University of California
Santa Barbara

Stable, Scalable, Reduced-Order Methods for Physical Simulation

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Media Arts and Technology

by

Qiaodong Cui

September 2020

The Dissertation of Qiaodong Cui is approved.

Professor Yon Visell

Professor Pradeep Sen, Committee Co-Chair

Professor Theodore Kim, Committee Co-Chair

July 2020

Stable, Scalable, Reduced-Order Methods for Physical Simulation

Acknowledgements

First of all, I would like to thanks my advisor, Professor Theodore Kim. I was inspired by his Wavelet Turbulence when I was doing an undergraduate project. That was when I decided to pursue a Ph.D. along this line of work. Even though we were physically separated during my study, he was always very patient to guide me. I can never forget his detailed email comment on my SIGGRAPH talks. This dissertation would be impossible without him.

I would also like to thanks my co-advisor, Professor Pradeep Sen. Working with him really broadened my knowledge. With his help, I was able to know there are many interesting topics in machine learning, computer vision, and computer graphics. I am also grateful to my mentor at Adobe, Dr. Timothy Langlois. I had a wonderful summer internship at Adobe Research. I would also like to thank my committee member, Professor Yon Visell.

I would also like to thanks the Media Arts and Technology community, which gave me both technical and creative inspiration. My thanks to my roommate, Weihao Qiu, for teaching me to drive and exchanging cooking skills. My thanks to my lab-mate, Yuxiang Wang, and Chieh-Chi Kao for helping me to find internships.

Finally, I would like to thank my parents for their endless support and encouragement.

Curriculum Vitæ
Qiaodong Cui

Education

2020	Ph.D. in Media Arts and Technology (focusing on computer graphics), University of California, Santa Barbara.
2015	B.S. in Computer Science and Technology, Beihang University

Experience

- **Research Assistant at UCSB** Sept.2016 to Current
  - Developed a reduced model fluid simulation method which is fast, memory efficient and scalable with analytical basis functions. Presented at **SIGGRAPH** 2018.
  - Researching fluid simulation with analytical basis functions in polar and spherical coordinate systems.
  - Explored upsampling of particle based liquid simulation(FLIP) with point based neural networks.
  - Explored acceleration of structure analysis/optimization with machine learning.
- **Research Intern at Adobe** June 2018 to Sept 2018
  - Developed a fast and robust stochastic structural optimization method. Submitted to Eurographics 2020.
  - Explored acceleration of failure analysis for 3D printing using neural networks.
- **Research Intern at DreamWorks Animation** June 2017 to Sept 2017
  - Implemented a method to quickly generate Poisson disk distribution on a surface mesh.
  - Implemented a method to smooth out the point distribution based on smoothed particle hydrodynamics simulation.
- **Research Assistant at UCSB** June. 2016 to Nov. 2016
  - Developed a fast image graph matching algorithm for large scale structure from motion. It offers over 4X speedup over state of art vocabulary tree methods. Accepted to 3DV 2017 as **oral**.
- **Research Assistant at Beihang University** Apr. 2014 to July. 2014
  - Developed a dynamic flame animation method. Accepted to Eurographics 2015.
  - Implemented a photorealistic flame rendering algorithm in PBRT.

Publications

Qiaodong Cui, Timothy Langlois, Pradeep Sen, and Theodore Kim, *"Fast and robust stochastic structural optimization"*. Computer Graphics Forum (Eurographics) 2020..

Qiaodong Cui, Pradeep Sen, Theodore Kim, *"Scalable Laplacian Eigenfluids"*, ACM Transactions on Graphics (SIGGRAPH 2018).

Qiaodong Cui, Victor Fragoso, Chris Sweeney and Pradeep Sen, *"GraphMatch: Efficient Large-Scale Graph Construction for Structure from Motion"*, in Proceedings of the IEEE International Conference on 3D Vision. IEEE, 2017. **Oral**

Qiaodong Cui, Zhaohui Wu, Zhong Zhou and Wei Wu, *"Target temperature driven dynamic flame animation"*, in Eurographics (Short Papers), 2015.

Yi Li, **Qiaodong Cui**, Fei Dou and Zhong Zhou, *"Automatic mesh animation preview"*, in IEEE International Conference on Multimedia and Expo (ICME), 2014.

Technical Skills

- **Skills:** fluid dynamics, finite elements analysis, machine learning, topology optimization, 3D reconstruction, photorealistic rendering.
- **Programming**: C/C++, Python, Tensorflow, OpenGL.
- **Tools**: Matlab, Git, CMake, Linux, Visual Studio, Illustrator, Premiere.

Abstract

Stable, Scalable, Reduced-Order Methods for Physical Simulation

by

Qiaodong Cui

Reduced-order methods are an attractive model for physical simulation in computer graphics. They aim to reduce the computational cost of a full-rank simulation by projecting onto a reduced set of bases. However, an improper application of such methods can be unstable and scale poorly, which prevents many methods from being used in practice.

In this work, we aim to improve the scalability and stability of reduced order methods for two such scenarios. For the Eigenfluids method of fluid simulation, we show that by carefully applying the discrete sine and cosine transforms, the prohibitive scaling of its memory usage can be reduced asymptotically. The simulation is further stabilized by using a variational approach with different basis functions. We also show that the basis functions can be made quite general by extending them to spherical and polar coordinate systems. By using an orthogonalization method, we show a wide variety of basis functions can be designed that maintain fast transformations. For stochastic structure optimization, which assesses whether a fabricated object will break under real-world conditions, we show that the computation can be made asymptotically faster by carefully exploiting certain tensor structures. We then stabilize the optimization by applying an alternate basis to the function gradients.

Contents

Chapter 1

Introduction

Note: A significant portion of this chapter has previously appeared as [1, 2].

With the increasing computing capabilities of modern computers, physical simulations have become a powerful tool in different areas. For example, in the film and game industries, physical simulations create a realistic yet virtual experience that's impossible to capture in the real world. In manufacturing, it helps to assess whether an object will fail in a virtual setting, therefore validating design choices and reducing costs. While much progress has been made in physical simulation over the last decade, efficient high-quality physical simulation remains an ongoing challenge.

Recently, model reduced methods have proven to be an attractive candidate to achieve efficient high-quality simulations. Reduced methods apply to a wide range of problems and can lead to orders of magnitude speed-ups. This, however, comes at a cost: bases matrices used by reduced order methods pose a serious memory bottleneck, and reduced dynamics are unstable unless carefully implemented. In this dissertation, I focus on two specific reduced-order simulations: fluid simulation using Laplacian eigenfunctions and stochastic structure optimizations.

1.1 Fluid Simulation

Many astonishing natural phenomena are governed by the motion of fluids. For example, they range from common life experiences like smoke swirling off a cigar and water splashing in a swimming pool, microscopic phenomena like blood cells being carried around in plasma, to macroscopic phenomena like ocean currents and planetary flows. Due to their ubiquitous presence in our lives, the study of fluid phenomena is of prominent importance in physics, mechanical engineering, weather prediction, and many other fields. In physics, there is a consensus that the Navier-Stokes equations are a good model to describe the motion of fluids. The equation is derived by applying Newton's second law to a continuum fluid body. Despite its importance, analytical solutions to this equation are impossible except in some special cases. The computational difficulties precluded numerical solutions until the invention of digital computers.

The earliest effort towards a numerical approach was introduced by Lewis Fry Richardson [3] in 1922. More than 20 years later, John Von Neumann and his colleagues began to use the ENIAC to compute weather using an approach similar to Lewis's book [4]. With the increasing computation power of computers, these earliest efforts evolved into a completely new field: computational fluid dynamics (CFD). In computer graphics, digitally capturing fluid-like motion occured much later when computers became an important tool in filmmaking.

In contrast to CFD, where physical accuracy is most important, fluid simulation in computer graphics is primarily governed by visual effects. The earliest work to capture fluid-like motion in computer graphics was not physically accurate. For example, particle systems were used to capture the fire effect in the film: *StarTrek II: The Wrath of Khan* [5]. Random turbulence was later developed to add more visual details with better physical accuracy [6]. Physically-based models were first applied in two dimensions. For example, Yaeger et al. [7] simulated a two-dimension planetary flow in film production, Kass and Miller [8] simulated liquids using shallow water equations. Three-dimensional fluid simulation in computer graphics was first shown by Foster and Metaxas [9] with clear visual advantages. However, the method is based on finite difference and explicit time solvers, so it became unstable unless a small time step was used.

The seminal work by Stam [10] presented the first work that is unconditionally stable in computer graphics, and thus become a standard approach of fluid simulation in computer graphics. However, the stabilities come at a price of uncontrollable numerical dissipation, where the fluid appears more viscous than intended. This prevents many visually interesting details from appearing, and later work sought to mitigate the numerical dissipation. Many works seek to compensate for the dissipated energy using techniques such as vorticity confinement [11] or IVOCK [12]. Alternatively, methods can be devised that are non-dissipative by construction, though these can involve asymmetric linear solves and non-linear Newton iterations [13].

The method of Laplacian Eigenfunctions [14], 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 [15].

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 [14] and 230 for [15]). 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 [16]–a potential solution was proposed in De Witt et al. [14]: 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 $\sim$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 [17, 18, 19, 20], so we perform an in-depth comparison in §4.3 that shows that our algorithm is capable of achieving significantly lower viscosities.

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.

While fluid simulations in rectangular domains are very common, certain types of fluid simulation may be more suitable for other coordinate systems. For example, simulation of planetary flow and soap bubbles is more suited to spherical coordinates. For this reason, simulation of fluid in non-Cartesian coordinates has also attracted attention [21, 22, 23, 24]. While representing analytical basis functions using fast transformations is attractive, they are constrained to a rectangular domain. I will show that analytical basis functions that support fast transformations can be found for polar and spherical coordinates, and therefore expand the potential applications of the Eigenfluids algorithm.

1.2 Stochastic Structure Optimization

Recently, a wide range of computational techniques have been developed to assist in the design of objects manufactured using additive fabrication (see, e.g., excellent surveys of the state-of-the-art [25, 26]). One key concern is the robustness of the manufactured object (i.e., the conditions in which it will break), and various *failure analysis* algorithms have been developed to estimate the failure cases of the object in order to improve its design.

Traditionally, failure analysis is performed for worst-case scenarios, and while such analyses are critical in certain scenarios (e.g., bridge construction), they can be overly conservative in others, such as figurine design. In these (more common) situations, it is more realistic to ask whether an object is robust in real-world situations, such as being dropped on the floor or down a flight of stairs. In this case, reinforcing portions of the object that are unlikely to experience an impact because they are already shielded by more robust regions is clearly sub-optimal. Unlikely mechanical scenarios should not artificially constrain the design space, and the manufactured designs should be optimized for the "common case."

Along these lines, Langlois et al. [27] recently proposed a *stochastic structural analysis* method that used a rigid body simulator to sample a more realistic space of real-world impacts and construct a probability map of predicted failures. This probability map was then incorporated as a constraint into a topology optimization, and used to solve a context-aware inverse design problem that produced geometrical models more robust to realistic impacts. Unfortunately, the method is quite computationally intensive, where even *a single optimization step* over a small $26 \times 34 \times 28$ model takes over an *hour* to compute .

In this dissertation, we observe that this computational expense arises for two main reasons. First, computing the probability gradients is quadratic in the number of elements. Second, the inertia gradient that arises from the rigid body simulation is computed using a finite difference scheme that leads to instabilities, which in turn negatively impacts the convergence of the optimization. We present an approach that is both asymptotically faster and more numerically robust than Langlois et al. [27]. We achieve both of these goals directly; no approximation is applied to the original algorithm. In the first case, we show that a careful analysis of the probability gradient can yield a computation that is only *linear* in the number of elements. In the second case, we use a combination of Gaussian Mixture Models and an alternate central-differencing method to arrive at a more stable version of the inertia gradient.

Even with these improvements, the optimization can still stall at local minima. We therefore introduce a *constrained restart* method that is able to make further progress and discover interesting new structures that were otherwise inaccessible to the original algorithm. Finally, we show that additional progress can be made by assuming that a thin sheath that is beyond the resolution of the numerical simulation will be printed along the exterior of the object.

Together, these components comprise a stochastic structural optimization method that is orders of magnitude faster than Langlois et al. [27], able to achieve previously-impractical resolutions, and capable of discovering previously-inaccessible designs.

1.3 Thesis Statement and Main Results

My thesis statement is the following:

Stable, scalable, reduced-order simulations can be obtained by carefully analyzing the underlying basis functions and physical systems.

To support this thesis, I developed stable, scalable, reduced-order algorithms for two different kinds of simulations: fluid simulation and structure optimization. In the first case, I will show that the bases used by the Laplacian Eigenfluids algorithm can be effectively represented by fast transformations, which removes the basis storage problem and leads to a more scalable version of the method. The method can be stabilized with a variational approach. In the second case, I will show an asymptotically faster stochastic structure optimization can be achieved by exploiting certain tensor structures. The method is then stabilized by using smoother basis functions to capture function gradients.

Chapters 3 and 4 introduce a scalable reduced-order fluid simulation algorithm based on Laplacian eigenfunctions. The contributions are:

- 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$^{\text{rd}}$-order advection tensor.
- Demonstration of support for real-time interaction.
- Lower viscosity flows than equivalent spectral methods.

Chapter 5 shows an extension of the scalable Laplacian Eigenfluids algorithm to polar and spherical coordinates. The contributions are:

- Generalized 2D analytical basis functions in polar coordinates and on the surface of a sphere.

- Generalized 3D analytical basis functions in spherical coordinates.
- An orthogonalization method that allows non-orthogonal basis functions to be used.

Chapter 6-7 shows a fast and robust version of the stochastic structure optimization method. The contributions are:

- Asymptotically faster computation of probability gradients
- A robust scheme for computing inertia gradients that stabilizes the optimization search direction
- A constrained restart strategy that allows global optimization to climb out of local minima
- A sheathing approach that robustly removes additional mass from the final design

1.4 Mathematical Notation

I 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., $\langle \mathbf{u}, \boldsymbol{\Psi} \rangle = \int_\Omega \mathbf{u} \cdot \boldsymbol{\Psi} d\Omega$.

The 3rd-order advection tensor $\mathfrak{C} \in \mathbb{R}^{r \times r \times r}$ 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 [28]. This yields a matrix, e.g., $\mathfrak{C} \times_3 \mathbf{w} = \mathbf{C}$, 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., $\mathbf{w}^T \mathfrak{C} \mathbf{w} = \mathbf{x}$ where $\mathbf{x} \in \mathbb{R}^r$.

1.5 Organization

The dissertation is organized as follows: Chapter 2 provides related work on physically-based fluid simulation and structure optimization. In chapter 3, I introduce one particular reduced-order fluid simulation: Laplacian Eigenfluids. In chapter 4, I describe several improvements to the Eigenfluids algorithm, which makes it scalable and stable. Chapter 5 provides an extension of Eigenfluids to polar and spherical coordinate systems. In chapter 6, background on stochastic structure optimization is provided. Finally, chapter 7 describes several contributions that lead to a more fast and robust stochastic structure optimization method.

Chapter 2

Background and Related Work

Note: A significant portion of this chapter has previously appeared as [1, 2].

2.1 Physical Simulation of Fluids

Before introducing the physical simulation of fluids, I will introduce the basic physical equation of the fluid phenomena. In physics, when the change of volume of the fluid under consideration is very small and can be ignored, it is considered as an incompressible. This greatly simplifies the physical phenomena. Assuming the density of the fluid is homogeneous, the fluid can be described by the incompressible Navier-Stokes (N-S) equations 2.1:

$$\begin{aligned} \dot{\mathbf{u}} &= -\mathbf{u} \cdot \nabla \mathbf{u} + \nu \nabla^2 \mathbf{u} - \nabla p + \mathbf{f} \\ \nabla \cdot \mathbf{u} &= 0. \end{aligned} \tag{2.1}$$

Here, terms $\nu, p, \mathbf{f}$ denote viscosity, pressure, and external forces. The term $-\mathbf{u} \cdot \nabla \mathbf{u}$ describes how the velocity is advected by the velocity field itself, therefore it is referred to as the advection term. The term $\nu\nabla^2\mathbf{u}$ describes how the velocity expands and diffuses through the media, therefore it is referred to as the diffusion term. The term $-\nabla p$ describes how internal pressure affects the velocity of the fluid. The pressure forms internally so that the fluid is incompressible. This implies the velocity field of the fluid is divergence-free: $\nabla \cdot \mathbf{u} = 0$. Finally, the term $\mathbf{f}$ describes external forces like gravity, buoyancy, or user interactions that can be applied to the fluid.

One key task of fluid simulation is to track the motion of the fluid. There are two major approaches: the Lagrangian approach and the Eulerian approach. In the Lagrangian approach, fluids are discretized by a particle system. Usually, one particle has at least three attributes: mass, position, and velocity. Each particle describes the velocity of the fluid near that particle. One way to solve the motion of the fluid is to move the particle around according to its velocity. The external forces like gravity and pressure will be computed for each particle, and the velocity of the particle is updated accordingly. Eulerian approach, on the other hand, discretizes the fluid by a static mesh. Each element of the mesh describes the quantities of the fluid at a fixed position, for example, velocity and density. The motion of the fluid is solved by evolving the fluid quantities stored at each element. In the Lagrangian approach, the advection of the fluid is trivially solved by advecting the particles. However, spatial derivatives are not straightforward, and the divergence-free condition is hard to maintain. In the Eulerian approach, spatial derivatives are easier to compute. For example, the finite difference can be used, although a careful treatment for advection is required for a robust method. In this dissertation, I will mostly focus more on the Eulerian approach.

2.1.1 Eulerian Simulation of Fluids

To simulate the fluid, continuous quantities like velocity $\mathbf{u}$, pressure p and force $\mathbf{f}$ in equation 2.1 are discretized. The most frequently used discretization method is a uniform grid as shown in figure 2.1.

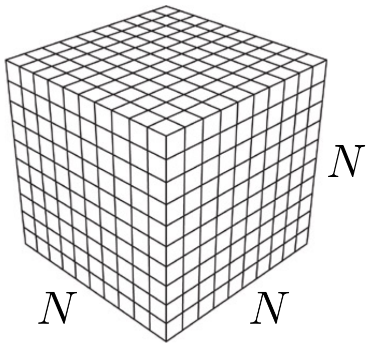


Figure 2.1: A typical discretization grid for 3D fluid simulation. Note the degree of freedom grows cubically.

The finite difference method [29] is most frequently used to discretize the N-S equation on a uniform grid due to its simplicity. The velocity of the fluid is the primary variable to solve. The task for fluid simulation is the following: Given the velocity field $\mathbf{u}^t$ at a time step t, solve the velocity at the next time-step $\mathbf{u}^{t+1}$.

The finite difference method has a long history of applications in computational fluid dynamics, one of the earliest works is introduced by Harlow and Welch [30], where they simulated a 2D incompressible fluid with the finite difference method and the forward explicit time-stepping. Following this work, Foster and Metaxas [9] introduced simulations of smoke with 3D N-S equations into computer graphics. However, the method uses explicit time stepping, where the time-step is constrained. Larger time steps may lead to instabilities, where the simulation "blows up" and has to restart with a smaller time-step. This greatly limits the method. Stam [10] introduced the first fluid simulation method which is unconditionally stable, regardless of the time-step. The method was widely adopted as a result.

Here I briefly introduce the work of Stam [10]. In this work, a splitting approach is used to solve the N-S equations in 2.1. As shown in figure 2.2, four sub-steps are involved to solve the velocity $\mathbf{u}$ at the next time step. Each sub-step solves one term of the N-S equation.

$$\begin{tikzcd} \mathbf{u}^{t} \arrow[r, "\text{add force}"] & \mathbf{u}^{t_1} \arrow[r, "\text{advect}"] & \mathbf{u}^{t_2} \arrow[r, "\text{diffuse}"] & \mathbf{u}^{t_3} \arrow[r, "\text{project}"] & \mathbf{u}^{t+1} \end{tikzcd}$$

Figure 2.2: A splitting approach to solve the N-S equation.

Add force: To add force to the simulation, the force weighted by time step is simply added to the velocity field:

$$\mathbf{u}^{t_1} = \mathbf{u}^{t} + \mathbf{f}\Delta t. \tag{2.2}$$

Advect: Stam introduced the classical semi-Lagrangian method to solve the advection term. This method can be described by:

$$\mathbf{u}^{t_2}(\mathbf{x}) = \mathbf{u}^{t_1}(\mathbf{x} - \mathbf{u}^{t_1}(\mathbf{x})\Delta t). \tag{2.3}$$

To evaluate the velocity $\mathbf{u}^{t_2}$ at a location $\mathbf{x}$, this method back-tracks the point $\mathbf{x}$ using the velocity at that point in sub-step t_1, the location is then evaluated as $\mathbf{x} - \mathbf{u}^{t_1}(\mathbf{x})\Delta t$. The velocity at this point is then interpolated and assigned to the velocity of the point $\mathbf{x}$ at the next sub-step t_2. Linear interpolations are usually used. This method is unconditionally stable because $\max(\mathbf{u}^{t_2}) \leq \max(\mathbf{u}^{t_1})$ introduced by the interpolation. However, the averaging caused by the interpolation introduces numerical dissipation, where sharp features are dissipated over time.

Diffuse: The diffusion term $\nu\nabla^2\mathbf{u}$ introduces a Poisson equation, which can be solved to obtain $\mathbf{u}^{t_3}$:

$$(I - \nu\Delta t\nabla^2)\mathbf{u}^{t_3} = \mathbf{u}^{t_2}. \tag{2.4}$$

After discretization, operator $(I - \nu\Delta t\nabla^2)$ becomes a sparse matrix. Iterative methods like conjugate gradient [28] are then used to solve the sparse linear system. In most fluid simulations, due to the numerical dissipation introduced by the advection step, viscosity ν is usually set to zero. This step is entirely skipped.

Project: Finally, the velocity $\mathbf{u}^{t_3}$ may not be a divergence-free velocity field. Therefore, it is projected to a divergence-free vector field. The Helmholtz decomposition states that any vector field can be decomposed into a divergence-free component and the gradient of a scalar field p:

$$\mathbf{u}^{t+1} = \mathbf{u}^{t_3} - \nabla p. \tag{2.5}$$

The term p is a pressure field because it enforces the velocity field to be divergence-free. The pressure p can be solved via a Poisson system:

$$\nabla^2 p = \nabla \mathbf{u}^{t_3}. \tag{2.6}$$

The gradient of the pressure is then subtracted from $\mathbf{u}^{t_3}$ to obtain the velocity at the next time step. The Poisson system requires solving a large sparse linear system. Assuming the viscosity step is skipped, this usually becomes the bottleneck of the algorithm.

The method is unconditionally stable; therefore, it is the first widely adopted method in computer graphics. However, both the pressure projection and the semi-Lagrangian advection schemes introduce numerical dissipations that many subsequent works have sought to mitigate.

2.1.2 Full Rank Methods

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 [31], multigrid hierarchies [32, 33], and sparsely paged grids [34].

Methods for re-injecting dissipated energy have also been explored, such as vorticity confinement [11], vortex particles [35], IVOCK [12], and turbulence methods that are applied as a post-process [36, 37, 38]. The structure-preserving properties of Lagrangian methods have also been leveraged in the form of vortex filament [39] [40], APIC [41], and PPIC [42] methods have also been developed.

Mullen et al. [13] 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 [43] algorithm also exhibits inviscid behavior, but it does not contain a viscosity parameter.

For most full rank methods, the degree of freedom grows as the resolution increases. This in turn requires more time to solve for the unknown variables. Therefore, fast full rank simulations of fluid are inherently difficult.

2.1.3 Reduced-Order Methods

Reduced-order methods mitigate the computational cost of the simulation by reducing the degree of freedom of the physical system. To do that, a set of basis functions (bases) are chosen for the vector of variables in the simulation. They are chosen in a way that a small number of basis functions can effectively represent the state space of the simulation.

This is similar to describing a signal in frequency space, where Fourier series are used as the basis function. The linear space composed by the basis functions is then called the subspace. If the entire physical system can be projected down to the subspace via the basis functions, the simulation can be entirely be performed in the subspace. This usually leads to orders of magnitude of speed-ups compared to full rank simulations.

Reduced-order methods were first introduced into computer graphics by Pentland and Williams [44] to simulate solids. Treuille et al. [45] first introduced the reduced-order simulation of fluid in computer graphics. The method however is not consistent with the full rank solver, and Kim and Delaney [46] introduced subspace fluid re-simulation, which closely matches the full rank simulation. In both methods, the bases matrix is numerically constructed from a set of velocity fields from exemplar simulations. Each column of the bases matrix is a principle component obtained via the principal component analysis (PCA) on the exemplar simulations. A full rank velocity field can then be projected to the subspace by the bases matrix, which I will refer to as the reduced coordinates. This is shown in figure 2.3.

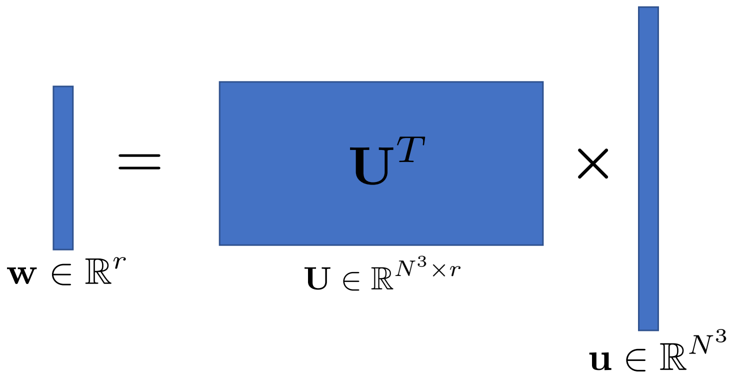

$=$
$\mathbf{U}^T$
$\times$
$\mathbf{w} \in \mathbb{R}^r$
$\mathbf{U} \in \mathbb{R}^{N^3 \times r}$
$\mathbf{u} \in \mathbb{R}^{N^3}$

Figure 2.3: Projection of a full rank velocity into subspace via a basis matrix. The term $\mathbf{w}$ is reduced coordinates. The term $\mathbf{U}$ is a bases matrix. The term $\mathbf{u}$ is a full rank velocity field.

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 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 [47] and JPEG-like compression [48], but the issue is far from resolved.

The method of Laplacian Eigenfunctions, which we refer to as *Eigenfluids*, was developed by De Witt et al. [14], and further stabilized using variational methods by Liu et al. [15]. While I will describe the method in detail later (§3), I 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 [45, 46, 49]. 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 et al. [50] also performed reduced fluid simulations in an analytic (Legendre) space, but traded spatial resolution for rendering efficiency.

2.1.4 Spectral Methods

This work makes extensive use of discrete sine (DST) and cosine (DCT) transforms, and thus shares connections with a variety of spectral fluid solvers. Stam [10, 51] 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 [52] showed that this approach could be extended to Dirichlet boundaries by using the DST and DCT, and Henderson [53] showed that it scales favorably over multiple processors. I 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 [54], and gained wider attention in fluid mechanics during the 1970s due to the work of [55, 56]. Many excellent texts exist that describe these methods Orszag [18, 19, 57], but relative to our current method, the use of Chebyshev and Legendre polynomials to handle non-periodic boundary conditions is the most relevant feature [17, 20]. I will perform an extensive comparison of our own method against a modern pseudo-spectral library [58] in §4.3, and show that our method can simulate flows with significantly lower viscosity.

2.1.5 Fluid Simulation in Non-Cartesian Coordinates

In most cases, the fluid is parameterized in 2D or 3D Cartesian coordinates, where the domain is usually rectangular. However, some fluid phenomena may better be parameterized with other coordinate systems. For example, spherical coordinates may better be suited to describe the planetary flow, where the fluid is constrained on the surface of a sphere.

Stam [21] introduced fluid simulation on Catmull-Clark surfaces of arbitrary topologies. The Stable Fluids method [10] is extended to arbitrary curvilinear coordinates and then mapped to the surface of the mesh. Following this work, [22] simulated surface waves on a character mesh. Thürey et al. [59] simulates wave details on the fluid surface to capture the surface tension effects. Kim et al. [60] upsampled the resolution of a liquid simulation by directly simulating waves on the liquid surface. Recently Hill and Henderson [23] introduced an efficient fluid simulation method on a sphere surface, where the geometry terms omitted by previous works are addressed. Yang et al. [24] extended this work using GPUs, achieving real-time performance. In this work, I will extend the

Eigenfluid algorithm [1] to polar coordinates and spherical coordinates.

2.2 Structure Optimization

Structural optimization, also referred as topology optimization, is a method that optimizes the material distribution of a shape given a set of loadings, boundary conditions, and objective and constraint functions. Since structural optimization was introduced by Bendsøe et al. [61, 62], it has attracted lots of attention in computer graphics. Traditionally, topology optimization involves the minimization of compliance, which I will briefly introduce here.

2.2.1 Compliance Optimization

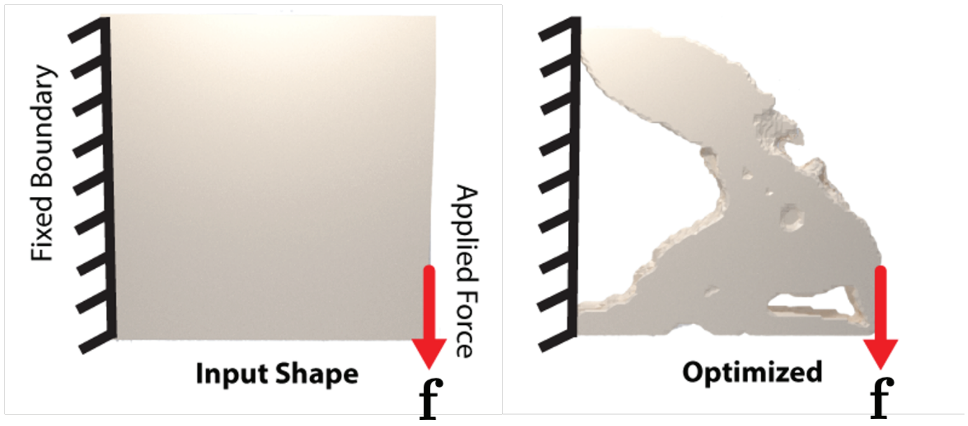


Figure 2.4: Left: The initial slab object that needs to be optimized. Right: The optimized slab object under the applied force and the boundary condition.

Figure 2.4 shows a typical setting of structure optimization. On the left is the initial condition of the optimization, where the slab is fully filled. The shape is typically parameterized by a finite element mesh. A density value is assigned to each mesh element, which is the primary variable for the optimization. Concatenate all density variables into a vector and denote it as $\boldsymbol{\omega}$.

Most materials used in structure optimization are stiff, for example, ABS plastics and metals. The material breaks when the deformation is very small, therefore, linear elasticity models are very frequently used. The nodal displacements ($\mathbf{u}$) of the finite element mesh can be computed as:

$$\mathbf{u} = \mathbf{K}^{-1}\mathbf{f}. \tag{2.7}$$

The term $\mathbf{K}$ denotes the stiffness matrix. The term $\mathbf{f}$ denotes the external force. One simplest objective function to minimize is the compliance: $\mathbf{u}^T\mathbf{K}\mathbf{u}$. Minimizing the compliance corresponds to maximizing the global stiffness [62], because the compliance is equal to $\mathbf{f}^T\mathbf{K}^{-T}\mathbf{f}$, assuming $\mathbf{f}$ is fixed during the optimization. The optimization can be written as:

$$\begin{aligned} &\min \mathbf{u}^T\mathbf{K}\mathbf{u} \\ &\text{s.t. } \mathbf{u} = \mathbf{K}^{-1}\mathbf{f}. \end{aligned} \tag{2.8}$$

Many different methods can be used to solve the above optimization problem. For example, the Optimality Criteria [63] method, the Sequential Linear Programming method or, the Method of Moving Asymptotes (MMA) [64]. All these methods are iterative in nature. I will focus on the MMA method here.

The MMA method can solve nonlinear optimizations with inequality constraints, it is a trust region method where information around the current optimization point is used to construct a convex approximation model at that point. The convex model is then solved to find the next iteration point. For each iteration, the value and gradient of both objective and constraint functions are evaluated externally, and then the information is supplied to the MMA optimizer to compute the next iteration point. For example, for the optimization problem in equation 2.8, given current iteration point $\boldsymbol{\omega}^t$, stiffness matrix is then evaluated at this point. Next, the equality constrain can be satisfied by evaluating $\mathbf{u}^t = \mathbf{K}^{-1}\mathbf{f}$. The value of compliance $(\mathbf{u}^t)^T\mathbf{K}\mathbf{u}^t$ and its derivative $(\mathbf{u}^t)^T\frac{\partial \mathbf{K}}{\partial \boldsymbol{\omega}}\mathbf{u}^t$ are then evaluated and supplied to the MMA optimizer.

Due to the simplicity of using compliance as the objective function, There are many works in computer graphics following this approach, e.g., [65, 66, 67]. However, compliance minimization can overfit to one loading condition and has difficulty predicting whether objects will fail under realistic conditions.

2.2.2 Other Formulations

Instead of minimizing compliance, minimizing the object's weight subject to a stress constraint offers a better guarantee of the robustness of the object. For example, Lee et al. [68] uses a constraint on the yield stress and minimizes the weight of the object. Similarly, Ulu et al. [69] optimizes the thickness of the shell to minimize the weight, subject to a constraint on the yield stress. Many of these methods use prescribed loadings. This can be useful in some particular cases, e.g., designing a bridge, where the loading can be prescribed in advance. However, in many other cases, the loading might be unknown beforehand.

To capture uncertainty, stochastic finite element analysis has been extensively studied in the engineering community. Stefanou [70] provides an excellent overview. There are two broad classes of methods: 1) the perturbation approach, which uses a Taylor series expansion of the system matrix and solution [71], and 2) the spectral stochastic finite element method, which represents each solution quantity with a series of random Hermite polynomials [72]. Monte Carlo simulation (MCS) [73] can be used in conjunction with these two methods, which models randomness by solving a deterministic problem many times using different samples of the random variables. These methods are very general, and can consider uncertainties in the loading, geometry, and material behavior of the problem. In our case, we care only about uncertainty in loading conditions, so do not need the complex, expensive machinery to approximate randomness in the system matrix provided by these methods. Our approach (and the previous approach we accelerate [27]) is akin to an MCS approach using model reduction to reduce the computational load.

To address uncertainty in the graphics community, worst case structural analysis was introduced by Zhou et al. [74]. The method computes a worst case loading scenario where it produces the worst possible stress distribution in the object. Along this line, several different works use the worst case loading to optimize the structure [75, 76, 77]. However, it is unknown how often the worst cast loading will be present in a realistic scenario, such as a figurine falling and hitting the ground.

To address this limitation, Langlois et al. [27] presented a method where the loading of the object was computed from a rigid body simulator that closely mimicked realistic loadings. The work also introduced semantically meaningful failure probabilities that better reflected real-word object failures. The work also presented a structural optimization scheme where the weight of the object was minimized under failure probability constraints. Still, this optimization scheme remained expensive due to the computation of the failure probability gradients.

This work immediately follows this line. I aim to optimize stochastic structural optimization through three specific contributions: acceleration of the gradient computation, a more robust probability gradient formulation, and a restart strategy to overcome non-optimal local minima during optimization.

Chapter 3

Laplacian Eigenfluids Background

Note: A significant portion of this chapter has previously appeared as [1].

3.1 Laplacian Eigenfluids Bases

In the Eigenfluids algorithm, $\mathbf{u}$ in the the Navier-Stokes equations 2.1 is encoded as a linear combination of vector eigenfunctions, $\boldsymbol{\Psi}$. These functions are defined according to a vector wave index, $\mathbf{k} = (k_x, k_y, k_z)$. Each $\boldsymbol{\Psi}$ 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., $\boldsymbol{\Psi} = \{\Phi_x(\mathbf{k}), \Phi_y(\mathbf{k}), \Phi_z(\mathbf{k})\}$. The total number of eigenfunctions being simulated is denoted r, which is the reduced simulation rank. The visualizations of the eigenfunctions is shown in figure 3.1.

We use the notation $\boldsymbol{\Psi}_i$ 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 $\boldsymbol{\Psi}$ simply as:

$$\mathbf{u} = \sum_{i=1}^{r} w_i \boldsymbol{\Psi}_i. \tag{3.1}$$

In a 2D domain defined over $\Omega \in [0, \pi]^2$, the eigenfunctions of Dewitt et al. [14] take

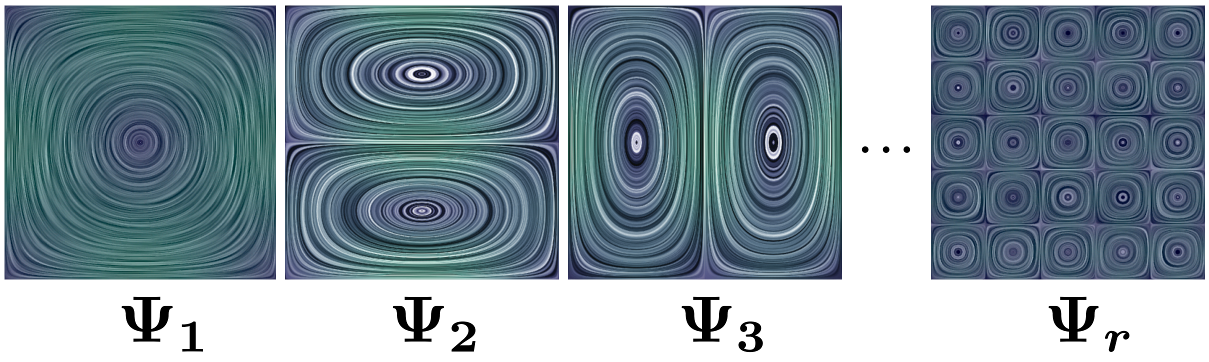


Figure 3.1: Visualization of Laplacian eigen-functions used by Dewitt et al.

the following form:

$$\begin{aligned}\Phi_x(\mathbf{k}) &= -\frac{1}{\eta_{\mathbf{k}}} k_y \sin(k_x x)\cos(k_y y)\\ \Phi_y(\mathbf{k}) &= \frac{1}{\eta_{\mathbf{k}}} k_x \cos(k_x x)\sin(k_y y),\end{aligned} \tag{3.2}$$

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., $\eta_{\mathbf{k}} = \sqrt{k_x^2 + k_y^2}$. A higher wavenumber $\mathbf{k}$ corresponds to a higher frequency of the eigenfunctions. As shown in figure 3.2:

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. 3.1 can be written as $\mathbf{u} = \mathbf{U}\mathbf{w}$.

We will show the eigenfunctions in equation 4.26 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., $\mathbf{\Psi}_\Gamma = 0$.

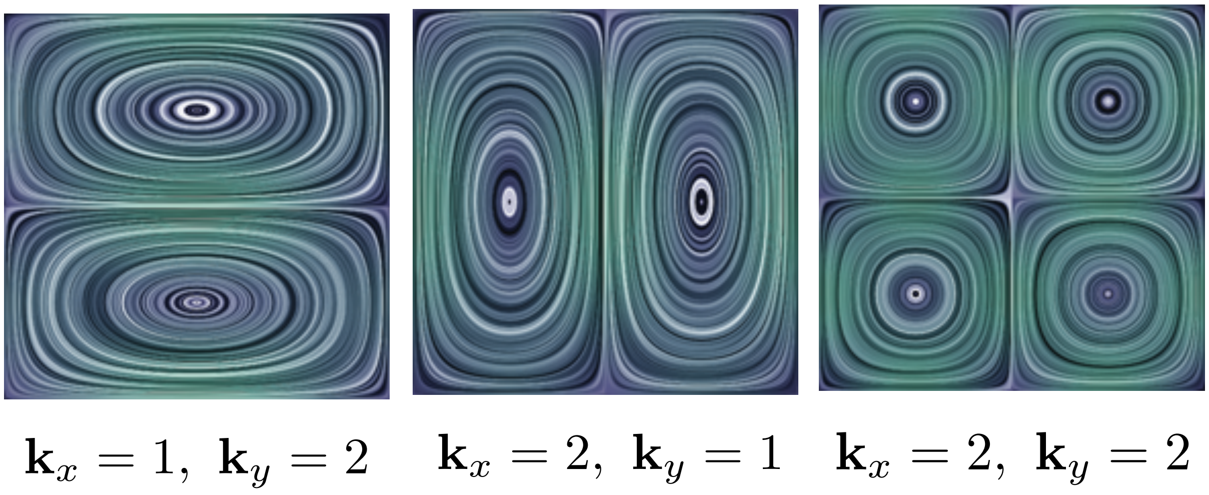

$\mathbf{k}_x = 1,\ \mathbf{k}_y = 2$
$\mathbf{k}_x = 2,\ \mathbf{k}_y = 1$
$\mathbf{k}_x = 2,\ \mathbf{k}_y = 2$

Figure 3.2: Visualization of Laplacian eigen-functions with different wavenumbers.

3.2 Laplacian Eigenfluids Dynamics

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 [10, 51], 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 $\mathbf{U}^T\mathbf{f} = \hat{\mathbf{f}}$.

$$\mathfrak{C}(g,h,i) = [\nabla \times (\phi_h \times \mathbf{\Psi}_i)] \cdot \phi_g \tag{3.3}$$

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 $\mathfrak{C}$ is then computed with entries in equation 3.3. The contribution of the advection operator to the time derivative can then be written,

$$\dot{w}_g = \sum_{h=1}^{r}\sum_{i=1}^{r} w_h w_i \mathfrak{C}(g,h,i), \tag{3.4}$$

which can be expressed in tensor form as $\dot{\mathbf{w}} = \mathbf{w}^T\mathfrak{C}\mathbf{w}$. Following [14], the complete equations can now be integrated using forward Euler,

$$\mathbf{w}^{t+1} = \left(\mathbf{w}^t + \Delta t \left(\mathbf{w}^t\right)^T \mathfrak{C}\mathbf{w}^t + \Delta t\hat{\mathbf{f}}\right) e^{\nu \Delta t \Lambda}, \tag{3.5}$$

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. Above time-stepping scheme is summarized in the following algorithm.

Algorithm 1 Time-stepping algorithm of Dewitt et al. [14]

```
 1: procedure TIME-STEPPING
 2:     $e_1 = \sum_{i=1}^{r} \mathbf{w}_i^2$,  // Store kinetic energy of velocity field
 3:     for $k$ from 1 to $r$ do
 4:         $\dot{\mathbf{w}}_k = \mathbf{w}^T \mathfrak{C} \times_3 \mathbf{w}$
 5:     end for
 6:     $\mathbf{w} \mathrel{+}= \dot{\mathbf{w}}\Delta t$,  // Explicit Euler integration
 7:     $e_2 = \sum_{i=1}^{r} \mathbf{w}_i^2$,  // Calculate energy after time step
 8:     $\mathbf{w} \mathrel{*}= \sqrt{e_1/e_2}$,  // Renormalize energy
 9:     for $k$ from 1 to $r$ do
10:         $\dot{\mathbf{w}}_k \mathrel{*}= \exp(\nu \lambda_k \Delta t)$
11:         $\dot{\mathbf{w}}_k \mathrel{+}= \mathbf{f}\Delta t$
12:     end for
13: end procedure
```

3.3 Discussion

The algorithm has shares the common bottleneck of the reduced order methods. 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 [14], these multiplies can take up to 84% of the running time, and similar stages in other algorithms [46] take up to 99%.

Alternatively, [14] observe that if analytic eigenfunctions are available, then storage issues can be eliminated entirely by recomputing the entries of **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×, and makes the overall algorithm prohibitively slow.

It's unclear whether the advection tensor computed in equation 3.3 preserves energy. However, Dewitt et al. used an energy renormalization step as shown in line 7 of algorithm 1. These renormalization steps reweighs all the basis coefficients when the energy is not preserved in the advection step, therefore may lead to non-physical behaviors. I'll analysis the properties of the advection tensor and how it connects to energy behavior in advection, and the stability of the algorithm when the renormalization step is dropped.

Chapter 4

Scalable Laplacian Eigenfluid

Note: A significant portion of this chapter has previously appeared as [1].

4.1 Fast and General Analytical Basis Functions

In this section, I 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, I 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.

4.1.1 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. [14] proposed eigenfunctions defined over $\Omega \in [0, \pi]^2$ that satisfy Dirichlet boundary conditions along its walls,

$$\begin{aligned} \Phi_x(\mathbf{k}) &= -\frac{1}{\eta_{\mathbf{k}}} k_y \sin(k_x x) \cos(k_y y) \\ \Phi_y(\mathbf{k}) &= \frac{1}{\eta_{\mathbf{k}}} k_x \cos(k_x x) \sin(k_y y), \end{aligned} \tag{4.1}$$

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:

$$\begin{aligned} \langle \mathbf{f}_x, \Phi_x(\mathbf{k}) \rangle &= -\frac{1}{\eta_{\mathbf{k}}} k_y \iint_\Omega \mathbf{f}_x \sin(k_x x) \cos(k_y y) \, dx \, dy \\ \langle \mathbf{f}_y, \Phi_y(\mathbf{k}) \rangle &= \frac{1}{\eta_{\mathbf{k}}} k_x \iint_\Omega \mathbf{f}_y \cos(k_x x) \sin(k_y y) \, dx \, dy. \end{aligned} \tag{4.2}$$

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:

$$\begin{aligned} \langle \mathbf{f}_x, \Phi_x(\mathbf{k}) \rangle &= -\frac{1}{\eta_{\mathbf{k}}} k_y \hat{\mathbf{f}}_x(\mathbf{k}) \\ \langle \mathbf{f}_y, \Phi_y(\mathbf{k}) \rangle &= \frac{1}{\eta_{\mathbf{k}}} k_x \hat{\mathbf{f}}_y(\mathbf{k}). \end{aligned} \tag{4.3}$$

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 [48], but they did not use it to accelerate an Eigenfluids simulation. The transformation also has fundamental connections to the spectral methods of Stam [10, 51], [52] and [53], but they all used the transform to accelerate pressure projection, not velocity reconstruction.

4.1.2 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,

$$g(x,y) = \big(a\cos(k_x x) + b\sin(k_x x)\big)\big(c\cos(k_y y) + d\sin(k_y y)\big), \tag{4.4}$$

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

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

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

$$k_x b_x c_x + k_y a_y d_y = 0 \qquad k_x b_x d_x - k_y a_y c_y = 0 \tag{4.5}$$

$$-k_x a_x c_x + k_y b_y d_y = 0 \qquad k_x a_x d_x + k_y b_y c_y = 0. \tag{4.6}$$

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

$$a_x b_x = 0 \qquad c_y d_y = 0 \tag{4.7}$$

$$a_y b_y = 0 \qquad c_x d_x = 0. \tag{4.8}$$

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:

$$\frac{\partial \Phi_x}{\partial x}\bigg|_{x=0,\pi} = 0 \qquad \Phi_y\big|_{y=0,\pi} = 0. \tag{4.9}$$

The values $b_x = c_y = 0$ select the trigonometric functions that satisfy these boundaries, as well as Eqn. 4.7. The solution now becomes,

$$\begin{aligned} \Phi_x(\mathbf{k}) &= a_x \cos(k_x x)\big(c_x \cos(k_y y) + d_x \sin(k_y y)\big) \\ \Phi_y(\mathbf{k}) &= d_y \sin(k_y y)\big(a_y \cos(k_x x) + b_y \sin(k_x x)\big), \end{aligned} \tag{4.10}$$

and two of the constraints from Eqn. 4.6 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:

$$\begin{aligned} \Phi_x(\mathbf{k}) &= a_x c_x \cos(k_x x)\cos(k_y y) \\ \Phi_y(\mathbf{k}) &= b_y d_y \sin(k_y y)\sin(k_x x). \end{aligned} \tag{4.11}$$

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

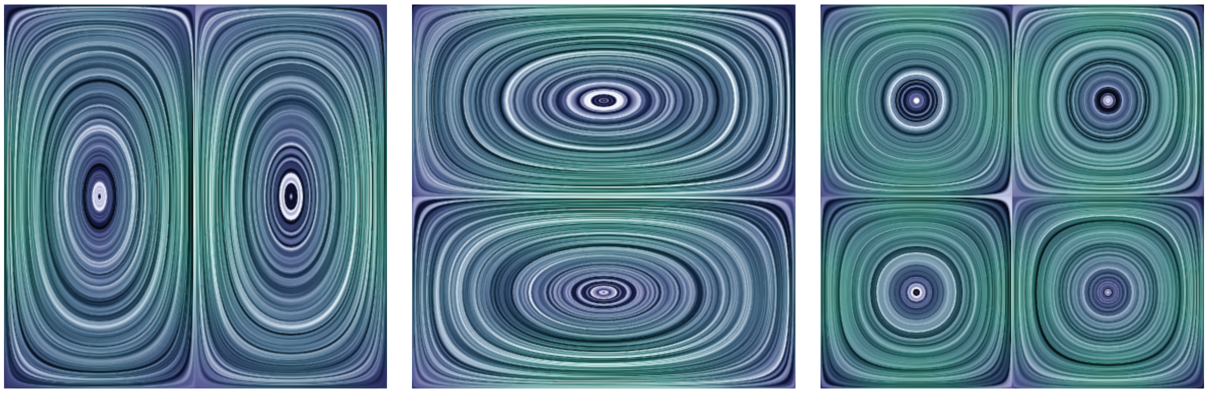
$$\begin{aligned} \Phi_x(\mathbf{k}) &= \frac{1}{\eta_{\mathbf{k}}} k_y \cos(k_x x)\cos(k_y y) \\ \Phi_y(\mathbf{k}) &= \frac{1}{\eta_{\mathbf{k}}} k_x \sin(k_x x)\sin(k_y y). \end{aligned} \tag{4.12}$$

These eigenfunctions are clearly amenable to DCT and DST acceleration, as they are all products of trigonometric functions. They are visualized in Fig. 4.1. 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.

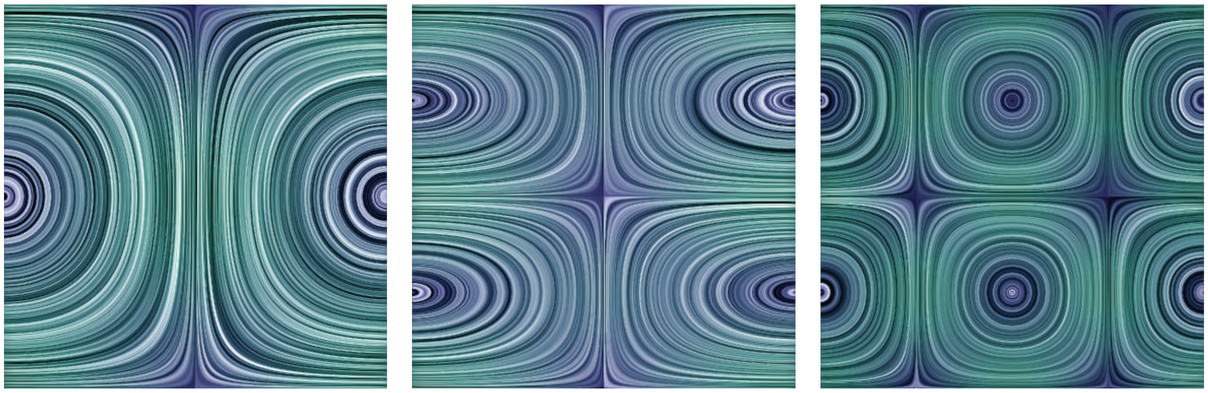
One Neumann Wall:

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. 4.9 are instead set to,

$$\left.\frac{\partial \Phi_x}{\partial x}\right|_{x=\pi} = 0 \qquad\qquad \Phi_x\big|_{x=0} = 0, \tag{4.13}$$



(a) Dirichlet boundaries along all walls



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

Figure 4.1: 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.

then the same eigenfunctions from Eqn. 4.26 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. $k_x \in \left(\mathbb{Z}^+ - \frac{1}{2}\right)$. This is illustrated by the function $\sin((k - \frac{1}{2})x)$, where k is a positive integer. The function is zero at $x = 0$, which satisfies the Dirichlet condition, and its derivative, $\cos((k - \frac{1}{2})x)$, is zero at $x = \pi$, which satisfies the Neumann condition.

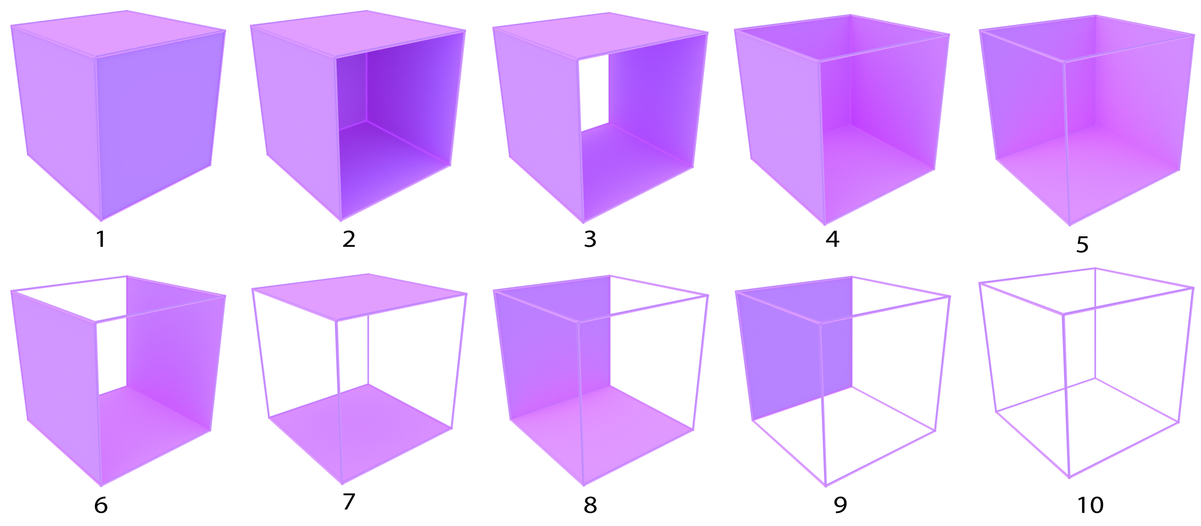


Figure 4.2: Visualization of all 10 possible boundary conditions, with associated indices.

3D Neumann Basis:

For a box domain with six boundary walls, there are 10 possible combinations of Dirichlet and Neumann boundary conditions (up to symmetry). Figure 4.2 is a visualization of all 10 combinations. We list all the basis functions we derived for these combinations.

1. Six Dirichlet walls:

$$\begin{cases} \Phi_x = a(\sin(k_x x)\cos(k_y y)\cos(k_z z)) \\ \Phi_y = b(\cos(k_x x)\sin(k_y y)\cos(k_z z)) \\ \Phi_z = c(\cos(k_x x)\cos(k_y y)\sin(k_z z)) \end{cases} \tag{4.14}$$

$k_x, k_y, k_z \in \mathbb{Z}^+ \cup 0$. The constants a, b, c need to satisfy the divergence free condition: $ak_x + bk_y + ck_z = 0$.

2. One Neumann wall, Φ_x at $x = \pi$:

$$\begin{cases} \Phi_x = a(\sin(k_x x)\cos(k_y y)\cos(k_z z)) \\ \Phi_y = b(\cos(k_x x)\sin(k_y y)\cos(k_z z)) \\ \Phi_z = c(\cos(k_x x)\cos(k_y y)\sin(k_z z)) \end{cases} \tag{4.15}$$

$k_x \in \left(\mathbb{Z}^+ - \frac{1}{2}\right) \cup 0$, $k_y, k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $ak_x + bk_y + ck_z = 0$.

3. Two Neumann walls, Φ_x at $x = 0, x = \pi$:

$$\begin{cases} \Phi_x = a(\cos(k_x x)\cos(k_y y)\cos(k_z z)) \\ \Phi_y = b(\sin(k_x x)\sin(k_y y)\cos(k_z z)) \\ \Phi_z = c(\sin(k_x x)\cos(k_y y)\sin(k_z z)) \end{cases} \tag{4.16}$$

$k_x, k_y, k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $-ak_x + bk_y + ck_z = 0$.

4. Two Neumann walls for Φ_x at $x = \pi$ and Φ_y at $y = \pi$:

$$\begin{cases} \Phi_x = a(\sin(k_x x)\cos(k_y y)\cos(k_z z)) \\ \Phi_y = b(\cos(k_x x)\sin(k_y y)\cos(k_z z)) \\ \Phi_z = c(\cos(k_x x)\cos(k_y y)\sin(k_z z)) \end{cases} \tag{4.17}$$

$k_x, k_y \in \left(\mathbb{Z}^+ - \frac{1}{2}\right) \cup 0$, $k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $ak_x + bk_y + ck_z = 0$.

5. Three Neumann walls, Φ_x at $x = \pi$, Φ_y at $y = \pi$ and Φ_z at $z = \pi$:

$$\begin{cases} \Phi_x = a(\sin(k_x x)\cos(k_y y)\cos(k_z z)) \\ \Phi_y = b(\cos(k_x x)\sin(k_y y)\cos(k_z z)) \\ \Phi_z = c(\cos(k_x x)\cos(k_y y)\sin(k_z z)) \end{cases} \tag{4.18}$$

$k_x, k_y, k_z \in \left(\mathbb{Z}^+ - \frac{1}{2}\right) \cup 0$. Divergence free condition: $ak_x + bk_y + ck_z = 0$.

6. Three Neumann walls, Φ_x at $x = \pi, x = 0$ and Φ_y at $y = \pi$:

$$\begin{cases} \Phi_x = a(\cos(k_x x)\cos(k_y y)\cos(k_z z)) \\ \Phi_y = b(\sin(k_x x)\sin(k_y y)\cos(k_z z)) \\ \Phi_z = c(\sin(k_x x)\cos(k_y y)\sin(k_z z)) \end{cases} \tag{4.19}$$

$k_y \in \left(\mathbb{Z}^+ - \frac{1}{2}\right) \cup 0$, $k_x, k_z \in \mathbb{Z}^+ \cup 0$, Divergence free condition: $-ak_x + bk_y + ck_z = 0$.

7. Four Neumann walls, Φ_x at $x = 0, x = \pi$ and Φ_z at $z = 0, z = \pi$:

$$\begin{cases} \Phi_x = a(\cos(k_x x)\cos(k_y y)\sin(k_z z)) \\ \Phi_y = b(\sin(k_x x)\sin(k_y y)\sin(k_z z)) \\ \Phi_z = c(\sin(k_x x)\cos(k_y y)\cos(k_z z)) \end{cases} \tag{4.20}$$

$k_x, k_y, k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $-ak_x + bk_y - ck_z = 0$.

8. Four Neumann walls, Φ_x at $x = \pi$, Φ_y at $y = \pi$ and Φ_z at $z = 0, z = \pi$.

$$\begin{cases} \Phi_x = a(\sin(k_x x)\cos(k_y y)\sin(k_z z)) \\ \Phi_y = b(\cos(k_x x)\sin(k_y y)\sin(k_z z)) \\ \Phi_z = c(\cos(k_x x)\cos(k_y y)\cos(k_z z)) \end{cases} \tag{4.21}$$

$k_x, k_y \in \left(\mathbb{Z}^+ - \frac{1}{2}\right) \cup 0$, $k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $ak_x + bk_y - ck_z = 0$

9. Five Neumann walls, Φ_x at $x=\pi$, Φ_y at $y=0, y=\pi$ and Φ_z at $z=0, z=\pi$:

$$\begin{cases} \Phi_x = a(\sin(k_x x)\sin(k_y y)\sin(k_z z)) \\ \Phi_y = b(\cos(k_x x)\cos(k_y y)\sin(k_z z)) \\ \Phi_z = c(\cos(k_x x)\sin(k_y y)\cos(k_z z)) \end{cases} \tag{4.22}$$

$k_x \in \left(\mathbb{Z}^+ - \frac{1}{2}\right) \cup 0$, $k_y, k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $ak_x - bk_y - ck_z = 0$

10. Six Neumann walls for all three axes.

$$\begin{cases} \Phi_x = a(\cos(k_x x)\sin(k_y y)\sin(k_z z)) \\ \Phi_y = b(\sin(k_x x)\cos(k_y y)\sin(k_z z)) \\ \Phi_z = c(\sin(k_x x)\sin(k_y y)\cos(k_z z)) \end{cases} \tag{4.23}$$

$k_x, k_y, k_z \in \mathbb{Z}^+ \cup 0$. Divergence free condition: $ak_x + bk_y + ck_z = 0$.

All the above basis functions need to be normalized, which places another constraint on the three constants. Assuming we are given a fixed wave number k_x, k_y, k_z, we need to solve for a, b and c. There are currently only two constraints, the normalization and divergence-free constraint. Another constraint needs to be added in order to determine a, b and c. Functions of the same form as ours are used to describe the electric field in a box cavity resonator, see e.g. [78]. In this case, a "direction of propagation" is chosen, and then the functions to describe the electric field are derived. For example, if the basis in equation 4.23 and the x axis is chosen as the "direction of propagation", then the constants become : $a = -(k_y^2 + k_z^2), b = k_x k_y, c = k_x k_z$. The constants are then normalized. We found that choosing the constants this way maximizes a under both the divergence-free and normalization constraint. Thus, the velocity along the direction x is maximized. One can also derive the same formula by solving the constrained maximization problem where a is maximized under the divergence-free and normalization constraints.

When we determine the constants a, b, c, we take the scene into consideration. For example, for a scene where the fluid flows predominantly along y direction (e.g. due to buoyancy), we choose the constant by maximizing the velocity along the y direction. If there is no prior knowledge about the direction of the fluid, we then choose the constant by maximize a, b and c individually, and then use the average value for each constant.

4.1.3 Computational Considerations

The velocity reconstruction method from §4.1.1 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 2D example is shown in figure 4.3.

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.

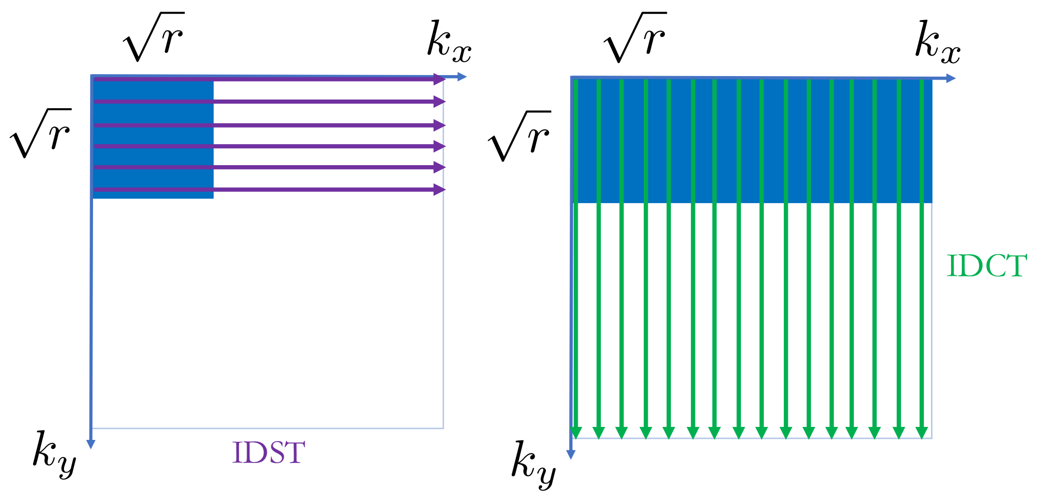


Figure 4.3: Pruned DCT in 2D coordinates. Left: Instead of doing N IDSTs along the first direction, we can prune it down to $\sqrt{r}$ IDSTs. Right: A full N IDCTs are then performed along the second direction.

4.2 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 [14] results in an unstable simulation, and that the variational form from [15] 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.

4.2.1 Advection Tensor

The method of [14] computes each entry of the advection tensor as $\mathfrak{C}(g, h, i) = [\nabla \times (\phi_h \times \mathbf{\Psi}_i)] \cdot \phi_g$, where $\mathbf{\Psi}_i$ 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 $[\nabla \times (\phi_h \times \mathbf{\Psi}_i)]$ does not. The phase mismatch will cause $[\nabla \times (\phi_h \times \mathbf{\Psi}_i)]$ 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 [14] 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, [15] showed a more principled way of enforcing this condition:

$$\mathfrak{C}(g, h, i) = \int_\Omega (\nabla \times \mathbf{\Psi}_i) \cdot (\mathbf{\Psi}_g \times \mathbf{\Psi}_h) d\Omega. \tag{4.24}$$

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 section 4.2.2.

The advection tensor in equation 4.24 preserves energy. To prove that, the energy of the fluid can be written as $E_t = \frac{1}{2}\Omega^T\Omega$. The derivative of the energy is then:

$$\dot{E}_t = \mathbf{w}^T\dot{\mathbf{w}} = \mathbf{w}^T\mathbf{C}\mathbf{w} = \frac{1}{2}\mathbf{w}^T(\mathbf{C} + \mathbf{C}^T)\mathbf{w} = 0. \tag{4.25}$$

where $\mathbf{C} = \mathfrak{C} \times_3 \mathbf{w}$. Matrix $\mathbf{C}$ is skew-symmetric due to equation 4.24. On the other hand, if energy derivative is zero for all $\mathbf{w}$, we know that matrix $\mathbf{C}$ is skew-symmetric, which in turn requires the 3D tensor must be anti-symmetric. Therefore the sufficient and necessary for the advection to preserve energy is that the advection tensor should be anti-symmetric.

4.2.2 Computing the advection tensor

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. $\mathbf{\Psi}_k = \mathbf{\Psi}(\mathbf{k})$, $\mathbf{\Psi}_l = \mathbf{\Psi}(\mathbf{l})$, $\mathbf{\Psi}_m = \mathbf{\Psi}(\mathbf{m})$. The normalized 2D Dirichlet basis functions ($\mathbf{\Psi}(\mathbf{k}) = \{\Phi_x(\mathbf{k}), \Phi_y(\mathbf{k})\}$) are:

$$\begin{aligned}\Phi_x(\mathbf{k}) &= -\frac{2}{\pi}\frac{k_y}{\eta_{\mathbf{k}}}\sin(k_x x)\cos(k_y y)\\ \Phi_y(\mathbf{k}) &= \frac{2}{\pi}\frac{k_x}{\eta_{\mathbf{k}}}\cos(k_x x)\sin(k_y y),\end{aligned} \tag{4.26}$$

where $\eta_{\mathbf{k}} = \sqrt{k_x^2 + k_y^2}$, and $\mathbf{\Psi}(\mathbf{l})$ and $\mathbf{\Psi}(\mathbf{m})$ have the same form as above but with $\mathbf{k}$ replaced with $\mathbf{l}$ and $\mathbf{m}$.

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

$$\nabla \times \mathbf{\Psi}_m = -\frac{2}{\pi}\eta_{\mathbf{m}}\sin(m_x x)\sin(m_y y). \tag{4.27}$$

After computing the term $\mathbf{\Psi}_k \times \mathbf{\Psi}_l$, we have

$$\begin{aligned}
&\mathfrak{C}(k,l,m) = \frac{2\eta_{\mathbf{m}}}{\pi^3 \eta_{\mathbf{l}} \eta_{\mathbf{k}}}[\\
&l_x k_y \int_0^\pi \sin(m_x x)(\sin((k_x + l_x)x) + sin((k_x - l_x)x))dx \\
&\int_0^\pi \sin(m_y y)(\sin((k_y + l_y)y) - \sin((k_y - l_y)y))dy- \\
&l_y k_x \int_0^\pi \sin(m_x x)(\sin((k_x + l_x)x) - \sin((k_x - l_x)x))dx \\
&\int_0^\pi \sin(m_y y)(\sin((k_y + l_y)y) + \sin((k_y - l_y)y))dy],
\end{aligned} \tag{4.28}$$

where the integral

$$\begin{aligned}
&\int_0^\pi \sin(m_x x)(\sin((k_x + l_x)x)dx = \\
&\frac{1}{2}\int_0^\pi \cos((m_x - k_x - l_x)x) - \cos((m_x + k_x + l_x)x)dx
\end{aligned} \tag{4.29}$$

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 $\mathfrak{C}(k,l,m) = \frac{\eta_{\mathbf{m}}(l_x k_y - l_y k_x)}{2\pi \eta_{\mathbf{l}} \eta_{\mathbf{k}}}$.

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:

$$\begin{aligned}\Phi_x(\mathbf{k}) &= \frac{2}{\pi}\frac{k_y}{\eta_{\mathbf{k}}}\cos(k_x x)\cos(k_y y)\\ \Phi_y(\mathbf{k}) &= \frac{2}{\pi}\frac{k_x}{\eta_{\mathbf{k}}}\sin(k_x x)\sin(k_y y).\end{aligned} \tag{4.30}$$

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

$$\nabla \times \boldsymbol{\Psi}_m = \frac{2}{\pi}\eta_{\mathbf{m}}\cos(m_x x)\sin(m_y y). \tag{4.31}$$

Similar as the Dirichlet case, we can then compute tensor entries as:

$$\begin{aligned}&\mathfrak{C}(k,l,m) = \frac{2\eta_{\mathbf{m}}}{\pi^3\eta_{\mathbf{l}}\eta_{\mathbf{k}}}[\\ &- l_y k_x \int_0^\pi \cos(m_x x)(\sin((k_x+l_x)x) + sin((k_x-l_x)x))dx\\ &\int_0^\pi \sin(m_y y)(\sin((k_y+l_y)y) + \sin((k_y-l_y)y))dy+\\ &l_x k_y \int_0^\pi \cos(m_x x)(\sin((k_x+l_x)x) - \sin((k_x-l_x)x))dx\\ &\int_0^\pi \sin(m_y y)(\sin((k_y+l_y)y) - \sin((k_y-l_y)y))dy].\end{aligned} \tag{4.32}$$

The major difference between equation 4.32 and equation 4.28 is that the integrand along x direction in equation 4.32 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 4.32 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.

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:

$$m_x = l_x + k_x \qquad m_x = l_x - k_x \qquad m_x = k_x - l_x \tag{4.33}$$
$$m_y = l_y + k_y \qquad m_y = l_y - k_y \qquad m_y = k_y - l_y \tag{4.34}$$
$$m_z = l_z + k_z \qquad m_z = l_z - k_z \qquad m_z = k_z - l_z. \tag{4.35}$$

In order for an entry in $\mathfrak{C}$ to be non-zero, one relation in each row of Eqns. 4.33-4.35 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. 4.33-4.35 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 [79, 80] is a direction for future work. However we will later show in §4.4 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 [81], 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:

$$\overline{\mathfrak{C}}(g,h,i) = b_g b_h b_i \cdot \mathfrak{C}(g,h,i). \tag{4.36}$$

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 §4.4.

4.2.3 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 [82] is difficult to apply in the spectral domain, and the non-linear advection tensor interferes with spectral eigenanalysis approaches, which are inherently linear [18]. 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 [15] in the form of their implicit trapezoidal update,

$$\mathbf{w}^{t+1} = \frac{\Delta t}{2}\mathbf{C}^{t+1}\mathbf{w}^{t+1} + \frac{\Delta t}{2}\mathbf{C}^{t}\mathbf{w}^{t} + \mathbf{w}^{t} + \hat{\mathbf{f}}, \tag{4.37}$$

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. 4.4 that a semi-implicit solve (i.e., a single Newton iteration in the style of [83]) was sufficient to maintain stability and good energy behavior.

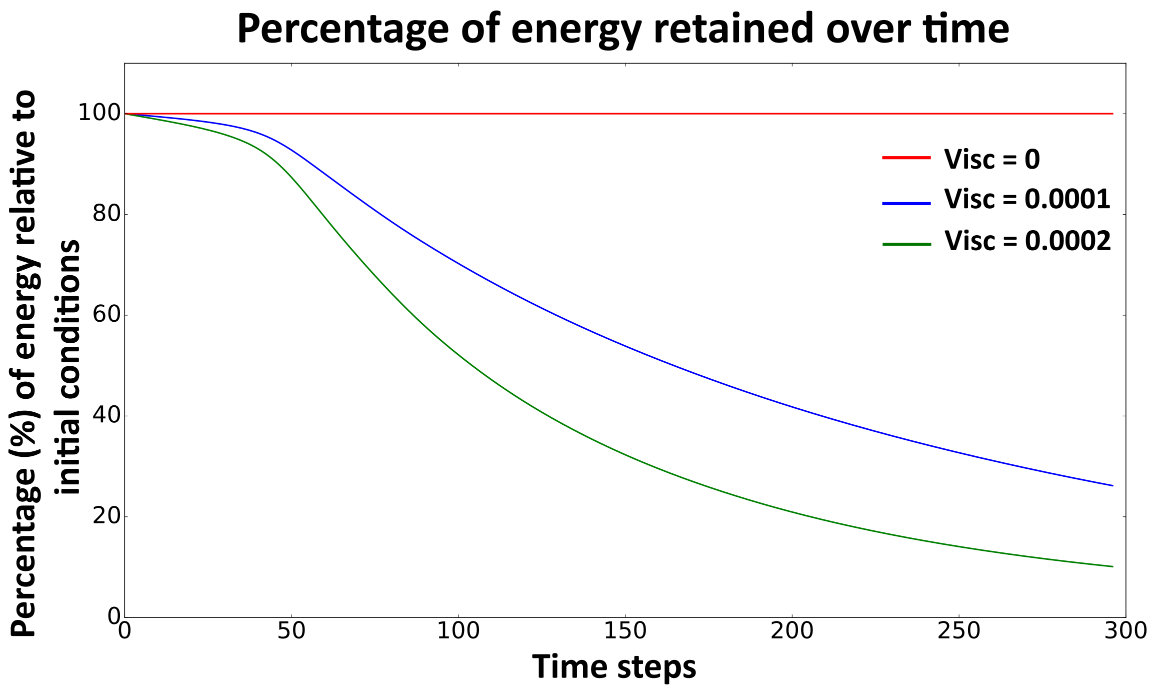


Figure 4.4: Energy of the simulation in Fig. 4.6 over time, for multiple viscosities. For the zero-viscosity regime, energy is stably preserved even when only a single Newton iteration is used.

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, $\mathbf{A}^T\mathbf{A}$, i.e., CGNR [84]. 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. 4.37 is sufficiently well-conditioned that this caveat does not apply. We can rewrite the system as:

$$\left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C}^{t+1}\right)\mathbf{w}^{t+1} = \frac{\Delta t}{2}\mathbf{C}^{t}\mathbf{w}^{t} + \mathbf{w}^{t} + \hat{\mathbf{f}}. \tag{4.38}$$

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:

$$\left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C}\right)^T \left(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C}\right) = \mathbf{I} + \frac{\Delta t^2}{4}\mathbf{C}^T\mathbf{C}. \tag{4.39}$$

The matrix $\mathbf{C}$ and its condition number are squared, but since $\Delta t \ll 1$, the squaring is offset by the $\frac{\Delta t^2}{4}$ 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.

4.3 Comparison with Spectral Methods

Since our method is closely related to spectral methods [57], we discuss and compare the approaches here. Specifically, we compare our algorithm to the collocation methods from Dedalus [58], a recent spectral library that has been successfully used to advance understanding in both general [85] and computational [86] 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 [14] in §4.2.1. 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.

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 [20], as they are employed to obtain spatial derivatives at specific collocation points [18]. While these differentiation matrices are never constructed explicitly, we can use them to determine the conservation properties of the underlying scheme. As shown in [20], 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. 4.5). 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. 4.5. Overall, compared to spectral collocation methods, our method handles various boundary conditions more easily and captures a wider variety of low viscosity flows.

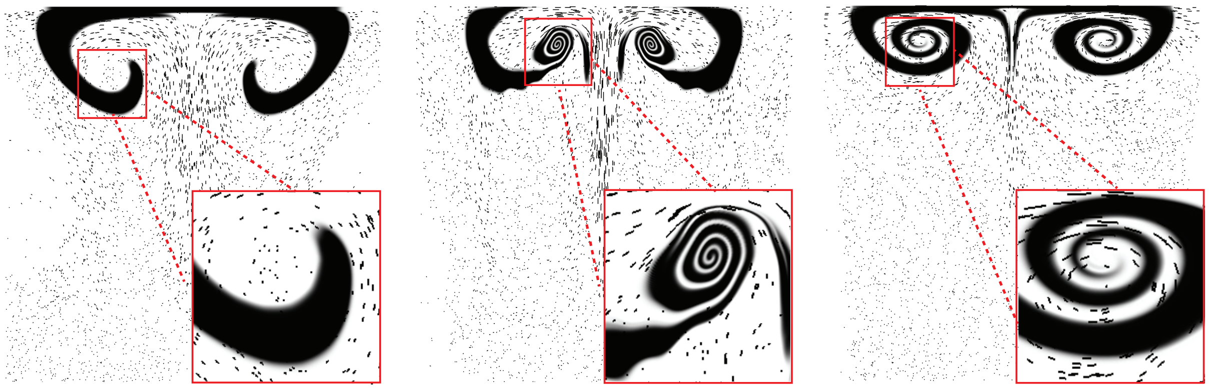


Figure 4.5: **Left:** A spectral simulation using 60×60 basis functions in Dedalus. The top and bottom walls were set to Dirichlet velocity boundaries, while the left and right were set to Neumann. We used the smallest possible viscosity that did not destabilize the simulation. **Middle:** An equivalent 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 $\nu = 2 \times 10^{-5}$.

4.3.1 Energy Conservation of Chebyshev Collocation Methods

In spectral collocation (pseudospectral) methods, differentiation matrices are used to compute the derivative of a given function on a grid. For example, given a discretized function $v(x_i), \quad i = 0, 1, ..., N-1$, the discretized derivative of v can be written as $\mathbf{v}' = \mathbf{D}\mathbf{v}$, where $\mathbf{D}$ is an $N \times N$ matrix, and $\mathbf{v}', \mathbf{v} \in \mathbb{R}^N$. Generally, $\mathbf{D}$ is a dense matrix. However, the matrix is usually never constructed explicitly, and a transformation method is used to compute the derivative $\mathbf{v}'$ given $\mathbf{v}$. For Fourier series and Chebyshev polynomials, the derivative can be evaluated in $N\log(N)$ time complexity using the FFT.

When the N-S equation is discretized in space, it is desirable for the spatially-discretized, time-continuous equation to conserve some important properties of the N-S equations. It is advised in [87] that the convection form of N-S equation will lead to instabilities at various Reynolds numbers, bcause the discretized convection form may not conserve momentum and energy. Thus, the rotation form of the N-S equations should be used instead. We can show that Chebyshev collocation does not conserve energy under rotation form when the viscosity is zero. In contrast to the Chebyshev differential matrix, the Fourier differential matrix is skew-symmetric, thus its semi-discretized equation conserves energy [20]. However, the Fourier basis assumes periodic boundary conditions.

The rotation form of the N-S equations is:

$$\begin{aligned} &\frac{\partial \mathbf{u}}{\partial t} + \mathbf{w} \times \mathbf{u} = -\nabla q + \nu \nabla^2 \mathbf{u} \\ &\nabla \cdot \mathbf{u} = 0 \end{aligned} \tag{4.40}$$

where $\mathbf{w} = \nabla \times \mathbf{u}$ and $q = p + 0.5|\mathbf{u}|^2$. This form is equivalent to the common convection form of the N-S equations.

First, let us denote spatially-discretized velocity as $\mathbf{u}^N$. Define the discrete gradient operator $\mathbb{G}_N q^N = \begin{bmatrix} \mathbf{D}_N^x q^N & \mathbf{D}_N^y q^N & \mathbf{D}_N^z q^N \end{bmatrix}^T$ where $\mathbf{D}$ is the Chebyshev differentiation matrix with size N along desired direction. And define the discrete divergence operator $\mathbb{D}_N \mathbf{u}^N = \mathbf{D}_N^x \mathbf{u}_x^N + \mathbf{D}_N^y \mathbf{u}_y^N + \mathbf{D}_N^z \mathbf{u}_z^N$. Omitting the viscosity, the space discretized inviscid N-S equation can be written as:

$$\begin{aligned} &\frac{d\mathbf{u}^N}{dt} + \mathbf{w}^N \times \mathbf{u}^N + \mathbb{G}_N q^N = 0 \\ &\mathbb{D}_N \mathbf{u}^N = 0 \end{aligned} \tag{4.41}$$

Taking the first equation of above and performing a dot product on both sides using $\mathbf{u}^N$

yields:

$$\frac{d|\mathbf{u}|^2}{dt} + (\mathbf{w}^N \times \mathbf{u}^N, \mathbf{u}^N) + (\mathbb{G}_N q^N, \mathbf{u}^N) = 0. \tag{4.42}$$

The product $(\mathbf{w}^N \times \mathbf{u}^N, \mathbf{u}^N)$ is zero since the cross product is orthogonal to $\mathbf{u}^N$. Thus, in order for energy to be conserved, $(\mathbb{G}_N q^N, \mathbf{u}^N)$ must be equal to zero. In addition to this, since $\mathbb{D}_N \mathbf{u}^N = 0$, we can construct the equation $(q^N, \mathbb{D}_N \mathbf{u}^N) = 0$. Thus, assuming that energy is conserved, the below two equations must hold:

$$\begin{aligned} (\mathbb{G}_N q^N, \mathbf{u}^N) &= (\mathbf{D}^x_N q^N, \mathbf{u}^N_x) + (\mathbf{D}^y_N q^N, \mathbf{u}^N_y) + (\mathbf{D}^z_N q^N, \mathbf{u}^N_z) = 0 \\ (q^N, \mathbb{D}_N \mathbf{u}^N) &= (q^N, \mathbf{D}^x_N \mathbf{u}^N_x) + (q^N, \mathbf{D}^y_N \mathbf{u}^N_y) + (q^N, \mathbf{D}^z_N \mathbf{u}^N_z) = 0 \end{aligned} \tag{4.43}$$

Writing the above two equations as matrix products yields:

$$\begin{aligned} &(\mathbf{u}^N_x)^T \mathbf{D}^x_N q^N + (\mathbf{u}^N_y)^T \mathbf{D}^y_N q^N + (\mathbf{u}^N_z)^T \mathbf{D}^z_N q^N = 0 \\ &(q^N)^T \mathbf{D}^x_N \mathbf{u}^N_x + (q^N)^T \mathbf{D}^y_N \mathbf{u}^N_y + (q^N)^T \mathbf{D}^z_N \mathbf{u}^N_z = 0. \end{aligned} \tag{4.44}$$

Taking the transpose of the first equation, and then adding the second one, we get:

$$\begin{aligned} (q^N)^T (\mathbf{D}^x_N + (\mathbf{D}^x_N)^T) \mathbf{u}^N_x + &(q^N)^T (\mathbf{D}^y_N + (\mathbf{D}^y_N)^T) \mathbf{u}^N_y + \\ &(q^N)^T (\mathbf{D}^z_N + (\mathbf{D}^z_N)^T) \mathbf{u}^N_z = 0. \end{aligned} \tag{4.45}$$

For arbitrary $\mathbf{u}$ and q, the above equation only holds when $\mathbf{D}_N + (\mathbf{D}_N)^T = 0$ (skew-symmetric), or when $\mathbf{D}_N = (\mathbf{D}_N)^T$ (symmetric). As shown in page 53 of [18], the

Chebyshev differentiation matrix is:

$$\mathbf{D}_N = \begin{bmatrix} \frac{2(N-1)^2+1}{6} & \cdots & 2\frac{(-1)^j}{1-x_j} & \cdots & \frac{1}{2}(-1)^{(N-1)} \\ \vdots & \ddots & & \frac{(-1)^{i+j}}{x_i-x_j} & \vdots \\ -\frac{1}{2}\frac{(-1)^i}{1-x_i} & & \frac{-x_j}{2(1-x_j^2)} & & \frac{1}{2}\frac{(-1)^{N-1+i}}{1+x_i} \\ \vdots & \frac{(-1)^{i+j}}{x_i-x_j} & & \ddots & \vdots \\ -\frac{1}{2}(-1)^{(N-1)} & \cdots & -2\frac{(-1)^{N-1+j}}{1+x_j} & \cdots & -\frac{2(N-1)^2+1}{6} \end{bmatrix} \tag{4.46}$$

where i, j is integer index from 0 to $N-1$, and $x_j = \cos(\frac{j\pi}{N-1})$. It is clear that this Chebyshev differentiation matrix is neither skew-symmetric, nor symmetric. So the equation 4.45 does not hold. Thus, $(\mathbb{G}_N q^N, \mathbf{u}^N) = 0$ does not hold since $\mathbb{D}_N \mathbf{u}^N = 0$, and Chebyshev collocation methods are not energy-conserving.

4.4 Implementation and Results

4.4.1 Implementation

We implemented our Scalable Laplacian Eigenfluids algorithm in C++. The CGNR algorithm was implemented by modifying the CG implementation in Eigen [88] to include an extra transposed matrix multiply. We used FFTW3 [89] to perform DCT and DST. Multi-threading was enabled 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 [52] used the semi-Lagrangian advection from Zephyr [90], but replaced the pressure projection with the DCT-based approach. All density advection was performed using a MacCormack scheme [91]. Similar to [14], [45], and [43], 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.

4.4.2 Results

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

Table 4.1: Running time and memory usage results of our DCT-based approach compared to caching a large matrix $\mathbf{U}$ of basis functions, or recomputing the entries of $\mathbf{U}$ 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.

Colliding Smoke Jets

. Our simplest example contains two blocks of smoke driven together by an initial impulse. As can be seen in Fig. 4.6, visually interesting details continue to appear as we increase the basis rank. The grid resolution is 220^3. As shown in Table 4.1, 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 [52], as their use of DCT more closely matches our approach than the original [10] 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 §4.1.3 is used, this stage of our method runs 30% faster. In Fig. 4.6, [52] 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. 4.7. 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. 4.7, more detail appears as we increase the basis rank.

The timings are shown in Table 5.1. For this scene, at the maximum rank of $r = 12600$, each frame takes 6.6 secs.

Thin Dirichlet Obstacles

. We also test our algorithm using static, Dirichlet obstacles. As observed by [14], 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. 4.8. 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.

Precomputing and Compressing the Tensor.

While it should be possible to directly address the entries that satisfy Eqns. 4.33-4.35, 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. 4.10, 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 4.2, 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 [79, 80] 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 4.2: 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. 4.10).

Real-time Interaction

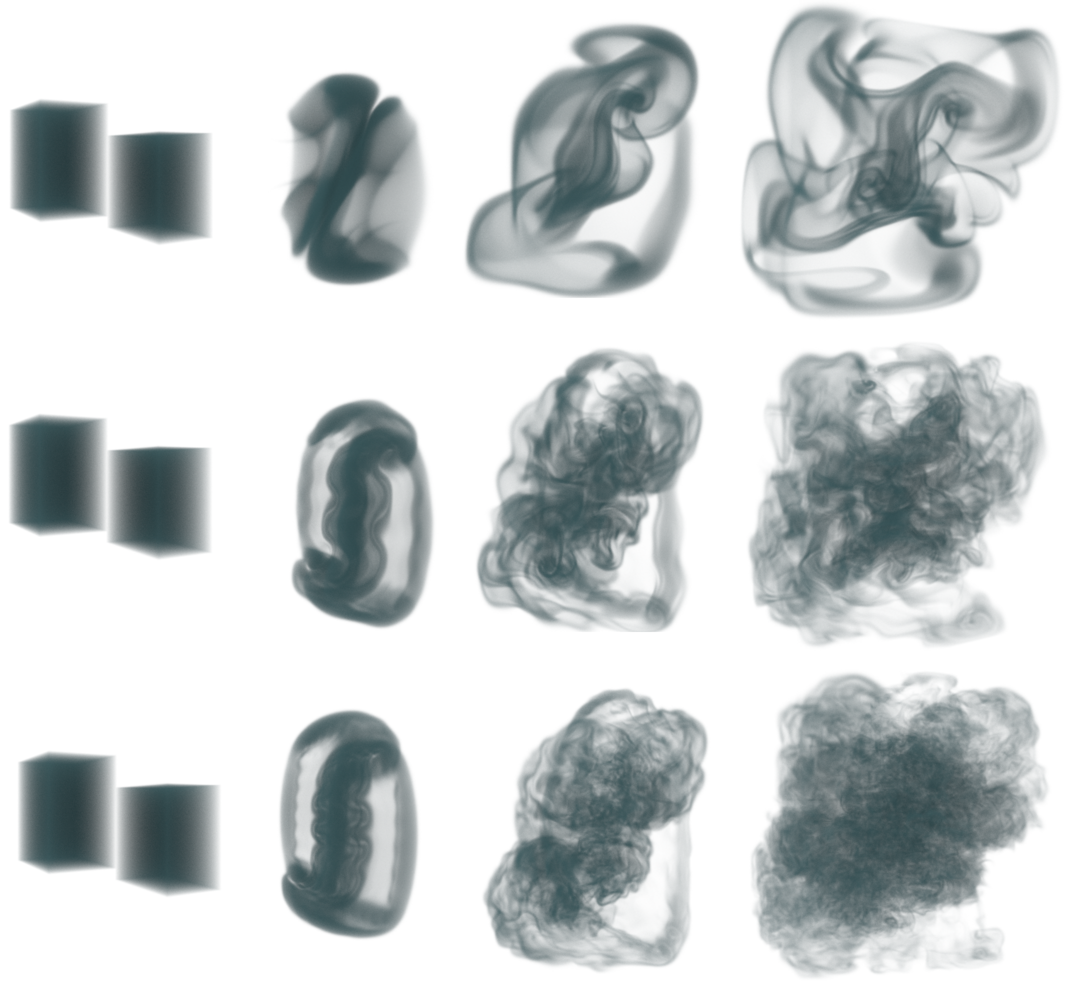
. Even with a large basis rank, our method is fast enough to run interactively. Fig. 4.11 shows an interactive simulation with $r = 1000$ and two Neumann walls. The advection tensor is compressed by dropping 80% of the smallest entries, and the grid resolution is $120 \times 60 \times 60$. The example runs at 13 FPS.

Directable Forward Scattering

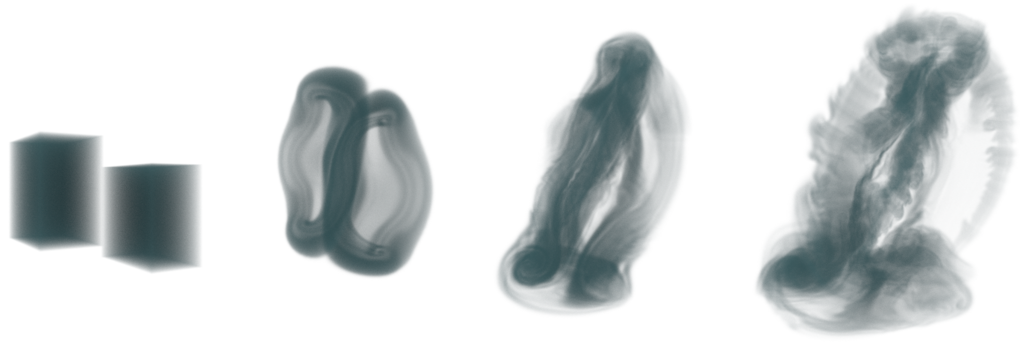
. As described in §4.2.1, we can re-weight the advection tensor to introduce directability into the phenomenon of forward scattering. In Fig. 4.12, we show the results of different forward scattering intensities using a various settings for c in the b_i function from Eqn. 4.36. When scattering is amplified, details emerge at higher frequencies much more quickly. In Fig. 4.13, we show that when a negative is introduced into the reweighting function, flows emerge that undergo ghostly reversals.

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

Table 4.3: Timing breakdown of our algorithm across all the different examples. The tensor † uses single precision floating point, and the rest use double.



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



(b) Results of the semi-Lagrangian / DCT method of [52].

Figure 4.6: 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 [52]. Our results are clearly more inviscid.

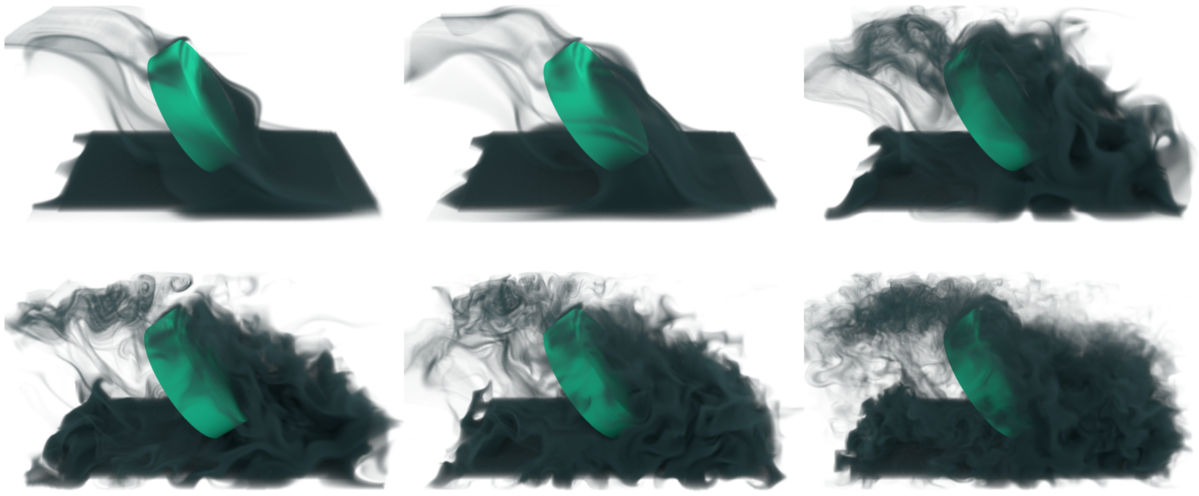


Figure 4.7: Paddle wheel scene: Top row: $r = 100, r = 200, r = 1000$ basis functions are used. Bottom row: $r = 2000, r = 4000, r = 12600$ basis functions are used. Previous approaches have only been able to achieve $r \approx 500$, and would have needed 2.25 TB of memory to simulate the $r = 12600$ case.

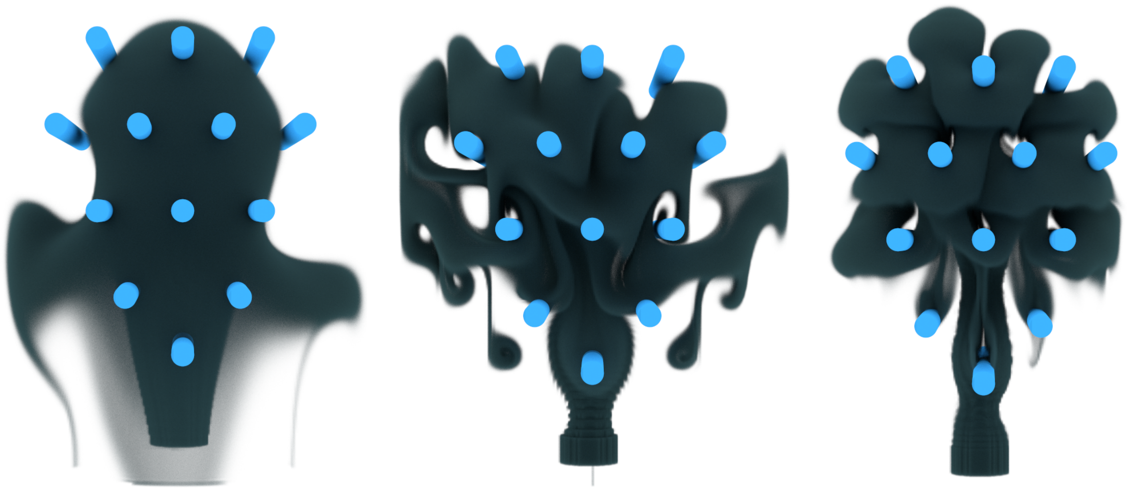


Figure 4.8: Larger bases resolve thinner obstacles, as shown in the Cylinders scene. The basis rank from left to right is $r = 200$, $r = 1000$, and $r = 7000$.

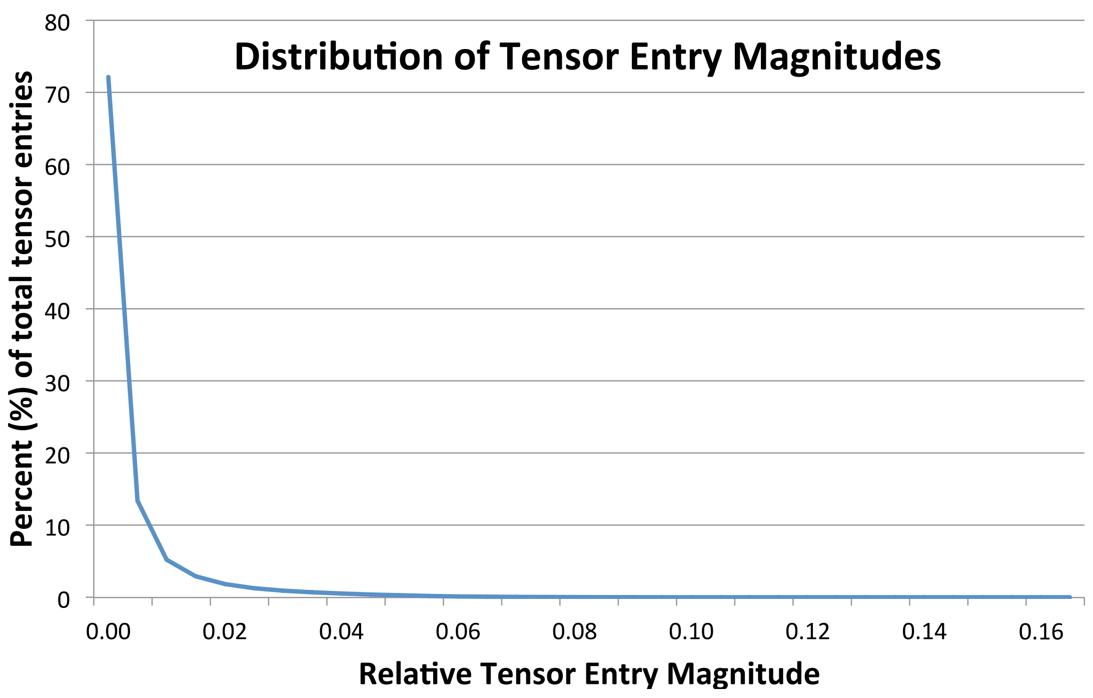


Figure 4.9: Distribution of magnitudes in the advection tensor. Most of the entries have values that are near-zero, and can be discarded without significantly influencing the overall fluid motion.

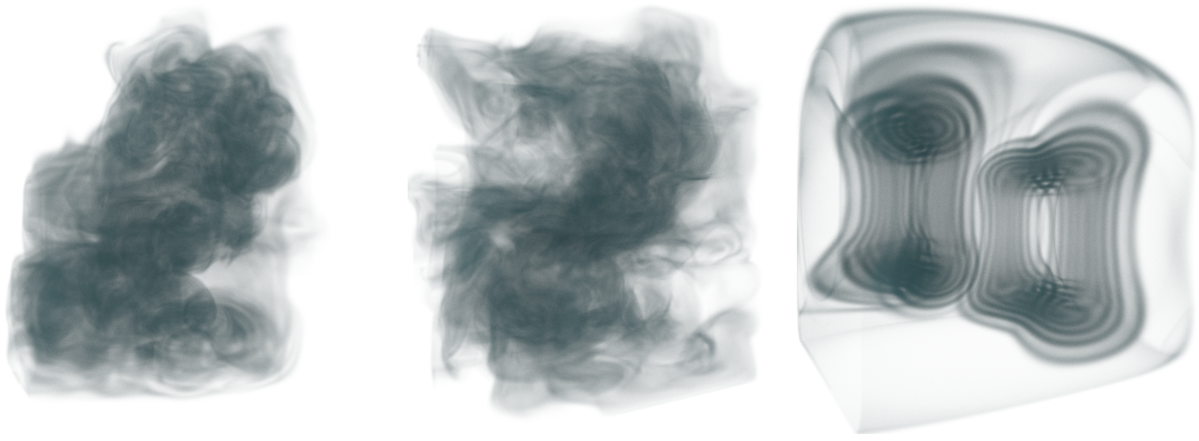


Figure 4.10: From left to right, 0%, 92%, 100% of the smallest tensor entries are discarded in an $r = 7000$ simulation. At 100%, no energy is transferred between basis functions, which creates a static velocity field.

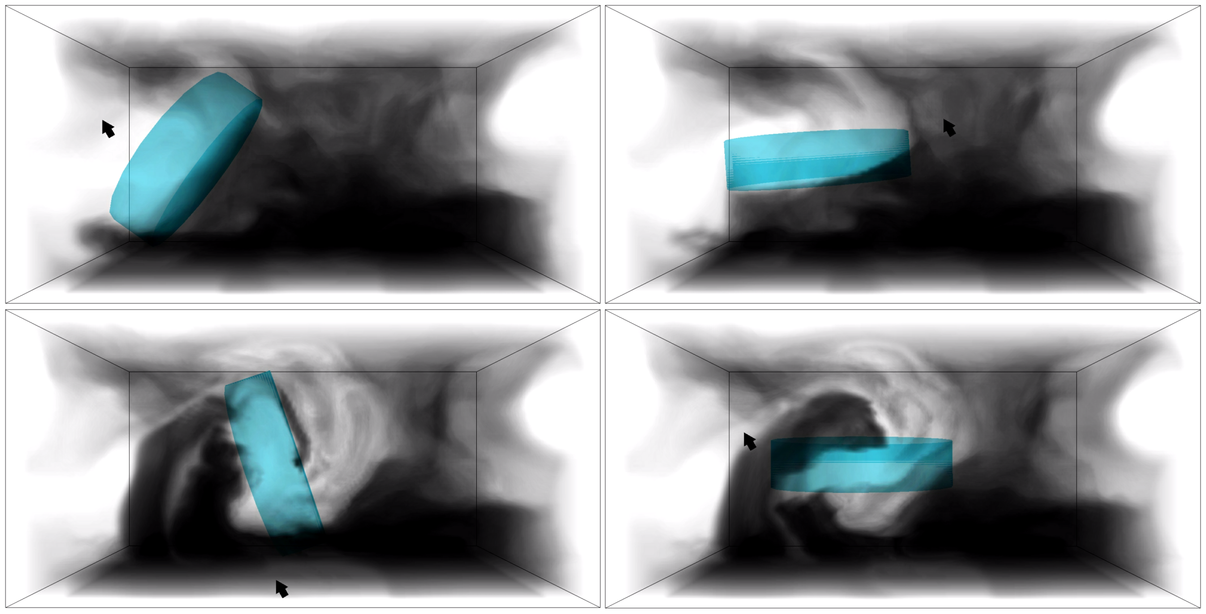


Figure 4.11: INTERACTIVE example, with $r = 1000$ basis functions and two Neumann walls.

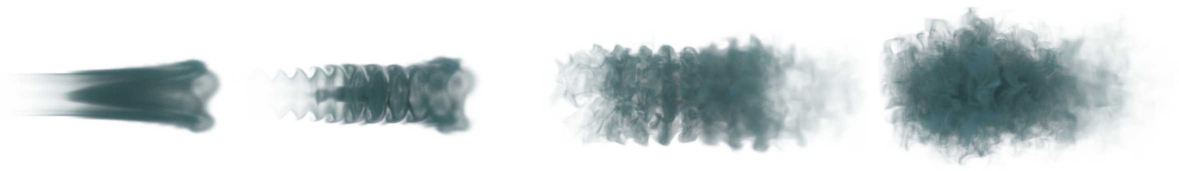


Figure 4.12: We introduce a directability parameter c that reweights the advection tensor according to Eqn. 4.36 and uses the function $b_k = (1 + c|\mathbf{k}|^2)$. Each image in the sequence shows the same simulation timestep, but with a different setting for c. From left to right, the settings are $c = 0$ (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.

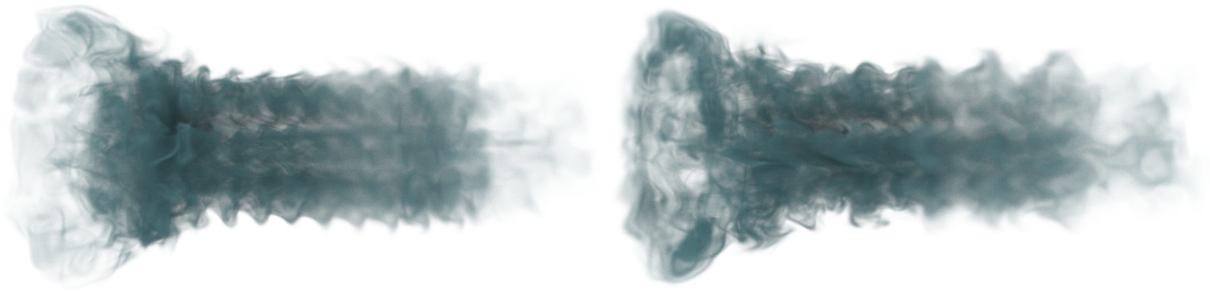


Figure 4.13: Using different tensor reweighting schemes, e.g. $b_k = -(1+c|\mathbf{k}|^2)$, a wider variety of flows are observed. Above, reweighted versions of the same simulation frame as Fig. 4.12 are shown.

Chapter 5

Eigenfluids in Polar and Spherical Coordinates

In this chapter, I introduce the Eigenfluid algorithm [1] in both polar and spherical coordinates. First, I will begin with 2D polar coordinates. The basis functions are derived, to support both fast transformations and the divergence-free condition. Next, an orthogonalization method is introduced to extend the basis functions to non-orthogonal cases. The basis functions are extended to the surface of a sphere, and full 3D spherical coordinates. Finally, the results are presented.

5.1 Eigenfluids in Polar Coordinates

5.1.1 The Polar Coordinate System

In this section, I introduce the polar coordinate system and the differential operators in this coordinate system.

The polar coordinate system is shown in figure 5.13. The basic transformations be-

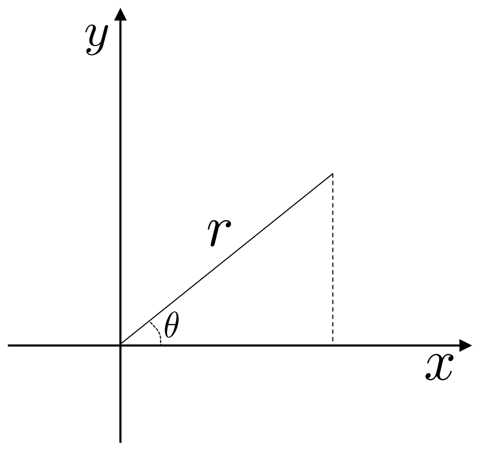


Figure 5.1: The polar coordinate system.

tween the polar coordinate system and the cartesian coordinate system are the following:

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta), \end{aligned} \tag{5.1}$$

and

$$\begin{aligned} r &= \sqrt{x^2+y^2} \\ \theta &= \operatorname{atan2}(y,x). \end{aligned} \tag{5.2}$$

The transformations between vector fields in the polar and cartesian coordinate system are the following:

$$\begin{bmatrix} \mathbf{u}_r \\ \mathbf{u}_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \end{bmatrix} \tag{5.3}$$

and

$$\begin{bmatrix}\mathbf{u}_x \\ \mathbf{u}_y\end{bmatrix} = \begin{bmatrix}\cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}\mathbf{u}_r \\ \mathbf{u}_\theta\end{bmatrix}. \tag{5.4}$$

Assuming a vector field $\mathbf{f}$ and a scalar field f, several important differential operators in polar coordinates can be expressed as:

$$\begin{aligned}
&\nabla^2 f(r,\theta) = \frac{\partial^2 f}{\partial r^2} + \frac{1}{r}\frac{\partial f}{\partial r} + \frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2} \\
&\nabla \cdot \mathbf{f} = \frac{1}{r}\left(\mathbf{f}_r + r\frac{\partial \mathbf{f}_r}{\partial r} + \frac{\partial \mathbf{f}_\theta}{\partial \theta}\right) \\
&\nabla \cdot \mathbf{f} = -\frac{1}{r}\sin(\theta)\frac{\partial \mathbf{f}_x}{\partial \theta} + \cos(\theta)\frac{\partial \mathbf{f}_x}{\partial r} + \frac{1}{r}\cos(\theta)\frac{\partial \mathbf{f}_y}{\partial \theta} + \sin(\theta)\frac{\partial \mathbf{f}_y}{\partial r} \\
&\nabla \times \mathbf{f} = \frac{1}{r}\left(\frac{\partial (r\mathbf{f}_\theta)}{\partial r} - \frac{\partial \mathbf{f}_r}{\partial \theta}\right).
\end{aligned} \tag{5.5}$$

Defining the simulation domain (Ω) as a unit circle, the integration of a scalar function in this domain can be computed as:

$$\int_\Omega f(r,\theta)d\Omega = \int_{r=0}^{1}\int_{\theta=0}^{2\pi} f(r,\theta)r dr d\theta. \tag{5.6}$$

The first step of simulating Eigenfluids in polar coordinates is to find appropriate velocity basis functions. As shown in [1], a desirable basis function should satisfy the following three conditions:

- Divergence-freeness
- Support fast transformations

- Orthogonality

As shown in appendix A, it is impossible to derive vector Laplacian eigenfunctions in polar coordinates. However, if the orthogonality condition is relaxed, it is possible to derive a set of divergence vector basis functions.

5.1.2 Generalized Divergence-free Vector Basis Functions

First of all, constructing the vector basis functions in polar components $(\mathbf{u}_r, \mathbf{u}_\theta)$ is better than in Cartesian components $(\mathbf{u}_x, \mathbf{u}_y)$, because it is easier to specify the boundary conditions along the border $(r = 1)$ in polar coordinates. Therefore, I will derive the basis functions in polar components.
Assuming both components of the velocity are separable functions:

$$\begin{aligned} \mathbf{u}_r &= A(r)\sin(n\theta) \\ \mathbf{u}_\theta &= C(r)\cos(n\theta), \end{aligned} \tag{5.7}$$

the divergence-free condition becomes:

$$\begin{aligned} &\nabla \cdot \mathbf{u} = (\mathbf{u}_r + r\frac{\partial \mathbf{u}_r}{\partial r} + \frac{\partial \mathbf{u}_\theta}{\partial \theta}) = 0 \\ &A + r\frac{\partial A}{\partial r} - nC = 0. \end{aligned} \tag{5.8}$$

From this observation, we can derive another set of possible basis functions:

$$\begin{aligned} \mathbf{u}_r &= A(r)\cos(n\theta) \\ \mathbf{u}_\theta &= C(r)\sin(n\theta). \end{aligned} \tag{5.9}$$

Given these assumptions, the radial terms $A(r)$ and $C(r)$ can be arbitrary functions as long as equation 5.8 is satisfied. As for fast transformations, because θ components are trigonometric functions, the only requirement is that $A(r), C(r)$ should satisfy the fast transformation. For example, we can choose $\sin, \cos$ function along the radial direction. Before choosing the specific form of functions along the radial direction, boundary conditions at $r = 1$ and $r = 0$ are needed to be considered.

Boundary Conditions

First, I will discuss the Dirichlet and Neumann boundary condition at the border. The Dirichlet boundary condition can be expressed as:

$$\mathbf{u}_r(r = 1) = 0, \tag{5.10}$$

which can be satisfied by finding functions that $A(r = 1) = 0$. Similarly, Neumann boundary condition is the following,

$$\left.\frac{d\mathbf{u}_r}{dr}\right|_{r=1} = \sin(n\theta)\left.\frac{dA(r)}{dr}\right|_{r=1} = 0, \tag{5.11}$$

which can be satisfied by finding function that $A'(r = 1) = 0$. To prevent a singularity at the center($r = 0$), as shown in [23], an additional boundary condition should be satisfied.

Assume we have a velocity field in Cartesian coordinates. To avoid the singularity at the center, assume we have a well-defined velocity at the pole in Cartesian coordinates: $\mathbf{u}_x(0, 0), \mathbf{u}_y(0, 0)$. Applying the velocity transformations at the center to equation 5.3

yields:

$$\begin{aligned} \mathbf{u}_r(r=0) &= \mathbf{u}_x(0,0)\cos(\theta) + \mathbf{u}_y(0,0)\sin(\theta) \\ \mathbf{u}_\theta(r=0) &= \mathbf{u}_y(0,0)\cos(\theta) - \mathbf{u}_x(0,0)\sin(\theta). \end{aligned} \tag{5.12}$$

This implies:

$$\frac{\partial \mathbf{u}_r}{\partial \theta} = \mathbf{u}_\theta, \quad \frac{\partial \mathbf{u}_\theta}{\partial \theta} = -\mathbf{u}_r, \quad r = 0. \tag{5.13}$$

Taking a derivative from equation 5.13, we have

$$\frac{\partial^2 \mathbf{u}_r}{\partial \theta^2} = -\mathbf{u}_r, \quad r = 0, \tag{5.14}$$

which implies the θ component of $\mathbf{u}_r$ at $r = 0$ should be a wave function with wavenumber equal to 1. The θ component $\mathbf{u}_\theta$ should then be equal to $\frac{\partial \mathbf{u}_r}{\partial \theta}$ at the center. Assuming we have a velocity field that is non-zero at the center, then the boundary condition in equation 5.13 should be satisfied. This implies n in equation 5.7 and 5.9 should be 1 when $A(r=0) \neq 0$. On the other hand, if $A(r=0) = 0, C(r=0) = 0$, this constraint does not apply.

Finally, the velocity field should be continuous along θ direction. This requires a periodic boundary condition along θ direction, which is the following:

$$\begin{aligned} A(r)\cos(n\theta) &= A(r)\cos(n\theta + 2n\pi) \\ C(r)\sin(n\theta) &= C(r)\sin(n\theta + 2n\pi). \end{aligned} \tag{5.15}$$

This can be trivially satisfied by requiring the wavenumber along θ direction to be an integer.

Considering the boundary conditions in equations 5.10, 5.11 and 5.13, I propose the following basis functions, which I will call principal basis functions.

Principal Basis Functions

The principal basis functions take sine mode along the radial direction. As a result, it is zero at the center. Both sine and cosine mode can be chosen along the angle direction, therefore we have two different choices. We use Φ to denote the principal basis functions, with a superscript denoting the index:

$$\begin{aligned} \Phi^0_r(r,\theta) &= \sin(i_1\pi r)\sin(i_2\theta), \quad i_1 > 0,\ i_2 > 0 \\ \Phi^0_\theta(r,\theta) &= \frac{1}{i_2}(\sin(i_1\pi r) + i_1\pi r\cos(i_1\pi r))\cos(i_2\theta), \end{aligned} \tag{5.16}$$

$$\begin{aligned} \Phi^1_r(r,\theta) &= \sin(i_1\pi r)\cos(i_2\theta), \quad i_1 > 0,\ i_2 > 0 \\ \Phi^1_\theta(r,\theta) &= -\frac{1}{i_2}(\sin(i_1\pi r) + i_1\pi r\cos(i_1\pi r))\sin(i_2\theta). \end{aligned} \tag{5.17}$$

Both basis functions are zero at the center

$$\begin{aligned} \Phi^0_r(0,\theta) = 0, \quad \Phi^0_\theta(0,\theta) = 0 \\ \Phi^1_r(0,\theta) = 0, \quad \Phi^1_\theta(0,\theta) = 0, \end{aligned} \tag{5.18}$$

and therefore they satisfy the boundary conditions in equation 5.13. The boundary condition at $r = 1$ can be controlled by the first wavenumber i_1. For example, when $i_1 \in \mathbb{Z}$, it satisfies the Dirichlet boundary condition (5.10). Similar to [1], the Neumann boundary condition can be obtained by offsetting the integer wavenumber by 0.5: $i_1 \in$

$\mathbb{Z}^+ - 0.5$. In figure 5.2 and figure 5.3, I show some visualizations of Φ^0 and Φ^1 with Dirichlet boundary conditions.

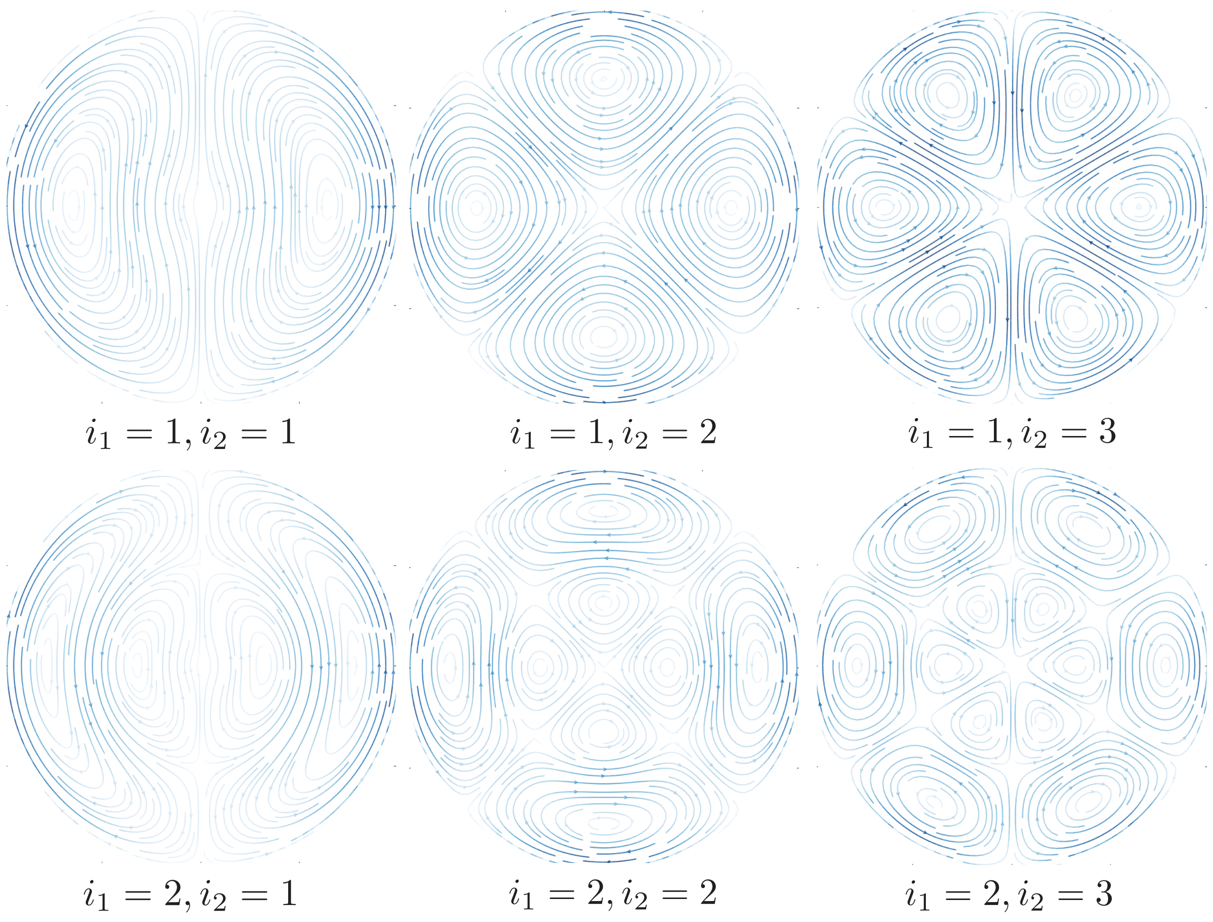


Figure 5.2: Basis function Φ^0 with Dirichlet boundary conditions.

As we can see, increasing the wavenumber along the radial direction(i_1) captures the higher frequency components along the radial direction. Increasing the wavenumber along the angle direction (i_2) captures the higher frequency components along the angular direction. It may seem Φ^1 can be obtained by rotating the basis function Φ^0 along the angle direction. However, this is impossible because the rotation angle is different for different wavenumber i_2. Therefore, basis functions Φ^1 complement Φ^0 in a non-trivial way. For this reason, we use both basis functions in our simulation for completeness.

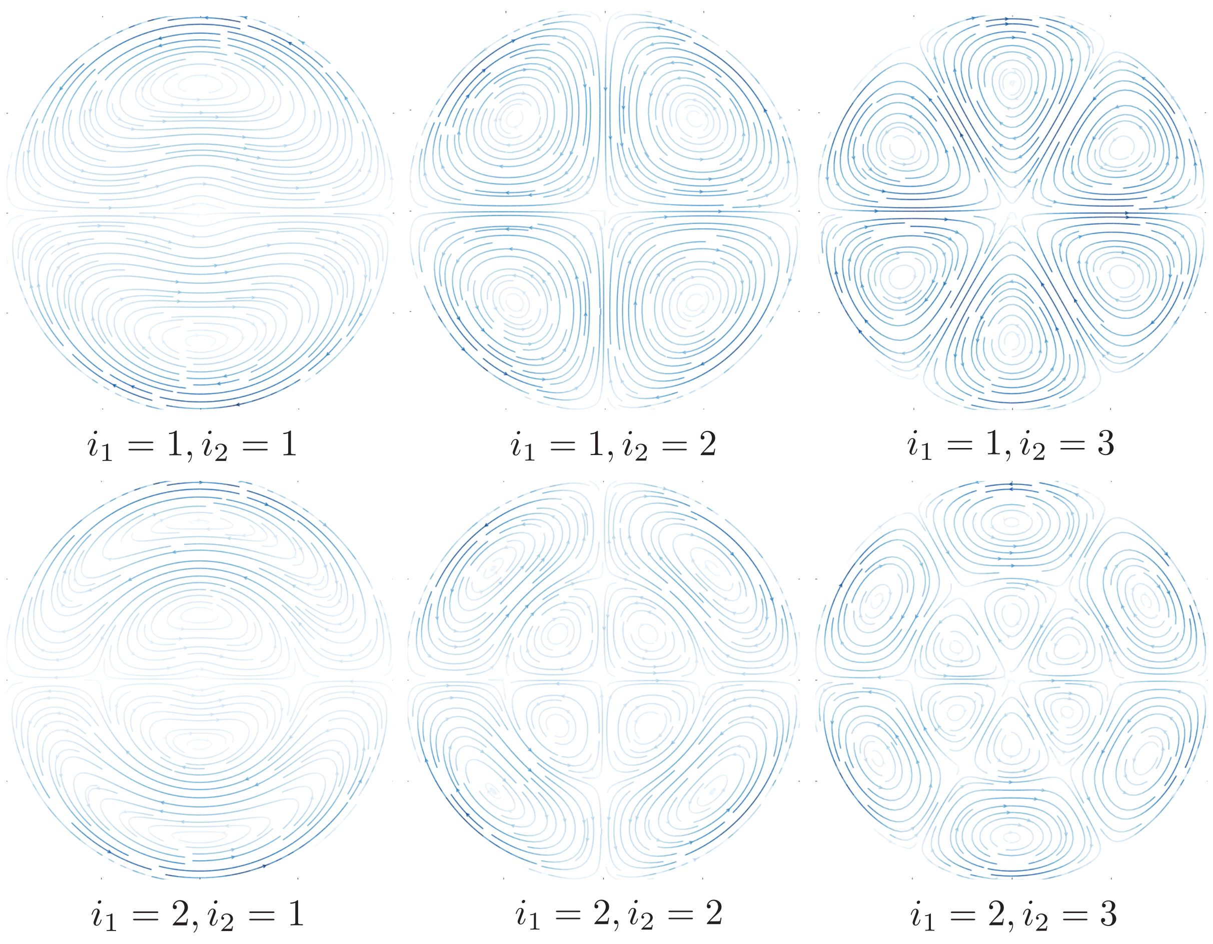
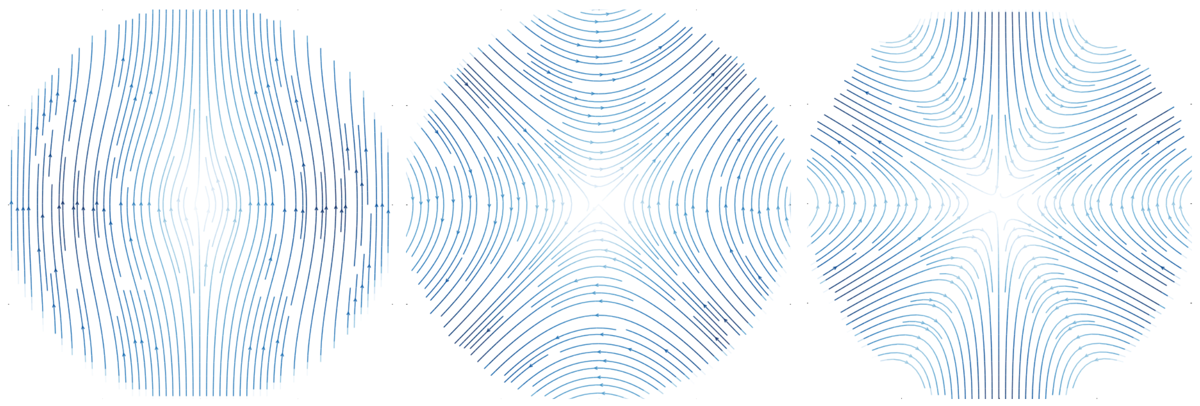


Figure 5.3: Basis function Φ^1 with Dirichlet boundary conditions.

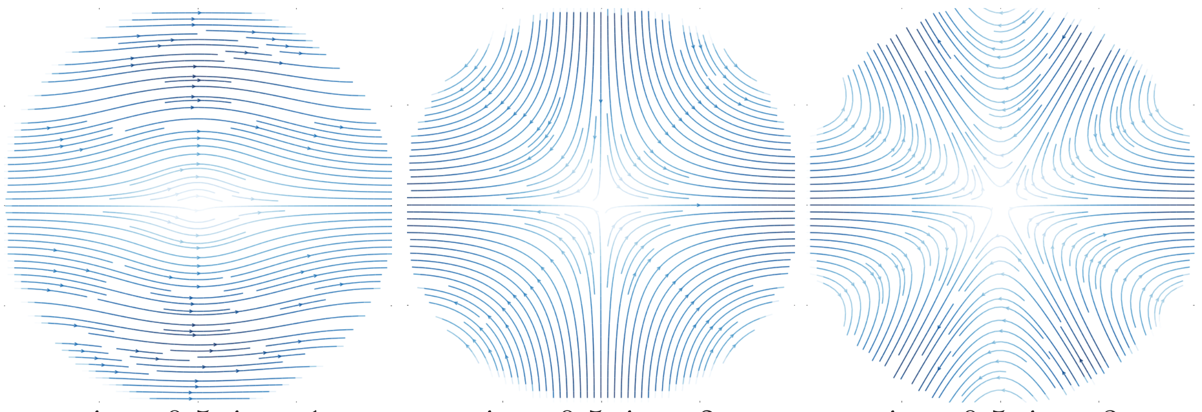
The Neumann boundary condition can be obtained by offsetting the integral wavenumber i_1 by 0.5. The visualization is shown in figure 5.4:

The basis functions satisfy all the boundary conditions and is smooth in the disk domain. However, since it is zero at the pole, the fluid cannot flow past the center if we use this basis function. To resolve this issue, I propose a set of enrichment basis functions, which are non-zero at the center.



$i_1 = 0.5, i_2 = 1$ $i_1 = 0.5, i_2 = 2$ $i_1 = 0.5, i_2 = 3$

(a) Basis functions Φ^0 with Neumann boundary conditions.



$i_1 = 0.5, i_2 = 1$ $i_1 = 0.5, i_2 = 2$ $i_1 = 0.5, i_2 = 3$

(b) Basis functions Φ^1 with Neumann boundary conditions.

Figure 5.4: Visualizations of basis functions Φ^0 and Φ^1 with Neumann boundary conditions.

Enrichment Basis Functions

The enrichment basis functions should be non-zero at the center. To this end, we select the cosine function along the radial direction, because it is non-zero at the center. Similar to the principal basis functions, we have two different choices along the angle direction, therefore we have two sets of enrichment basis functions. We use Ψ to denote the enrichment basis functions, with a superscript as the index:

$$\begin{aligned}\Psi_r^0 &= \cos(i_1\pi r)\sin(\theta),\ i_1 \geq 0 \\ \Psi_\theta^0 &= (\cos(i_1\pi r) - i_1\pi r\sin(i_1\pi r))\cos(\theta),\end{aligned} \tag{5.19}$$

$$\begin{aligned}\Psi_r^1 &= \cos(i_1\pi r)\cos(\theta),\ i_1 \geq 0 \\ \Psi_\theta^1 &= (-\cos(i_1\pi r) + i_1\pi r\sin(i_1\pi r))\sin(\theta),\end{aligned} \tag{5.20}$$

Both enrichment basis functions are non-zero at the center:

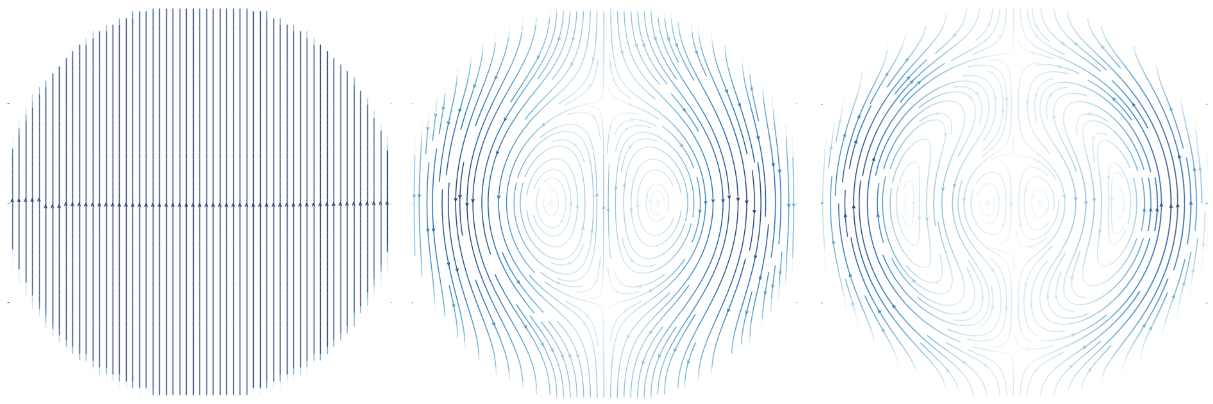
$$\begin{aligned}\Psi_r^0(r=0) &= \sin(\theta) \\ \Psi_\theta^0(r=0) &= \cos(\theta) \\ \Psi_r^1(r=0) &= \cos(\theta) \\ \Psi_\theta^1(r=0) &= -\sin(\theta).\end{aligned} \tag{5.21}$$

Additionally, they satisfy the center boundary condition as shown in equation 5.13. If we compute the velocity at the pole in the Cartesian coordinates, we have

$$\begin{aligned}\mathbf{u}_x^0 &= 0,\ \mathbf{u}_y^0 = 1, \\ \mathbf{u}_x^1 &= 1,\ \mathbf{u}_y^1 = 0,\end{aligned} \tag{5.22}$$

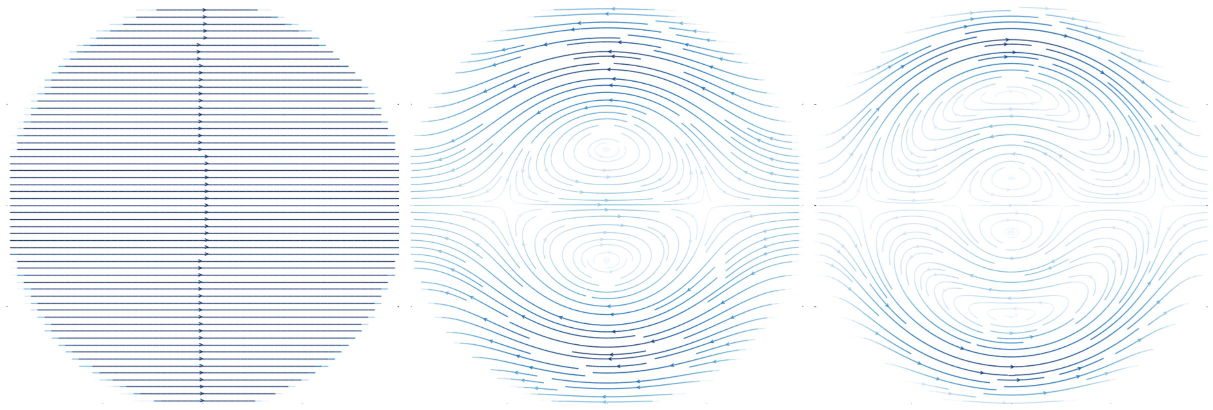
where $\mathbf{u}^0, \mathbf{u}^1$ correspond to Ψ^0 and Ψ^1. This corresponds to unit vectors along the y and x directions in Cartesian coordinates. Therefore, these two basis functions are sufficient to resolve any velocity at the center.

Finally, to capture the circular motion of the fluid around the center, I add the



$i_1 = 1, i_2 = 1$ $i_1 = 2, i_2 = 1$ $i_1 = 3, i_2 = 1$

(a) Basis functions Ψ^0 with Neumann boundary conditions.



$i_1 = 1, i_2 = 1$ $i_1 = 2, i_2 = 1$ $i_1 = 3, i_2 = 1$

(b) Basis functions Ψ^1 with Neumann boundary conditions.

Figure 5.5: Visualizations of enrichment basis functions Ψ^0 and Ψ^1 with Neumann boundary conditions.

following enrichment basis functions:

$$\begin{aligned} \Psi_r^2 &= 0 \\ \Psi_\theta^2 &= \sin(i_1 \pi r),\ i_1 > 0. \end{aligned} \tag{5.23}$$

Fast Transformations and Discretization

The principal basis functions and enrichment basis functions all support fast transformations. The basis functions are expressed in polar coordinates, therefore we discretized the basis functions on a uniform polar grid as shown in figure 5.6.

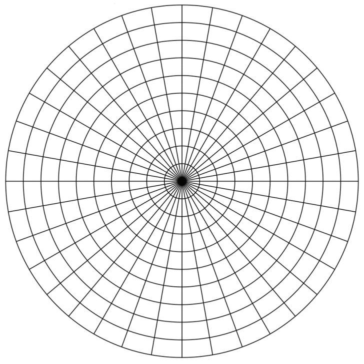


Figure 5.6: The polar discretization grid.

The velocity reconstruction and force projection are then performed on this uniform polar grid. For example, to reconstruct the r component of basis function Φ^0 as shown in equation 5.16, we apply

$$\mathbf{u} = \sum_{i=1}^{r} \mathbf{w}_i \sin(i_1 \pi r) \sin(i_2 \theta), \tag{5.24}$$

which can be done by first putting the coefficients into the corresponding position in the polar grid, and then performing an inverse DST along the r direction, performing an inverse DST along the θ direction. For force projection, assuming the force field is in a polar grid, we first need to weight the force field by r along the radial direction, due to the r which appears in the integration formula in equation 5.6. Then, the force is transformed by a DST along r direction and a DST along θ direction. Finally, the projection coefficients on each basis can be extracted from the corresponding position in the polar grid.

Finally, to advect the scalar density field, we interpolate the velocity on the polar grid onto the Cartesian grid using equation 5.22. Similarly, a force field on the Cartesian grid is first interpolated onto the polar grid, and then the force projection is performed. Unlike the interpolation of Stam advection [10], this interpolation does not affect the dynamics of the Eigenfluid. The density field is also discretized on a Cartesian grid, which is then advected by the velocity field obtained from interpolation.

5.2 Polar Eigenfluids Dynamics

We use the same dynamic framework as in [15] and [1]. However, as the basis functions are not orthogonal, we need to extract a set of orthogonal basis functions from them. The extracted orthogonal basis functions are then used in the simulation. The orthogonal basis functions will be linear combinations of principal and enrichment basis functions Φ and Ψ, therefore it is also divergence-free.

5.2.1 Orthogonalization

To extract a set of divergence-free basis functions, one standard method is the Gram–Schmidt process. Let's say we have a set of r non-orthogonal basis functions Φ_i. The extracted orthogonal basis functions are denoted as $\mathbf{v}_i$. We use $\mathbf{V}$ and $\boldsymbol{\phi}$ to denote the vector formed by these basis functions. The Gram–Schmidt process can be described as:

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \vdots \\ \mathbf{v}_r \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & a_{r3} & \dots & a_{rr} \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \vdots \\ \Phi_r \end{bmatrix}, \tag{5.25}$$

$$\mathbf{V} = \mathbf{A\Phi},$$

where $\mathbf{A}$ is a block triangular matrix that denotes the transformation between a set of non-orthogonal bases Φ_i and a set of orthogonal bases $\mathbf{v}_i$. Therefore the set of orthogonal basis functions is a linear transformation of the original basis functions with a transfer matrix $\mathbf{A}$. The Gram–Schmidt algorithm can be used to compute the transfer matrix $\mathbf{A}$. However, here I proposed a slightly different approach.

Computing the Transfer Matrix A

To compute the transfer matrix $\mathbf{A}$, assuming $\Phi_i,\ i = 1, 2, 3, ..., r$ are linearly independent, we use $\mathbf{H}$ to denote the inner product matrix between all basis functions: $\mathbf{H}_{ij} = \langle \Phi_i, \Phi_j \rangle$. The inner product matrix should be symmetric positive definite because Φ_i are linearly independent. $\mathbf{V}$ is a set of orthogonal basis functions, therefore $\mathbf{VV}^T = \mathbf{I}$. We have:

$$\mathbf{A\Phi\Phi}^T\mathbf{A}^T = \mathbf{AHA}^T = \mathbf{I}. \tag{5.26}$$

The matrix $\mathbf{H}$ is symmetric. It has eigen-decomposition $\mathbf{UDU}^T = \mathbf{H}$. Matrix $\mathbf{U}$ is orthogonal and $\mathbf{D}$ is diagonal with diagonal entries as eigenvalues. This eigen-decompoition can be efficiently computed via a symmetric QR decomposition, see from [28]. We use an efficient implementation in *Eigen* matrix library [88]. The transfer matrix can then be $\mathbf{A} = \mathbf{D}^{-\frac{1}{2}}\mathbf{U}^T$.

Numerically, matrix $\mathbf{H}$ is well-conditioned if $\mathbf{\Phi}$ are close to orthogonal. $\mathbf{H}$ becomes badly-conditioned or can even have zero eigenvalues if $\mathbf{\Phi}$ are far from orthogonal, or has linearly dependent elements. To ensure a well-conditioned transfer matrix $\mathbf{A}$, components correspond to smaller eigenvalues of $\mathbf{H}$ should be discarded, because the inverse of eigenvalues appear in the transfer matrix $\mathbf{A}$. Intuitively, this removes some basis functions that are overlapping with other basis functions. This thresholding on eigenvalues also discards columns of matrix $\mathbf{U}$, therefore the transfer matrix $\mathbf{A}$ is also resized.

Assuming we extract m orthogonal components from $\mathbf{\Phi}$, we have $\mathbf{A} \in \mathbb{R}^{m\times r}$, and $\mathbf{V}$ is a vector of orthogonal basis of size m. The $\mathbf{D} \in \mathbb{R}^{m\times m}$ is a diagonal matrix where only m large eigenvalues are kept. For the basis functions I derived in polar coordinates, 10% of the basis functions are discarded from the set $\mathbf{\Phi}$, or $m = 0.9r$, to keep the condition number of matrix $\mathbf{H}$ below 10.

Advection Tensor Under Orthogonal Basis

To simulate the fluid using orthogonal basis functions, one obvious way is to directly compute the new advection tensor under the orthogonal basis $\mathbf{v}_i$ as a pre-process:

$$\begin{aligned}
\mathfrak{C}(g,h,i) &= \int_0^1 \int_0^{2\pi} (\nabla \times \mathbf{v}_i)\cdot(\mathbf{v}_g \times \mathbf{v}_h)dr d\theta \\
&= \sum_{l,m,n} a_{il}a_{gm}a_{hn} \int_0^1 \int_0^{2\pi} (\nabla \times \Phi_l)\cdot(\Phi_m \times \Phi_n)dr d\theta \\
&= \sum_{l,m,n} a_{gm}a_{hn}a_{il}\mathfrak{C}^o(m,n,l),
\end{aligned} \tag{5.27}$$

where $\mathfrak{C}^o$ is the old advection tensor computed with non-orthogonal bases. Coefficients a_{ij} are entries of the transfer matrix $\mathbf{A}$. The new advection tensor is skew-symmetric:

$$\begin{aligned}
\mathfrak{C}(g,h,i) &= \sum_{l,m,n} a_{gm}a_{hn}a_{il}\mathfrak{C}^o(m,n,l) \\
\mathfrak{C}(h,g,i) &= \sum_{l,m,n} a_{hm}a_{gn}a_{il}\mathfrak{C}^o(m,n,l) = \sum_{l,m,n} a_{gm}a_{hn}a_{il}\mathfrak{C}^o(n,m,l) = -\mathfrak{C}(g,h,i).
\end{aligned} \tag{5.28}$$

Assuming $\mathfrak{C}^o$ is $O(r^3)$ dense, and transformation $\mathbf{A}$ is also $O(r^2)$ dense, this will mostly take $O(r^4)$ to compute. For example, for each index i, computing the contraction $\mathbf{C}_i = \sum_l a_{il}\mathfrak{C}^o(m,n,l) \in \mathbb{R}^{r\times r}$ takes $O(r^3)$. Then we cache $\mathbf{C}_i$ in memory, and compute the tensor entries for all g, h. $\mathfrak{C}(h,g,i) = \sum_{m,n} a_{hm}a_{gn}\mathbf{C}_i$. To do this, we can compute a matrix vector product : $\mathbf{c}_{g,i} = \sum_n a_{gn}\mathbf{C}_i$, which takes $O(r^2)$ for each g. Finally we compute the tensor entry $\mathfrak{C}(h,g,i) = \sum_m a_{hm}\mathbf{c}_{g,i}$ and takes $O(r)$ for each h.

In summary, we need to compute $\mathbf{C}_i$ a total of r times, each takes $O(r^3)$, therefore this part takes $O(r^4)$. Next, we need to compute $\mathbf{c}_{g,i}$ a total of r^2 times, each taking $O(r^2)$, therefore this part also takes $O(r^4)$. Finally, we need to compute $\sum_m a_{hm}\mathbf{c}_{g,i}$ a total of r^3 times, each taking $O(r)$, and this part is also $O(r^4)$. Thus, we can compute the tensor transformation 5.27 in at most $O(r^3)$ complexity with $O(r^2)$ additional memory. Therefore, computing the new tensor given the tensor $\mathfrak{C}^o$ will take $O(r^4)$ computations. A better solution is to apply transfer matrix $\mathbf{A}$ on the fly when computing the advection, without computing the new tensor as a pre-process.

A Modified Conjugate Gradient Solver

We use the trapezoidal update rule to solve for advection. As shown in [1], the equation we need to solve is:

$$(\mathbf{I} - \frac{\Delta t}{2}\mathbf{C})\mathbf{w}^{t+1} = \frac{\Delta t}{2}\mathbf{C}\mathbf{w}^t + \mathbf{w}^t + \mathbf{f}. \tag{5.29}$$

The matrix $\mathbf{C}$ is the contraction $\mathfrak{C} \times_3 \mathbf{w}^t$. By using equation 5.27, we can expand $\mathbf{C}$ in terms of the old advection tensor

$$\begin{aligned} \mathbf{C}_{gh} &= \mathfrak{C} \times_3 \mathbf{w}^t = a_{gm}a_{hn}a_{il}\mathbf{w}_i\mathfrak{C}^o(m,n,l) \\ &= a_{gm}a_{hn}\mathbf{y}_l\mathfrak{C}^o(m,n,l) = a_{gm}a_{hn}\mathfrak{C}^o(m,n,l) \times_3 \mathbf{y} = a_{gm}a_{hn}C^o_{mn} \\ &= a_{gm}\mathbf{C}^o_{mn}a_{hn} = (\mathbf{A}\mathbf{C}^o)_{gn}a_{hn} = (\mathbf{A}\mathbf{C}^o\mathbf{A}^T)_{gh}, \end{aligned} \tag{5.30}$$

where $\mathbf{y} = \mathbf{A}^T\mathbf{w}$, $\mathbf{C}^o = \mathfrak{C}^o \times_3 \mathbf{y}$. Therefore equation 5.29 can be written as

$$(\mathbf{I} - \frac{\Delta t}{2}\mathbf{A}\mathbf{C}^o\mathbf{A}^T)\mathbf{w}^{t+1} = \frac{\Delta t}{2}\mathbf{A}\mathbf{C}^o\mathbf{A}^T\mathbf{w}^t + \mathbf{w}^t + \mathbf{f}. \tag{5.31}$$

So, instead of doing a tensor transformation using equation 5.27, we can still use the old tensor, compute old matrix $\mathbf{C}^o$ with $\mathbf{C}^o = \mathfrak{C}^o \times_3 \mathbf{y}$, and then use matrix $\mathbf{A}\mathbf{C}^o\mathbf{A}^T$ as the new system matrix. This matrix is obviously skew-symmetric, and the system should be energy conserving. The only change here is that three vector matrix products are required for the conjugate gradient solver.

Finally, to compute the fast transformations, we can use the transfer matrix to recover the coefficients of the original basis functions or to project the coefficients of the original basis functions onto the orthogonal basis functions.

Fast Transformations with Transfer Matrix A

The velocity reconstruction is computed as:

$$\mathbf{u} = \sum_i \mathbf{w}_i \mathbf{v}_i = \mathbf{w}^T V = \mathbf{w}^T \mathbf{A} \Phi. \tag{5.32}$$

where $\mathbf{w}$ is a set of basis coefficients. $\mathbf{A}$ is the transformation matrix, $\mathbf{u}$ is the full rank velocity. We already know Φ admits fast transformations. To do fast transformations over the orthogonal velocity basis $\mathbf{V}$, we can first transform the coefficients by computing $\mathbf{w}^T\mathbf{A}$, and then do fast transformations using basis Φ.

The force projection is computed as:

$$\mathbf{w}_i = \langle \mathbf{f}, \mathbf{v}_i \rangle = \langle \mathbf{f}, \sum_k a_{ik} \Phi_k \rangle = \sum_k a_{ik} \langle \mathbf{f}, \Phi_k \rangle, \tag{5.33}$$

$$\mathbf{w} = \mathbf{A} \begin{bmatrix} \langle \mathbf{f}, \Phi_1 \rangle \\ \langle \mathbf{f}, \Phi_2 \rangle \\ \vdots \\ \langle \mathbf{f}, \Phi_r \rangle \end{bmatrix}. \tag{5.34}$$

Therefore we can compute the fast transformations under the Φ basis, and obtain a vector of coefficients, then multiply with the matrix $\mathbf{A}$ to obtain the coefficients under the orthogonal basis functions.

In summary, I dropped the requirement of basis functions being orthogonal. Therefore we can choose any set of divergence-free functions, and use them as bases. This is compatible with the previous work [1].

5.2.2 Quasi-viscosity

The orthogonalized basis functions $\mathbf{v}_i$ are no longer Laplacian eigenfunctions, therefore the Laplacian $\nabla^2\mathbf{u}$ no longer projects the basis to itself. While a Galerkin treatment of viscosity is possible, here we use an approximate quasi-viscosity treatment to this problem.

The point-wise decay of coefficients in Laplacian eigenfluids is

$$w_k^{t+1} = w_k^t e^{\mu\lambda_i \Delta t}, \tag{5.35}$$

where λ_i is the eigenvalue for eigenfunction Φ_i. In the previous work $\lambda_i = -(i_1^2 + i_2^2)$. Therefore λ_i describes the frequency of basis function Φ_i. In this work, we still use λ_i as a measurement of the frequency of the basis functions, even though it is no longer an eigenvalue of the basis functions. This is still valid as increasing i_1, i_2 will lead to higher frequency components. The orthogonal basis functions $\mathbf{v}$ are linear combinations of $\mathbf{\Phi}$. For each orthogonal basis function $\mathbf{v}_i$, I compute a frequency characterization of that basis function using the transfer matrix $\mathbf{A}$,

$$\boldsymbol{\lambda}^* = \mathbf{A}\boldsymbol{\lambda}, \tag{5.36}$$

where $\boldsymbol{\lambda}$ is a vector concatenating all the square wavenumbers $-i_1^2 - i_2^2$. $\boldsymbol{\lambda}^*$ is then a vector of the same size with basis coefficients $\mathbf{w}$, and we then decay each basis coefficients according to:

$$w_k^{t+1} = w_k^t e^{\mu\boldsymbol{\lambda}_i^* \Delta t}. \tag{5.37}$$

I use this to approximately capture viscosity effect for the Eigenfluids in polar and spherical coordinates.

5.3 Eigenfluids in 2D Spherical Coordinates

In this section, I introduce Eigenfluid simulation on the surface of a sphere.

5.3.1 Spherical Coordinate System

First I introduce the spherical coordinate system and several important differential operators.

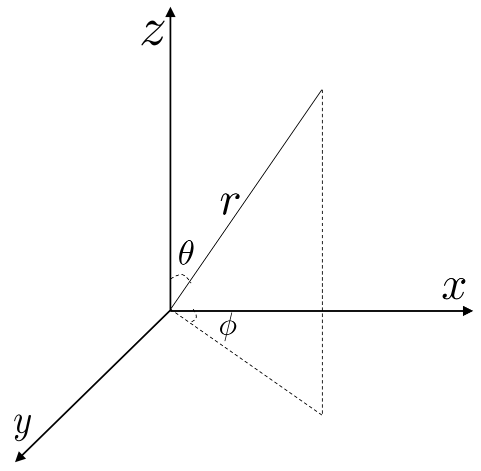


Figure 5.7: The spherical coordinate system.

The polar coordinate system is shown in figure 5.13. The basic transformations between the spherical coordinate system and the Cartesian coordinate system are the following:

$$\begin{aligned} x &= r\sin(\theta)\cos(\phi) \\ y &= r\sin(\theta)\sin(\phi) \\ z &= r\cos(\theta), \end{aligned} \tag{5.38}$$

$$\begin{aligned} r &= \sqrt{x^2+y^2+z^2} \\ \theta &= \operatorname{acos}(\frac{z}{r}) \\ \phi &= \operatorname{atan2}(y,x). \end{aligned} \tag{5.39}$$

Given a vector field in spherical coordinates, the vector field can be transformed to Cartesian coordinates by the following:

$$\begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{bmatrix} = \begin{bmatrix} \sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) & -\sin(\phi) \\ \sin(\theta)\sin(\phi) & \cos(\theta)\sin(\phi) & \cos(\phi) \\ \cos(\theta) & -\sin(\theta) & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_r \\ \mathbf{u}_\theta \\ \mathbf{u}_\phi \end{bmatrix}, \tag{5.40}$$

On the other hand, the inverse of the above transformation is:

$$\begin{bmatrix} \mathbf{u}_r \\ \mathbf{u}_\theta \\ \mathbf{u}_\phi \end{bmatrix} = \begin{bmatrix} \sin(\theta)\cos(\phi) & \sin(\theta)\sin(\phi) & \cos(\theta) \\ \cos(\theta)\cos(\phi) & \cos(\theta)\sin(\phi) & -\sin(\theta) \\ -\sin(\phi) & \cos(\phi) & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{bmatrix}. \tag{5.41}$$

Given a vector field $\mathbf{f}$, a few important differential operators in spherical coordinates can be expressed as:

$$\begin{aligned}
&\nabla \cdot \mathbf{f} = \frac{1}{r^2}\frac{\partial(r^2\mathbf{f}_r)}{\partial r} + \frac{1}{r\sin(\theta)}\frac{\partial}{\partial\theta}(\mathbf{f}_\theta\sin(\theta)) + \frac{1}{r\sin(\theta)}\frac{\partial\mathbf{f}_\phi}{\partial\phi},\\
&\nabla \times \mathbf{f} = \frac{1}{r\sin(\theta)}\left(\frac{\partial}{\partial\theta}(\mathbf{f}_\phi\sin(\theta))\right) - \frac{\partial\mathbf{f}_\theta}{\partial\phi})\hat{\mathbf{r}} +\\
&\frac{1}{r}\left(\frac{1}{\sin(\theta)}\frac{\partial\mathbf{f}_r}{\partial\phi} - \frac{\partial}{\partial r}(r\mathbf{f}_\phi)\right)\hat{\boldsymbol{\theta}} + \frac{1}{r}\left(\frac{\partial}{\partial r}(r\mathbf{f}_\theta) - \frac{\partial\mathbf{f}_r}{\partial\theta}\right)\hat{\boldsymbol{\phi}}.
\end{aligned} \tag{5.42}$$

Assume the simulation domain(Ω) as a unit sphere: $r \in [0,1], \theta \in [0,\pi], \phi \in [0,2\pi)$, the integration of a scalar function in this domain can be computed as:

$$\int_\Omega f(r,\theta,\phi)d\Omega = \int_{r=0}^{1}\int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} f(r,\theta,\phi)r^2\sin(\theta)drd\theta d\phi. \tag{5.43}$$

5.3.2 Divergence-free Vector Basis Functions

While it is possible to compose divergence-free and orthogonal vector basis functions using spherical harmonics (A.3). It is unclear how these basis functions can be generalized to include the entire sphere. Also, fast transformations of spherical harmonics may pose a challenge. Therefore I instead derive new basis functions using trigonometric functions. Similar to the case in polar coordinates, I will begin with the divergence-free condition, and then derive the divergence-free basis functions which support fast transformations. Because the fluid is constrained on the surface, the degrees of freedom are θ and ϕ. Therefore I will derive the vector basis functions expanded in θ and ϕ components.

The divergence-free condition on the surface of a sphere can be expressed as:

$$\begin{aligned}
&\nabla \cdot \mathbf{u} = \frac{1}{r\sin(\theta)}\frac{\partial}{\partial\theta}(\sin(\theta)\mathbf{u}_\theta) + \frac{1}{r\sin(\theta)}\frac{\partial \mathbf{u}_\phi}{\partial\phi} = 0 \\
&\frac{\partial}{\partial\theta}(\sin(\theta)\mathbf{u}_\theta) + \frac{\partial \mathbf{u}_\phi}{\partial\phi} = 0.
\end{aligned} \tag{5.44}$$

Assuming both $\mathbf{u}_\theta$ and $\mathbf{u}_\phi$ are separable functions along θ and ϕ, we have:

$$\nabla \cdot \mathbf{u} = \frac{\partial}{\partial\theta}(\sin(\theta)C(\theta))D(\phi) + E(\theta)\frac{\partial F(\phi)}{\partial\phi}, \tag{5.45}$$

where $\mathbf{u}_\theta = C(\theta)D(\phi), \mathbf{u}_\phi = E(\theta)F(\phi)$. Assuming $D(\phi) = \frac{\partial F(\phi)}{\partial\phi}$ and resolving the ϕ direction, the divergence free condition then becomes:

$$\nabla \cdot \mathbf{u} = \sin(\theta)\frac{\partial C}{\partial\theta} + \cos(\theta)C + E = 0. \tag{5.46}$$

Therefore, we can choose function C that satisfies the fast transformation, and then choose E accordingly. Before we choose the specific form of C, boundary conditions should be specified.

Boundary Conditions

As pointed out in [23], there are two boundary conditions at the north and south pole $\theta = 0, \theta = \pi$:

$$\begin{aligned}
&\frac{\partial \mathbf{u}_\phi}{\partial\phi} = -\mathbf{u}_\theta, \quad \frac{\partial \mathbf{u}_\theta}{\partial\phi} = \mathbf{u}_\phi, \quad \theta = 0 \\
&\frac{\partial \mathbf{u}_\phi}{\partial\phi} = \mathbf{u}_\theta, \quad \frac{\partial \mathbf{u}_\theta}{\partial\phi} = -\mathbf{u}_\phi, \quad \theta = \pi.
\end{aligned} \tag{5.47}$$

This implies the wavenumbers of $\mathbf{u}_\phi, \mathbf{u}_\theta$ along ϕ direction are both one, albeit in the opposite direction along $\theta = 0, \theta = \pi$. Another boundary condition is the periodic boundary condition along the direction ϕ, which can be easily satisfied by using an integer as the second wavenumber. Because the fluid flows on a closed surface, there is no need for Dirichlet or Neumann boundary conditions.

Principal Basis Functions

For the principal basis functions, we choose sine modes along the θ direction. Therefore the basis functions are zero at both the north and the south pole, and the boundary conditions there are trivially satisfied:

$$\begin{aligned}\Phi^0_\theta &= \sin(i_1\theta)\cos(i_2\phi)\\ \Phi^0_\phi &= -\frac{1}{i_2}(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta))\sin(i_2\phi),\ i_1 > 0,\ i_2 > 0,\end{aligned} \tag{5.48}$$

$$\begin{aligned}\Phi^1_\theta &= \sin(i_1\theta)\sin(i_2\phi)\\ \Phi^1_\phi &= \frac{1}{i_2}(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta))\cos(i_2\phi),\ i_1 > 0,\ i_2 > 0,\end{aligned} \tag{5.49}$$

$$\begin{aligned}\Phi^2_\theta &= 0\\ \Phi^2_\phi &= \sin(i_1\theta),\ i_1 > 0.\end{aligned} \tag{5.50}$$

To capture the circular flow, one additional basis function Φ^2 is added. It corresponds to flow along the latitude direction. Similar to before, for the fluid to flow past the north and south pole, I introduce the enrichment basis functions as follows.

Enrichment Basis Functions

The enrichment basis functions take cosine modes along the θ direction. The basis functions are non-zero at both poles. The second wavenumber is set to one to satisfy the boundary condition at the pole. They are

$$\begin{aligned}\Psi^0_\theta &= \cos(i_1\theta)\cos(\phi)\\ \Psi^0_\phi &= (-\cos(\theta)\cos(i_1\theta) + i_1\sin(\theta)\sin(i_1\theta))\sin(\phi), \quad i_1 \geq 0,\end{aligned} \tag{5.51}$$

$$\begin{aligned}\Psi^1_\theta &= \cos(i_1\theta)\sin(\phi)\\ \Psi^1_\phi &= (\cos(\theta)\cos(i_1\theta) - i_1\sin(\theta)\sin(i_1\theta))\cos(\phi), \quad i_1 \geq 0.\end{aligned} \tag{5.52}$$

At the north pole ($\theta = 0$), the enrichment basis functions satisfy the boundary condition:

$$\begin{aligned}\Psi^0_\theta(\theta = 0) &= \cos(\phi)\\ \Psi^0_\phi(\theta = 0) &= -\sin(\phi)\\ \Psi^1_\theta(\theta = 0) &= \sin(\phi)\\ \Psi^1_\phi(\theta = 0) &= \cos(\phi).\end{aligned} \tag{5.53}$$

At the south pole ($\theta = \pi$), the enrichment basis functions satisfies the boundary condition:

$$\begin{aligned}
\Psi^0_\theta(\theta = \pi) &= \cos(i_1\pi)\cos(\phi) \\
\Psi^0_\phi(\theta = \pi) &= \cos(i_1\pi)\sin(\phi) \\
\Psi^1_\theta(\theta = \pi) &= \cos(i_1\pi)\sin(\phi) \\
\Psi^1_\phi(\theta = \pi) &= -\cos(i_1\pi)\cos(\phi).
\end{aligned} \tag{5.54}$$

We can compute the Cartesian velocity at both poles for both basis functions. For Ψ^0, we have:

$$\begin{aligned}
&\mathbf{u}_x(\theta = 0) = 1, \quad \mathbf{u}_y(\theta = 0) = 0 \\
&\mathbf{u}_x(\theta = \pi) = (-1)^{i_1+1}, \quad \mathbf{u}_y(\theta = \pi) = 0.
\end{aligned} \tag{5.55}$$

For Ψ^1, we have:

$$\begin{aligned}
&\mathbf{u}_x(\theta = 0) = 0, \quad \mathbf{u}_y(\theta = 0) = 1 \\
&\mathbf{u}_x(\theta = \pi) = 0, \quad \mathbf{u}_y(\theta = \pi) = (-1)^{i_1+1}.
\end{aligned} \tag{5.56}$$

Similar to the enrichment basis functions in 2D, the enrichment basis function presented here form two orthogonal directions at both poles.

5.3.3 Fast Transformations and Discretization

We discretize the velocity field on an even 2D θ, ϕ grid, which maps to a surface of a sphere as shown below:

In contrast to the 2D polar case, I do not interpolate the velocity field onto a Cartesian grid. The density field is also discretized on a uniform sphere surface grid. The density is then advected using the same algorithm as in [23]. The density field is then used as a

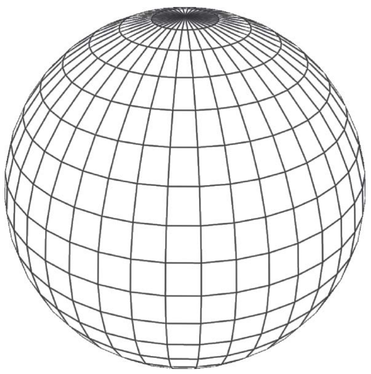


Figure 5.8: Uniform grid on the surface of a sphere.

texture map for a sphere.

Similar to the polar 2D case, both the principal and enrichment basis functions support the fast transformation. As we can see the basis function 5.48 and 5.49 require multiple transformations. The following two observations can be leveraged to reduce computation.

First of all, when transforming terms like $(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta))\sin(i_2\phi)$, one way is to do the transformation separately:

$$\begin{aligned}&\sum \mathbf{w}_i(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta))\sin(i_2\phi) = \\ &\cos(\theta)\sum \mathbf{w}_i\sin(i_1\theta)\sin(i_2\phi) + \sin(\theta)\sum \mathbf{w}_i\ i_1\cos(i_1\theta)\sin(i_2\phi).\end{aligned} \tag{5.57}$$

This corresponds to doing two 2D DCTs. However, 2D DCTs are separable, meaning doing a 2D DCT is equal to doing 1D DCTs along each dimension. Therefore a better way is to first to do a batch of 1D DCTs that transforms $(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta))$ along the first dimension, and then do a second batch of 1D DCTs that transforms $\sin(i_2\phi)$ along the second dimension. This corresponds to transforming the first line of equation 5.57 directly, and saves one DCT compared to doing them separately and adding the result together.

The second observation is that when transforming terms like $\cos(\theta)\sin(i_1\theta)$, one way to do this is to first use DST to transform $\sin(i_1\theta)$, and then weight the results by the scalar field $\cos(\theta)$. A better way is by observing:

$$\sum \mathbf{w}_i \cos(\theta)\sin(i_1\theta) = \frac{1}{2}\sum \mathbf{w}_i \sin((i_1+1)\theta) + \mathbf{w}_i \sin((i_1-1)\theta). \tag{5.58}$$

Therefore we can put coefficients $\frac{1}{2}\mathbf{w}_i$ in spots corresponding wavenumber i_1+1 and i_1-1, and then do a single transformation of DST, that would be equal to doing a DST and then weighting the results by weighting with $\cos(\theta)$.

In summary, in the case of transforming the first part of ϕ term, $(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta))\sin(i_2\phi)$, we can do the following:

$$\begin{aligned}&\sum \mathbf{w}_i(\cos(\theta)\sin(i_1\theta) + i_1\sin(\theta)\cos(i_1\theta)) = \\ &\frac{1}{2}\sum \mathbf{w}_i(\sin((i_1+1)\theta) + \sin((i_1-1)\theta) + i_1\sin((i_1+1)\theta) - i_1\sin((i_1-1)\theta)).\end{aligned} \tag{5.59}$$

Therefore, we can put the coefficients $\frac{1}{2}\mathbf{w}_i(1+i_1)$, $\frac{1}{2}\mathbf{w}_i(1-i_1)$ into spots corresponding to i_1+1, i_1-1 respectively, and do a single transformation instead.

5.4 Eigenfluids in 3D Spherical Coordinates

In this section, I will present the Eigenfluid simulation in full 3D spherical coordinates.

5.4.1 General Forms of Vector Basis Functions

Similar to the 2D case, the divergence-free condition should be satisfied when deriving the basis functions in full 3D spherical coordinates. There are multiple ways that this constraint could be satisfied, which leads to different basis functions. In this section, I will derive 3 different forms that will be used to derive the final basis functions. First I will begin with the divergence-free condition shown in equation 5.42:

$$\begin{aligned}\nabla \cdot \mathbf{u} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2\mathbf{u}_r) + \frac{1}{r\sin(\theta)}\frac{\partial}{\partial \theta}(\sin(\theta)\mathbf{u}_\theta) + \frac{1}{r\sin(\theta)}\frac{\partial \mathbf{u}_\phi}{\partial \phi} = 0 \\ \frac{1}{r^2\sin(\theta)}\left(\sin(\theta)\frac{\partial}{\partial r}(r^2\mathbf{u}_r) + r\frac{\partial}{\partial \theta}(\sin(\theta)\mathbf{u}_\theta) + r\frac{\partial \mathbf{u}_\phi}{\partial \phi}\right) = 0.\end{aligned} \tag{5.60}$$

Compared to the flow on the surface of a sphere, the velocity component along r is needed ($\mathbf{u}_r$). Assuming both three velocity components are separable functions along different dimensions, where $\mathbf{u}_r = A(r,\theta)B(\phi), \mathbf{u}_\theta = C(r,\theta)D(\phi), \mathbf{u}_\phi = E(r,\theta)F(\phi)$, we have:

$$\nabla \cdot \mathbf{u} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2A(r,\theta))B(\phi) + \frac{1}{r\sin(\theta)}\frac{\partial}{\partial \theta}(\sin(\theta)C(r,\theta))D(\phi) + \frac{1}{r\sin(\theta)}E(r,\theta)\frac{\partial F(\phi)}{\partial \phi}. \tag{5.61}$$

The ϕ component can be determined by letting $B(\phi) = D(\phi) = \frac{\partial F(\phi)}{\partial \phi}$, and above equation can be simplified as:

$$\nabla \cdot \mathbf{u} = r\sin(\theta)\frac{\partial A_r}{\partial r}A_\theta + 2\sin(\theta)A_rA_\theta + \sin(\theta)\frac{\partial C_\theta}{\partial \theta}C_r + \cos(\theta)C_rC_\theta + E_rE_\theta = 0. \tag{5.62}$$

There will be three combinations that will lead to useful divergence-free basis functions.

Choice One

First of all, the equation 5.62 can be grouped as:

$$\sin(\theta)(r\frac{\partial A_r}{\partial r}A_\theta + A_r A_\theta + \frac{\partial C_\theta}{\partial \theta}C_r) + \sin(\theta)A_r A_\theta + \cos(\theta)C_r C_\theta + E_r E_\theta = 0. \tag{5.63}$$

We can assume:

$$\begin{aligned} &r\frac{\partial A_r}{\partial r}A_\theta + A_r A_\theta + \frac{\partial C_\theta}{\partial \theta}C_r = 0 \\ &\sin(\theta)A_r A_\theta + \cos(\theta)C_r C_\theta + E_r^1 E_\theta^1 + E_r^2 E_\theta^2 = 0. \end{aligned} \tag{5.64}$$

This gives:

$$\begin{aligned} &\frac{\partial C_\theta}{\partial \theta} = A_\theta \\ &r\frac{\partial A_r}{\partial r} + A_r + C_r = 0 \\ &\sin(\theta)A_r A_\theta + E_r^1 E_\theta^1 = 0 \\ &\cos(\theta)C_r C_\theta + E_r^2 E_\theta^2 = 0. \end{aligned} \tag{5.65}$$

One solution of above equation is:

$$\begin{aligned}
\Phi_r^0 &= \sin(i_1\pi r)\cos(i_2\theta)\sin(i_3\phi) \\
\Phi_\theta^0 &= -\frac{1}{i_2}(i_1\pi r\cos(i_1\pi r)+\sin(i_1\pi r))\sin(i_2\theta)\sin(i_3\phi) \\
\Phi_\phi^0 &= -\frac{1}{i_2 i_3}[-i_2\sin(i_1\pi r)\sin(\theta)\cos(i_2\theta)+(i_1\pi r\cos(i_1\pi r)+\sin(i_1\pi r))\cos(\theta)\sin(i_2\theta)]\cos(i_3\phi).
\end{aligned} \tag{5.66}$$

In general, the solution of this combination can be written as:

$$\begin{aligned}
\Phi_r^0 &= A(i_1\pi r)B(i_2\theta)C(i_3\phi) \\
\Phi_\theta^0 &= \frac{1}{i_2^2}(A(i_1\pi r)+rA'(i_1\pi r))B'(i_2\theta)C(i_3\phi) \\
\Phi_\phi^0 &= \frac{1}{i_3^2}\left(\frac{1}{i_2^2}(A(i_1\pi r)+rA'(i_1\pi r))\cos(\theta)B'(i_2\theta)+A(i_1\pi r)\sin(\theta)B(i_2\theta)\right)C'(i_3\phi),
\end{aligned} \tag{5.67}$$

where A, B, C can be sin or cos functions.

Choice Two

Another way to group equation 5.62 is:

$$\sin(\theta)\left(r\frac{\partial A_r}{\partial r}A_\theta + 2A_rA_\theta + \frac{\partial C_\theta}{\partial \theta}C_r\right) + \cos(\theta)C_rC_\theta + E_rE_\theta = 0. \tag{5.68}$$

We can assume:

$$\begin{aligned} &r\frac{\partial A_r}{\partial r}A_\theta + 2A_rA_\theta + \frac{\partial C_\theta}{\partial \theta}C_r = 0 \\ &\cos(\theta)C_rC_\theta + E_rE_\theta = 0, \end{aligned} \tag{5.69}$$

which gives:

$$\begin{aligned} &A_\theta = \frac{\partial C_\theta}{\partial \theta} \\ &r\frac{\partial A_r}{\partial r} + 2A_r + C_r = 0 \\ &\cos(\theta)C_rC_\theta + E_rE_\theta = 0. \end{aligned} \tag{5.70}$$

A general form of the solution is:

$$\begin{aligned} \Phi_r &= A(i_1\pi r)B(i_2\theta)C(i_3\phi) \\ \Phi_t &= \frac{1}{i_2^2}(2A(i_1\pi r) + rA'(i_1\pi r))B'(i_2\theta)C(i_3\phi) \\ \Phi_p &= \frac{1}{i_2^2 i_3^2}(2A(i_1\pi r) + rA'(i_1\pi r))\cos(\theta)B'(i_2\theta)C'(i_3\phi). \end{aligned} \tag{5.71}$$

Choice Three

To avoid the boundary condition at the pole, A_θ and C_θ can be weighted by $\sin(\theta)$, because $\sin(\theta)$ is zero at both pole. Assuming $A_\theta = A_\theta^* \sin(\theta), C_\theta = C_\theta^* \sin(\theta)$ then,

$$\sin(\theta)\left(r\frac{\partial A_r}{\partial r}A_\theta^*\sin(\theta) + 2A_rA_\theta^*\sin(\theta) + \sin(\theta)\frac{\partial C_\theta^*}{\partial \theta}C_r\right) + \sin(2\theta)C_rC_\theta^* + E_rE_\theta = 0. \tag{5.72}$$

This leads to:

$$\begin{aligned}
&\frac{\partial C_\theta^*}{\partial \theta} = A_\theta^* \\
&r\frac{\partial A_r}{\partial r} + 2A_r + C_r = 0 \\
&\sin(2\theta)C_r C_\theta^* + E_r E_\theta = 0.
\end{aligned} \tag{5.73}$$

A general form of the basis function is:

$$\begin{aligned}
\Phi_r &= A(i_1\pi r)\sin(\theta)B(i_2\theta)C(i_3\phi) \\
\Phi_t &= \frac{1}{i_2^2}(2A(i_1\pi r) + rA'(i_1\pi r))\sin(\theta)B'(i_2\theta)C(i_3\phi) \\
\Phi_p &= \frac{1}{i_2^2 i_3^2}(2A(i_1\pi r) + rA'(i_1\pi r))\sin(2\theta)B'(i_2\theta)C'(i_3\phi).
\end{aligned} \tag{5.74}$$

5.4.2 Basis Functions

Using the general forms are derived in the previous section, I will derive the principal basis functions and enrichment basis functions.

Principal Basis Functions

To derive the principal basis functions, we use choice 3 as shown in 5.74. The basis functions are weighted with $\sin(\theta)$, therefore they are zero along the pole line. Furthermore, I choose the sine mode along r direction for the Φ_r component, so they are also zero at the center. In this way, they will satisfy all the boundary conditions. For these basis functions, we are free to choose the three wave-numbers along each direction. They are

$$\begin{aligned}
&\Phi^0_r = \sin(i_1 r)\sin(\theta)\cos(i_2\theta)\sin(i_3\phi)\\
&\Phi^0_\theta = -\frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))\sin(\theta)\sin(i_2\theta)\sin(i_3\phi)\\
&\Phi^0_\phi = -\frac{1}{i_2 i_3}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))\sin(2\theta)\sin(i_2\theta)\cos(i_3\phi)\\
&i_1 > 0,\ i_2 > 0,\ i_3 > 0,
\end{aligned} \tag{5.75}$$

and

$$\begin{aligned}
&\Phi^1_r = \sin(i_1 r)\sin(\theta)\cos(i_2\theta)\cos(i_3\phi)\\
&\Phi^1_\theta = -\frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))\sin(\theta)\sin(i_2\theta)\cos(i_3\phi)\\
&\Phi^1_\phi = \frac{1}{i_2 i_3}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))\sin(2\theta)\sin(i_2\theta)\sin(i_3\phi)\\
&i_1 > 0,\ i_2 > 0,\ i_3 > 0.
\end{aligned} \tag{5.76}$$

The velocity at the boundaries is:

$$\begin{aligned}
&\Phi^0_*(r=0) = 0,\ \Phi^0_*(\theta=0) = 0,\ \Phi^0_*(\theta=\pi) = 0,\\
&\Phi^1_*(r=0) = 0,\ \Phi^1_*(\theta=0) = 0,\ \Phi^1_*(\theta=\pi) = 0.
\end{aligned} \tag{5.77}$$

Because the principal basis functions are zero both at the pole and the center, I propose another set of enrichment basis functions which are non-zero at those locations.

Enrichment Basis Functions at $\theta = 0, \pi$

First, I propose two basis functions that enrich the velocity along the pole, except at the center. These basis functions are choice 2 in equation 5.71. For these basis functions, Φ_r is zero at the pole, therefore the velocity points in tangent directions, similar to the basis functions on a sphere. Two wave-numbers are free, while the third one along ϕ is constrained to be 1. These basis functions are zero at the center. The first one is:

$$\begin{aligned}
&\Psi_r^2 = \sin(i_1 r)\sin(i_2\theta)\sin(\phi)\\
&\Psi_\theta^2 = \frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))\cos(i_2\theta)\sin(\phi)\\
&\Psi_\phi^2 = \frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))(\cos(\theta)\cos(i_2\theta))\cos(\phi).\\
&i_1 > 0, i_2 \geq 0
\end{aligned} \tag{5.78}$$

When $i_2 = 0$, the coefficients $\frac{1}{i_2}$ is set to 1 instead, which still makes a valid basis function. At the pole and the center, we have:

$$\begin{aligned}
&\Psi_r^2(\theta = 0) = 0\\
&\Psi_\theta^2(\theta = 0) = H\sin(\phi),\ \Phi_\phi^2(\theta = 0) = H\cos(\phi)\\
&\Psi_r^2(\theta = \pi) = 0\\
&\Psi_\theta^2(\theta = \pi) = \cos(i_2\pi)H\sin(\phi),\ \Phi_\phi^2(\theta = \pi) = -\cos(i_2\pi)H\cos(\phi),
\end{aligned} \tag{5.79}$$

where $H = \frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))$. We can see it satisfies the boundary condition at both poles. The other one is:

$$\begin{aligned}
&\Psi_r^3 = \sin(i_1 r)\sin(i_2\theta)\cos(\phi)\\
&\Psi_\theta^3 = \frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))\cos(i_2\theta)\cos(\phi)\\
&\Psi_\phi^3 = -\frac{1}{i_2}(i_1 r\cos(i_1 r) + 2\sin(i_1 r))(\cos(\theta)\cos(i_2\theta))\sin(\phi).\\
&i_1 > 0, i_2 \geq 0
\end{aligned} \tag{5.80}$$

When $i_2 = 0$, the coefficients $\frac{1}{i_2}$ is set to 1 instead. At the pole and the center, we have the velocity value as:

$$\begin{aligned}
&\Psi_r^3(\theta = 0) = 0\\
&\Psi_\theta^3(\theta = 0) = H\cos(\phi),\ \Phi_\phi^3(\theta = 0) = -H\sin(\phi)\\
&\Psi_r^3(\theta = \pi) = 0\\
&\Psi_\theta^3(\theta = \pi) = \cos(i_2\pi)H\cos(\phi),\ \Phi_\phi^3(\theta = \pi) = \cos(i_2\pi)H\sin(\phi).
\end{aligned} \tag{5.81}$$

It also satisfies boundary conditions at both poles. Finally, the principal basis functions and the above two enrichment basis functions are all zero at the center. To enrich the velocity at the center, I propose three extra basis functions for the center.

Enrichment Basis Functions at the Center

The first enrichment basis function is Ψ^4:

$$\begin{aligned}
\Psi^4_r &= \cos(i_1 r)\cos(\theta) \\
\Psi^4_\theta &= (-\cos(i_1 r) + \frac{1}{2} i_1 r \sin(i_1 r))\sin(\theta) \\
\Psi^4_\phi &= 0 \\
i_1 &\geq 0.
\end{aligned} \tag{5.82}$$

The velocity at the center and the poles for this basis function is:

$$\begin{aligned}
\Psi^4(r=0) &= \begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\end{bmatrix}^T \\
\Psi^4(\theta=0) &= \begin{bmatrix}\cos(i_1 r) & 0 & 0\end{bmatrix}^T \\
\Psi^4(\theta=\pi) &= \begin{bmatrix}-\cos(i_1 r) & 0 & 0\end{bmatrix}^T.
\end{aligned} \tag{5.83}$$

This is the last column of the transformation matrix shown in equation 5.41, therefore it points towards z direction in Cartesian coordinates. The next enrichment basis function takes the choice 1 from equation 5.67:

$$\begin{aligned}
\Psi^5_r &= \cos(i_1 r)\sin(\theta)\sin(\phi) \\
\Psi^5_\theta &= (-i_1 r \sin(i_1 r) + \cos(i_1 r))\cos(\theta)\sin(\phi) \\
\Psi^5_\phi &= (-i_1 r \sin(i_1 r)\cos^2(\theta) + \cos(i_1 r))\cos(\phi) \\
i_1 &\geq 0.
\end{aligned} \tag{5.84}$$

The velocity at the center of this basis function is:

$$\begin{aligned}
\Psi^5(r=0) &= \begin{bmatrix}\sin(\theta)\sin(\phi) & \cos(\theta)\sin(\phi) & \cos(\phi)\end{bmatrix}^T \\
\Psi^5(\theta=0) &= \begin{bmatrix}0 & (\cos(i_1 r) - i_1 r\sin(i_1 r))\sin(\phi) & (\cos(i_1 r) - i_1 r\sin(i_1 r))\cos(\phi)\end{bmatrix}^T \\
\Psi^5(\theta=\pi) &= \begin{bmatrix}0 & -(\cos(i_1 r) - i_1 r\sin(i_1 r))\sin(\phi) & (\cos(i_1 r) - i_1 r\sin(i_1 r))\cos(\phi)\end{bmatrix}^T.
\end{aligned} \tag{5.85}$$

This is the second column of the transformation matrix shown in equation 5.41, therefore it points towards y direction in Cartesian coordinates.

Finally, the enrichment basis function along x direction at the center is Ψ^6. It also takes choice 1 from the equation 5.67:

$$\begin{aligned}
&\Psi^6_r = \cos(i_1 r)\sin(\theta)\cos(\phi) \\
&\Psi^6_\theta = (-i_1 r\sin(i_1 r) + \cos(i_1 r))\cos(\theta)\cos(\phi) \\
&\Psi^6_\phi = -(-i_1 r\sin(i_1 r)\cos^2(\theta) + \cos(i_1 r))\sin(\phi) \\
&i_1 \geq 0.
\end{aligned} \tag{5.86}$$

The velocity at the center:

$$\begin{aligned}
\Psi^6(r=0) &= \begin{bmatrix}\sin(\theta)\cos(\phi) & \cos(\theta)\cos(\phi) & -\sin(\phi)\end{bmatrix}^T \\
\Psi^6(\theta=0) &= \begin{bmatrix}0 & (\cos(i_1 r) - i_1 r\sin(i_1 r))\cos(\phi) & -(\cos(i_1 r) - i_1 r\sin(i_1 r))\sin(\phi)\end{bmatrix}^T \\
\Psi^6(\theta=\pi) &= \begin{bmatrix}0 & -(\cos(i_1 r) - i_1 r\sin(i_1 r))\cos(\phi) & -(\cos(i_1 r) - i_1 r\sin(i_1 r))\sin(\phi)\end{bmatrix}^T.
\end{aligned} \tag{5.87}$$

This is the first column of the transformation matrix shown in equation 5.41, therefore it points towards x direction in Cartesian coordinates.

In conclusion, these enrichment basis functions satisfy the boundary condition at $\theta = 0, \pi$. Also, they satisfy the boundary condition at $r = 0$, because each basis function projects to $\mathbf{e}_z, \mathbf{e}_y, \mathbf{e}_x$ at the center, accordingly. For these basis functions, two wavenumbers along θ and ϕ are constrained to be 1, to satisfy the boundary condition at the poles. Only one wavenumber along r is free to choose.

Finally, to capture the circular motion of the fluid along the polar axis, I add the following enrichment basis functions:

$$\begin{aligned} &\Psi^7_r = 0 \\ &\Psi^7_\theta = 0 \\ &\Psi^7_\phi = \sin(i_1 r)\sin(i_2\theta) \\ &i_1 > 0,\ i_2 > 0. \end{aligned} \tag{5.88}$$

Boundary Conditions

For the 3D basis functions in a sphere, there will be a boundary condition along the line $\theta = 0$. This is discussed in the case of the surface of a sphere, as shown in equation 5.47. Here, I instead focus on the boundary conditions at the center.

For the boundary condition at the center, we can compute three velocity terms for each Cartesian direction using coordinate transformations.

$$\begin{bmatrix} \mathbf{u}_r \\ \mathbf{u}_\theta \\ \mathbf{u}_\phi \end{bmatrix} = \begin{bmatrix} \sin(\theta)\cos(\phi) & \sin(\theta)\sin(\phi) & \cos(\theta) \\ \cos(\theta)\cos(\phi) & \cos(\theta)\sin(\phi) & -\sin(\theta) \\ -\sin(\phi) & \cos(\phi) & 0 \end{bmatrix} \mathbf{u}, \tag{5.89}$$

where $\mathbf{u}$ can be assigned with $\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z$ to compute each of the boundary terms.

Similar to the surface of a sphere, we have a periodic boundary condition along ϕ,

$\mathbf{u}(\phi+2\pi) = \mathbf{u}(\phi)$. This can be satisfied by letting the wavenumber along ϕ be an integer.

The other boundary condition we need to consider is at the border of the sphere. Similar to polar 2D case, we can control the boundary condition by changing the first wavenumber of the basis function. For the basis functions 5.75, 5.76, 5.78 and 5.80, Dirichlet boundary condition at $r = 1$ can be satisfied by letting $i_1 \in \mathbb{N}$. Neumann boundary condition can be satisfied by letting $i_1 \in \mathbb{N} + 0.5$. For the basis functions 5.82, 5.84 and 5.86, Dirichlet boundary condition can be satisfied by letting $\mathbb{N}+0.5$. Neumann boundary condition can be satisfied by letting $i_1 \in \mathbb{N}$.

5.4.3 A Better Transformation Scheme

In this section, I show how to use the least number of DCTs to transform all the basis functions.

Transformation of $\mathbf{u}_r$ of $\Phi^0, \Phi^1, \Phi^2, \Phi^3$

The transformation of r component of Φ^0 can be expressed as:

$$\begin{gathered}\mathbf{u}_r = \sum_i \mathbf{w}_i^0 \sin(i_1 r)\sin(\theta)\cos(i_2\theta)\sin(i_3\phi) = \\ \frac{1}{2}\sum_i \mathbf{w}_i^0 \sin(i_1 r)[\sin((i_2+1)\theta) - \sin((i_2-1)\theta)]\sin(i_3\phi).\end{gathered} \tag{5.90}$$

Instead of doing two 3D DCTs and adding the results together, we can add $\frac{1}{2}\mathbf{w}_i^0$ and $-\frac{1}{2}\mathbf{w}_i^0$ into slots which corresponds to wavenumber i_2+1 and i_2-1, respectively, and then do one 3D DCT. Because the final velocity along r is the summarization of r terms from all basis functions, we can group them as:

$$\begin{gathered}\mathbf{u}_r = \sum_i \mathbf{w}_i^0 \Phi_r^0 + \sum_i \mathbf{w}_i^1 \Phi_r^1 + \sum_i \mathbf{w}_i^2 \Phi_r^2 + \sum_i \mathbf{w}_i^3 \Phi_r^3 = \\ \left(\sum_i \mathbf{w}_i^0 \Phi_r^0 + \sum_i \mathbf{w}_i^2 \Phi_r^2 \right) + \left(\sum_i \mathbf{w}_i^1 \Phi_r^1 + \sum_i \mathbf{w}_i^3 \Phi_r^3 \right).\end{gathered} \tag{5.91}$$

The Φ_r^0, Φ_r^2 are grouped because they share the transformations $\sin(r), \sin(\theta), \sin(\phi)$ along all three directions. Therefore, we can simply add the contributions of Φ_r^0, Φ_r^2 together, before doing any transformations. Similarly, Φ_r^1, Φ_r^3 share the transformations $\sin(r), \sin(\theta), \cos(\phi)$, therefore we can also add contributions of them together. Now, Φ_r^0, Φ_r^2 and Φ_r^1, Φ_r^3 share transformations along r, θ. To do that, we can do a DST along ϕ for Φ_r^0, Φ_r^2, and a DCT along ϕ for Φ_r^1, Φ_r^3, adding the results together, and then doing a DST along r and θ. The flow graph looks like the following:

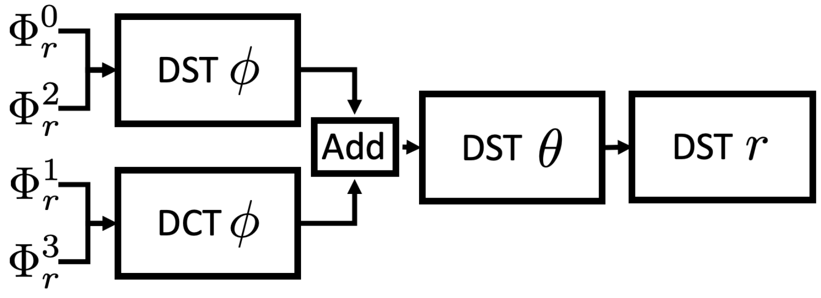


Figure 5.9: Flow graph of transformations.

Therefore, we can transform the r component of $\Phi^0, \Phi^1, \Phi^2, \Phi^3$ using only 4 DCTs, where each DCT transforms along one direction. This also trivially supports trimmed DCTs along the first two directions.

5.4.4 Computing Dot Products of Basis Functions

In this section I briefly describe the method I use to compute the dot product between all different basis functions. The integration of a scalar function in 3D spherical coordinates is described in equation 5.43. First of all, the velocity components $\hat{\mathbf{r}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}$ are orthogonal, therefore:

$$\left\langle \Phi^l, \Phi^m \right\rangle = \left\langle \Phi^l_r, \Phi^m_r \right\rangle + \left\langle \Phi^l_\theta, \Phi^m_\theta \right\rangle + \left\langle \Phi^l_\phi, \Phi^m_\phi \right\rangle . \tag{5.92}$$

Each of the Φ^l components are multiplications of separable functions along r, θ, ϕ. The term $\left\langle \Phi^l_r, \Phi^m_r \right\rangle$ can be evaluated as:

$$\left\langle \Phi^l_r, \Phi^m_r \right\rangle = \int_{r=0}^{1} A^l(r) A^m(r) r^2 dr \int_{\theta=0}^{\pi} B^l(\theta) B^m(\theta) \sin(\theta) d\theta \int_{\phi=0}^{2\pi} C^l(\phi) C^m(\phi) d\phi. \tag{5.93}$$

The integration along with r, θ, ϕ can be evaluated individually. This is better than computing all combinations at once. For example, assume we have total 8 basis functions, computing them at once will result in 64 combinations. However, as we can see there are only 2 choices of function A, we only need 4 combinations along each direction.

As for the advection tensor, computing all combinations at once will results in 64*8 = 512 combinations. However, by decomposing them along each direction, we only need 8 combinations along each direction.

5.5 Implementation and Results

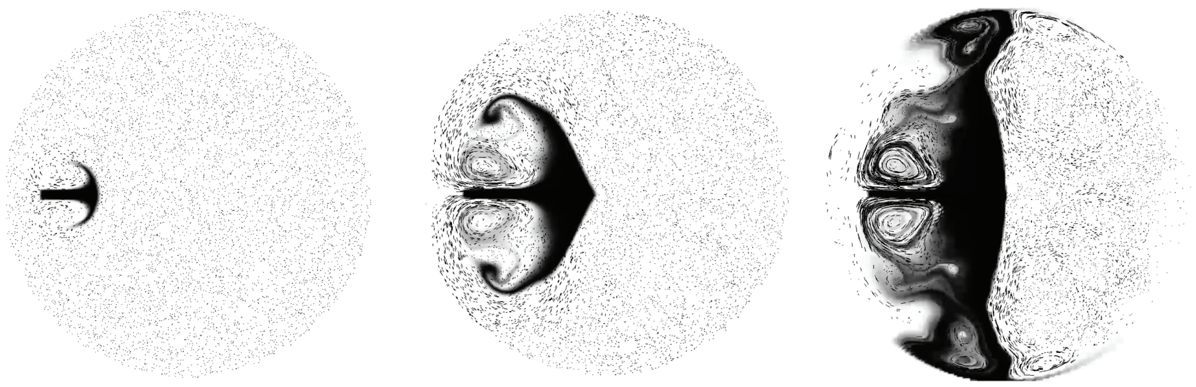
We implemented our algorithm in C++. The FFTW3 library [89] is used to perform the DCTs and DSTs. Multithreading was enabled using OpenMP whenever possible. We use the Eigen library [88] to perform basic linear algebra operations, including the eigendecomposition of the inner product matrix. All our results were run on a desktop with 48 GB of memory and 6 cores running at 3.7 GHz. To ensure a well-conditioned transfer matrix that transforms the basis functions into orthogonal, we discarded the smallest 10% of eigenvalues of the inner product matrix $\mathbf{H}$ in all our examples. The pruned DCT mentioned in [1] is used whenever possible. We do not compress the advection tensor in any of our examples.

5.5.1 Polar 2D Coordinates

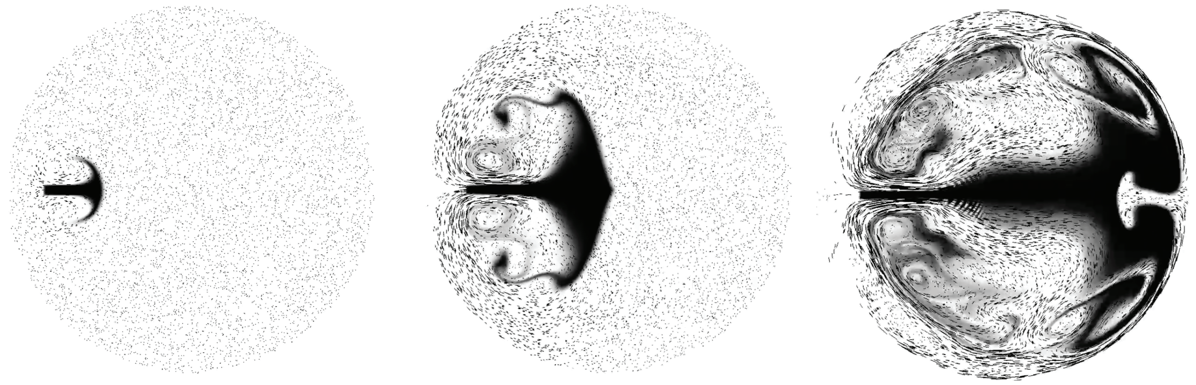
Below I show some results of Eigenfluid simulations in polar coordinate systems. In all examples, I interpolated the velocity under the polar grid onto a Cartesian grid of resolution 512×512. The polar grid is rectangular, where more pixels are allocated along the angular direction to ensure an even discretization. The resolution of the polar grid roughly matches the resolution of the Cartesian grid.

Dirichlet and Neuman boundary conditions: In figure 5.10 I show the simulation in polar coordinates with Dirichlet and Neumann boundary conditions. In the simulation, the plume is driven by a buoyancy force. Fig (a) of 5.10 is simulated with only principal basis functions 5.16 and 5.17 only. As we can see, it is difficult for the plume to pass through the center, because the principal basis functions are zero at the center. Adding enrichment basis functions as shown in 5.19 and 5.23 resolves this issue, which allows the plume to pass through the center.

Colliding smoke in polar coordinates: In figure 5.11 we animate two smoke plumes



(a) Simulation without enrichment basis functions.

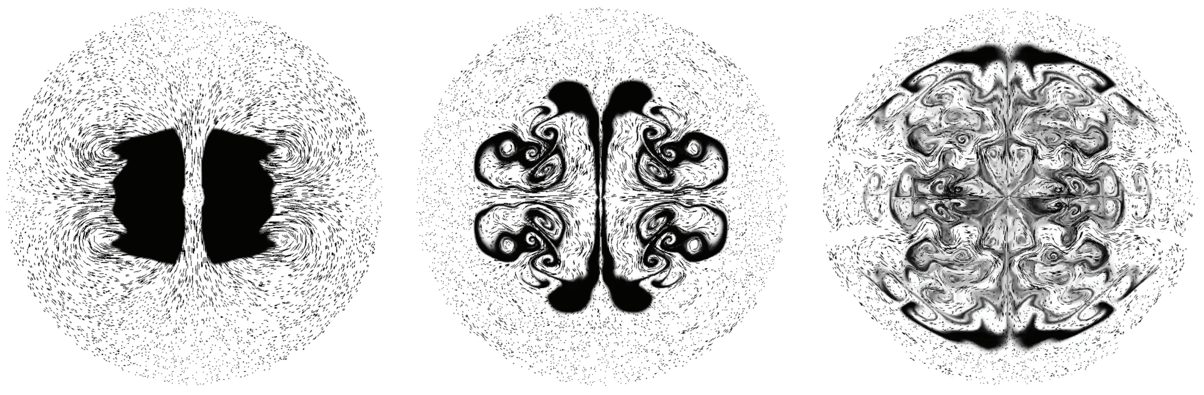


(b) Simulation with enrichment basis functions for the center.

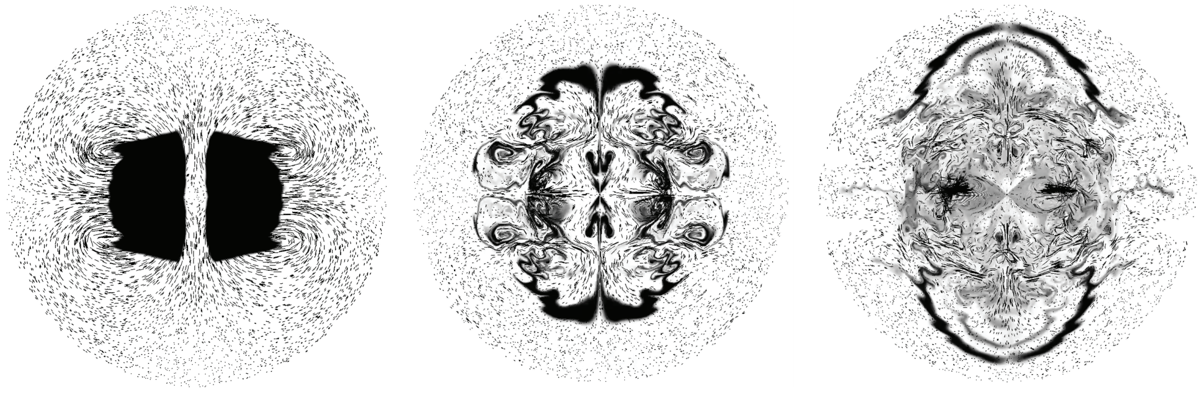
Figure 5.10: Simulation of a plume without and with the enrichment basis functions. The enrichment basis function allows the plume to pass through the center.

colliding with each other. In figure (a) we show the simulation with 300 basis functions. Adding more basis functions will allow more detailed vortices to appear. Because the polar basis functions support fast transformations, we retain the scalability in [1].

Boundary conditions in polar coordinates: In figure 5.12 we simulate the fluid under Neumann and the Dirichlet boundary conditions. This is achieved by changing the wavenumber along with radial directions of the basis functions. The plume bounces back into the simulation domain under the Dirichlet boundary condition. The Neumann boundary condition allows the plume to flow out of the domain.



(a) Simulation with 300 basis functions.



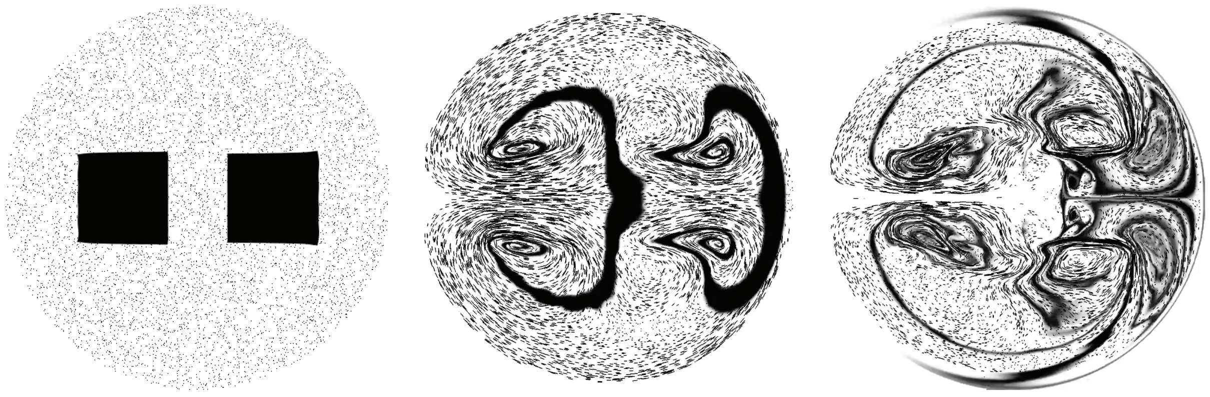
(b) Simulation with 1600 basis functions.

Figure 5.11: Simulation of two plumes colliding with each other, the viscosity is set to zero. The plume scatters into finer vortices with more basis functions added.

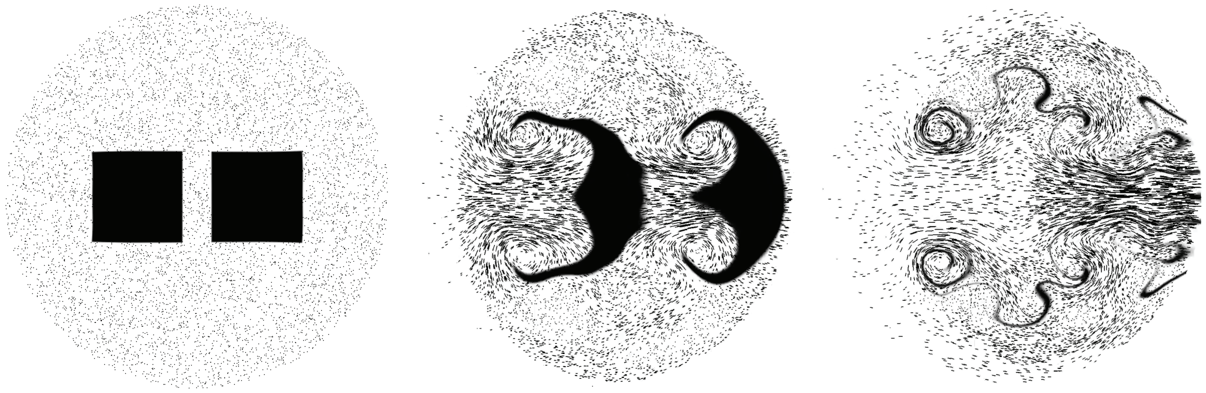
5.5.2 2D Spherical Coordinates

In this section, I show the results using the spherical 2D basis functions. The basis functions are best suited for simulating surface flow, for example, planetary flows or soap bubbles. All of our examples are simulated on a (θ, ϕ) grid of resolution 512×1024. The density field is obtained by splatting density particles that are passively advected along the velocity field. The density field is then used as a texture map on the sphere.

Planetary flow on sphere surface: In figure 5.13 we simulate a planetary flow. Usually, the planetary flow is driven by both pressure gradient and Coriolis force [7]. However, our method is pressure-free, so we are not able to capture the effect of forces caused by



(a) Simulation with Dirichlet basis functions.



(b) Simulation with Neumann basis functions.

Figure 5.12: Simulation of a pulse under Dirichlet and Neumann basis functions. Neumann basis functions can allow the fluid to flow out of the domain, while Dirichlet basis functions can not.

an uneven pressure field. Still, we show that our basis function can be sufficient to result in a detailed planetary flow simulation, assuming appropriate force are injected. In this example, we initialized the density to form a stripe pattern. Then, I injected forces that are alternating in direction along the θ direction, which points toward the ϕ direction. The forces are continuously injected into the simulation, and a viscosity of $\nu = 0.01$ is used to dissipate the energy.

Turbulent planetary flow: In figure 5.14 a turbulent planetary flow is obtained by injecting extra Coriolis force [7]. The Coriolis force pulls the flow along the tangent direction, where the magnitude is proportional to the velocity. This results in a more

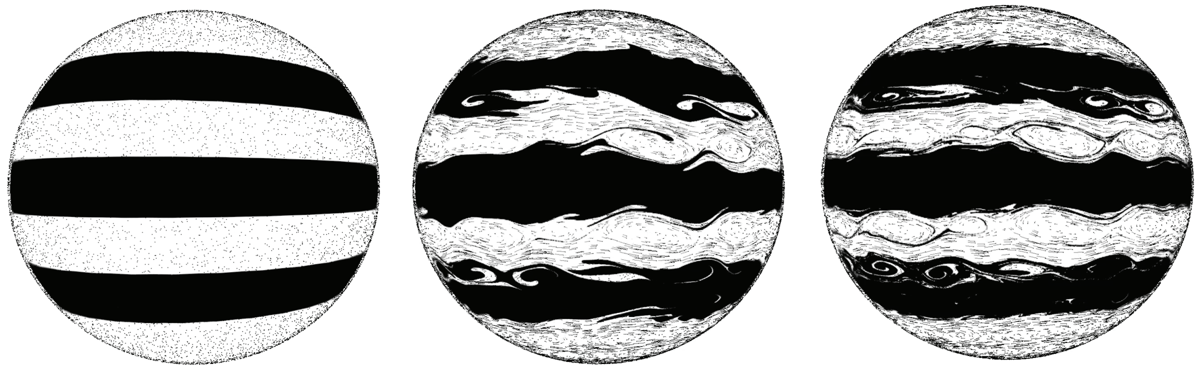


Figure 5.13: A simulation of planetary flow.

turbulent look. In figure (b) we show the periodic texture that is wrapped around the sphere. Notice the intricate details formed by the basis function.

5.5.3 Full 3D Spherical Coordinates

Finally, we show the results of the full 3D spherical simulation. Similar to the case of a polar grid, velocity is spatted onto a Cartesian grid. Density is then advected on the Cartesian grid with a MacCormack method [91]. In all the examples the resolution of the Cartesian grid is 128^3, where the resolution of the spherical grid is $64 \times 128 \times 256$, along (r, θ, ϕ), respectively. We ray trace the density field with PBRT [92].

Impulse flow: In figure 5.15 we show an animation where a block of smoke is pushed by an impulse. The viscosity in this example is zero. Figure (a) shows the simulation with 300 basis functions. As we can see, 300 basis functions already produces detailed vortices. Figure (b) shows the same simulation with 1700 basis functions. More details appear as more basis functions are added.

Boundary comparison in 3D: In figure 5.16 we show two simulations with different boundary conditions, both with 700 basis functions. Similar to the simulation on the polar grid, the Neumann boundary condition allows the fluid to flow out of the domain.

Also, notice the fluid can freely flow past the center contributed by the enrichment basis functions.

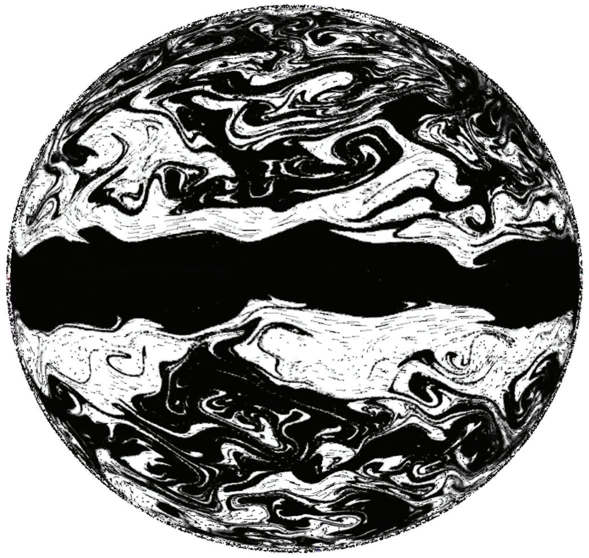
Plume Gravity: In 5.17 we show a simulation with a block of smoke pulled down by gravity. In this case, the viscosity is set to zero.

Circular Force: Finally, the enrichment basis function 5.88 can resolve the circular motion of the fluid along the polar axis. In figure 5.18, we inject a circular force, where a slab of smoke is pushed around by it. In this case, the viscosity is set to zero.

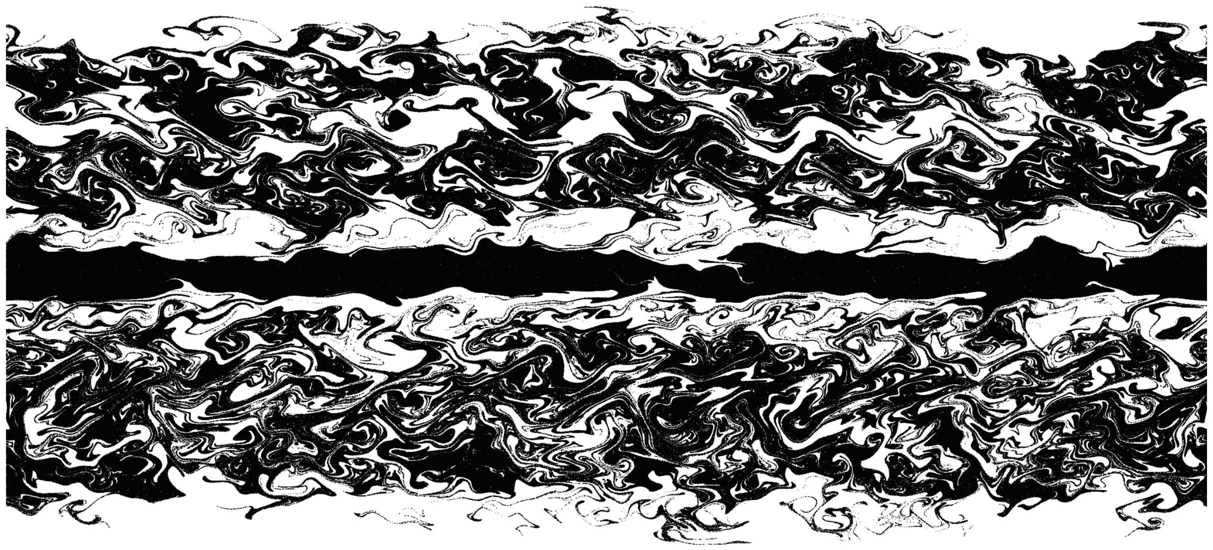
The timing of all our examples is shown in table 5.1.

Scene	IMPULSE FLOW		PLANETARY FLOW	COLLIDING SMOKES	
Grid Resolution	128^3		512×1024	512^2	
Basis functions	Spherical 3D		Spherical 2D	Polar 2D	
Basis Dimension	1700	300	4000	1600	300
Tensor Contraction	0.122 secs	0.0011 secs	1.25 secs	0.310 secs	0.009 secs
Linear Solver	0.017 secs	0.0002 secs	0.265 secs	0.064 secs	0.0015 secs
DCT/ DST	0.135 secs	0.135 secs	0.100 secs	0.133 secs	0.133 secs
Density Advection	0.064 secs	0.064 secs	0.820 secs	0.0027 secs	0.0026 secs
Total	0.338 secs	0.200 secs	2.44 secs	0.534 secs	0.222 secs

Table 5.1: Timing breakdown of our algorithm across all the different examples.

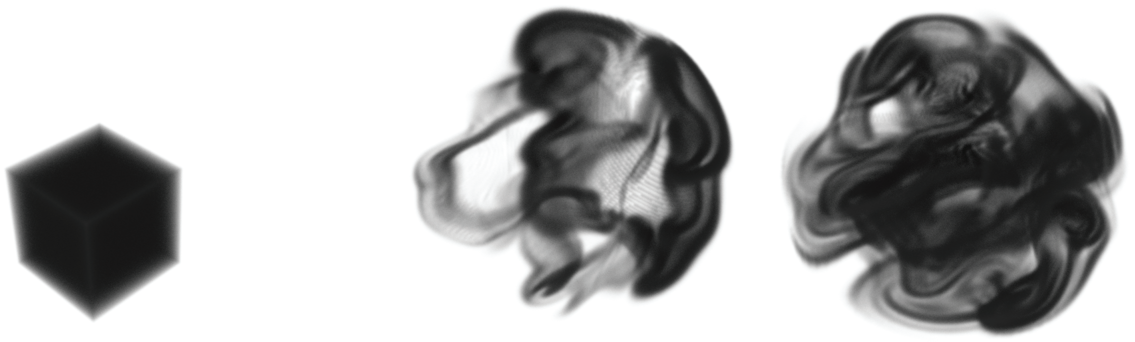


(a) A turbulent planetary flow simulation.

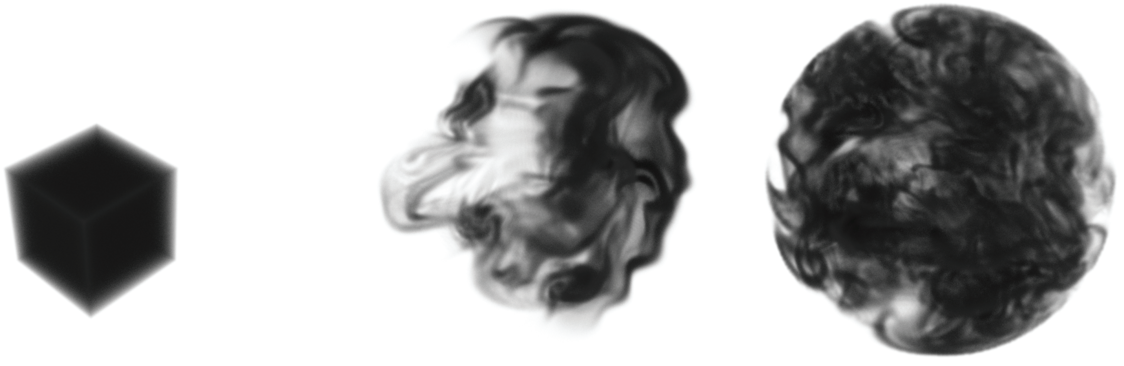


(b) The Periodic texture produced by the method, which is then wrapped onto a sphere in figure (a).

Figure 5.14: A turbulent planetary flow simulation with 4000 basis functions.

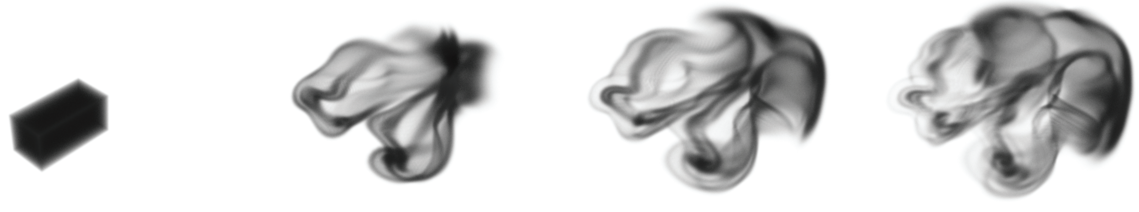


(a) A simulation with 300 basis functions.

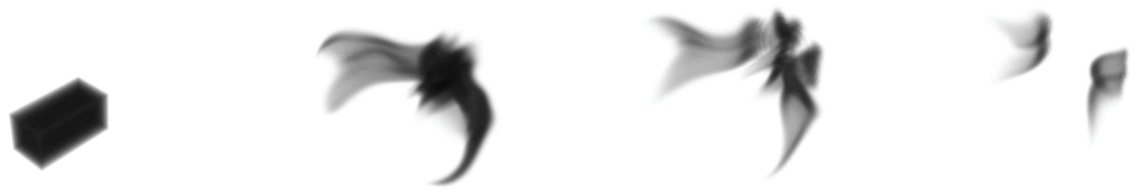


(b) A simulation with 1700 basis functions.

Figure 5.15: Simulation of an impulse under 3D spherical coordinates.



(a) A simulation with Dirichlet boundary conditions.



(b) The same simulation with Neumann boundary conditions.

Figure 5.16: Two simulations are shown with Dirichlet and Neumann boundary conditions. Notice the Neumann boundary condition allows the smoke to flow out of the domain.

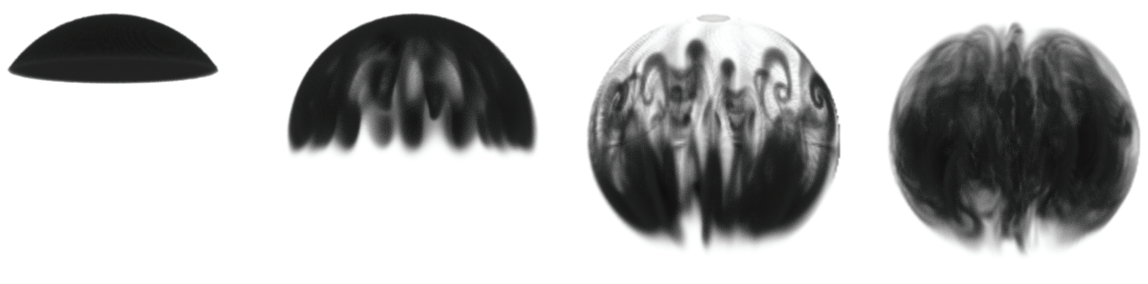


Figure 5.17: A simulation of smoke pulled down by gravity.

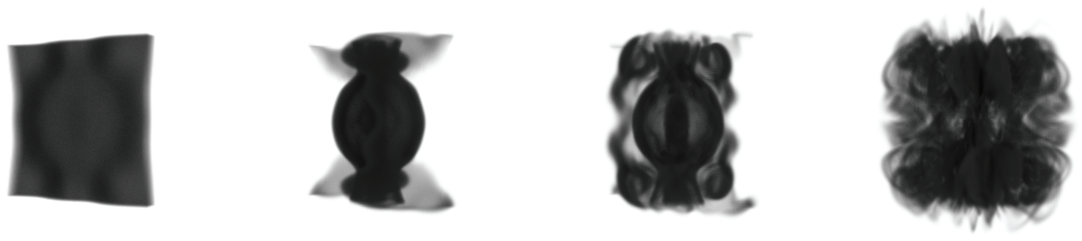


Figure 5.18: Simulation of a slab of smoke twisted by circular forces.

Chapter 6

Stochastic Structure Optimization Background

Note: A significant portion of this chapter has previously appeared as [2].

In this chapter, I summarize the stochastic structural optimization technique of Langlois et al. [27].

Finite Element Method (FEM)

We use a hexahedral uniform grid as our FEM discretization and compute the displacement ($\mathbf{u}$) and element Cauchy stresses ($\boldsymbol{\sigma}$) that arise from an external force ($\mathbf{f}$):

$$\begin{aligned} \mathbf{K}\mathbf{u} &= \mathbf{f} \\ \boldsymbol{\sigma} &= \mathbf{C}\mathbf{B}\mathbf{u}. \end{aligned} \tag{6.1}$$

Above, $\mathbf{K} \in \mathbb{R}^{3n \times 3n}$ is the stiffness matrix, $\boldsymbol{\sigma} \in \mathbb{R}^{6m}$ is a vector of per-element Cauchy stresses, $\mathbf{C} \in \mathbb{R}^{6m \times 6m}$ is a constitutive matrix, and $\mathbf{B} \in \mathbb{R}^{6m \times 3n}$ maps $\mathbf{u}$ to Cauchy strains. Stresses and strains are evaluated at the center of each voxel. The quantity m

is the total number of mesh elements, and n is the number of vertices.

Using the per-element stress,

$$\boldsymbol{\sigma}_e = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{31} \\ \sigma_{12} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{23} & \sigma_{33} \end{bmatrix} \tag{6.2}$$

we compute a scalar, per-element von Mises stress:

$$\begin{aligned} S(\boldsymbol{\sigma}_e) =& \frac{1}{2}\left[(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2\right] + \\ & 3\left(\sigma_{23}^2 + \sigma_{31}^2 + \sigma_{12}^2\right). \end{aligned} \tag{6.3}$$

A yield stress $\hat{\sigma}$ completes our failure criteria. An element fails if $S(\boldsymbol{\sigma}_e) > \hat{\sigma}$, and we define "object failure" as the failure of any individual element.

Stochastic Failure Probability

Prior to optimization, we need to estimate an object's failure probability under various force distributions. Langlois et al. [27] estimated real-world loadings by using a rigid-body simulation to generate force samples. For each sample, they then computed the maximal von Mises stress across the whole object, and then estimated a probability distribution function (PDF) of maximal stresses. The object fails if its maximal stress is greater than the yield stress, so the survival probability is computed by integrating the PDF from 0 to $\hat{\sigma}$. The failure probability is one minus the survival probability.

Many force samples (≈ 5000) are needed to accurately represent the PDF, so principal component analysis (PCA) is used to reduce its computational complexity. We denote each force sample as $\mathbf{f}^i$, where $i = 1 \ldots n_s$, and n_s is the total number of samples. In lieu of performing an FEM analysis for each sample, they compute a reduced force basis, $\bar{\mathbf{F}} \in \mathbb{R}^{3n \times r}$, where r is the number of principal components. Each sample is then represented with a reduced coordinate, $\boldsymbol{\alpha}^i \in \mathbb{R}^r$, where $\mathbf{f}^i \approx \bar{\mathbf{F}}\boldsymbol{\alpha}^i$. The element stresses for each sample i can then be computed as:

$$\boldsymbol{\sigma}^i = \mathbf{CBK}^{-1}\bar{\mathbf{F}}\boldsymbol{\alpha}^i. \tag{6.4}$$

By pre-solving $\mathbf{CBK}^{-1}$ for each column in $\bar{\mathbf{F}}$, significant savings can be achieved when $r \ll n_s$ (e.g. $r \approx 100$).

The normalized, whole-object stress for each $\mathbf{f}^i$ is then:

$$s^i = \frac{1}{\hat{\sigma}} \max_e(S(\boldsymbol{\sigma}_e^i)) \begin{cases} e = 1 \dots m \\ i = 1 \dots n_s \end{cases} \tag{6.5}$$

Using all the stress samples s^i, we can construct a probability distribution function (PDF) $p(s)$ for the whole-object stress. The corresponding cumulative distribution (CDF) function is denoted as $P(s)$. The probability of the object survival $P(s < 1)$ is then:

$$P(s < 1) = \int_0^1 p(s) ds. \tag{6.6}$$

Topology Optimization

The failure probability is then used as a constraint in a topology optimization that adds or subtracts materials from some initial design. The overall goal is to reduce the final object weight while satisfying a user defined failure probability Θ:

$$\begin{aligned} &\min \sum_{e=1}^{m} \omega_e \\ &\text{s.t.} \;\; P(s < 1) > 1 - \Theta. \end{aligned} \tag{6.7}$$

Above, $\boldsymbol{\omega}$ is a vector of element densities, such that $\forall e\ \omega_e \in [0, 1]$, which is usually initialized to be fully filled ($\forall e,\ \omega_e = 1$). Eqn. 6.7 is optimized using the Method of Moving Asymptotes (MMA) [64], which requires both the object and constraint gradients. The objective gradient is straightforward to efficiently compute, but the constraint gradient is a major bottleneck because the existing approach is quadratic in the number of elements. We will show that it is possible to reduce its complexity to linear.

Chapter 7

Fast and Robust Stochastic Optimization

Note: A significant portion of this chapter has previously appeared as [2].

In this chapter, I will present our fast and robust stochastic optimization algorithm.

First, in section §7.1, I analyze the complexity of computing the probability gradients. By carefully leveraging the structure of the problem, we found the *quadratic* complexity of previous methods [27] can be reduced to *linear* with respect to the number of elements. In practice, this results in a roughly two-order of magnitude speedup.

Next, in section §7.2, I show that the existing approach leads to unreliable probability gradients. This negatively impacts the convergence of the optimization and, in turn, the final design. I show how to stabilize these gradients, which leads to higher-quality shapes.

The optimization can still stall at local minima. To address this, I introduce (§7.3) a constrained restart method that identifies and constrains promising structures when the optimization stalls, and applies a perturbation to bump the state out of its local minimum. Finally, I make additional progress by allowing the outer shell of the object to erode, and later restore the visual appearance of the object by adding a lightweight sheath as a post-process.

7.1 Asymptotic Analysis and Acceleration

First, I will analyze the existing method [27] to show that it runs in $O(m^2)$. Then I will show that an identical computation can be done in $O(m)$.

7.1.1 The Previous Quadratic Method

We begin by expanding the derivative of $p(s)$ from Eqn. 6.6 in terms of s^i using the chain rule:

$$\frac{\partial P(s<1)}{\partial \boldsymbol{\omega}} = \int_0^1 \sum_{i=1}^{n_s} \frac{\partial p}{\partial s^i}\frac{\partial s^i}{\partial \boldsymbol{\omega}} ds. \tag{7.1}$$

The $\frac{\partial s^i}{\partial \boldsymbol{\omega}}$ term is computed by replacing the discontinuous max function in Eqn. 6.5 with a smoother L^p norm. The density term ω_e is also multiplied beforehand to avoid singularities as shown in [68], $s^i \approx ||\sqrt{\omega_e^i} S(\boldsymbol{\sigma}_e^i)||_p / \hat{\sigma}$, yielding:

$$\begin{aligned}\frac{\partial s^i}{\partial \boldsymbol{\omega}} =& \frac{1}{\hat{\sigma}} \left(\sum_{e=1}^{m} \sqrt{\omega_e^i} S(\boldsymbol{\sigma}_e^i)^p \right)^{\frac{1}{p}-1} \\ & \sum_{e=1}^{m} (\sqrt{\omega_e^i} S(\boldsymbol{\sigma}_e^i))^{p-1} \left(\frac{1}{2\sqrt{\omega_e^i}} S(\boldsymbol{\sigma}_e^i) + \frac{\partial S}{\partial \boldsymbol{\sigma}_e^i}\frac{\partial \boldsymbol{\sigma}_e^i}{\partial \boldsymbol{\omega}} \right).\end{aligned} \tag{7.2}$$

Combining equation 7.2 with equation 7.1, we obtain:

$$\frac{\partial P(s<1)}{\partial \boldsymbol{\omega}} = \sum_{i=1}^{n_s} a^i \left(\mathbf{b}^i + \left(\frac{\partial \boldsymbol{\sigma}^i}{\partial \boldsymbol{\omega}} \right)^T \mathbf{c}^{\mathbf{i}} \right), \tag{7.3}$$

where

$$\begin{aligned}
a^i &= \int_0^1 \frac{\partial p}{\partial s^i} ds \frac{1}{\hat{\sigma}} \left(\sum_{e=1}^{m} \sqrt{\omega_e^i} S(\boldsymbol{\sigma}_e^i)^p \right)^{\frac{1}{p}-1} \\
\mathbf{b}^i &= \sum_{e=1}^{m} (\sqrt{\omega_e^i} S(\boldsymbol{\sigma}_e^i))^{p-1} S(\boldsymbol{\sigma}_e^i)/(2\sqrt{\omega_e^i}) \\
\mathbf{c}_e^i &= (\sqrt{\omega_e^i} S(\boldsymbol{\sigma}_e^i))^{p-1} \frac{\partial S}{\partial \boldsymbol{\sigma}_e^i}.
\end{aligned}$$

The term $\frac{\partial S}{\partial \boldsymbol{\sigma}_e^i}$ can be computed from equation 6.3. Next, from equation 6.4 we compute:

$$\frac{\partial \boldsymbol{\sigma}^i}{\partial \boldsymbol{\omega}} = \underbrace{-\mathbf{CBK}^{-1} \frac{\partial \mathbf{K}}{\partial \boldsymbol{\omega}} \bar{\mathbf{U}} \boldsymbol{\alpha}^i}_{\mathbf{I}} + \underbrace{\mathbf{CBK}^{-1} \frac{\partial \bar{\mathbf{F}}}{\partial \boldsymbol{\omega}} \boldsymbol{\alpha}^i}_{\mathbf{II}} + \underbrace{\mathbf{CB}\bar{\mathbf{U}} \frac{\partial \boldsymbol{\alpha}^i}{\partial \boldsymbol{\omega}}}_{\mathbf{III}}, \tag{7.4}$$

where $\bar{\mathbf{U}} = \mathbf{K}^{-1}\bar{\mathbf{F}}$. Here, term **I** computes the derivative of the stiffness matrix, term **II** computes the gradient of the reduced force basis, and term **III** computes the gradient of the reduced force coordinates.

Combining equations 7.3 and 7.4, we obtain the final probability derivative, which can be computed as:

$$\frac{\partial P(s<1)}{\partial \boldsymbol{\omega}} = (\mathbf{K}^{-1}\mathbf{Y}\bar{\mathbf{U}}^T) : \frac{\partial \mathbf{K}}{\partial \boldsymbol{\omega}} + (\mathbf{K}^{-1}\mathbf{Y}) : \frac{\partial \bar{\mathbf{F}}}{\partial \boldsymbol{\omega}} + \mathbf{x} + \mathbf{t}, \tag{7.5}$$

where

$$\begin{aligned}
\mathbf{Y} &= \sum_{i=1}^{n_s} \mathbf{B}^T \mathbf{C}^T \mathbf{c}^i \otimes \boldsymbol{\alpha}^i \qquad \mathbf{t} = \sum_{i=1}^{n_s} a^i \mathbf{b}^i \\
\mathbf{x} &= \sum_{i=1}^{n_s} \frac{\partial \boldsymbol{\alpha}^i}{\partial \boldsymbol{\omega}} \bar{\mathbf{U}}^T \mathbf{B}^T \mathbf{C}^T \mathbf{c}^i.
\end{aligned}$$

The main complexity in Eqn. 7.5 lies in the second term, $(\mathbf{K}^{-1}\mathbf{Y}) : \frac{\partial \bar{\mathbf{F}}}{\partial \boldsymbol{\omega}}$. We show in Appendix 7.1.2 for a single element e, the force derivative on the right of the double-contraction can be written as

$$\frac{\partial \bar{\mathbf{F}}}{\partial \boldsymbol{\omega}_e} = \bar{\mathbf{F}}\mathbf{W}_e. \tag{7.6}$$

$\mathbf{W}_e \in \mathbb{R}^{r\times r}$ is a dense matrix that has to be evaluated for *each* element. A naïve evaluation for one element then becomes mr^2 because $\bar{\mathbf{F}} \in \mathbb{R}^{3n\times r}$ and $m \propto n$ due to the uniform grid discretization. Computing the force derivatives for all samples is then $O(m^2r^2)$.

7.1.2 Evaluation of the Derivative of Reduced Force Vectors

For completeness, we summarize the derivative of the reduced force basis vectors $\bar{\mathbf{F}}$, which is described in the supplemental material of [27]. A finite element force sample $\mathbf{f}^i \in \mathbb{R}^{3n}$ is obtained by mapping a rigid body force $\mathbf{l}^i$ via a projection matrix J: $\mathbf{f}^i = \mathbf{J}\mathbf{l}^i$. Each rigid body force sample $\mathbf{l}^i \in \mathbb{R}^{3n_u+6}$ is of the form:

$$\mathbf{l}^i = \begin{bmatrix}\mathbf{l}_1 & \dots & \mathbf{l}_{n_u} & \mathbf{f}^{com} & \boldsymbol{\tau}^{com}\end{bmatrix}, \tag{7.7}$$

where $\mathbf{l}_{3(j-1)} \in \mathbb{R}^3, j = 1, \dots, n_u$ are the contact forces sampled at the surface of the object, n_u is the total number of possible contact positions, and $\mathbf{f}^{com}, \boldsymbol{\tau}^{com} \in \mathbb{R}^3$ are the inertial force and torque acting on the center of the mass.

The matrix $\mathbf{J}$ is of the form:

$$\mathbf{J} = \begin{bmatrix}\mathbf{J}^m & \mathbf{J}^{com} & \mathbf{J}^{\tau}\end{bmatrix}, \tag{7.8}$$

where $\mathbf{J}^m \in \mathbb{R}^{3n\times 3n_u}$ maps the surface contact points to volumetric element vertices, and $\mathbf{J}^{com}, \mathbf{J}^{\tau} \in \mathbb{R}^{3n\times 3}$ map the inertial forces and torques acting on the center of mass to element vertices.

Given n_s rigid body force samples, $\mathbf{L} = \begin{bmatrix} \mathbf{l}^1 & \dots & \mathbf{l}^{n_s} \end{bmatrix}$, the reduced force basis $\bar{\mathbf{F}}$ is the eigenvectors of the covariance matrix of the force samples, $\mathbf{JL}$. Therefore, we have $\mathbf{JLL}^T\mathbf{J}^T \approx \bar{\mathbf{F}}\mathbf{\Lambda}\bar{\mathbf{F}}^T$, where $\mathbf{\Lambda} \in \mathbb{R}^{r\times r}$ is a diagonal matrix of eigenvalues of the covariance matrix $\mathbf{JLL}^T\mathbf{J}^T$. From here we follow the procedure of [93]. We have:

$$\frac{\partial \mathbf{JLL}^T\mathbf{J}^T}{\partial \omega_e} = \frac{\partial \bar{\mathbf{F}}}{\partial \omega_e}\mathbf{\Lambda}\bar{\mathbf{F}}^T + \bar{\mathbf{F}}\frac{\partial \mathbf{\Lambda}}{\partial \omega_e}\bar{\mathbf{F}}^T + \bar{\mathbf{F}}\mathbf{\Lambda}\frac{\partial \bar{\mathbf{F}}^T}{\partial \omega_e}. \tag{7.9}$$

Multiplying Eqn. 7.9 with $\bar{\mathbf{F}}^T$ from the left and $\bar{\mathbf{F}}$ from the right yields:

$$\bar{\mathbf{F}}^T\frac{\partial \mathbf{JLL}^T\mathbf{J}^T}{\partial \omega_e}\bar{\mathbf{F}} = \bar{\mathbf{F}}^T\frac{\partial \bar{\mathbf{F}}}{\partial \omega_e}\mathbf{\Lambda} + \frac{\partial \mathbf{\Lambda}}{\partial \omega_e} + \mathbf{\Lambda}\frac{\partial \bar{\mathbf{F}}^T}{\partial \omega_e}\bar{\mathbf{F}}. \tag{7.10}$$

Denote $\mathbf{B}_e = \bar{\mathbf{F}}^T\frac{\partial \mathbf{JLL}^T\mathbf{J}^T}{\partial \omega_e}\bar{\mathbf{F}}$. The matrix $\mathbf{W}_e = \bar{\mathbf{F}}^T\frac{\partial \bar{\mathbf{F}}}{\partial \omega_e}$ is antisymmetric [93], therefore $\frac{\partial \mathbf{\Lambda}}{\partial \omega_e} = diag(\mathbf{B}_e)$, so we have:

$$\mathbf{B}_e - diag(\mathbf{B}_e) = \mathbf{W}_e\mathbf{\Lambda} + \mathbf{\Lambda}\mathbf{W}_e^T. \tag{7.11}$$

Exploiting the skew-symmetry of $\mathbf{W}_e$ and setting $\boldsymbol{\lambda} = diag(\mathbf{\Lambda})$, we can derive a simple update equation:

$$(\lambda_i - \lambda_j)\mathbf{W}_{ije} = \mathbf{B}^*_{ij}. \tag{7.12}$$

where $\mathbf{B}^* = \mathbf{B}_e - diag(\mathbf{B}_e)$. We solve this equation for each $\mathbf{W}_{ije}$. Finally we can compute the approximate gradients as $\frac{\partial \bar{\mathbf{F}}}{\partial \omega_e} = \bar{\mathbf{F}}\mathbf{W}_e$. The matrix $\mathbf{J}$ has a closed form derivative and we compute the derivative of $\mathbf{LL}^T$ using finite differences of the rigid body force samples, so $\frac{\partial \mathbf{JLL}^T\mathbf{J}^T}{\partial \omega_e}$ is straightforward to evaluate.

7.1.3 Evaluation of Derivative $\frac{\partial \mathbf{J}}{\partial \omega_e}$

Here we explain how to compute the derivative $\frac{\partial \mathbf{J}}{\partial \omega_e}$. The general form of $\mathbf{J}$ is shown in equation 7.8. The matrix $\mathbf{J}^m$ maps a surface index to full volume index. Because the FEM mesh does not change during optimization, $\frac{\partial \mathbf{J}^m}{\partial \omega_e} = 0$. We only need to consider last 6 columns of $\mathbf{J}$: $\begin{bmatrix} \mathbf{J}^{com} & \mathbf{J}^{\tau} \end{bmatrix}$.

Before going into the details of $\mathbf{J}^{com}$ and $\mathbf{J}^{\tau}$, denote the total mass of the object as M, and the total number of element nodes as n. The mass for each node i is m_i, the fill ratio for each element e is ω_e, and the mass of each element is $d_e = \rho\omega_e$, where ρ is the density.

Because we use hexahedral elements in the FEM mesh, we compute the node mass as:

$$m_i = \sum_{j \in N(i)} \frac{1}{8} d_j, \tag{7.13}$$

where $N(i)$ denotes the set of element indices adjacent to node i.

The matrix $\mathbf{J}^{com}$ maps the center of mass force to each element. It is a tiled diagonal matrix of 3×3 of the following form:

$$\mathbf{J}^{com} = \frac{1}{M} \begin{bmatrix} \vdots & \vdots & \vdots \\ m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \\ \vdots & \vdots & \vdots \end{bmatrix}. \tag{7.14}$$

To compute $\frac{\partial \mathbf{J}^{com}}{\partial \omega_e}$ for element e, we need to evaluate:

$$\frac{\partial (m_i/M)}{\partial \omega_e} = \frac{\partial m_i}{\partial \omega_e}\frac{1}{M} - \frac{\partial M}{\partial \omega_e}\frac{m_i}{M^2}, \quad i = 1, \ldots, n. \tag{7.15}$$

The second term on the right hand side of equation 7.15 is non zero for all $i = 1, \dots, n$, because all element nodes contribute to the total mass M:

$$-\frac{\partial M}{\partial \omega_e}\frac{m_i}{M^2} = -\sum_{k \in N(e)} \frac{\partial m_k}{\partial \omega_e}\frac{m_i}{M^2}. \tag{7.16}$$

Because element e contributes $\frac{1}{8}d_e$ to node m_k, we have

$$-\frac{1}{8}\sum_{k \in N(e)} \frac{\partial d_e}{\partial \omega_e}\frac{m_i}{M^2} = -\frac{1}{8}\sum_{k \in N(e)} \rho\frac{m_i}{M^2} = -\rho\frac{m_i}{M^2}, \quad i = 1, \dots, n. \tag{7.17}$$

Denote this part of the derivative as $\hat{\mathbf{J}}^c$. It is a dense tiled matrix of the form:

$$\hat{\mathbf{J}}^c = \frac{\rho}{M^2}\begin{bmatrix} \vdots & \vdots & \vdots \\ -m_i & 0 & 0 \\ 0 & -m_i & 0 \\ 0 & 0 & -m_i \\ \vdots & \vdots & \vdots \end{bmatrix}, \tag{7.18}$$

where every $i = 1 \dots n$ is tiled with the 3×3 diagonal matrix.

Now consider the first part of equation 7.15, which is non zero only for nodes i that are adjacent to element e. We have:

$$\frac{\partial m_i}{\partial \omega_e}\frac{1}{M} = \frac{\partial m_k}{\partial \omega_e}\frac{1}{M} = \frac{\rho}{8M}, \quad for \quad k \in N(e). \tag{7.19}$$

This part of the derivative is $\tilde{\mathbf{J}}_e^c$, a *sparse* matrix of the form:

$$\tilde{\mathbf{J}}_e^c = \begin{bmatrix} \vdots & \vdots & \vdots \\ \frac{\rho}{8M} & 0 & 0 \\ 0 & \frac{\rho}{8M} & 0 \\ 0 & 0 & \frac{\rho}{8M} \\ \vdots & \vdots & \vdots \end{bmatrix}, \tag{7.20}$$

where only node indices $k \in N(e)$ are filled with the 3×3 diagonal matrix. So in summary:

$$\frac{\partial \mathbf{J}^{com}}{\partial \omega_e} = \hat{\mathbf{J}}^c + \tilde{\mathbf{J}}_e^c. \tag{7.21}$$

The matrix $\mathbf{J}^\tau$ maps the center of mass torque to the elements. It is a tiled 3×3 skew-symmetric matrix, where each matrix denotes a cross product:

$$\mathbf{J}^\tau = \frac{1}{M} \begin{bmatrix} \vdots & \vdots & \vdots \\ 0 & \mathbf{r}_{iz} m_i & -\mathbf{r}_{iy} m_i \\ -\mathbf{r}_{iz} m_i & 0 & \mathbf{r}_{ix} m_i \\ \mathbf{r}_{iy} m_i & -\mathbf{r}_{ix} m_i & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}, \tag{7.22}$$

where $\mathbf{r}_i = \mathbf{p}_i - \mathbf{q}$ denotes the vector pointing from the center of mass ($\mathbf{q}$) to the position of node i ($\mathbf{p}_i$). Subscripts x, y, z denote the respective components of that vector. To compute its derivative, consider the z component:

$$\frac{\partial(\mathbf{r}_{iz} m_i / M)}{\partial \omega_e} = \frac{\partial \mathbf{r}_{iz}}{\partial \omega_e} \frac{m_i}{M} + \mathbf{r}_{iz} \frac{\partial (m_i / M)}{\partial \omega_e}, \quad i = 1, \ldots, N. \tag{7.23}$$

The second term of 7.23 is computed with equations 7.15 and 7.19, and results in a constant part which is included in equation 7.28, as well as the following non-constant sparse term:

$$\tilde{\mathbf{J}}_e^r = \frac{\rho}{8M} \begin{bmatrix} \vdots & \vdots & \vdots \\ 0 & \mathbf{r}_{kz} & -\mathbf{r}_{ky} \\ -\mathbf{r}_{kz} & 0 & \mathbf{r}_{kx} \\ \mathbf{r}_{ky} & -\mathbf{r}_{kx} & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}. \tag{7.24}$$

Similar to equation 7.20, only node indices $k \in N(e)$ are filled. For the first term in equation 7.23, we have:

$$\frac{\partial \mathbf{r}_{iz}}{\partial \omega_e}\frac{m_i}{M} = \frac{\partial(\mathbf{p}_{iz} - \mathbf{q}_z)}{\partial \omega_e}\frac{m_i}{M} = -\frac{\partial \mathbf{q}_z}{\partial \omega_e}\frac{m_i}{M}. \tag{7.25}$$

Since $\mathbf{q}_z = \sum_{k=1}^{n} \mathbf{p}_{kz} m_k / M$, denote $\boldsymbol{s} = \sum_{k=1}^{N} \mathbf{p}_{kz} m_k$, we have:

$$-\frac{\partial(\boldsymbol{s}/M)}{\partial \omega_e}\frac{m_i}{M} = -\frac{\partial \boldsymbol{s}}{\partial \omega_e}\frac{m_i}{M^2} + \frac{\mathbf{q}_z m_i}{M^2}\frac{\partial M}{\partial \omega_e}, \quad i = 1, \ldots, N. \tag{7.26}$$

The second part of this equation is equal to $\rho\frac{\mathbf{q}_z m_i}{M^2}$ for all i, so we have the total contribution to the constant part of derivative as:

$$(\mathbf{q}_z - \mathbf{r}_{iz})\frac{\rho m_i}{M^2}, \quad i = 1, \ldots, N, \tag{7.27}$$

we denote this part as $\hat{\mathbf{J}}^{\tau}$, which is constant for all elements:

$$\hat{\mathbf{J}}^{\tau} = \frac{\rho}{M^2}\begin{bmatrix} \vdots & \vdots & \vdots \\ 0 & (\mathbf{q}_z - \mathbf{r}_{iz})m_i & -(\mathbf{q}_y - \mathbf{r}_{iy})m_i \\ -(\mathbf{q}_z - \mathbf{r}_{iz})m_i & 0 & (\mathbf{q}_x - \mathbf{r}_{ix})m_i \\ (\mathbf{q}_y - \mathbf{r}_{iy})m_i & -(\mathbf{q}_x - \mathbf{r}_{ix})m_i & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}. \tag{7.28}$$

For the first term of equation 7.26, we have:

$$\begin{aligned} \frac{\partial \boldsymbol{s}}{\partial \omega_e}\frac{m_i}{M^2} &= -\sum_{k\in N(e)} \mathbf{p}_{kz}\frac{\partial m_k}{\partial \omega_e}\frac{m_i}{M^2} \\ &= -\sum_{k\in N(e)} \frac{1}{8}\mathbf{p}_{kz}\rho\frac{m_i}{M^2}, \quad i = 1, \ldots, N. \end{aligned} \tag{7.29}$$

The summation of node position $\sum_{k\in N(e)} \frac{1}{8}\mathbf{p}_{kz}$ is the z component of the position of element e. Using the z component of the centroid of element e: $_{\mathtt{ez}} = \sum_{\mathtt{k\in N(e)}} \frac{1}{8}\mathbf{p}_{\mathtt{kz}}$, equation 7.29 can be computed as:

$$\frac{1}{M}\begin{bmatrix} \vdots & \vdots & \vdots \\ m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \\ \vdots & \vdots & \vdots \end{bmatrix} \frac{\rho}{M}\begin{bmatrix} 0 & -_{\mathtt{ze}} & _{\mathtt{ye}} \\ _{\mathtt{ze}} & 0 & -_{\mathtt{xe}} \\ -_{\mathtt{ye}} & _{\mathtt{xe}} & 0 \end{bmatrix} = \mathbf{J}^{com}\tilde{\mathbf{J}}^{\tau}_{e}, \tag{7.30}$$

where the matrix $\tilde{\mathbf{J}}^{\tau}_{e}$ denotes the right 3×3 matrix of element e. In summary, the derivative of $\mathbf{J}$ is:

$$\frac{\partial \mathbf{J}}{\partial \omega_e} = \begin{bmatrix} 0 & \hat{\mathbf{J}}^c & \hat{\mathbf{J}}^\tau \end{bmatrix} + \begin{bmatrix} 0 & \tilde{\mathbf{J}}^c_e & \tilde{\mathbf{J}}^r_e + \mathbf{J}^{com}\tilde{\mathbf{J}}^\tau_e \end{bmatrix}. \tag{7.31}$$

7.1.4 Our Linear Method

We first observe that in equation 7.6, the per-element force basis matrix $\bar{\mathbf{F}}$ is fixed, and only the smaller $\mathbf{W}_e \in \mathbb{R}^{r\times r}$ ever changes. Second, we observe that the final quantity $(\mathbf{K}^{-1}\mathbf{Y}) : \frac{\partial \bar{\mathbf{F}}}{\partial \boldsymbol{\omega}}$ is all that matters; the per-element intermediate $\frac{\partial \bar{\mathbf{F}}}{\partial \boldsymbol{\omega}_e}$ is not strictly required. Therefore, if we pre-contract $\bar{\mathbf{F}}$ with $\mathbf{K}^{-1}\mathbf{Y}$, we obtain a smaller matrix, $\bar{\mathbf{F}}^T\mathbf{K}^{-1}\mathbf{Y} \in \mathbb{R}^{r\times r}$. Each element can then be computed as $\bar{\mathbf{F}}^T\mathbf{K}^{-1}\mathbf{Y} : \mathbf{W}_e$, which is only $O(r^2)$ per element.

Specifically, each element must compute the product

$$g_e = \mathbf{K}^{-1}\mathbf{Y} : (\bar{\mathbf{F}}\mathbf{W}_e), \tag{7.32}$$

where g_e is the entry of the second term in Eqn. 7.5 for element e. Both $\mathbf{Z} = \mathbf{K}^{-1}\mathbf{Y}$ and $\bar{\mathbf{F}}$ are static, per-element, $\mathbb{R}^{3n\times r}$ matrices that we use to rewrite g_e as

$$\begin{aligned} g_e &= \sum_{i,j} (\mathbf{Z})_{ij} (\bar{\mathbf{F}}\mathbf{W}_e)_{ij} = \sum_{i,j,k} \mathbf{Z}_{ij}\bar{\mathbf{F}}_{ik}\mathbf{W}_{kje} = \sum_{i,j,k} \bar{\mathbf{F}}_{ik}\mathbf{Z}_{ij}\mathbf{W}_{kje} \\ &= (\bar{\mathbf{F}}^T\mathbf{Z}) : \mathbf{W}_e. \end{aligned} \tag{7.33}$$

Since $\bar{\mathbf{F}}^T\mathbf{Z} \in \mathbb{R}^{r\times r}$ is fixed for all e, we can precompute it in $O(mr^2)$ time. At runtime, an $O(r^2)$ contraction is performed over m elements, yielding an $O(mr^2)$ overall running time.

So far, we have only examined the derivative of the force basis in Eqn. 7.6. However, naïvely evaluating $\mathbf{W}_e$ also takes $O(m^2)$ time. We show in the following section that this can also be reduced to $O(m)$ by leveraging matrix sparsity and similar pre-computations.

7.1.5 Enhanced Evaluation for $\mathbf{W}_e$

To evaluate $\mathbf{W}_e$, first we need to evaluate $\mathbf{B}_e$, and then compute $\mathbf{W}_e$ according to equations 7.11 and 7.12. Using equation 7.9 and expanding derivatives, we have:

$$\begin{aligned}&\mathbf{B}_e = \mathbf{B}_e^1 + \mathbf{B}_e^2 + (\mathbf{B}_e^1)^T \\ &\mathbf{B}_e^1 = \bar{\mathbf{F}}^T \frac{\partial \mathbf{J}}{\partial \omega_e} \mathbf{L}\mathbf{L}^T \mathbf{J}^T \bar{\mathbf{F}} \qquad \mathbf{B}_e^2 = \bar{\mathbf{F}}^T \mathbf{J} \frac{\partial \mathbf{L}\mathbf{L}^T}{\partial \omega_e} \mathbf{J}^T \bar{\mathbf{F}}.\end{aligned} \tag{7.34}$$

First we consider the evaluation of $\mathbf{B}_e^1$. Because only the $\frac{\partial \mathbf{J}}{\partial \omega_e}$ factor is different from element to element, we can precompute $\mathbf{L}\mathbf{L}^T\mathbf{J}^T\bar{\mathbf{F}} \in \mathbb{R}^{(n_u+6)\times r}$ in $O(n_u^2 r + mr)$ for all the elements. The major complication is to compute $\bar{\mathbf{F}}^T \frac{\partial \mathbf{J}}{\partial \omega_e}$ efficiently for all elements.

From Appendix 7.1.3, $\frac{\partial \mathbf{J}}{\partial \omega_e}$ is composed of three parts:

$$\frac{\partial \mathbf{J}}{\partial \omega_e} = \begin{bmatrix} 0 & \hat{\mathbf{J}}^c & \hat{\mathbf{J}}^\tau \end{bmatrix} + \begin{bmatrix} 0 & \tilde{\mathbf{J}}_e^c & \tilde{\mathbf{J}}_e^r + \mathbf{J}^{com}\tilde{\mathbf{J}}_e^\tau \end{bmatrix}, \tag{7.35}$$

where $\hat{\mathbf{J}}^c, \hat{\mathbf{J}}^\tau \in \mathbb{R}^{3n\times 3}$ is constant for all the elements, $\tilde{\mathbf{J}}_e^c$, $\tilde{\mathbf{J}}_e^r \in \mathbb{R}^{3n\times 3}$ is different from element to element (but its sparsity is of $O(c)$, where c is a constant), $\mathbf{J}^{com} \in \mathbb{R}^{3n\times 3}$ is a constant matrix, and $\tilde{\mathbf{J}}_e^\tau \in \mathbb{R}^{3\times 3}$ differs from element to element (but it is a small matrix).

Computing $\bar{\mathbf{F}}^T \frac{\partial \mathbf{J}}{\partial \omega_e}$ naively for all elements poses a complexity of $O(m^2 r)$. However,

$$\bar{\mathbf{F}}^T \frac{\partial \mathbf{J}}{\partial \omega_e} = \begin{bmatrix} 0 & \bar{\mathbf{F}}^T\hat{\mathbf{J}}^c & \bar{\mathbf{F}}^T\hat{\mathbf{J}}^\tau \end{bmatrix} + \begin{bmatrix} 0 & \bar{\mathbf{F}}^T\tilde{\mathbf{J}}_e^c & \bar{\mathbf{F}}^T\tilde{\mathbf{J}}_e^r + \bar{\mathbf{F}}^T\mathbf{J}^{com}\tilde{\mathbf{J}}_e^\tau \end{bmatrix} \tag{7.36}$$

where $\bar{\mathbf{F}}^T\hat{\mathbf{J}}^c, \bar{\mathbf{F}}^T\hat{\mathbf{J}}^\tau$ and $\mathbf{X} = \bar{\mathbf{F}}^T\mathbf{J}^{com}$ can be precomputed for all elements in $O(mr)$. The only terms which are needed per-element are $\bar{\mathbf{F}}^T\tilde{\mathbf{J}}_e^c, \bar{\mathbf{F}}^T\tilde{\mathbf{J}}_e^r$ and $\mathbf{X}\tilde{\mathbf{J}}_e^\tau$. Because the three matrices $\tilde{\mathbf{J}}_e^c, \tilde{\mathbf{J}}_e^r$ and $\tilde{\mathbf{J}}_e^\tau$ have a constant number of non-zero entries, this can be done for all elements in $O(mr)$. Therefore, computing $\bar{\mathbf{F}}^T \frac{\partial \mathbf{J}}{\partial \omega_e}$ can be reduced from $O(m^2 r)$ to $O(mr)$. The final matrix products $\bar{\mathbf{F}}^T \frac{\partial \mathbf{J}}{\partial \omega^e}$ and $(\mathbf{L}\mathbf{L}^T\mathbf{J}^T\bar{\mathbf{F}})$ take $O(mr^2)$ for all elements.

Therefore, $\mathbf{B}_e^1$ can be computed in $O(mr^2)$.

Now consider the evaluation of $\mathbf{B}_e^2$. Finite differences are used to evaluate $\frac{\partial(\mathbf{L}\mathbf{L}^T)}{\partial\omega_e}$:

$$\frac{\partial(\mathbf{L}\mathbf{L}^T)}{\partial\omega_e} = \sum_{i=1}^{i=10} \frac{\partial(\mathbf{L}\mathbf{L}^T)}{\partial M_i} \frac{\partial M_i}{\partial\omega_e}. \tag{7.37}$$

The terms $\frac{\partial(\mathbf{L}\mathbf{L}^T)}{\partial M_i}$ are 10 matrices which can be precomputed in $O(n_u n_s)$ for all elements. The matrix product $(\bar{\mathbf{F}}^T\mathbf{J})\frac{\partial\mathbf{L}\mathbf{L}^T}{\partial\omega^e}(\mathbf{J}^T\bar{\mathbf{F}})$ takes $O(m(rn_u^2 + r^2 n_u))$ for all elements. This is the total complexity for $\mathbf{B}_e^2$.

This part can also be accelerated through further precomputation. Denote $\mathbf{A}_i = \frac{\partial(\mathbf{F}^T\mathbf{F})}{\partial M_i}$, $\mathbf{H} = \bar{\mathbf{F}}^T\mathbf{J} \in \mathbb{R}^{r\times(n_u+6)}$. We have:

$$\mathbf{B}_e^2 = \mathbf{H}\sum_{i=1}^{i=10} \mathbf{A}_i \frac{\partial M_i}{\partial\omega_e}\mathbf{H}^T = \sum_{i=1}^{i=10} \mathbf{H}\mathbf{A}_i\mathbf{H}^T\frac{\partial M_i}{\partial\omega_e}. \tag{7.38}$$

Since only $\frac{\partial M_i}{\partial\omega_e}$ changes between elements, we can precompute $\mathbf{H}\mathbf{A}_i\mathbf{H}^T \in \mathbb{R}^{r\times r}$ before doing the summation. This reduces the complexity for evaluation of $\mathbf{B}_e^2$ to $O(mr^2)$, and also reduces the total complexity for $\mathbf{B}_e$ to $O(mr^2)$.

We show in Table 7.1 the running time of computing the total probability gradients using the previous quadratic method (estimated) and our linear method. Our method is asymptotically faster, and accelerates this stage of the method by two orders of magnitude, effectively removing it as the bottleneck of the method.

	Time (s)		
Resolution	[27]	**Our Method**	**Speedup**
$28 \times 32 \times 36$	$1.84^{\dagger}$	0.041	**44.9**×
$28 \times 44 \times 28$	$6.18^{\dagger}$	0.044	**140**×
$40 \times 64 \times 60$	$41.7^{\dagger}$	0.128	**326**×

Table 7.1: Running time of Eqn. 7.6 using the quadratic method of Langlois et al.[27] and our linear method. †Estimated time.

7.2 Stabilizing the Inertia Gradients

Previous Method

We examine the relevant term from Eqn. 7.5,

$$\mathbf{x} = \sum_{i=1}^{n_s} \frac{\partial \boldsymbol{\alpha}^i}{\partial \boldsymbol{\omega}} \bar{\mathbf{U}}^T \mathbf{B}^T \mathbf{C}^T \mathbf{c}^i, \tag{7.39}$$

that measures how changing the voxel densities $\boldsymbol{\omega}$ influence the rigid body force samples $\boldsymbol{\alpha}^i$. The previous method [27] evaluates this term using finite differences.

Given n_s samples $\boldsymbol{\alpha}^i \in \mathbb{R}^r$, we assume each entry of $\boldsymbol{\alpha}^i$ is sampled from a 1D PDF, and take its finite difference over that distribution. The j^{th} entry of each force sample α_j is denoted α. We assume α is a random variable and that α_j^i is drawn from the PDF $c(\alpha)$.

Instead of computing a finite difference for the sample α_j^i, we perform a finite difference over its distribution. For any α, we can compute its CDF as $C(\alpha)$, and retrieve α_j^i by sampling its inverse using $\alpha_j^i = C^{-1}(u)$, where u is a uniform random variable. Using the properties of the CDF, we obtain:

$$\begin{aligned}\frac{\partial C(C^{-1}(u))}{\partial \boldsymbol{\omega}} &= \frac{\partial u}{\partial \boldsymbol{\omega}} = 0 \\ \frac{\partial C(C^{-1}(u))}{\partial \boldsymbol{\omega}} &= \frac{\partial C}{\partial \alpha}\bigg|_{\alpha_j^i} \frac{\partial C^{-1}}{\partial \boldsymbol{\omega}}\bigg|_{u} + \frac{\partial C}{\partial \boldsymbol{\omega}}\bigg|_{\alpha} = 0.\end{aligned} \tag{7.40}$$

By manipulating Eqn. 7.40, we obtain the derivative of α:

$$\frac{\partial \alpha}{\partial \boldsymbol{\omega}} = \frac{\partial C^{-1}}{\partial \boldsymbol{\omega}}\bigg|_{u} = -\frac{1}{c(\alpha)}\frac{\partial C}{\partial \boldsymbol{\omega}}\bigg|_{\alpha}. \tag{7.41}$$

We insert $\alpha = \alpha_j^i$ into Eqn. 7.41 to compute the gradient for each random sample. Therefore, an important step in computing the inertia gradients is building the distributions $C(\alpha)$ and $c(\alpha)$ over the samples α_j^i, where $i = 1 \dots n_s$.

Langlois et al. [27] computed $C(\alpha)$ and $c(\alpha)$ by fitting a uniform 1D finite element grid over the samples, which represents them as a sum of basis functions: $c(\alpha) = \sum_{i=1}^{k} a_i \psi_i(\alpha)$. Here, k is the number of elements of this 1D finite element grid, a_i is a shape coefficient, and $\psi_i(\alpha)$ is a symmetric hat function of element i. A Galerkin method is then used to solve for a_i. However, this approach can fit the data poorly. In the top of Fig. 7.1,

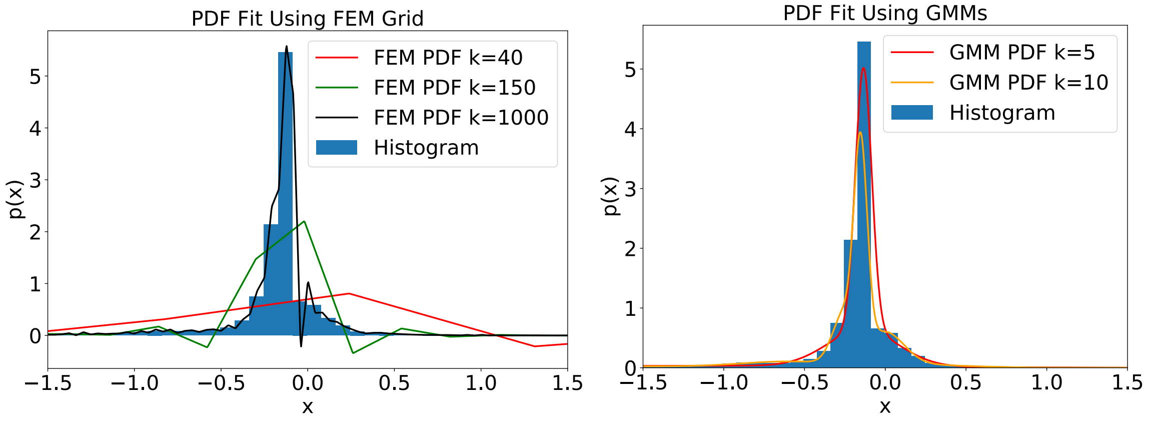


Figure 7.1: The PDF constructed from an FEM grid fits the data poorly and leads to unstable probability gradients. The distribution has a long tail; we have zoomed into a portion of the x axis. Our GMM fit does not exhibit these artifacts.

we show the PDF $c(\alpha)$ that results from this approach using n elements. For a small n, the histogram is fit poorly. As n increases, ringing artifacts appear. This leads to instabilities when computing $\frac{1}{c(\alpha)}$ in Eqn. 7.41 because the ringing artifacts can cause the PDF to become negative.

Stabilized Inertia Gradients

We use Gaussian Mixture Models (GMMs) to address this problem. GMMs are widely used to capture discrete distributions [94], and we found that they yield results superior to the FEM approach.

We first expand $c(\alpha)$ using a set of Gaussians:

$$c(\alpha) = \sum_{i=1}^{k} \pi_i \mathcal{N}(\alpha \,|\, \mu_k, \sigma_k), \tag{7.42}$$

where k is the number of Gaussians, π_k is a weight, and $\mathcal{N}(\alpha \,|\, \mu_k, \sigma_k)$ is a Gaussian with mean μ_k and variance σ_k. The parameters, π_k, σ_k, μ_k, can be computed from α_j^i using expectation maximization [94], and we found that $k = 10$ usually yields good results. This leads to the superior distribution representations we show in the bottom of Fig. 7.1.

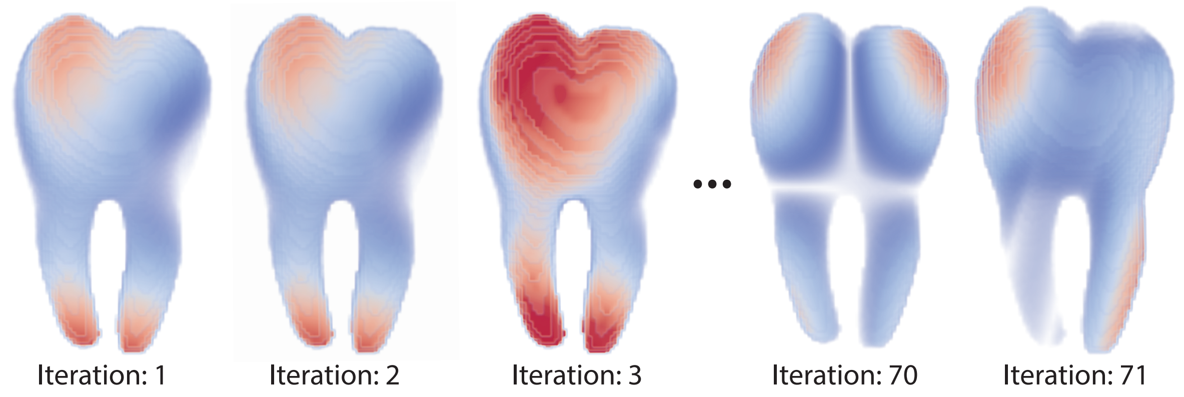
Returning to Eqn. 7.41, $\frac{\partial C}{\partial \boldsymbol{\omega}}$ is also evaluated using finite differences. We assume that all the potential contact positions of the rigid-body lies on the surface of the object. So the rigid-body can be parameterized by its mass, center of mass, and moment of inertia. As a result, α_j^i and its CDF $C(\alpha)$ can be parameterized using a total of 10 variables. Denoting the parameters as $M_i, i = 1 \ldots 10$, the finite difference is computed as:

$$\left.\frac{\partial C}{\partial \boldsymbol{\omega}}\right|_{\alpha} = \sum_{i=1}^{10} \left.\frac{\partial C}{\partial M_i}\right|_{\alpha} \frac{\partial M_i}{\partial \boldsymbol{\omega}}. \tag{7.43}$$

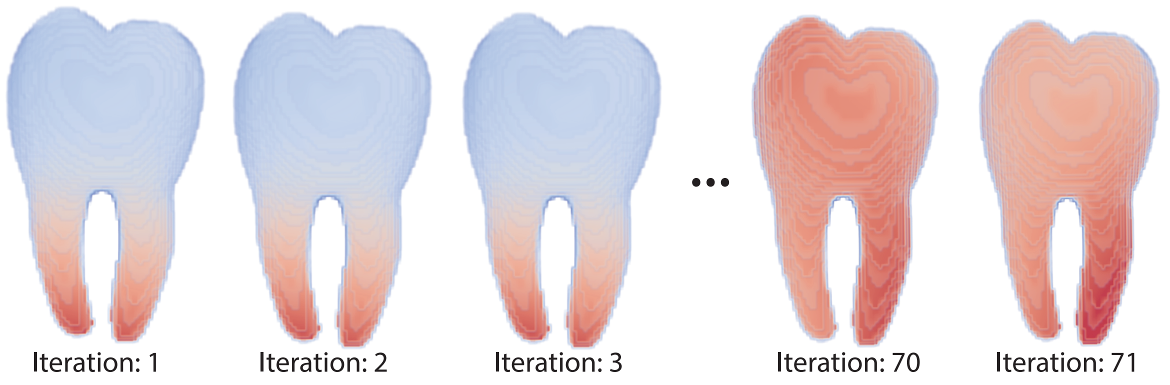
The $\frac{\partial M_i}{\partial \boldsymbol{\omega}}$ is straightforward, and we use a centered difference for $\frac{\partial C}{\partial M_i}$:

$$\left.\frac{\partial C}{\partial M_i}\right|_{\alpha} = \frac{1}{2}\frac{C(M_i + \Delta M)|_{\alpha} - C(M_i - \Delta M)|_{\alpha}}{\Delta M}. \tag{7.44}$$

Evaluating the above equation involves two extra rounds of rigid body simulation with $M_i + \Delta M$ and $M_i - \Delta M$ perturbations applied to each parameter, for a total of 20 rounds.



(a) Previous, unstable probability gradients



(b) Our stable probability gradients

Figure 7.2: Probability gradients from the first three optimization frames and two frames near the end. The previous method changes wildly, even when the underlying densities change very little. Our method changes slowly, in step with the density changes.

The resulting probability distribution leads to much stabler inertia gradients, as shown in Fig. 7.2. In that figure, we show the probability gradients from the first three frames of the optimization as well as two frames near the end. The element densities change only slightly during these frames, and yet the FEM-based gradients oscillate wildly. Using our method, the gradient becomes very stable, and greatly improves the convergence of the optimization (Fig. 7.12).

7.3 Constrained Restart Strategy

Eqn. 6.7 has a non-linear constraint and a large number of variables, so we use the Method of Moving Asymptotes (MMA) [64] as our optimization algorithm.

The non-linearity of the probability constraint causes the optimization to often fall out of the feasible region. MMA can recover, but often at the cost of oscillatory behavior that converges to sub-optimal local minima [64]. One popular technique for addressing this well-known problem is the Solid Isotropic Material with Penalization (SIMP) model [62], but we found it to be insufficient for our case as shown in the following section.

7.3.1 Optimization with Different SIMP Parameters

SIMP penalizes intermediate densities in structural optimization by exponentiating them when computing the stiffness matrix. This makes the intermediate densities uneconomical in the optimization, attempting to force them to go to either zero or one. The exponent is a parameter, where increasing it will further penalize the intermediate densities, but render the optimization more difficult.

In figure 7.4, we use SIMP with an exponent of 5. We experimented with different choices and found it does not improve the results. As shown in figure 7.3, small SIMP exponents (2 or 3) lead to an unacceptable number of intermediate densities. Increasing the exponent above 10 often stall the optimization at the very beginning. We need a different approach to overcome the local minima.

Instead, we found the following *constrained restart* method to be effective, which can

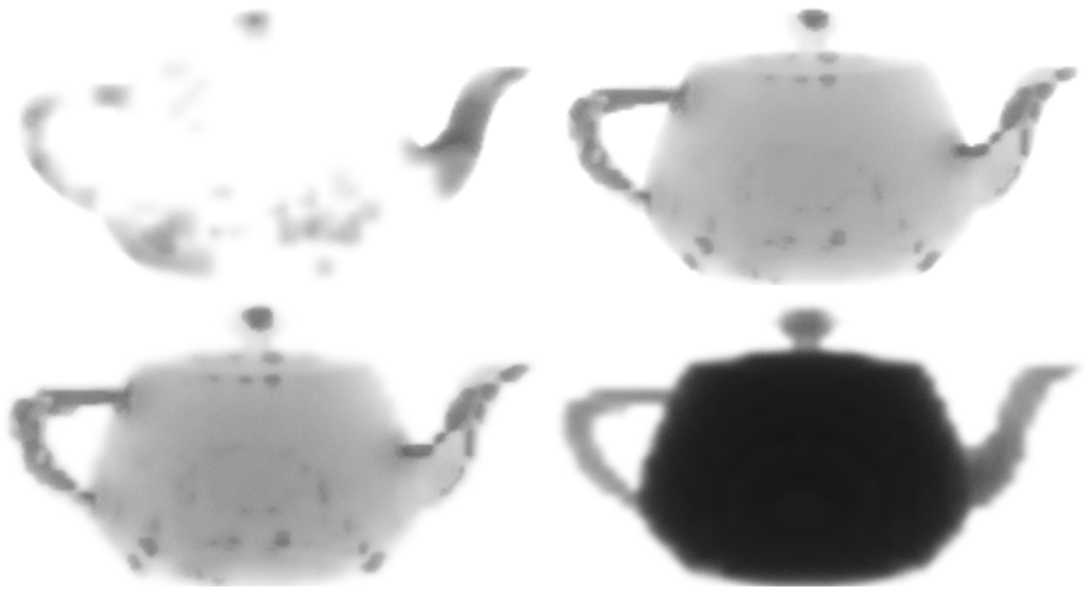


Figure 7.3: Top row: Converged results using SIMP exponent 2,3 respectively. Bottom row: Converged results using SIMP exponent 5, 10, respectively.

be viewed as a form of either block-cyclic reduction [28] or sand-filling [95]. We observe that when the optimization stalls, it usually has found a preliminary, but promising, reinforcement structure. Therefore, we perform multiple optimization passes where the promising structures from the previous iteration are used as an initial guess.

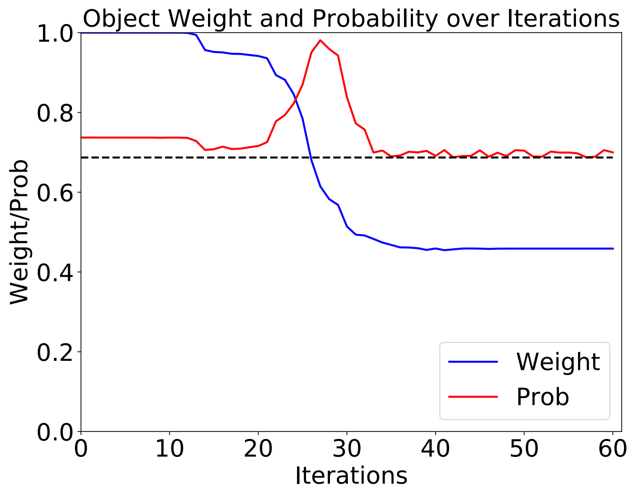
We isolate these structures by applying threshold c to the current solution and constraining the results to $\omega_e = 1$ in Eqn. 6.7.

Next, we add a perturbation to push the global solution state out of its current local minimum.

We compute extension density field $\boldsymbol{\beta}$ of the stalled solution $\boldsymbol{\omega}^{i-1}$ and then use $\min(\boldsymbol{\omega}^{i-1} + \boldsymbol{\beta}, 1)$ as the initial state for the next pass. The extension field is computed as:

$$\boldsymbol{\beta} = \max\left(1 - \frac{\mathrm{SDF}(V)}{b_\beta}, 0\right), \tag{7.45}$$

where SDF is a signed distance field, V is the set of constrained voxels and b_β is a



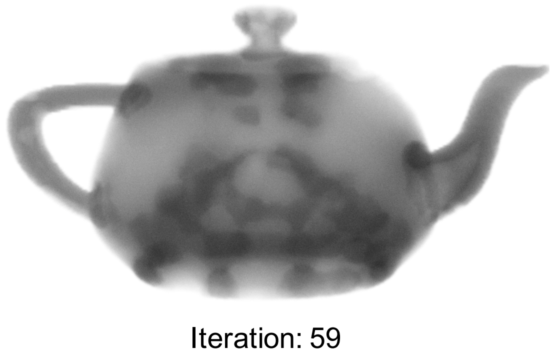


Figure 7.4: Left: The object weight and survival probability during the optimization. Right: The local optimum at iteration 59.

bandwidth parameter. Fig. 7.5 illustrates these quantities. The strategy essentially inflates the existing reinforcement structure for the next optimization pass.

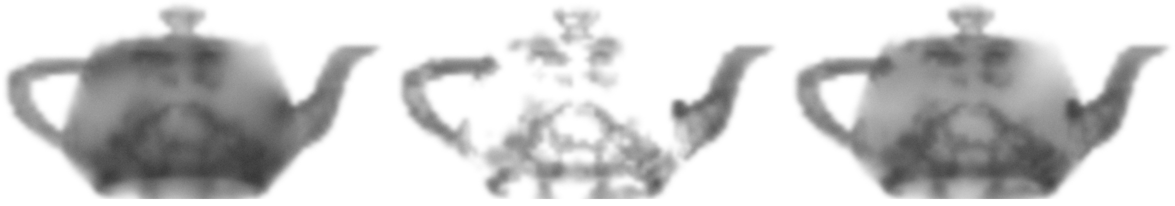


Figure 7.5: Left: Solution $\boldsymbol{\omega}^{i-1}$ after one optimization pass. Middle: The reinforcement structure V from $\boldsymbol{\omega}^{i-1}$. Right: Extension field $\boldsymbol{\beta}$ of V.

Using the extension field $\boldsymbol{\beta}$ and constraints on V, we run the next optimization pass. We found this to be effective in perturbing the solution from local minima and finding sparser and more interesting structures, e.g. as shown in Fig. 7.6. The complete optimization is listed in Algorithm 2. We found that $n_{opt} = 3$ rounds of optimization with MMA produces converged results. As shown in Fig. 7.7, $n_{opt} > 3$ produces little change. In all our computations we set $c = 0.7$ and $b_{\beta} = 16$ grid cells.

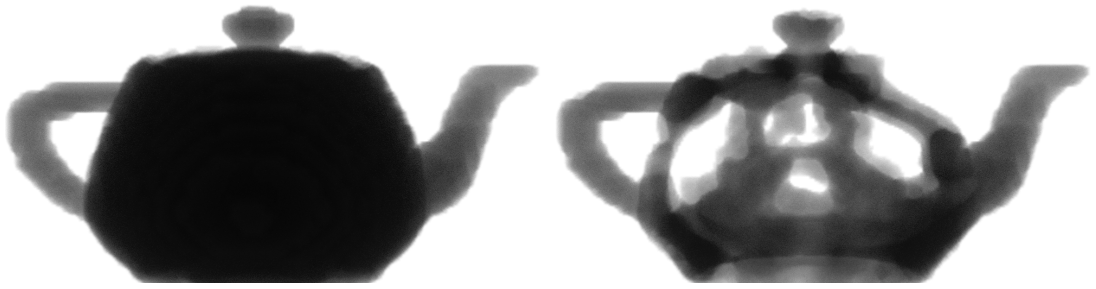


Figure 7.6: Left: Initial state $\min(\boldsymbol{\omega}^{i-1} + \boldsymbol{\beta}, 1)$ for the next round of optimization. Right: Converged result $\boldsymbol{\omega}^{i}$ after the next round.

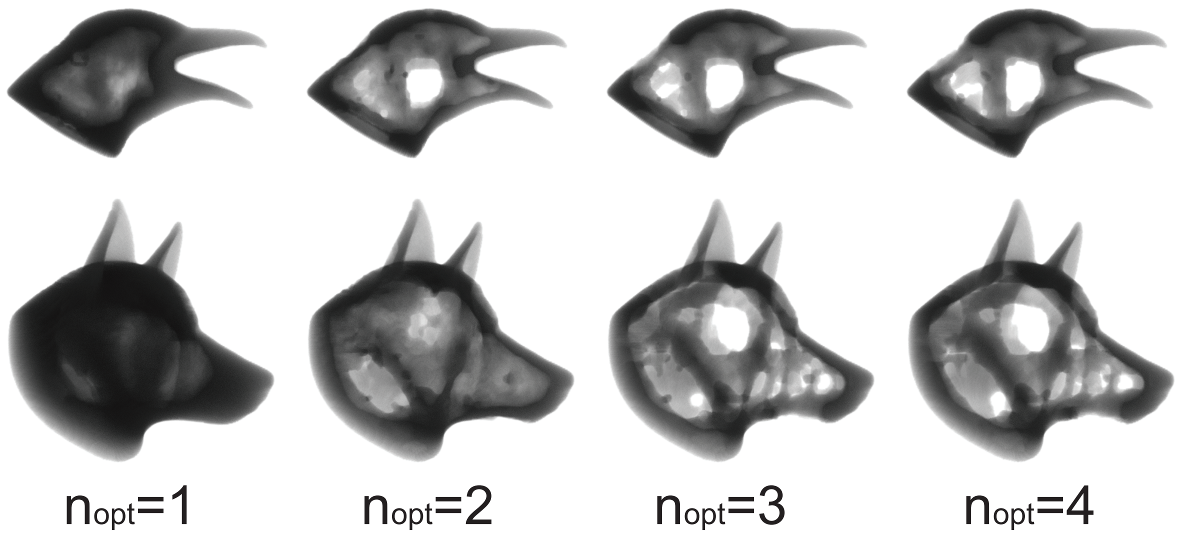


Figure 7.7: Results with different n_{opt}. We set $n_{opt} = 3$ because $n_{opt} > 3$ yielded negligible improvements.

7.3.2 Sheathing Post-Process

Many algorithms constrain the exterior of the object during the optimization [75, 27]. However, the final object then has an outer shell that is the thickness of the (quite coarse) voxel grid. However, as the shell is thickened, it can become the main reinforcement structure in the model [75], which biases the optimization towards shell-like designs, and results in heavier objects. Instead, we only constrain the shell at force contact locations, allowing the optimizer to find a lighter reinforcement structure, and allowing surface

Algorithm 2 Incremental Shape Optimization

```
Input: User-defined geometry , user defined failure probability $\Theta$
 1: procedure SHAPE OPTIMIZATION
 2:     $\boldsymbol{\omega}^1 \leftarrow \mathbf{1}$
 3:     Optimize $\boldsymbol{\omega}^1$ using MMA to convergence
 4:     $V \leftarrow$ voxels in $\boldsymbol{\omega}^1$ with density larger than $c$
 5:     for $i$ from 2 to $n_{opt}$ do
 6:         Compute $\boldsymbol{\beta}$ from $V$
 7:         $\boldsymbol{\omega}^i \leftarrow \min(\boldsymbol{\omega}^{i-1} + \boldsymbol{\beta}, 1)$
 8:         Constrain densities of $V$ to 1
 9:         Optimize $\boldsymbol{\omega}^i$ using MMA to convergence
10:         $V \leftarrow$ voxels in $\boldsymbol{\omega}^i$ with density larger than $c$
11:     end for
12:     $V \leftarrow \emptyset$
13:     $\boldsymbol{\omega}^f \leftarrow \boldsymbol{\omega}^m$
14:     Optimize $\boldsymbol{\omega}^f$ using MMA to convergence
15: end procedure
```

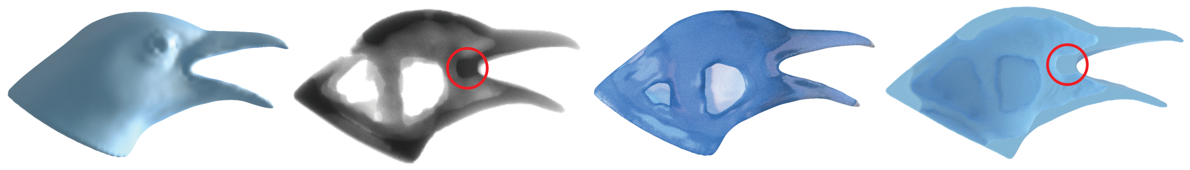


Figure 7.8: A raven model optimized using our method. From left to right: The initial object, the optimized density field, the final shape, and a cut-away view of the final shape. Our method automatically reinforces the corner (commissure) of the beak, circled in red, which is likely to break under real-world impacts. Material is subtracted from regions that are unlikely to experience impacts.

regions which are unlikely to experience contact to be hollowed out. To preserve the surface geometry, we perform a post-process that adds a thin "sheath" of material, far below the resolution that we can simulate, along the original surface. We show in §7.4.4 that this sheathing has a minimal impact on the object's performance.

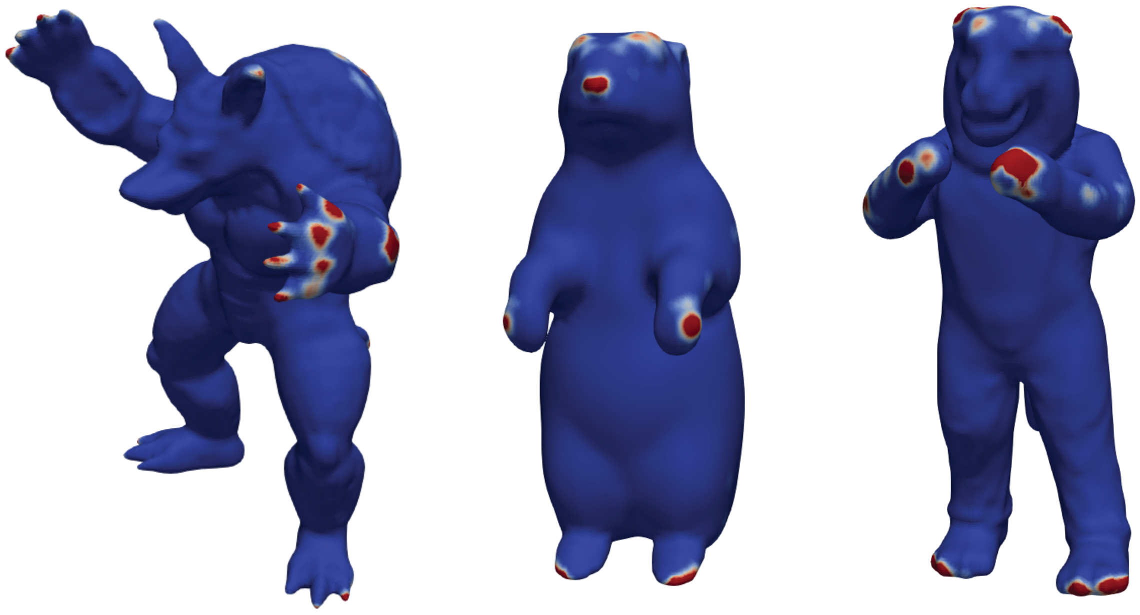


Figure 7.9: Force contact locations for different examples. Surfaces marked red are possible contact locations.

7.4 Implementation and Results

7.4.1 Implementation Details

We initially voxelize the surface mesh according to the resolutions in Tbl. 7.4. We use Bullet [96] to obtain rigid body force samples. For all our examples, we use the following scenario: the shape falls 1 meter with a random initial orientation and small random angular velocity, and hits a flat plane. We record the three initial contact events when the shape hits the ground. We use $n_s = 5000$ rigid body simulations in all our examples and kept 90% of the total variance when performing PCA on the force samples. To avoid checkerboard patterns, we augment our cost function with the energy term of from Schumacher et al. [97]. We constrain the rigid body contact locations to $\omega_e = 1$ to ensure that they do not change during optimization. As shown in Fig 7.9, these locations are usually very sparse and lie on the object's convex hull. We use the following material parameters: Young's modulus = 2.2 GPa, density = 1.037 g/cm^3, and yield stress = 0.031 GPa. The object is scaled so that the maximal bounding box dimension is 15 cm.

We implemented our algorithm in C++. OpenMP multi-threading was used whenever possible, Eigen [88] is used for most matrix operations, Intel's Paradiso was used to solve linear systems, and Armadillo [98] was used for the GMMs. All our results were run on a desktop with 192 GB of memory and a 2.4 GHz, 20-core Intel 6148.

7.4.2 Optimization Results

Fig. 7.10 shows that our method adds densities to locations of potential high stress. In particular, fragile locations such as the head of the beaver, the extremities of the camel, the handle and spout tip of the teapot, the beak corner (commissure) on the crow (Fig. 4.7), and the pelvis of the Armadillo (Fig. 7.15, top) are reinforced during optimization. Conversely, material is removed from low-stress regions. We do not constrain the surface, so interesting contact-dependent structures form, such as the removal of the beaver's back, the tiger's chest, and the divot between the camel's humps.

To further explore the dependence of our algorithm on real-world contacts, we optimized a beaver model with and without its tail (Fig. 7.11). The optimizer aggressively subtracts material along the beaver's back when collisions in that region become unlikely. Fig. 7.12 compares our results to Langlois et al. [27], and shows that our method consistently produces a superior (lighter) object. Tbl. 7.2 shows the survival probability remains similar, and we achieve up to 3.22× lighter objects. Fig. 7.13 shows the object weights and survival probabilities during optimization. Langlois et al. [27] stalls early, while our method makes steady progress.

Model	Method	Weight (gram)	Improvement	Survival Probability
Rabbit	[27]	97.3		0.630
	Ours	36.7	**2.65**×	0.662
Penguin	[27]	94.8		0.650
	Ours	29.4	**3.22**×	0.767
Molar	[27]	63.6		0.598
	Ours	32.3	**1.96**×	0.621

Table 7.2: The final object weight and survival probabilities of our method and Langlois et al. [27]. Our method consistently produces lighter objects with similar survival probabilities.

7.4.3 Post-Processing

As is common in many algorithms, we optimize over continuous densities (Fig. 7.14, left).

To obtain the final mesh (Fig. 7.14, middle), we use marching cubes on the density field [99] to obtain the $\omega = 0.5$ isocontour. Since we did not constrain the exterior shell of the object, we attach a thin shell of width $\frac{dx}{4}$ using the SDF of the original mesh, where dx is length of one hexahedron (Fig. 7.14, right). We show in §7.4.4 that this thin shell has a negligible effect on the final object's weight and survival probability.

7.4.4 Optimization Validation

We ran several comparisons to validate our sheathing post-process. First, we compared the results of the optimization with and without shell constraints (Tbl. 7.3). The shell constraint consistently produces heavier results, even through the survival probability is almost identical. We then added a thin sheath to the results without the shell constraint. The survival probability remains essentially the same (in the Armadillo case, it actually *improves*), and the object weight remains significantly below that found using the shell constraint.

Model	**Configuration**	**Weight (gram)**	**Survival Probability**
Armadillo	with shell constraint	86	0.706
	w/o shell constraint	56	0.706
	sheathing post-process	66	0.700
Raven	with shell constraint	63	0.624
	w/o shell constraint	42	0.624
	sheathing post-process	48	0.632
Teapot	with shell constraint	75	0.737
	w/o shell constraint	52	0.737
	sheathing post-process	62	0.747
Tiger	with shell constraint	65	0.813
	w/o shell constraint	45	0.813
	sheathing post-process	52	0.819

Table 7.3: Results with and without the shell constraint, and with the sheathing post-process from §7.3.2.

Additionally, we visualized the von Mises stresses for several models in Fig. 7.15. The heavier shell-constrained models produce larger rigid body impact forces, and therefore larger stresses. The sheath post-processed models have almost the same stress distribution as the original shell-unconstrained results, which indicate that the regions of likely impact have been effectively reinforced.

7.5 Physical Validation

We printed five copies of our Raven model with the sheath and five copies without. The results are shown in Fig. 7.16.

We then dropped each model from a height of 1.5 meters until a breakage occurred. Each drop used a random initial orientation, and the results are listed in Tbl. 7.5.

We computed the expected survival probabilities from these breakage statistics, and found that they are close to the probability predicted by our simulation.

			Time (s)						
Model	**Resolution**	**# Iters**	**Sampling**	**GMMs**	**FEM Solve**	**Grad**	**Total**	**Volume Reduction**	**Survival Probability**
Armadillo	$56 \times 64 \times 52$	189	72.52	31.40	8.36	289.17	401.45	67.4%	0.706
Camel	$20 \times 64 \times 52$	142	70.14	20.00	3.71	140.34	234.19	64.8%	0.734
Beaver(tail)	$56 \times 28 \times 64$	162	60.55	22.56	6.28	200.84	290.23	70.4%	0.617
Raven	$32 \times 64 \times 40$	159	62.11	31.00	5.00	260.70	328.18	73.1%	0.624
Teapot	$64 \times 36 \times 44$	168	60.80	43.00	7.51	282.51	393.83	71.7%	0.737
Dog	$40 \times 64 \times 60$	103	74.69	32.44	11.8	341.81	460.71	77.4%	0.754
Tiger	$32 \times 40 \times 64$	134	67.36	31.04	4.93	170.77	274.10	67.1%	0.813
Penguin	$36 \times 48 \times 40$	97	21.74	42.11	11.5	180.51	255.85	71.3%	0.767
Rabbit	$28 \times 44 \times 48$	103	17.28	35.15	6.57	99.72	158.72	68.9%	0.662
Hand	$40 \times 32 \times 24$	98	30.72	28.28	3.80	57.66	120.46	68.8%	0.868
Molar	$32 \times 28 \times 48$	91	33.41	41.87	5.45	127.03	207.76	60.0%	0.621
Panda	$28 \times 32 \times 36$	95	17.22	35.12	4.54	90.30	147.18	68.2%	0.776

Table 7.4: Timing breakdown (in seconds), volume reduction, and survival probability across different examples. The volume reduction is computed with final after the sheathing post-process.

In Fig. 7.17 we show two breakage patterns from the sheathed and un-sheathed objects. Breaks occur around the beak or contact locations. In all five of our tests, as predicted by our simulation, the sheath did not come into contact with the ground.

sheathed		un-sheathed	
weight	# of drops	weight	# of drops
44.02 g	4	41.68 g	5
45.00 g	4	42.62 g	3
44.84 g	3	42.48 g	3
44.44 g	3	41.77 g	2
44.89 g	2	41.26 g	2

Table 7.5: Number of experimental drops of the sheathed and un-sheathed objects before breakage. Our predicted probability of remaining intact is respectively 0.632 and 0.624. The expected probability of remaining intact computed from the breakage statistics and assuming a binomial distribution is respectively 0.688 and 0.666.

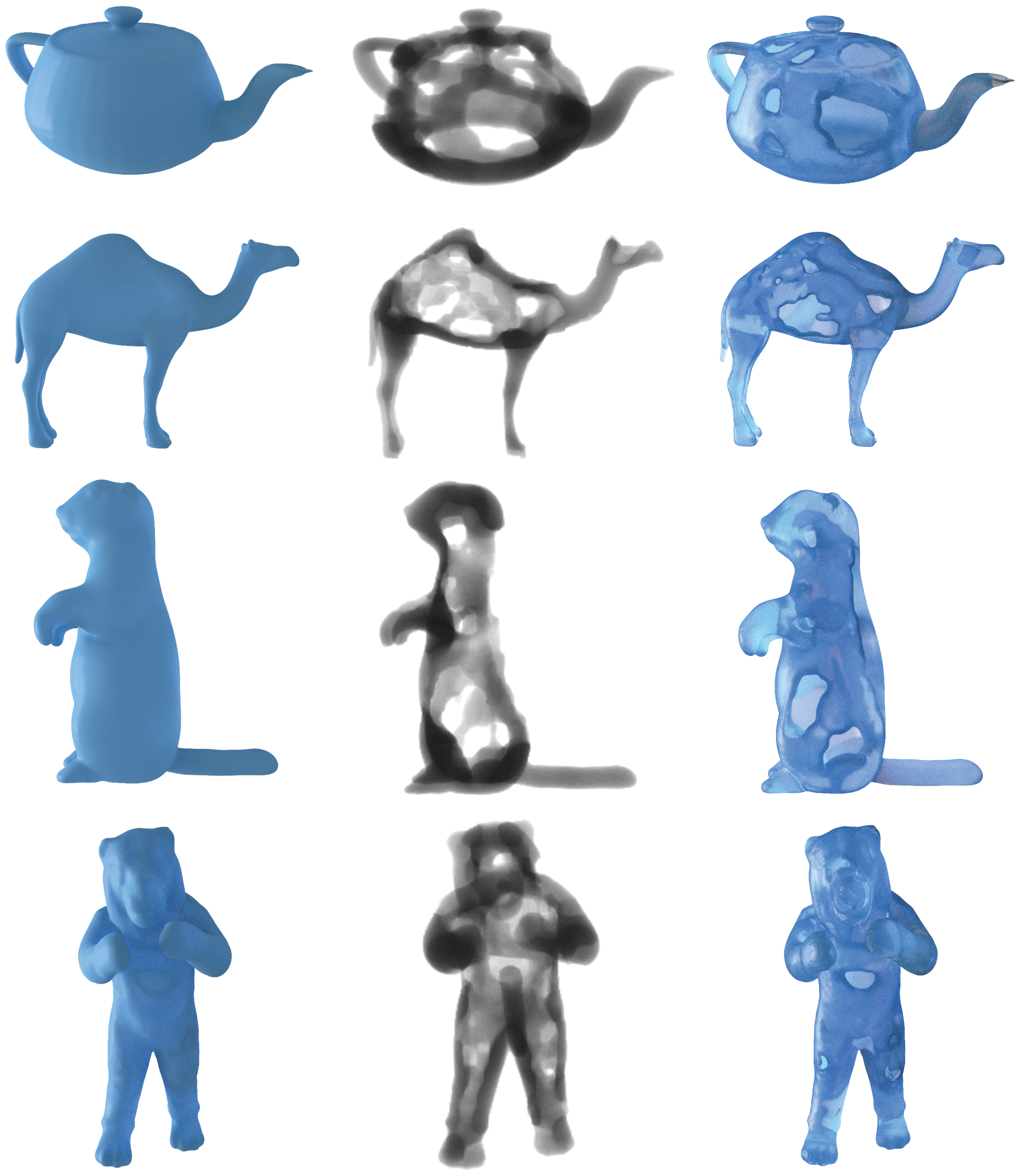


Figure 7.10: Left: The teapot, camel, beaver, and tiger surface meshes. Middle: Final optimized density field. Right: Final meshed results, rendered translucently to show internal structure.

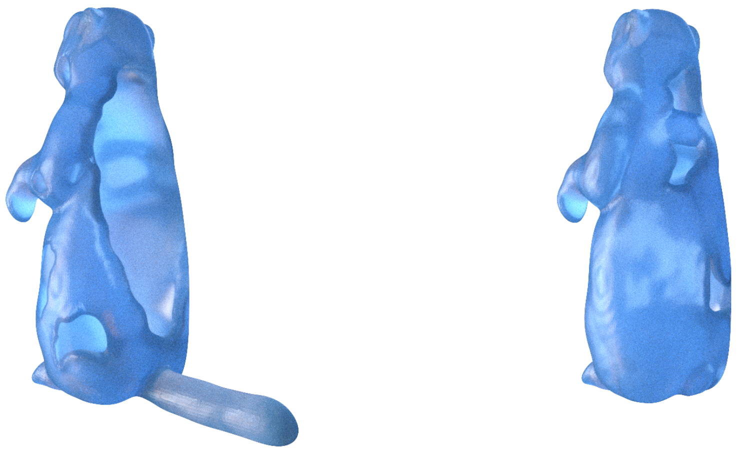


Figure 7.11: Left: Optimized result of a beaver with its tail. Collision with its back becomes very unlikely, so the region is entirely hollowed out. Right: With the tail removed, the back experiences collisions, so the material remains.

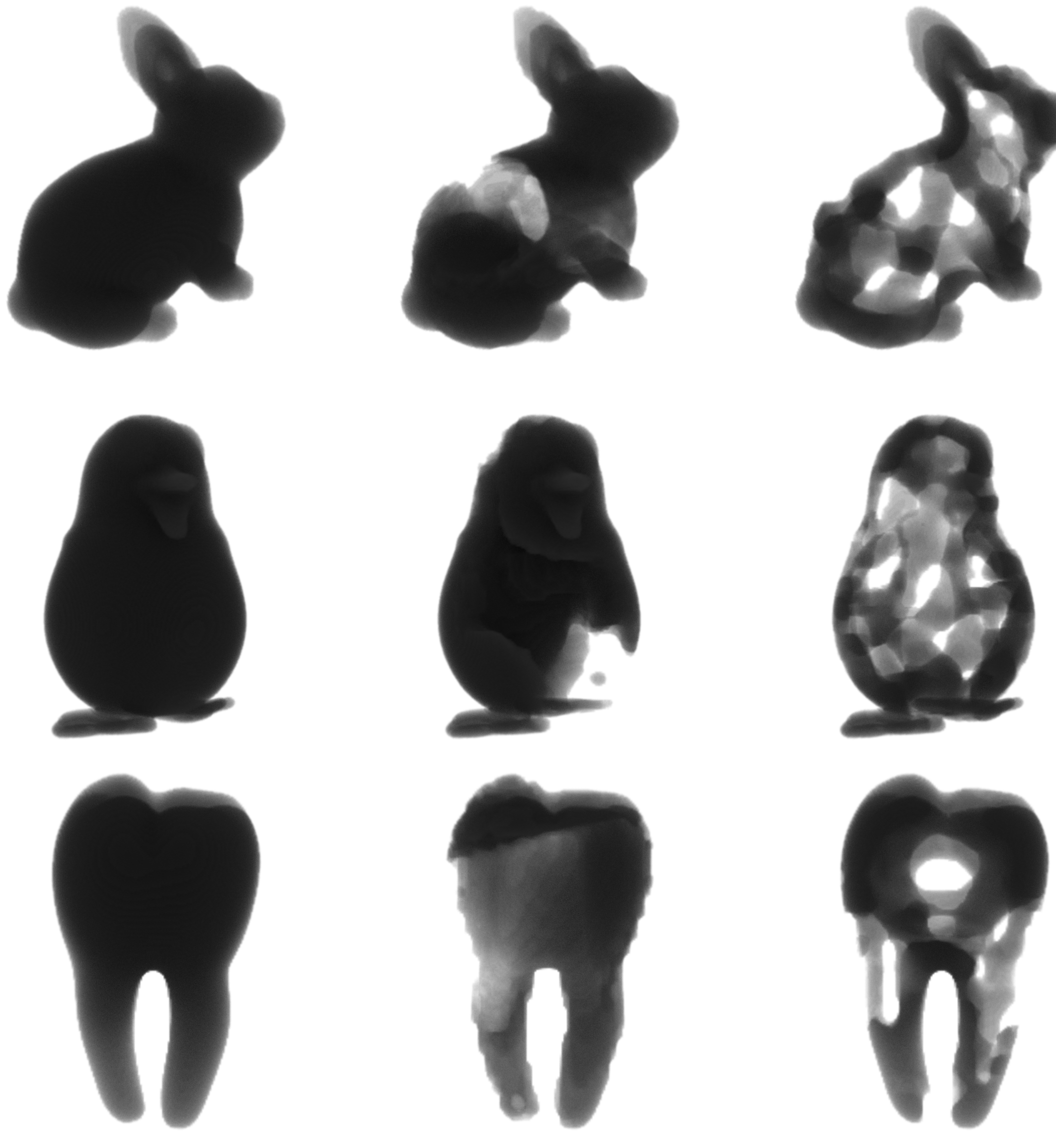


Figure 7.12: The Langlois et al. [27] algorithm vs. ours. Left: Original shape. Middle: Stalled result from Langlois et al. Right: Improved result using ours.

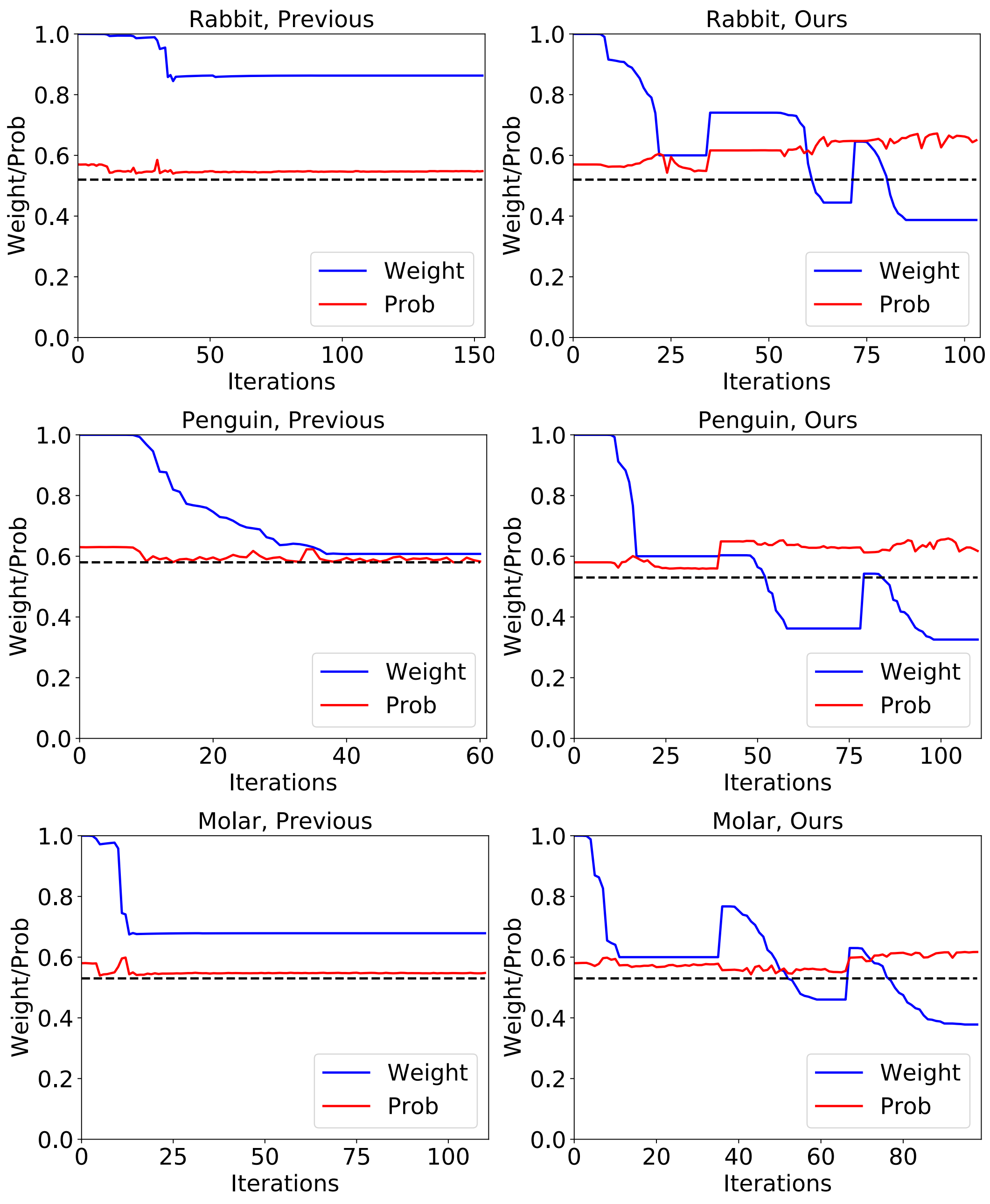


Figure 7.13: Left: The object weights and survival probabilities when optimizing using Langlois et al. [27]. The convergence consistently stalls. Right: The same plots using our method. Top to bottom: The rabbit, penguin, and molar models. The dashed line is the constraint probability. In our method, the curves jump at optimization restarts, but eventually reach lower weights.

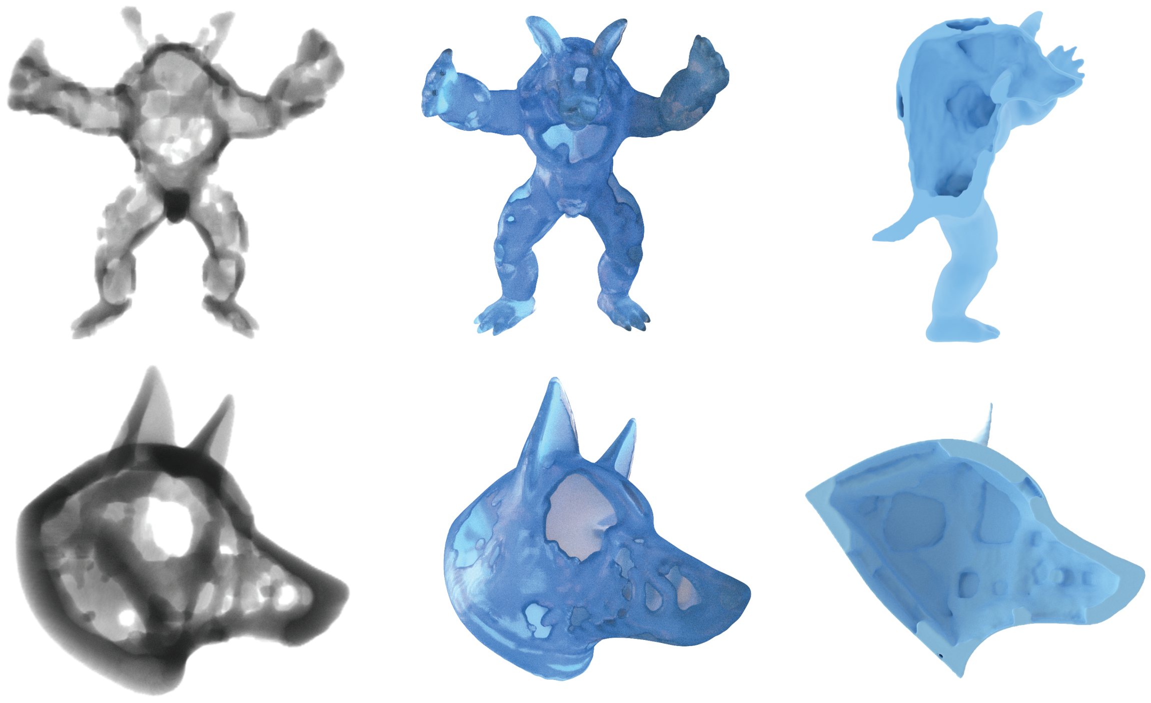


Figure 7.14: Left: Optimized density field. Middle: Post processed final shapes. Right: Cut view of the post processed shapes.

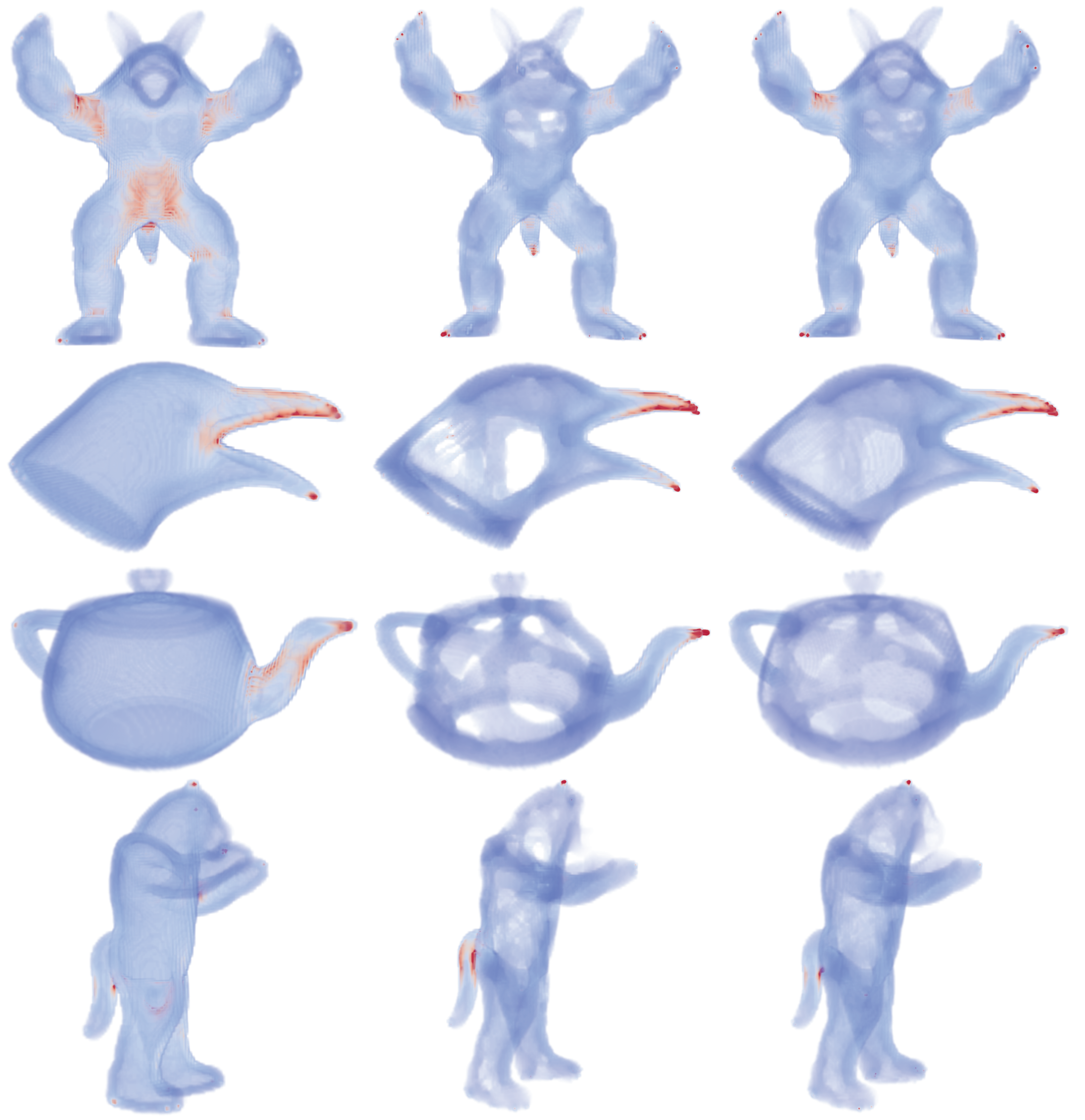


Figure 7.15: Left: The von Mises stress of models with a shell constraint. Middle: Stresses without a shell constraint. Right: Stresses after a sheath is added. The stresses appear in essentially the same regions. Notice the optimization with a shell constraint produces a higher stress in the pelvis of the armadillo.

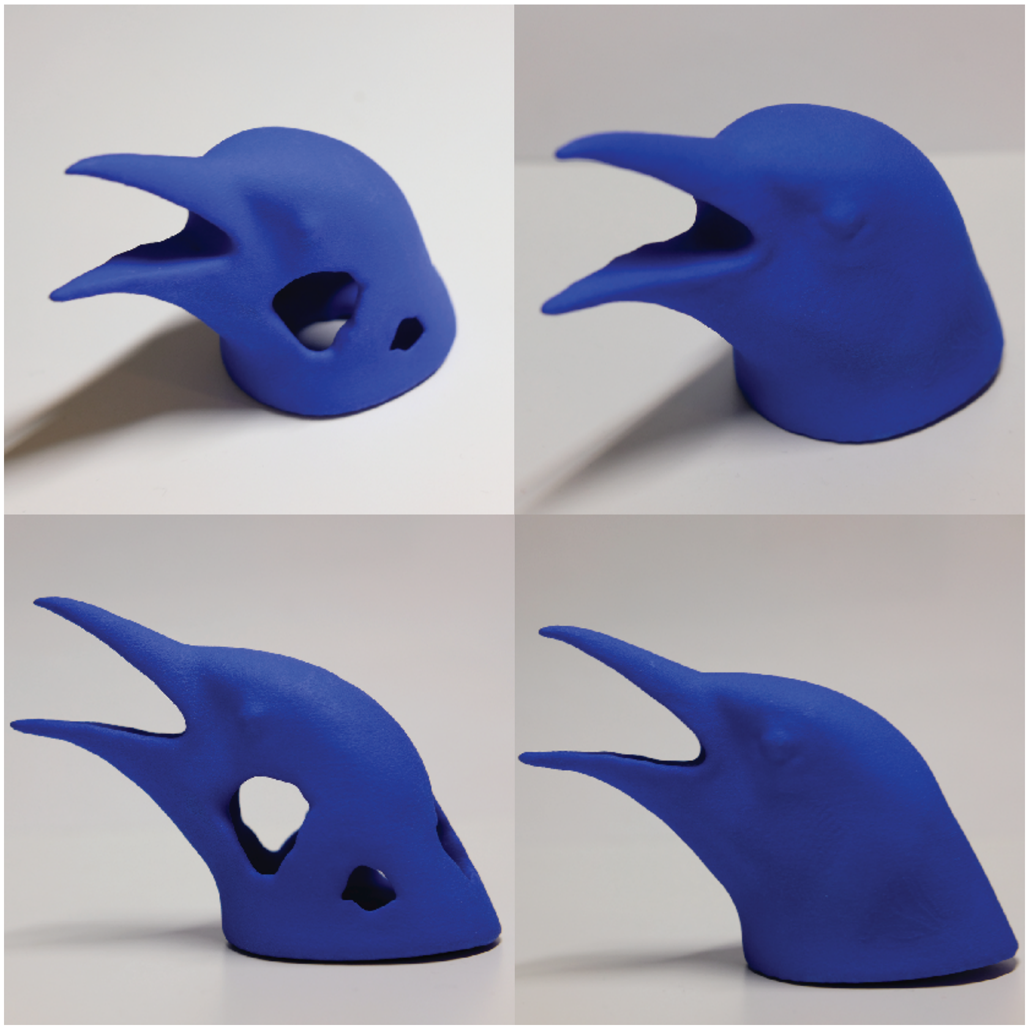


Figure 7.16: Photos of the 3D printed results from the raven in Fig. 7.8.

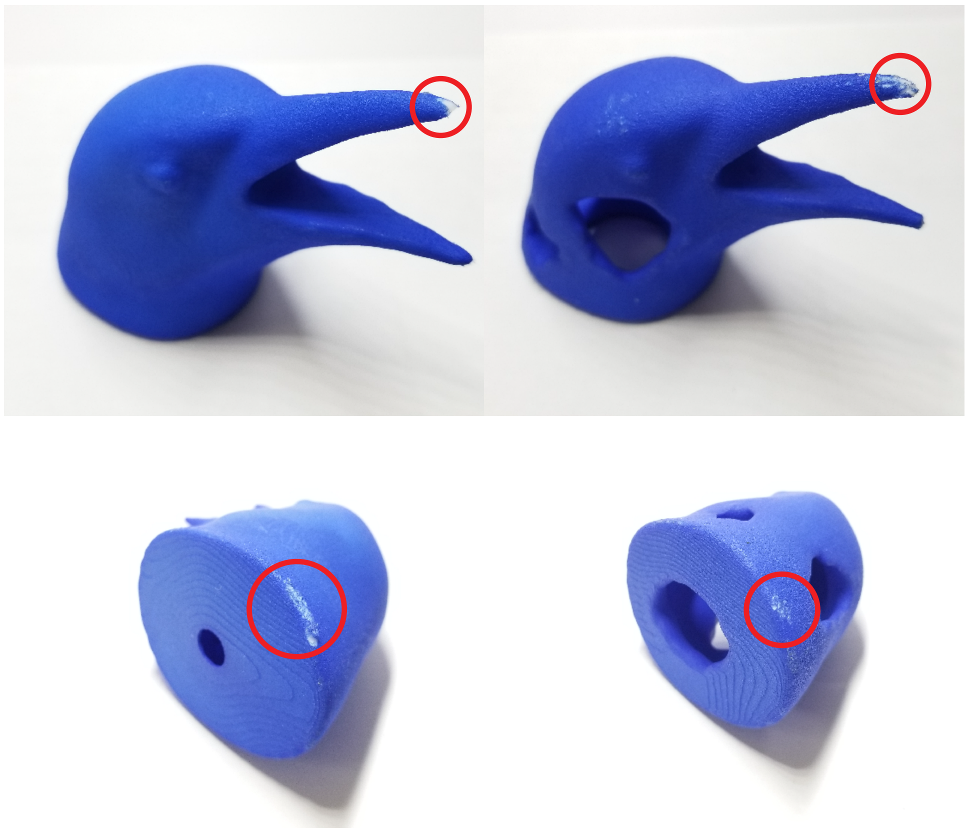


Figure 7.17: Photographs of breakage patterns. Left: Breakages of the un-sheathed object. Right: Breakages of the sheathed object.

Chapter 8

Conclusion

Note: A significant portion of this chapter has previously appeared as [1, 2].

In this dissertation, I have described several methods of enhancing two specific existing reduced-order methods. The new methods enable scalable and stable simulations, which lead to visually interesting simulations that are not possible with previous approaches. The methods presented covers two different scenarios: fluid simulations and structure optimization. This further establishes reduced-order methods are applicable to a wide range of physical simulations.

8.1 Summary of Results

First of all, I have described a version of the Eigenfluids algorithm that removes the memory limitations imposed by basis storage. This is achieved by using analytical Laplacian eigenfunctions and implicitly representing them using fast transformations. The basis functions also generalize to Neumann boundary conditions. I showed the linear system can be solved using common symmetric solvers. I presented the directable dynamics of the fluid by modifying the advection tensor and showed the possibility to compress such tensor effectively. The method is made more practical as we are able to improve the scalability of the original algorithm by over two orders of magnitude.

Next, I described an extension of Eigenfluids to polar and spherical coordinate systems, which allows the method to capture a variety of flows on different geometries. For example: planetary flow, 2D flow inside a circle, and the 3D flow inside a sphere. To achieve this, a new set of analytical basis functions were derived for each different domain. The new basis functions admit fast transformations, allowing efficient velocity reconstruction. While designing the basis functions, I showed they can be made more general by lifting orthogonality requirements. An orthogonalization approach is presented to reintroduce orthogonality into the simulation. The approach allows the advection tensor to be expressed using the original analytical basis functions, which allows its entries to be computed analytically as well.

Finally, I described an asymptotically faster and more robust method for stochastic structural optimization. I reduced the previous quadratic complexity to linear, which results in a two order of magnitude speed-up. This is achieved by leveraging the matrix structures in gradient computations. The method is further stabilized by computing robust gradients with GMMs. The optimization is improved by using a constrained restart method to overcome the local minima. A more faster and robust stochastic structural optimization is achieved, exhibiting more detailed and visually interesting optimized shapes.

8.2 Limitations

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 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., [13])

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 [80, 48], or by discovering compact new structures in the tensor, such as Kronecker product formulations [28].

After extending the basis functions to spherical coordinates, we no longer require the basis functions to be orthogonal. This implies they are no longer eigenfunctions of the Laplacian operator. Therefore, the diffusion term no longer projects to the basis function itself. We approximate the diffusion effect by computing a number to characterize the frequency that is similar to the eigenvalues of the original Laplacian eigenfunctions [1]. We then exponentially decay the coefficients using this number. While this captures the viscosity effect, a better treatment like Galerkin projection can be developed.

In the stochastic structural optimization, we hold the contact points fixed during the optimization, which enables us to use a finite difference method to compute the inertia gradients (i.e. the gradient of the rigid body simulator). However, this means that our method cannot handle topological changes along the surface, so it limited to examples where the external surface is prescribed. As a direction for future work, allowing the gradients to incorporate shape changes would broaden the possible application areas.

8.3 Future Work

First of all, 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 [47] still need to devised for such elements.

We have not yet explored the extension of Eigenfluids to include liquid surfaces. Although [52] 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 [100] 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.

Our simulations are fast and memory-efficient because the simulation domain is limited to simple shapes like boxes. 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 [14, 15]. 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.

By extending the basis functions to polar and spherical coordinates, we can simulate the fluid over more domains like in a circle or a sphere. Because Eigenfluids do not exhibit numerical viscosity, they are good candidates to enhance small scale vortex details. Also, Eigenfluids in spherical coordinates closely resembles a particle. Therefore, one direct extension of this is to use Eigenfluids to upsample existing simulations like in the vortex particle method [35, 101]. New methods of incorporating such Eigenfluids particles into the base flow need to be devised.

The introduction of the orthogonalization approach no longer requires the basis functions to be orthogonal when they are being designed. The basis functions can be made quite general, therefore one promising direction for future research is to extend Eigenfluids to more domains, for example, ellipsoids or even general curvilinear coordinates as shown in [21]. As I showed in appendix A.3, the basis functions are likely not unique. Therefore exploring basis functions that satisfy other constraints, for example, a sparse advection tensor is promising as well. In contrast to the rectangular domain, increasing the wavenumber of the basis functions in spherical coordinates do not refine the spatial domain uniformly. Therefore, designing basis functions that uniformly refine spatial domain is desirable.

In the stochastic structural optimization, while the constrained restart method works in practice, it remains to be seen if the reinforcement structure can be identified and constrained within a single optimization pass to improve convergence. Even with our improvements, computing the probability gradient can still be a bottleneck due to its dense linear algebra operations, which limits the resolution of our method. One way to reduce this cost could be to use a sparse grid [66].

Appendix A

Laplacian Eigenfunctions in Polar and Spherical Coordinates

To find a vector basis functions in a polar domain, one may start with the scalar Laplacian eigenfunctions in polar coordinates, similar to the Cartesian case. As shown in [102], scalar Laplacian eigenfunctions in a disk domain ($[0,1] \times [0,2\pi]$) is the following:

$$\begin{aligned} f_{n,k}(r,\theta) &= J_n(\alpha_{n,k} r)\cos(n\theta) \\ f_{n,k}(r,\theta) &= J_n(\alpha_{n,k} r)\sin(n\theta), \quad n \neq 0, \end{aligned} \tag{A.1}$$

where $J_n(\alpha_{n,k} r)$ is the Bessel function of the first kind. The above function is orthogonal. Transformations over Bessel function of the first kind (the Hankel transform) can be computed efficiently in $O(N\log(N))$ using FFT [103]. But it remains to be seen if divergence-free vector basis functions can be constructed using scalar eigenfunctions. In the following sections, I will attempt to construct a divergence-free vector field using scalar eigenfunctions in A.1.

There are two directions toward this goal: The first is to decompose velocity into a radial $\mathbf{u}_r$ and a tangent component $\mathbf{u}_\theta$. The second is to decompose it into Cartesian components: $\mathbf{u}_x$ and $\mathbf{u}_y$.

A.1 Decomposing Velocity Basis as $\mathbf{u}_r$ and $\mathbf{u}_\theta$

In this case, we have: $\mathbf{u} = \mathbf{u}_r\mathbf{e}_r + \mathbf{u}_\theta\mathbf{e}_\theta$, where $\mathbf{e}_r, \mathbf{e}_\theta$ are unit vectors along $\mathbf{r}$ and its tangent direction. Assume two velocity components are separable: $\mathbf{u}_r = A(r)B(\theta)$, $\mathbf{u}_\theta = C(r)D(\theta)$. The divergence-free condition is:

$$\begin{aligned} \mathbf{u}_r + r\frac{\partial \mathbf{u}_r}{\partial r} + \frac{\partial \mathbf{u}_\theta}{\partial \theta} = 0 \\ AB + rB\frac{\partial A}{\partial r} + C\frac{\partial D}{\partial \theta} = 0. \end{aligned} \tag{A.2}$$

We can choose B, D in a way that $\frac{\partial D}{\partial \theta} = B$, for example, $B = \cos(n\theta), D = \frac{1}{n}\sin(n\theta)$. Then we have:

$$A + r\frac{\partial A}{\partial r} + C = 0. \tag{A.3}$$

Both $\mathbf{u}_r, \mathbf{u}_\theta$ have to choose from equation A.1 in order for $\mathbf{u}$ to be an eigenfunction of the Laplacian operator. This requires $A(r)$ and $C(r)$ to be Bessel functions. However, equation A.3 indicates that $A+rA'$ has to be a Bessel function as well. This is impossible given that A is a Bessel function, this can be proven by contradiction.

First, A is a Bessel function:

$$r^2A'' + rA' + (r^2 - n^2)A = 0. \tag{A.4}$$

Assuming $A+rA'$ is another Bessel function, then the following equation must be satisfied:

$$r^3A''' + 4r^2A'' + 2rA' + (r^2 - m^2)(A + rA') = 0. \tag{A.5}$$

Take the derivative of equation A.4 and add with the above, we have:

$$r^3A''' + 4r^2A'' + 2rA' + (3r^2 - n^2)A + r(r^2 - n^2)A' = 0. \tag{A.6}$$

Subtract above equation from A.5, we have:

$$(-2r^2 + n^2 - m^2)A + r(n^2 - m^2)A' = 0 \tag{A.7}$$

This is a different from the equation A.4. So the solution is not a Bessel function. This contradicts with the assumption that A is a Bessel function. Therefore it is impossible to satisfy equation A.3 if both A and C are Bessel functions. So it is impossible to construct a divergence free basis using Laplacian eigenfunctions in a disk domain, in the form: $\mathbf{u}_r = A(r)B(\theta)$, $\mathbf{u}_\theta = C(r)D(\theta)$.

A.2 Decomposing Velocity Basis as $\mathbf{u}_x$ and $\mathbf{u}_y$

Another possible direction is decomposing $\mathbf{u}$ in Cartesian coordinates: $\mathbf{u} = \mathbf{u}_x\mathbf{e}_x + \mathbf{u}_y\mathbf{e}_y$. Both $\mathbf{u}_x, \mathbf{u}_y$ can be composed from equation A.1. First assume the basis is the following:

$$\begin{aligned}\mathbf{u}_x &= aJ_n(\alpha_n r)\sin(n\theta)\\ \mathbf{u}_y &= bJ_n(\alpha_n r)\cos(n\theta).\end{aligned} \tag{A.8}$$

The divergence of above vector field is :

$$\begin{aligned}\nabla\cdot\mathbf{u} &= -\frac{1}{r}\sin(\theta)\frac{\partial\mathbf{u}_x}{\partial\theta}+\cos(\theta)\frac{\partial\mathbf{u}_x}{\partial r}+\frac{1}{r}\cos(\theta)\frac{\partial\mathbf{u}_y}{\partial\theta}+\sin(\theta)\frac{\partial\mathbf{u}_y}{\partial r}\\ &= -\frac{1}{r}\sin(\theta)naJ_n(\alpha_n r)\cos(n\theta)+a\alpha_n\cos(\theta)\sin(n\theta)J_n(\alpha_n r)'\\ &-\frac{1}{r}\cos(\theta)nbJ_n(\alpha_n r)\sin(n\theta)+b\alpha_n\sin(\theta)\cos(n\theta)J_n(\alpha_n r)'\\ &= -\frac{n}{r}[a\sin(\theta)\cos(n\theta)+b\cos(\theta)\sin(n\theta)]J_n(\alpha_n r)\\ &+[a\cos(\theta)\sin(n\theta)+b\sin(\theta)\cos(n\theta)]\alpha_n J_n(\alpha_n r)',\end{aligned} \tag{A.9}$$

which can be simplified into the following:

$$\begin{aligned}&\nabla\cdot\mathbf{u} =\\ &= \frac{\alpha_n}{2}(a-b)J_{n-1}(\alpha_n r)\sin((n-1)\theta)-\frac{\alpha_n}{2}(a+b)J_{n+1}(\alpha_n r))\sin((n+1)\theta),\end{aligned} \tag{A.10}$$

because the Bessel function satisfies the following properties:

$$\begin{aligned}J_n(r) &= \frac{r}{2n}(J_{n-1}(r)+J_{n+1}(r))\\ J_n(r)' &= \frac{1}{2}(J_{n-1}(r)-J_{n+1}(r)).\end{aligned} \tag{A.11}$$

Therefore, both $a-b=0$ and $a+b=0$ must be satisfied. This is not possible unless $a=0, b=0$.

Mixing different frequencies of Laplacian eigenfunctions can lead to a divergence-free vector field. For example, if $a = -b$, we have the non-zero divergence be: $a\alpha_n J_{n-1}(\alpha_n r)\sin((n-1)\theta)$. We could add another band of Laplacian eigenfunctions that cancel out this term. For example, the following basis is divergence-free:

$$\begin{aligned} \mathbf{u}_x &= J_n(\alpha_n r)\sin(n\theta) + J_{n-2}(\alpha_n r)\sin((n-2)\theta) \\ \mathbf{u}_y &= -J_n(\alpha_n r)\cos(n\theta) + J_{n-2}(\alpha_n r)\cos((n-2)\theta). \end{aligned} \tag{A.12}$$

The above basis mixes two frequencies unless $n = 1$, where it becomes:

$$\begin{aligned} \mathbf{u}_x &= J_1(\alpha_1 r)\sin(\theta) \\ \mathbf{u}_y &= -J_1(\alpha_1 r)\cos(\theta). \end{aligned} \tag{A.13}$$

The above analysis also holds if cosine modes are used for $\mathbf{u}_x$ and sine modes are used for $\mathbf{u}_y$, or combinations of cosine and sine modes are used for θ. Mixing two bands of Laplacian eigenfunctions may lead to non-orthogonal basis functions. Therefore, it is unclear how two construct a divergence-free basis using only a single band of Laplacian eigenfunction, unless $n = 1$.

Therefore, it seems difficult to construct a divergence-free vector field using scalar Laplacian eigenfunctions in polar coordinates.

A.3 Laplacian Eigenfunctions on the Surface of a Sphere

It can be shown that the vector spherical harmonics form orthogonal bases on the surface of a sphere. Define the orbital angular momentum operator as $\mathbf{L} = -i\mathbf{r} \times \nabla$. One type of vector spherical harmonics are:

$$\begin{aligned}\mathbf{X} &= \frac{1}{\sqrt{l(l+1)}}\mathbf{L}Y_l^m \\ \mathbf{X}_\theta &= \frac{-mY_l^m}{\sqrt{(l(l+1))}\sin(\theta)} \\ \mathbf{X}_\phi &= \frac{-i}{\sqrt{(l(l+1))}}\frac{\partial Y_l^m}{\partial\theta}\end{aligned} \tag{A.14}$$

The vector field $\mathbf{X}$ is divergence free because:

$$\begin{aligned}&\nabla\cdot\mathbf{X} = \frac{1}{r\sin(\theta)}[\frac{\partial}{\partial\theta}(\sin(\theta)\mathbf{X}_\theta) + \frac{\partial\mathbf{X}_\phi}{\partial\phi}] \\ &\frac{1}{\sqrt{(l(l+1))}}\frac{1}{r\sin(\theta)}[-m\cot(\theta)Y_l^m - m(\frac{\partial Y_l^m}{\partial\theta} - \cot(\theta)Y_l^m) - i\frac{\partial}{\partial\phi}(\frac{\partial Y_l^m}{\partial\theta})] = \\ &\frac{1}{\sqrt{(l(l+1))}}\frac{1}{r\sin(\theta)}[-m\frac{\partial Y_l^m}{\partial\theta} + m\frac{\partial Y_l^m}{\partial\theta}] = 0\end{aligned} \tag{A.15}$$

The vector field $\mathbf{X}$ is also shown to be orthogonal in [104].

Bibliography

[1] Q. Cui, P. Sen, and T. Kim, *Scalable laplacian eigenfluids*, *ACM Transactions on Graphics (TOG)* **37** (2018), no. 4 1–12.

[2] Q. Cui, T. Langlois, P. Sen, and T. Kim, *Fast and Robust Stochastic Structural Optimization*, *Computer Graphics Forum* (2020).

[3] L. F. Richardson, *Weather prediction by numerical process.* Cambridge university press, 2007.

[4] J. C. Hunt, *Lewis fry richardson and his contributions to mathematics, meteorology, and models of conflict*, *Annual Review of Fluid Mechanics* **30** (1998), no. 1 xiii–xxxvi.

[5] W. T. Reeves, *Particle systems—a technique for modeling a class of fuzzy objects*, *ACM Transactions On Graphics (TOG)* **2** (1983), no. 2 91–108.

[6] J. Stam and E. Fiume, *Turbulent wind fields for gaseous phenomena*, in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pp. 369–376, 1993.

[7] L. Yaeger, C. Upson, and R. Myers, *Combining physical and visual simulation—creation of the planet jupiter for the film "2010"*, *Acm Siggraph Computer Graphics* **20** (1986), no. 4 85–93.

[8] M. Kass and G. Miller, *Rapid, stable fluid dynamics for computer graphics*, in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 49–57, 1990.

[9] N. Foster and D. Metaxas, *Modeling the motion of a hot, turbulent gas*, in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 181–188, 1997.

[10] J. Stam, *Stable fluids*, in *Proceedings of SIGGRAPH*, pp. 121–128, 1999.

[11] R. Fedkiw, J. Stam, and H. W. Jensen, *Visual simulation of smoke*, in *Proceedings of SIGGRAPH*, pp. 15–22, 2001.

[12] X. Zhang, R. Bridson, and C. Greif, *Restoring the missing vorticity in advection-projection fluid solvers*, *ACM Trans. Graph.* **34** (July, 2015) 52:1–52:8.

[13] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun, *Energy-preserving integrators for fluid animation*, *ACM Trans. Graph.* **28** (2009), no. 3 38:1–38:8.

[14] T. De Witt, C. Lessig, and E. Fiume, *Fluid simulation using laplacian eigenfunctions*, *ACM Trans. Graph.* **31** (2012), no. 1 10:1–10:11.

[15] B. Liu, G. Mason, J. Hodgson, Y. Tong, and M. Desbrun, *Model-reduced variational fluid simulation*, *ACM Trans. Graph.* **34** (2015), no. 6 244:1–244:12.

[16] A. Angelidis. Personal Communication, 2017.

[17] D. Gottlieb and S. A. Orszag, *Numerical analysis of spectral methods: theory and applications.* SIAM, 1977.

[18] L. N. Trefethen, *Spectral methods in MATLAB.* SIAM, 2000.

[19] J. P. Boyd, *Chebyshev and Fourier spectral methods.* Dover Publications, 2001.

[20] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang, *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics.* Springer, 2007.

[21] J. Stam, *Flows on surfaces of arbitrary topology*, *ACM Transactions On Graphics (TOG)* **22** (2003), no. 3 724–731.

[22] R. Angst, N. Thuerey, M. Botsch, and M. Gross, *Robust and efficient wave simulations on deforming meshes*, in *Computer Graphics Forum*, vol. 27, pp. 1895–1900, Wiley Online Library, 2008.

[23] D. J. Hill and R. D. Henderson, *Efficient fluid simulation on the surface of a sphere*, *ACM Transactions on Graphics (TOG)* **35** (2016), no. 2 1–9.

[24] B. Yang, W. Corse, J. Lu, J. Wolper, and C.-F. Jiang, *Real-time fluid simulation on the surface of a sphere*, *Proceedings of the ACM on Computer Graphics and Interactive Techniques* **2** (2019), no. 1 1–17.

[25] M. Livesu, S. Ellero, J. Martínez, S. Lefebvre, and M. Attene, *From 3d models to 3d prints: an overview of the processing pipeline*, in *Computer Graphics Forum*, vol. 36, pp. 537–564, Wiley Online Library, 2017.

[26] W. Matusik and A. Schulz, *Computational fabrication*, in *ACM SIGGRAPH Courses*, pp. 7:1–7:305, 2019.

[27] T. Langlois, A. Shamir, D. Dror, W. Matusik, and D. I. W. Levin, *Stochastic structural analysis for context-aware design and fabrication*, *ACM Trans. Graph.* **35** (Nov., 2016) 226:1–226:13.

[28] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU Press, 2012.

[29] J. C. Strikwerda, *Finite difference schemes and partial differential equations*, vol. 88. Siam, 2004.

[30] F. H. Harlow and J. E. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, *The physics of fluids* **8** (1965), no. 12 2182–2189.

[31] F. Losasso, F. Gibou, and R. Fedkiw, *Simulating water and smoke with an octree data structure*, *ACM Trans. Graph.* **23** (Aug., 2004) 457–462.

[32] A. McAdams, E. Sifakis, and J. Teran, *A parallel multigrid poisson solver for fluids simulation on large grids*, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 65–74, 2010.

[33] F. Ferstl, R. Westermann, and C. Dick, *Large-scale liquid simulation on adaptive hexahedral grids*, *IEEE Transactions on Visualization and Computer Graphics* **20** (2014), no. 10 1405–1417.

[34] R. Setaluri, M. Aanjaneya, S. Bauer, and E. Sifakis, *Spgrid: A sparse paged grid structure applied to adaptive smoke simulation*, *ACM Trans. Graph.* **33** (Nov., 2014) 205:1–205:12.

[35] A. Selle, N. Rasmussen, and R. Fedkiw, *A vortex particle method for smoke, water and explosions*, *ACM Trans. Graph.* **24** (July, 2005) 910–914.

[36] T. Kim, N. Thürey, D. James, and M. Gross, *Wavelet turbulence for fluid simulation*, *ACM Trans. Graph.* **27** (Aug., 2008) 50:1–50:6.

[37] H. Schechter and R. Bridson, *Evolving sub-grid turbulence for smoke animation*, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–7, 2008.

[38] R. Narain, J. Sewall, M. Carlson, and M. C. Lin, *Fast animation of turbulence using energy transport and procedural synthesis*, *ACM Trans. Graph.* **27** (Dec., 2008).

[39] S. Weissmann and U. Pinkall, *Filament-based smoke with vortex shedding and variational reconnection*, *ACM Trans. Graph.* **29** (July, 2010) 115:1–115:12.

[40] Y. Zhu and R. Bridson, *Animating sand as a fluid*, *ACM Trans. Graph.* **24** (July, 2005) 965–972.

[41] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin, *The affine particle-in-cell method*, *ACM Trans. Graph.* **34** (July, 2015) 51:1–51:10.

[42] C. Fu, Q. Guo, T. Gast, C. Jiang, and J. Teran, *A polynomial particle-in-cell method*, *ACM Trans. Graph.* **36** (Nov., 2017) 222:1–222:12.

[43] A. Chern, F. Knöppel, U. Pinkall, P. Schröder, and S. Weissmann, *Schrödinger's smoke*, *ACM Trans. Graph.* **35** (July, 2016) 77:1–77:13.

[44] A. Pentland and J. Williams, *Good vibrations: Modal dynamics for graphics and animation*, in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pp. 215–222, 1989.

[45] A. Treuille, A. Lewis, and Z. Popović, *Model reduction for real-time fluids*, *ACM Trans. Graph.* **25** (July, 2006) 826–834.

[46] T. Kim and J. Delaney, *Subspace fluid re-simulation*, *ACM Trans. Graph.* **32** (2013), no. 4 62:1–62:9.

[47] M. Wicke, M. Stanton, and A. Treuille, *Modular bases for fluid dynamics*, *ACM Trans. Graph.* **28** (July, 2009) 39:1–39:8.

[48] A. D. Jones, P. Sen, and T. Kim, *Compressing fluid subspaces*, in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 77–84, 2016.

[49] M. Stanton, Y. Sheng, M. Wicke, F. Perazzi, A. Yuen, S. Narasimhan, and A. Treuille, *Non-polynomial galerkin projection on deforming meshes*, *ACM Trans. Graph.* **32** (July, 2013) 86:1–86:14.

[50] M. Gupta and S. G. Narasimhan, *Legendre fluids: A unified framework for analytic reduced space modeling and rendering of participating media*, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 17–25, 2007.

[51] J. Stam, *A simple fluid solver based on the fft*, *J. Graph. Tools* **6** (Sept., 2002) 43–52.

[52] B. Long and E. Reinhard, *Real-time fluid simulation using discrete sine/cosine transforms*, in *Symposium on Interactive 3D Graphics and Games*, pp. 99–106, 2009.

[53] R. D. Henderson, *Scalable fluid simulation in linear time on shared memory multiprocessors*, in *Proceedings of the Digital Production Symposium*, DigiPro '12, pp. 43–52, 2012.

[54] C. Lanczos, *Trigonometric interpolation of empirical and analytical functions*, *J. Math Phys.* **17** (1938), no. 1-4 123–199.

[55] S. A. Orszag, *Numerical methods for the simulation of turbulence*, *The Physics of Fluids* **12** (1969), no. 12 II–250.

[56] S. A. Orszag, *Accurate solution of the orr–sommerfeld stability equation*, *Journal of Fluid Mechanics* **50** (1971), no. 4 689–703.

[57] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. Zang, *Spectral methods in fluid dynamics.* Springer, 1988.

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

[59] N. Thürey, C. Wojtan, M. Gross, and G. Turk, *A multiscale approach to mesh-based surface tension flows*, *ACM Transactions on Graphics (TOG)* **29** (2010), no. 4 1–10.

[60] T. Kim, J. Tessendorf, and N. Thuerey, *Closest point turbulence for liquid surfaces*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 2 1–13.

[61] M. P. Bendsøe, *Optimal shape design as a material distribution problem*, *Structural optimization* **1** (1989), no. 4 193–202.

[62] M. P. Bendsøe and O. Sigmund, *Optimization of structural topology, shape, and material*, vol. 414. Springer, 1995.

[63] O. Sigmund, *A 99 line topology optimization code written in matlab*, *Structural and multidisciplinary optimization* **21** (2001), no. 2 120–127.

[64] K. Svanberg, *A class of globally convergent optimization methods based on conservative convex separable approximations*, *SIAM Journal on Optimization* **12** (2002), no. 2 555–573.

[65] W. Wang, T. Y. Wang, Z. Yang, L. Liu, X. Tong, W. Tong, J. Deng, F. Chen, and X. Liu, *Cost-effective printing of 3d objects with skin-frame structures*, *ACM Trans. Graph.* **32** (Nov., 2013) 177:1–177:10.

[66] H. Liu, Y. Hu, B. Zhu, W. Matusik, and E. Sifakis, *Narrow-band topology optimization on a sparsely populated grid*, *ACM Trans. Graph.* **37** (Dec., 2018) 251:1–251:14.

[67] L. Lu, A. Sharf, H. Zhao, Y. Wei, Q. Fan, X. Chen, Y. Savoye, C. Tu, D. Cohen-Or, and B. Chen, *Build-to-last: Strength to weight 3d printed objects*, *ACM Trans. Graph.* **33** (July, 2014) 97:1–97:10.

[68] E. Lee, K. A. James, and J. R. Martins, *Stress-constrained topology optimization with design-dependent loading*, *Structural and Multidisciplinary Optimization* **46** (2012), no. 5 647–661.

[69] E. Ulu, J. McCann, and L. B. Kara, *Structural design using laplacian shells*, in *Computer Graphics Forum*, vol. 38, pp. 85–98, Wiley Online Library, 2019.

[70] G. Stefanou, *The stochastic finite element method: Past, present and future*, *Computer Methods in Applied Mechanics and Engineering* **198** (2009), no. 9 1031 – 1051.

[71] W. K. Liu, T. Belytschko, and A. Mani, *Probabilistic finite elements for nonlinear structural dynamics*, *Computer Methods in Applied Mechanics and Engineering* **56** (1986), no. 1 61 – 81.

[72] R. G. Ghanem and P. D. Spanos, *Stochastic Finite Elements: A Spectral Approach*. Springer-Verlag, 1991.

[73] M. Papadrakakis and V. Papadopoulos, *Robust and efficient methods for stochastic finite element analysis using monte carlo simulation*, *Computer Methods in Applied Mechanics and Engineering* **134** (1996), no. 3 325 – 340.

[74] Q. Zhou, J. Panetta, and D. Zorin, *Worst-case structural analysis*, *ACM Trans. Graph.* **32** (July, 2013) 137:1–137:12.

[75] E. Ulu, J. Mccann, and L. B. Kara, *Lightweight structure design under force location uncertainty*, *ACM Trans. Graph.* **36** (July, 2017) 158:1–158:13.

[76] J. Panetta, A. Rahimian, and D. Zorin, *Worst-case stress relief for microstructures*, *ACM Trans. Graph.* **36** (July, 2017) 122:1–122:16.

[77] C. Schumacher, J. Zehnder, and M. Bächer, *Set-in-stone: Worst-case optimization of structures weak in tension*, *ACM Trans. Graph.* **37** (Dec., 2018) 252:1–252:13.

[78] D. K. Cheng *et. al.*, *Field and wave electromagnetics*. Pearson Education India, 1989.

[79] S. S. An, T. Kim, and D. L. James, *Optimizing cubature for efficient integration of subspace deformations*, *ACM Trans. Graph.* **27** (Dec., 2008) 165:1–165:10.

[80] M. Hasan, E. Velazquez-Armendariz, F. Pellacini, and K. Bala, *Tensor clustering for rendering many-light animations*, *Computer Graphics Forum* **27** (2008), no. 4 1105–1114.

[81] U. Frisch, *Turbulence: The Legacy of AN Kolmogorov*. Cambridge University Press, 1995.

[82] R. Bridson, *Fluid simulation for computer graphics*. CRC Press, 2015.

[83] D. Baraff and A. Witkin, *Large steps in cloth simulation*, in *Proceedings of SIGGRAPH*, pp. 43–54, 1998.

[84] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[85] S. Davidovits and N. J. Fisch, *Sudden viscous dissipation of compressing turbulence*, *Physical Review Letters* **116** (2016), no. 10.

[86] G. M. Vasil, K. J. Burns, D. Lecoanet, S. Olver, B. P. Brown, and J. S. Oishi, *Tensor calculus in polar coordinates using jacobi polynomials*, *Journal of Computational Physics* **325** (2016) 53 – 73.

[87] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. Zang, *Spectral methods in fluid dynamics*. Springer, 1988.

[88] G. Guennebaud, B. Jacob, *et. al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[89] M. Frigo and S. G. Johnson, *The design and implementation of FFTW3*, *Proceedings of the IEEE* **93** (2005), no. 2 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[90] T. Kim, "Zephyr." http://www.tkim.graphics/RESIM/source.html, 2013.

[91] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac, *An unconditionally stable maccormack method*, *Journal of Scientific Computing* **35** (2008), no. 2 350–371.

[92] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[93] H. Xu, F. Sin, Y. Zhu, and J. Barbič, *Nonlinear material design using principal stretches*, *ACM Trans. Graph.* **34** (July, 2015) 75:1–75:11.

[94] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[95] R. C. Bernardi, M. C. Melo, and K. Schulten, *Enhanced sampling techniques in molecular dynamics simulations of biological systems*, *Biochimica et Biophysica Acta (BBA)-General Subjects* **1850** (2015), no. 5 872–877.

[96] E. Coumans *et. al.*, *Bullet physics library*, *Open source: bulletphysics. org* **15** (2013), no. 49 5.

[97] C. Schumacher, B. Bickel, J. Rys, S. Marschner, C. Daraio, and M. Gross, *Microstructures to control elasticity in 3d printing*, *ACM Transactions on Graphics (TOG)* **34** (2015), no. 4 1–13.

[98] C. Sanderson and R. Curtin, *Armadillo: a template-based c++ library for linear algebra*, *Journal of Open Source Software* **1** (2016), no. 2 26.

[99] W. E. Lorensen and H. E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, in *ACM SIGGRAPH computer graphics*, vol. 21, pp. 163–169, ACM, 1987.

[100] N. Heo and H.-S. Ko, *Detail-preserving fully-eulerian interface tracking framework*, *ACM Trans. Graph.* **29** (Dec., 2010) 176:1–176:8.

[101] T. Pfaff, N. Thuerey, J. Cohen, S. Tariq, and M. Gross, *Scalable fluid simulation using anisotropic turbulence particles*, in *ACM SIGGRAPH Asia 2010 papers*, pp. 1–8. 2010.

[102] D. S. Grebenkov and B.-T. Nguyen, *Geometrical structure of laplacian eigenfunctions*, *siam REVIEW* **55** (2013), no. 4 601–667.

[103] J. D. Talman, *Numerical fourier and bessel transforms in logarithmic variables*, *Journal of computational physics* **29** (1978), no. 1 35–48.

[104] B. Carrascal, G. Estevez, P. Lee, and V. Lorenzo, *Vector spherical harmonics and their application to classical electrodynamics*, *European Journal of Physics* **12** (1991), no. 4 184.