Bridson. 2008. Evolving Sub-grid Turbulence for Smoke Animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–7.

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

Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. 2005. A Vortex Particle Method for Smoke, Water and Explosions. *ACM Trans. Graph.* 24, 3 (July 2005), 910–914.

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A Sparse Paged Grid Structure Applied to Adaptive Smoke Simulation. *ACM Trans. Graph.* 33, 6, Article 205 (Nov. 2014), 12 pages.

Jos Stam. 1999. Stable Fluids. In *Proceedings of SIGGRAPH*. 121–128.

Jos Stam. 2002. A Simple Fluid Solver Based on the FFT. *J. Graph. Tools* 6, 2 (Sept. 2002), 43–52.

Matt Stanton, Yu Sheng, Martin Wicke, Federico Perazzi, Amos Yuen, Srinivasa Narasimhan, and Adrien Treuille. 2013. Non-polynomial Galerkin Projection on Deforming Meshes. *ACM Trans. Graph.* 32, 4, Article 86 (July 2013), 14 pages.

Lloyd N Trefethen. 2000. *Spectral methods in MATLAB*. SIAM.

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model Reduction for Real-time Fluids. *ACM Trans. Graph.* 25, 3 (July 2006), 826–834.

Geoffrey M. Vasil, Keaton J. Burns, Daniel Lecoanet, Sheehan Olver, Benjamin P. Brown, and Jeffrey S. Oishi. 2016. Tensor calculus in polar coordinates using Jacobi polynomials. *J. Comput. Phys.* 325 (2016), 53 – 73.

Steffen Weissmann and Ulrich Pinkall. 2010. Filament-based Smoke with Vortex Shedding and Variational Reconnection. *ACM Trans. Graph.* 29, 4, Article 115 (July 2010), 12 pages.

Martin Wicke, Matt Stanton, and Adrien Treuille. 2009. Modular Bases for Fluid Dynamics. *ACM Trans. Graph.* 28, 3, Article 39 (July 2009), 8 pages.

Xinxin Zhang, Robert Bridson, and Chen Greif. 2015. Restoring the Missing Vorticity in Advection-projection Fluid Solvers. *ACM Trans. Graph.* 34, 4, Article 52 (July 2015), 8 pages.

Yongning Zhu and Robert Bridson. 2005. Animating Sand As a Fluid. *ACM Trans. Graph.* 24, 3 (July 2005), 965–972.