Université de Montréal

Iterative Solvers for Physics-based Simulations and Displays

par

Olivier Mercier

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

9 février 2018

SOMMAIRE

La génération d'images et de simulations réalistes requiert des modèles complexes pour capturer tous les détails d'un phénomène physique. Les équations mathématiques qui composent ces modèles sont compliquées et ne peuvent pas être résolues analytiquement. Des procédures numériques doivent donc être employées pour obtenir des solutions approximatives à ces modèles. Ces procédures sont souvent des algorithmes itératifs, qui calculent une suite convergente vers la solution désirée à partir d'un essai initial. Ces méthodes sont une façon pratique et efficace de calculer des solutions à des systèmes complexes, et sont au coeur de la plupart des méthodes de simulation modernes.

Dans cette thèse par article, nous présentons trois projets où les algorithmes itératifs jouent un rôle majeur dans une méthode de simulation ou de rendu. Premièrement, nous présentons une méthode pour améliorer la qualité visuelle de simulations fluides. En créant une surface de haute résolution autour d'une simulation existante, stabilisée par une méthode itérative, nous ajoutons des détails additionels à la simulation. Deuxièmement, nous décrivons une méthode de simulation fluide basée sur la réduction de modèle. En construisant une nouvelle base de champ de vecteurs pour représenter la vélocité d'un fluide, nous obtenons une méthode spécifiquement adaptée pour améliorer les composantes itératives de la simulation. Finalement, nous présentons un algorithme pour générer des images de haute qualité sur des écrans multicouches dans un contexte de réalité virtuelle. Présenter des images sur plusieurs couches demande des calculs additionels à coût élevé, mais nous formulons le problème de décomposition des images afin de le résoudre efficacement avec une méthode itérative simple.

Mots-clés : Solveurs itératifs, simulation de fluides, augmentation de detail, réduction de modèle, écrans multicouches

SUMMARY

Realistic computer-generated images and simulations require complex models to properly capture the many subtle behaviors of each physical phenomenon. The mathematical equations underlying these models are complicated, and cannot be solved analytically. Numerical procedures must thus be used to obtain approximate solutions. These procedures are often iterative algorithms, where an initial guess is progressively improved to converge to a desired solution. Iterative methods are a convenient and efficient way to compute solutions to complex systems, and are at the core of most modern simulation methods.

In this thesis by publication, we present three papers where iterative algorithms play a major role in a simulation or rendering method. First, we propose a method to improve the visual quality of fluid simulations. By creating a high-resolution surface representation around an input fluid simulation, stabilized with iterative methods, we introduce additional details atop of the simulation. Second, we describe a method to compute fluid simulations using model reduction. We design a novel vector field basis to represent fluid velocity, creating a method specifically tailored to improve all iterative components of the simulation. Finally, we present an algorithm to compute high-quality images for multifocal displays in a virtual reality context. Displaying images on multiple display layers incurs significant additional costs, but we formulate the image decomposition problem so as to allow an efficient solution using a simple iterative algorithm.

Keywords: Iterative solvers, fluid simulation, upres, model reduction, multifocal displays

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

RHS	Right-hand side	RAM	Random-access memory
CFD	Computational fluid dynamics	CPU	Central processing unit
SPH	Smoothed particle hydrodynamics	FPS	Frames per second
FLIP	Fluid implicit particle	GPU	Graphics processing unit
VR	Virtual reality	HMD	Head-mounted display
1/2/3D	One/Two/Three dimensions	RMSE	Residual mean square error
HMD	Head-mounted display	VAC	Vergence-accommodation conflict
VAC	Vergence-accommodation conflict	LCD	Liquid-crystal display
CPT	Closest point turbulence	MTF	Modulation transfer function
STAR	State of the art review	PSF	Point spread function
K	Thousands	M/G/TB	Megabyte/Gigabyte/Terabyte
SVD	Singular-value decomposition	OLED	Organic light-emitting diode
AWCM	Adaptive wavelet collocation method	LED	Light-emitting diode
DFW	Divergence-free wavelet	DOF	Depth of field
SDF	Signed distance function	IAD	Interaxial distance
Hz/GHz	Hertz/GigahertzSLD	SLD	Superluminescent diode

Chapter 1

INTRODUCTION

The demand for realistic computer-generated images is constantly increasing to satisfy the growing quality requirements of modern movies, video games, and virtual reality systems. Consequently, the need for efficient physics-based rendering and simulation methods has drastically increased over the last decade. Improving realism, however, requires complicated models and inevitably increases the amount of data that needs to be processed. Despite continuous improvements in algorithmic methods and computing power, the production of realistic virtual environments remains a challenging problem.

The model chosen to represent a physical phenomenon is important when designing a rendering or simulation method, but ultimately, it is the computational method used to solve the model that has a direct impact its efficacy and usability. As such, finding the right balance between the physical accuracy of a model and the computational practicality of the algorithm used to solve it is key when designing new methods.

This thesis presents three instances where a physically-based model is altered in order to better suit an algorithmic solver. Two of the projects we present deal with the simulation of liquids and smoke, while the third project relates to the computation of high-quality images for virtual reality systems. The main contribution of each project is directly related to the use of an iterative algorithm to compute solutions to each of the proposed models.

The thesis is organized as follows: the remaining sections of this chapter introduce the main concepts of iterative solvers, fluid simulations, and virtual reality displays. The three following chapters then present each publication, along with a short introduction and, in the case of the first project, an additional section to discuss significant progress towards an extension of the paper (Section 2.C). Finally, our conclusion discusses additional avenues of future work for all three projects.

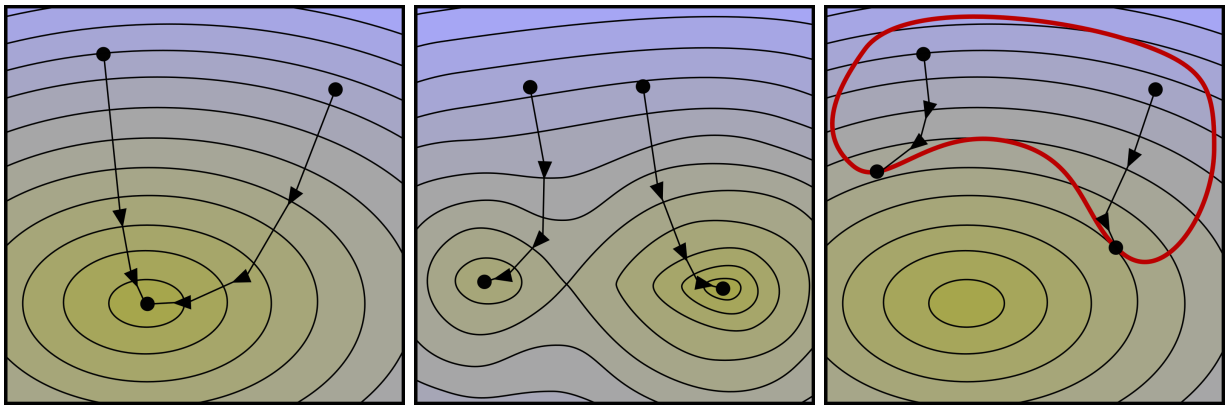
1.1. Iterative Solvers for Optimization Problems

A broad set of dynamic processes can be formulated in terms of optimizing an objective function under a given set of constraints. For instance, simple management problems can be

defined as the maximization of profits while satisfying resource and time constraints. This basic idea is key to the formulation of many more complicated problems. For instance, it is the basis for machine learning, where the proficiency of a learning network to accomplish a given task is optimized, constrained by the number and type of nodes in the network. Furthermore, any linear problem $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ are known, and vector $\boldsymbol{x}$ is the unknown, can also be formulated as the solution of the unconstrained minimization problem

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b}, \tag{1.1}$$

which further increases the class of problems solvable through optimization.



(a) Convex objective (b) Non-convex objective (c) Constrained convex objective

Figure 1.1. Solving global minimization problems with iterative methods. Level curves of an objective function are shown, and the color gradient represents the function value (increasing values from yellow to blue). a) Convex optimization leads to the single correct minimum. b) If the objective is not convex, two initial guesses can lead to different answers. Determining that we have reached the globally minimal solution is not always trivial. c) Even if the objective is convex, constraints (interior of red curve) can lead to different solutions.

In practical applications, the minimization objective is often too complex to be solved directly, for instance by analytically solving for points where the gradient is zero. However, evaluating the function is generally simple, and so is evaluating its first derivatives [1]. This leads to the use of descent methods: an initial guess $\boldsymbol{x}^{(0)}$ is chosen, and iteratively moved in the direction of the negative gradient of the objective function, i.e., the direction that most rapidly reduces the value of the objective. Figure 1.1 illustrates a descent method for various scenarios. In the simplest case (Figure 1.1a), the objective function is convex, and descent methods always converge to the global minimum. However, the objective can in general have many local minima (Figure 1.1b), in which case the choice of initial guess $\boldsymbol{x}^{(0)}$ can lead to different solutions. In this case, it is important to understand the general structure of the

1. If analytic derivatives are difficult to evaluate, they can be approximated with finite differences

objective function, so that the convergence behavior can be predicted, and the initial guess can be chosen appropriately.

Many problems deal with constrained optimization (Figure 1.1c), where the solution is restricted to a given subset of the search space. Descent methods can still be used in this case, by projecting each iteration back into the constraint set. Note that here, even if the objective function is convex, descent methods can still lead to multiple solutions. A knowledge of the structure of the constraints is therefore also required to guide the method to the desired solution.

This simple idea of progressively stepping towards the solution of a minimization problem is the basis for many complex, modern iterative methods. Since the behavior of the iterations depends on the structure of the objective and constraints, dozens of optimization methods have been developed over the years to properly adjust to specific classes of problems. A common approach when designing new physical models is to therefore model the problem accurately as a minimization problem, and to then apply an existing iterative method that best suits the chosen model. In this thesis, we instead argue that it is often beneficial to sacrifice some accuracy in the model in exchange for enabling the use of simpler iterative solvers. The three papers presented in Chapters 2, 3 and 4 illustrate methods where altering the model allows us to use more basic iterative methods, which both simplifies implementation and accelerates computations.

1.2. Fluid Simulations

The term *fluid* is used to group a large variety of substances, such as water, air, smoke, tar, whipped cream and corn starch. These fluids all behave differently, but they can all be modelled by variations of the incompressible Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} = -(\nabla \mathbf{u})\mathbf{u} - \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{1.2}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{1.3}$$

defined on some simulation domain $\Omega \subset \mathbb{R}^3$. In Equation 1.2, $\mathbf{u}$, p and $\mathbf{f}$ are functions of space with an implicit dependence on time t. More specifically, scalars p and ν are the pressure and viscosity, vector fields $\mathbf{u}$ and $\mathbf{f}$ represent the fluid velocity and external forces, ∇ is the gradient operator for scalars and the Jacobian for vectors, and ∇^2 is the Laplace operator. The RHS terms of Equation 1.2 are, from left to right, the advection, pressure, diffusion, and force terms, and Equation 1.3 is the incompressibility constraint. Note that these equations only represent the velocity and pressure of the fluid. In most cases, the fluid density is represented independently and is passively transported following the velocity field $\mathbf{u}$.

This system of partial differential equations is nonlinear due to the advection term, which makes it difficult to directly solve for **u**. The Navier-Stokes equations are also well known for their turbulent behavior, meaning that slightly different initial conditions can lead to drastically different solutions. Because of numerical imprecisions, it is therefore hard or impossible to compute a physically-accurate simulation of every small structure of the fluid. Fortunately, in most applications, and especially in computer graphics, it is sufficient to properly simulate the coarse behavior of the fluid and to only capture the statistical properties of the finer, turbulent scales, in order to give a realistic appearance to the fluid.

To be solved on a computer, the Navier-Stokes equations need to be discretized to transfer from the physical, continuous formulation to a finite representation. Depending on the situation, different structures can be used to represent the fluid, such as grids or point sets. The size of these structures, i.e., the number of grid nodes or the number of points, is referred to as the resolution of the fluid representation. Using a higher resolution to solve the Navier-Stokes equations generally yields a more accurate solution, but requires a larger computational effort.

Using the discretized Navier-Stokes model, solvers are designed to compute the fluid dynamics and evolve the fluid in time. There exists a large and constantly evolving variety of fluid solvers, but most follow the general structure shown in Figure 1.2, where an initial fluid is iteratively evolved frame-by-frame to create the simulation.

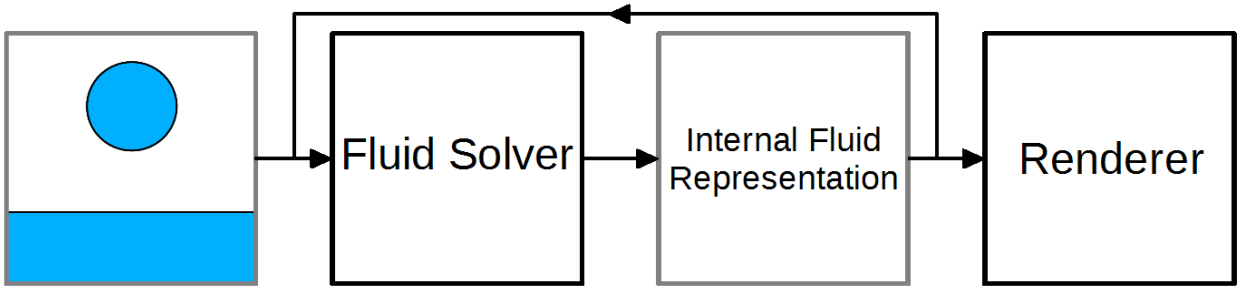


Figure 1.2. Typical fluid solver pipeline. An initial fluid condition (left) is given to the fluid solver (middle left) which represents the fluid using some internal representation such as grids or particles (middle right). The fluid solver updates this fluid representation iteratively to simulate its evolution. Each simulation frame created by the fluid solver is then forwarded to a renderer (right), such as a rasterizer or a ray tracer, to transform the abstract fluid representations into images.

1.2.1. Computational Fluid Dynamics vs. Computer Graphics

The methods employed to solve the Navier-Stokes equations greatly depend on the application domain. In the computational fluid dynamics (CFD) community, the solutions are used mostly for prediction purposes, e.g., to compute the lift of an airplane wing or to

predict the movement of clouds for weather forecasts. These applications require accurate algorithms that correctly predict the real, physical behavior of fluids. Quality metrics for such applications can therefore be devised by directly comparing the simulation to a measured experiment in a simple case, such as the flow of a fluid past a circular object (Figure 1.3).

In computer graphics, the physical properties of the fluid are arguably less important than the final appearance of the animation sequence. The model therefore only needs to faithfully represent enough of the fluid dynamics in order to convey a convincing fluid behavior. In effect, ground truth comparisons are rarely used in computer graphics, since a solution that does not match reality is not necessarily unwanted. It is actually often desirable to design methods that can controllably deviate from physics in order to, e.g., satisfy the artistic needs of a movie scene [17, 12]. Fluid simulation methods are therefore more properly compared according to computational times, memory requirements, appearance, and flexibility.

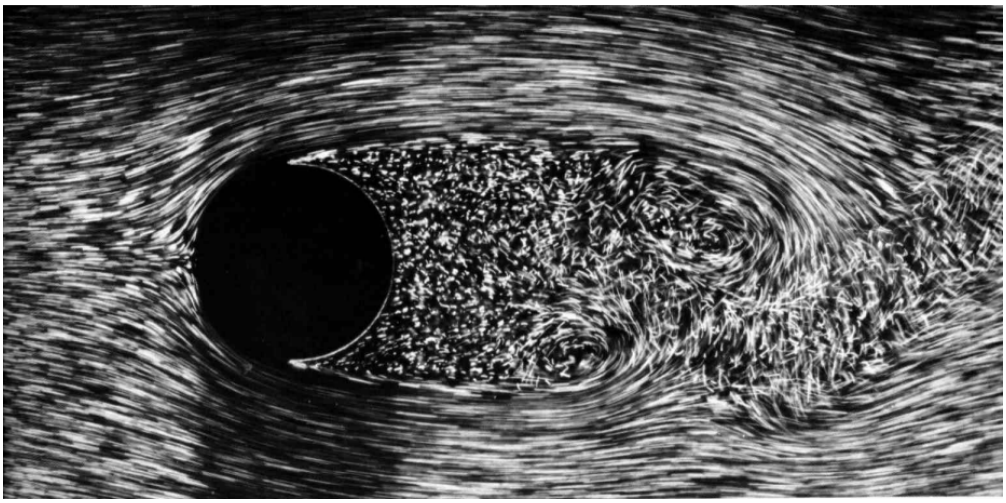


Figure 1.3. Visualization of a real air flow around a sphere [18]. Simulations in CFD try to recreate and predict the fluid behavior of such real scenarios.

1.2.2. Eulerian vs. Lagrangian Approaches

A wide variety of fluid solvers have been developed over the last few decades, and they can generally be separated in three broad categories depending on the fluid discretization they use: Eulerian, Lagrangian, and hybrid methods, all illustrated in Figure 1.4.

Eulerian methods discretize the fluid on a grid. The simulation domain is embedded in an $N \times N \times N$ regular grid, and each grid cell stores a value for $\mathbf{u}$ and the fluid density. This reduces simulations to solving partial differential equations on a grid, a problem which has been widely studied and is well understood. The canonical Eulerian method is that of Stam [16]. This method uses an operator splitting approach to iteratively solve the forces, advection, viscosity, and pressure incompressibility steps of the Navier-Stokes equation. The

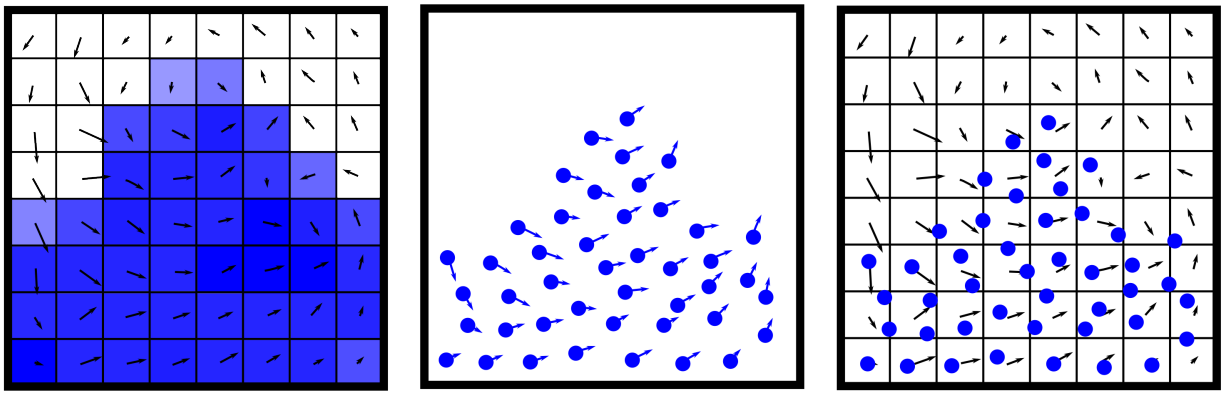


Figure 1.4. Three different fluid discretizations. Left: Eulerian, densities and velocities are stored on a grid. Middle: Lagrangian, fluid is represented by particles that store their velocity. Right: Hybrid, the grid is used to resolve velocities, but the fluid is represented as particles.

incompressibility step solves the constraint of Equation 1.3, and is generally the most expensive operation, as it requires solving a linear system of size proportional to the entire grid. The grid representation of Eulerian methods limits the fluid structure sizes and tends to create artificial smoothness. Many methods have since improved the approach to better its efficiency [8, 7] and ameliorate its sharpness [4, 15, 14], but fully Eulerian simulations remain limited.

Lagrangian methods instead discretize a fluid as a set of points called particles. This provides a significnatly better representation compared to grids, since particles allow to represent arbitrarily small fluid structures such as droplets and splashes. Particles also allow to focus the fluid data only in regions where fluid is present, which can provide significant improvements in term of memory and computation speed. As such, Lagrangian methods like smoothed particle hydrodynamics (SPH) [10] approaches and position-based fluids [9] are widely used for interactive applications. However, they can offer an unrealistic and less robust behavior, and are prone to instabilities in corner cases. Their implementation also often requires more care, since they do not benefit from the simple and regular structures present in grid-based methods.

Hybrid methods combine the Eulerian and Lagrangian approaches to obtain a detailed fluid representation with stable, realistic behavior. A example commonly used in computer graphics is the Fluid Implicit Particle (FLIP) method [20], which uses a grid to solve the differential equations like an Eulerian method, but stores data on particles that are advected in the velocity field like a Lagrangian method. Many improved hybrid methods have been developped [2, 3, 6] and they are currently the preferred option for high-quality fluid solvers in the film industry.

1.2.3. Smoke vs. Liquid Simulations

One important distinction between the first two papers of this thesis is that one deals with the simulation of water, while the other simulates smoke. Although they are both fluids, and are both governed by the Navier-Stokes equations, they have important conceptual differences that need to be handled.

Smoke clouds are usually represented as single-phase fluids, meaning smoke is simply represented as air with a varying concentration of passive smoke particles. Smoke clouds do not have a well-defined, sharp boundary, and since they are not fully opaque, it is important to correctly represent their entire volumetric structure for simulation and rendering. As such, Eulerian or hybrid representations are often preferred for smoke simulations. Smoke solvers are usually easier to implement, and the challenge of modern smoke simulations resides almost entirely in the large grid resolution requirements, resulting in high computational costs.

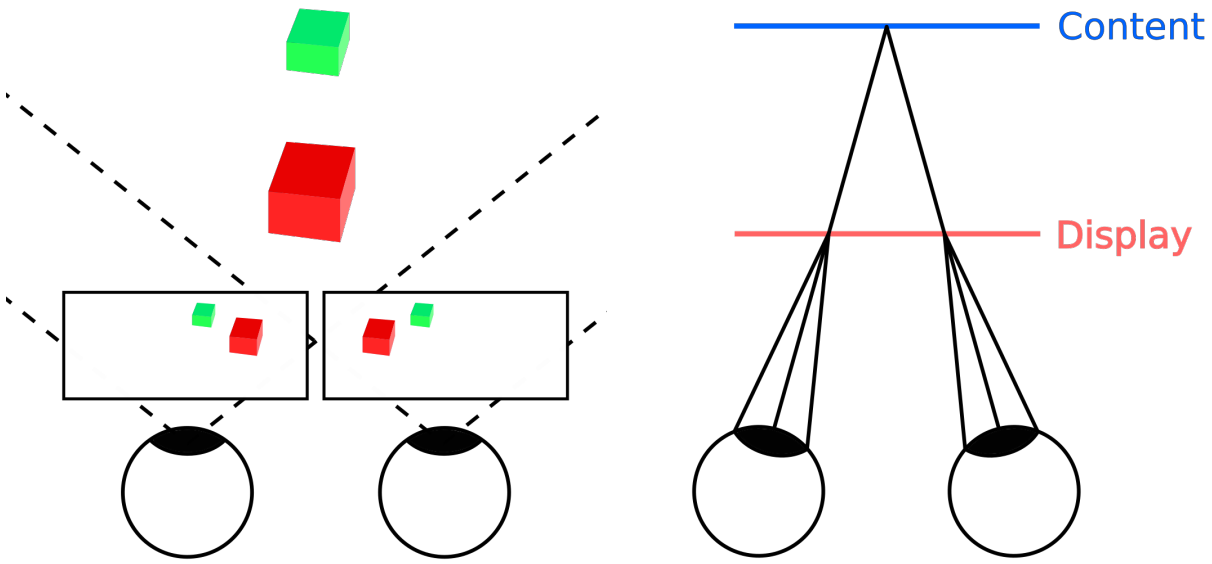
In contrast, liquid simulations use a two-phase representation, meaning each point in the simulation domain is either inside or outside of the fluid. Liquids have a clear boundary, and benefit from the precision of particle-based representations used in Lagrangian or hybrid solvers. The surface of a liquid requires specific attention, since it is the main visible component of a liquid, and has a direct effect on the simulations in the form of surface tension. Liquids simultaneously exhibit a wide variety of distinct behaviors, such as large eddies, surface waves, and misty splashes. This makes liquids usually more difficult to simulate realistically compared to smoke, and liquid models often need to be specialized to focus only on a subset of all possible fluid behaviors.

1.3. Virtual Reality Displays

Despite their seemingly recent mainstream popularity, virtual reality (VR) systems have been around since the second part of the 20th century. Even though early systems were technologically light-years away from modern implementations, they had the ability to convey the impression of depth in 3D imagery. Although a complete VR system also requires to simulate inputs for all senses through haptic devices and motions peripherals, displaying high-quality 3D imagery to the user is paramount in creating a realistic VR experience.

Creating a coarse impression of depth is not a difficult task. As shown in Figure 1.5a, by simply displaying different images to each eye, the brain can combine the images and interpret the result as 3D content. These images can be simply computed by separately projecting the scene onto each eye. This also generally does not require complicated hardware, as primitive cardboard glasses with red and cyan filters can achieve the effect [2].

2. This method of generating 3D effects is known as *anaglyph 3D*.



(a) (b)

Figure 1.5. a) By displaying different images to each eye, VR display can trick the brain into combining the images interpreting the result as a 3D scene. a) Since the left eye sees the green object to the left of the red object, while the right eye sees the opposite, the brain interprets the combination as the red object being closer than the green object. b) Using binocular images, content can be represented at any depth in the scene. However, the physical display is located at a fixed depth. The eye must accommodate at the distance of the display to get a sharp image, which leads to a conflict between the accommodation and vergence of the eyes (See Section 1.3.1).

However, creating a *convincing* impression of depth is a surprisingly difficult task, plagued by the subtleties of the human visual system. Even though virtual reality systems are usually implemented as head-mounted displays (HMD) that naturally follows the user's head movements, eyes constantly move with respect to the HMD. As the user's eyes rotate to look at different parts of the scene, or as the HMD slips on the user's head, the center of projection used to compute the displayed images does not accurately match the actual position of the user's eyes, which creates image distortions and unrealistic results. To construct a perfect VR display, eye-tracking is therefore needed to adapt the computed images to the dynamic location of the eyes, significantly complicating hardware implementations.

Furthermore, the optics of the eyes are not modelled accurately by the pinhole camera model commonly used in computer graphics. The eye has a pupil with a finite radius, and a lens that allows it to focus the image at different depths. This optical system is also not represented accurately by a thin lens model, since aberrations in the cornea and lens creates image distortions that vary widely between users. These distortions, as well as the distortions caused by the optics of the HMD, need to be compensated by the displayed

images in order to perfectly recreate the desired image on the user's retina. We refer to the book by Schwartz [13] for an introduction to the optics of the eye.

Many effects further complicate the eye's optical system by constantly varying it in time. For instance, the variation in image deformation is not uniform as the eyes rotate, and microfluctuations [19] constantly change the focus of the eye. Such effects are difficult to measure externally, and prevent a high-accuracy prediction of the image distortion by the eye.

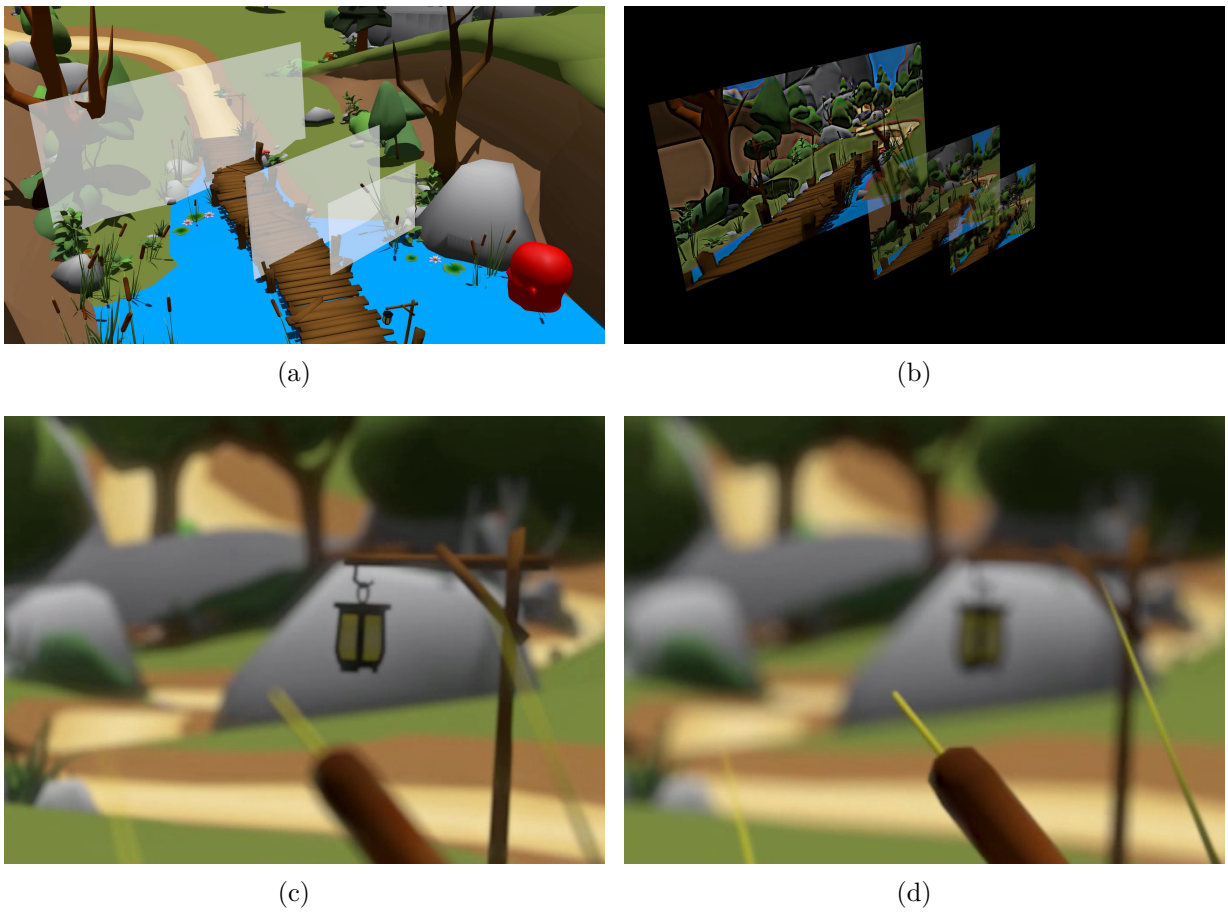
1.3.1. Solving the Vergence-Accommodation Conflict

To convey a realistic immersion into a virtual environment, a user must be able to focus freely on any part of the scene. For natural viewing conditions, the eyes will naturally *verge* (i.e., point) towards the focused target, and the eye's focusing mechanism will *accommodate* to the desired focus distance, as in Figure 1.5b. However, if only one image is presented at a fixed distance from each eye, the focus of the eye needs to adjust to the depth of the display, and not the depth of the virtual content, in order to receive a sharp signal at the retina. This creates an issue known as the vergence-accommodation conflict (VAC), where the vergence and accomodation of the eyes are not set to the same distance. This is known to be a significant cause of discomfort, and leads to unnatural viewing conditions [5].

Instead of a single display plane, one solution to the VAC is to use volumetric displays, which can display content with real depth properties. In effect, the perfect volumetric display would be a super-fast 3D printer with dynamic lighting fitted in front of the user's face, capable of accurately building any 3D scene in real-time. Since this is obviously far from possible with current technology, approximations must be employed instead.

One possibility is to use multifocal displays, which use multiple transparent additive displays placed at different depths in the scene, as shown in Figure 1.6a for three displays. This allows the user to focus at three different depths while still receiving sharp image signals. However, because the images are combined additively, the content of the out-of-focus displays appears blurred and interferes with the in-focus content. Computing the images to show on each display in order to accurately reconstruct image at each focal depth is not a simple problem, and leads to unintuitive results, as shown in Figure 1.6b. Still, since multifocal displays support each viewing direction by more than one pixel, the additional degrees of freedom can effectively reproduce 3D content at multiple focal depths. In Figure 1.6c-d, the combination of three images allows the user to focus on the cattail and on the lamppost without changing the content of the images.

Simple methods can be used to compute the images to show on each display, such as the linear blending method of Akeley et al. [1], depicted in Figure 1.7. This approach shows each element of the scene on the display planes that are closest to it in depth, smoothly blending between displays. This creates images like in Figure 1.8, where the scene is effectively split



(a) (b)

(c) (d)

Figure 1.6. a) Three virtual displays are positioned at various depth to cover a larger accommodation range. b) The scene is decomposed into three images to be presented on the displays. c-d) Given the additional degrees of freedom enabled by the multiple displays, the three images can be combined to provide proper focus at potentially many more depths. In this case, the combination of images allows the user to properly focus on the lamppost or on the cattail using a static set of three images.

into regions of different depths; elements near the viewer are sent to the nearest display, and background elements are sent to the furthest display from the user.

Although this approach is simple to compute, it does not always provide an accurate reconstruction of the scene. Figure 1.9 shows a simple scene composed of two rectangular object overlapping each other where linear blending does not properly reconstruct the region near the occlusion.

Better images can be obtained by formulating the problem as a minimization. Since many perceptual factors come into play when determining image quality, choosing the best objective function to minimize is not an easy task. The work we present in Chapter 4 follows the optimal decomposition work of Narain et al. [11] and defines the objective based only on a geometric formulation of the blurring of each display as the eye focuses in the accommodation

range. Even though this objective function greatly simplifies the human visual system, it gives a better image reconstruction than linear blending, as shown in Figure 1.9c. The paper presented in Chapter 4 aims to adjust the optimal decomposition formulation to make it more easily solvable by an iterative solver.

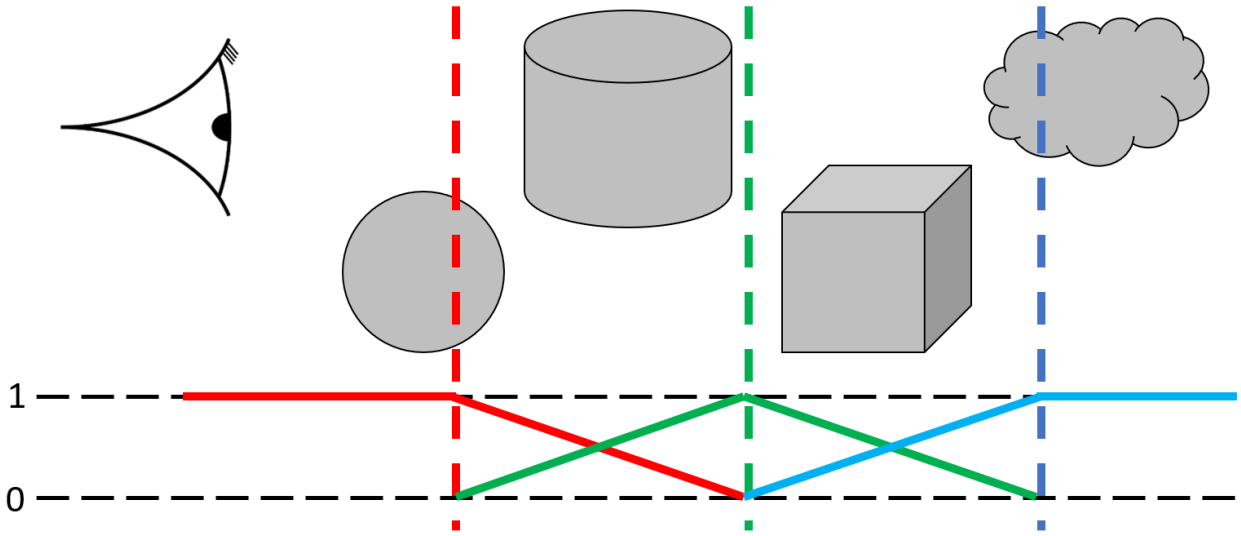
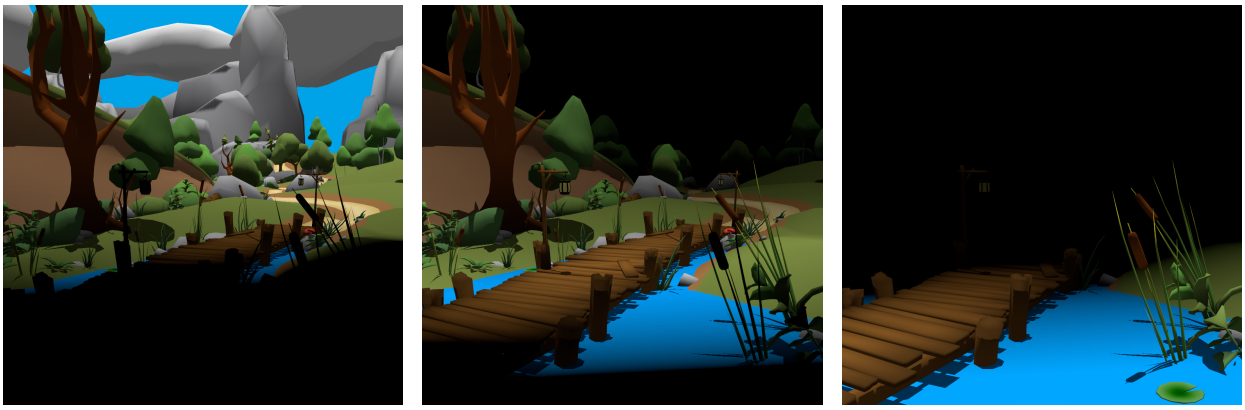
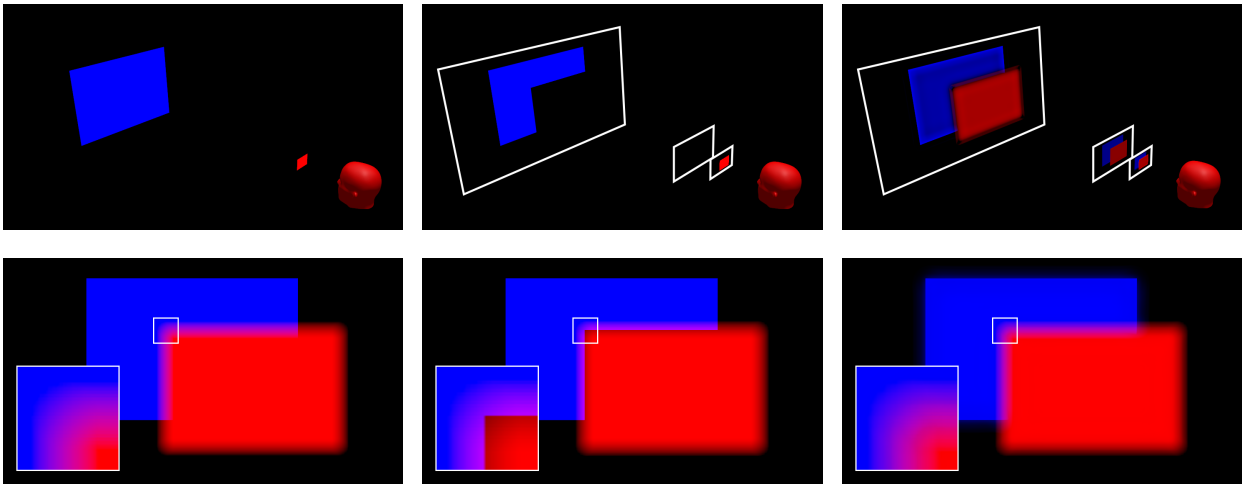


Figure 1.7. Linear decomposition of a scene onto the three displays located at the dashed lines. The objects are spread onto each display proportionally to their distance to the display location. The bottom of the image shows the linear weighting used to distribute the images onto each display plane.



(a) Back display (b) Middle display (c) Front display

Figure 1.8. Images obtained from linear decomposition on each display. Because of the linear weight, the scene transitions smoothly in depth between the three displays.



(a) Real observed image (b) Linear blending (c) Optimal decomposition

Figure 1.9. Two rectangular objects overlap in front of a user (top), resulting in an observed image with occlusion as the user focuses on the blue rectangle (bottom). a) In a perfect volumetric display, the region near the occlusion of the two rectangles presents a smooth transition. b) Using linear blending, the large depth discrepancy prevents the occlusion from being properly reconstructed, and a sharp edge is visible at the occlusion boundary. c) Using optimal decomposition, the reconstruction is much closer to the real observed image in (a).

References

[1] Akeley, K., S. J. Watt, A. R. Girshick and M. S. Banks. 2004, A stereo display prototype with multiple focal distances, *ACM Transactions on Graphics (TOG)*, vol. 23, nº 3, p. 804–813, ISSN 0730-0301.

[2] Batty, C., F. Bertails and R. Bridson. 2007, A fast variational framework for accurate solid-fluid coupling, in *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, p. 100.

[3] Boyd, L. and R. Bridson. 2012, Multiflip for energetic two-phase fluid simulation, *ACM Transactions on Graphics (TOG)*, vol. 31, nº 2, p. 16.

[4] Fedkiw, R., J. Stam and H. W. Jensen. 2001, Visual simulation of smoke, in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, p. 15–22.

[5] Hoffman, D. M., A. R. Girshick, K. Akeley and M. S. Banks. 2008, Vergence–accommodation conflicts hinder visual performance and cause visual fatigue, *Journal of vision*, vol. 8, nº 3, p. 33–33.

[6] Jiang, C., C. Schroeder, A. Selle, J. Teran and A. Stomakhin. 2015, The affine particle-in-cell method, *ACM Transactions on Graphics (TOG)*, vol. 34, nº 4, p. 51.

[7] Klingner, B. M., B. E. Feldman, N. Chentanez and J. F. O'brien. 2006, Fluid animation with dynamic meshes, in *ACM Transactions on Graphics (TOG)*, vol. 25, ACM, p. 820–825.

[8] Losasso, F., F. Gibou and R. Fedkiw. 2004, Simulating water and smoke with an octree data structure, in *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, p. 457–462.

[9] Macklin, M. and M. Müller. 2013, Position based fluids, *ACM Transactions on Graphics (TOG)*, vol. 32, nº 4, p. 104.

[10] Müller, M., D. Charypar and M. Gross. 2003, Particle-based fluid simulation for interactive applications, in *ACM SIGGRAPH/Eurographics Symposium on Computer animation*, p. 154–159.

[11] Narain, R., R. A. Albert, A. Bulbul, G. J. Ward, M. S. Banks and J. F. O'Brien. 2015, Optimal presentation of imagery with focus cues on multi-plane displays, *ACM Transactions on Graphics (TOG)*, vol. 34, nº 4, ISSN 0730-0301.

[12] Raveendran, K., N. Thuerey, C. Wojtan and G. Turk. 2012, Controlling liquids using meshes, in *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*, Eurographics Association, p. 255–264.

[13] Schwartz, S. H. 2013, Geometrical and visual optics, McGraw Hill Professional.

[14] Selle, A., R. Fedkiw, B. Kim, Y. Liu and J. Rossignac. 2008, An unconditionally stable maccormack method, *Journal of Scientific Computing*, vol. 35, nº 2-3, p. 350–371.

[15] Selle, A., N. Rasmussen and R. Fedkiw. 2005, A vortex particle method for smoke, water and explosions, in *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, p. 910–914.

[16] Stam, J. 1999, Stable fluids, in *Proceedings of 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 121–128.

[17] Thürey, N., R. Keiser, M. Pauly and U. Rüde. 2009, Detail-preserving fluid control, *Graphical Models*, vol. 71, nº 6, p. 221–228.

[18] Van Dyke, M. and M. Van Dyke. 1982, An album of fluid motion, .

[19] Zhu, M., M. J. Collins and D. Robert Iskander. 2004, Microfluctuations of wavefront aberrations of the eye, *Ophthalmic and Physiological Optics*, vol. 24, nº 6, p. 562–571.

[20] Zhu, Y. and R. Bridson. 2005, Animating sand as a fluid, in *ACM Transactions on Graphics (TOG)*, vol. 24(3), ACM, p. 965–972.

Chapter 2

SURFACE TURBULENCE FOR PARTICLE-BASED LIQUID SIMULATIONS

This first paper presents a method for improving particle-based fluid simulations. A low-resolution simulation is taken as input and augmented using a finer, particle-based surface shell. The surface is then displaced to generated additional detalis along the surface.

A key component of our method is the creation and maintenance of the point-based surface. It is based on a global constrained minimization problem, where the surface is made as smooth as possible within prescribed limits around the input simulation. This construction improves upon previous work, designing a smooth band constraint to contain the surface, which allows for the application of a simple and efficient iterative scheme.

Following the paper, section 2.C presents a previously unpublished extension of the method to the wrinkling of viscous fluids.

2.A. Publication

This paper was published in ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2015, in November 2015. It has been reformatted to appropriately follow the format of this thesis.

Olivier Mercier and Cynthia Beauchemin are the two principal authors of this project. Mercier was primarily involved in the numerical analysis of the surface creation method and the design of the wave seeding and evolution algorithms, as well as the writing and presentation of the paper.

Surface Turbulence for Particle-Based Liquid Simulations

Olivier Mercier, *Université de Montréal*

Cynthia Beauchemin, *Université de Montréal*

Nils Thuerey, *Technische Universität München*

Theodore Kim, *University of California, Santa Barbara*

Derek Nowrouzezahrai, *Université de Montréal*

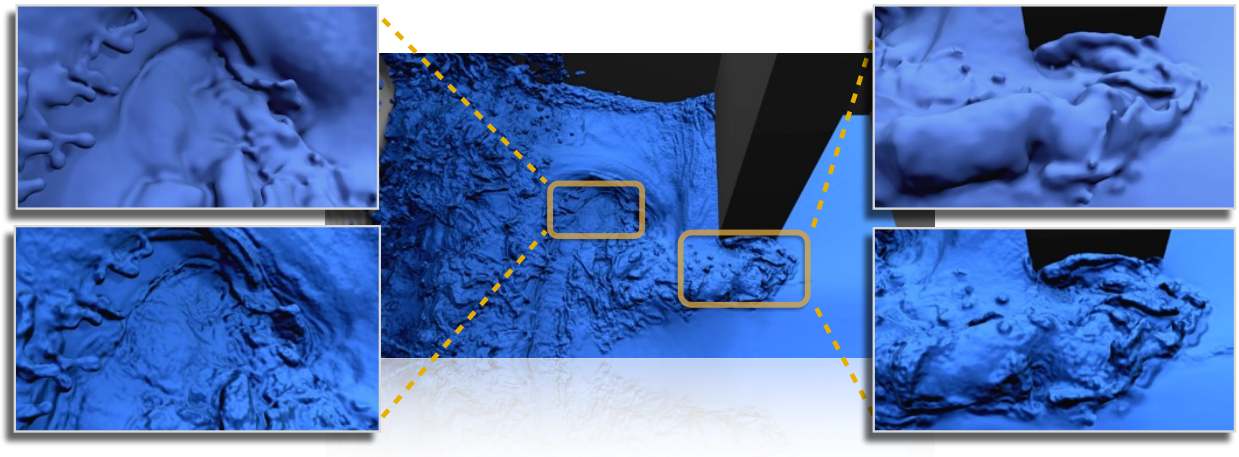


Figure 2.1. We apply our turbulence model to a high-resolution FLIP simulation ($>12{\times}10^6$ particles). Zoom-ins compare the unmodified input surface (top) to our output (bottom). Even at high resolutions, the input simulation fails to resolve small scale details, which our method is capable of adding. In this extreme example, our entire post-process adds an overhead of roughly a third of the full simulation time.

Résumé

Nous présentons une méthode pour améliorer la résolution apparente des simulations de liquides basées sur des représentations par particules. À partir d'une simulation à basse résolution, notre méthode produit un ensemble de points dense, temporellement cohérent, et régulier. Une simulation Lagrangienne de vagues de surface est ensuite calculée sur cet ensemble de points à haute résolution. Nous développons de nouvelles méthodes pour la génération et la simulation de vagues sur un ensemble de points que nous utilisons pour générer des détails à haute résolution. Nous évitons les manipulations de maillage susceptibles aux erreurs, et propageons robustement les vagues sans avoir besoin de notions explicites de

connectivité entre les points. Notre méthode de génération de vagues combine une évaluation robuste de la courbure avec plusieurs bandes d'oscillations, injecte des vagues avec des structures arbitrairement fines, et gère adéquatement les obstacles. Nous générons des surface fluides détaillées à partir de simulation de basse résolution par un processus a posteriori indépendant de la simulation originale qui peut être appliqué à la plupart des algorithmes de simulation fluides basés sur des particules.

Mots-clés : Simulation Lagrangienne de fluides, turbulence par vagues de surface.

Abstract

We present a method to increase the apparent resolution of particle-based liquid simulations. Our method first outputs a dense, temporally coherent, regularized point set from a coarse particle-based liquid simulation. We then apply a surface-only Lagrangian wave simulation to this high-resolution point set. We develop novel methods for seeding and simulating waves over surface points, and use them to generate high-resolution details. We avoid error-prone surface mesh processing, and robustly propagate waves without the need for explicit connectivity information. Our seeding strategy combines a robust curvature evaluation with multiple bands of seeding oscillators, injects waves with arbitrarily fine-scale structures, and properly handles obstacle boundaries. We generate detailed fluid surfaces from coarse simulations as an independent post-process that can be applied to most particle-based fluid solvers.

Keywords: Lagrangian fluid simulation, wave turbulence.

2.1. Introduction

Simulating the behavior of fluids is a long standing problem that often requires visual details resolved to a very fine resolution. Simulating smoke and liquids are the two most common cases, and specialized approaches have been developed for each.

For *smoke* animation, Eulerian approaches are common. Here, performance scales with the underlying grid's resolution and resolving details at fine scales quickly becomes prohibitive. *Fluid up-res* methods address this problem by applying fine-scale turbulence models atop coarser simulations, generating detailed results without explicitly simulating at a fine resolution [45, 55, 62]. These methods are practical for art-direction since they guarantee that the coarse behavior will not change when fine details are added.

In contrast, detailed *liquid* simulations are still performed at full resolution, regardless of whether a grid- or particle-based approach is used, with high-quality particle-based simulators, such as Fluid Implicit Particle (FLIP) [82, 23] or Smoothed Particle Hydrodynamics (SPH) [53, 56], having seen rapid, recent adoption.

We address the discrepancy with an *"up-res"* technique for particle-based liquid simulations. While an Eulerian *closest point turbulence* (CPT) method was recently developed for level set-based liquids [44], it can only be applied to particle-based data by converting the data to an Eulerian grid, discarding many of the simulation's rich details. We instead add turbulent details *directly* to particles, solving the wave equation in a Lagrangian setting. We first convert a set of input particles from a coarse liquid simulation into a high-quality, high-density surface point set. We then perform a wave simulation that adds high-frequency features to the liquid surface, in the form of bump or displacement maps. We use standard surfacing to obtain a detailed, high-resolution liquid surface.

To our knowledge, ours is the first comprehensive up-res technique for particle-based simulations, making the following contributions:

- robust, temporally coherent, meshless surface generation, that yields smooth, simulation-ready surfaces,
- smooth constraints to ensure that our surface remains spatially and temporally faithful to the underlying particle set,
- a robust, curvature-based method for initiating surface waves,
- a novel discrete Laplace operator that is provably well-suited for meshless point representations, in addition to
- an efficient simulation strategy that synthesizes details across scales onto our high-density surface.

Our method is agnostic to the source of particle data, and can be applied to FLIP, SPH, and position-based works [51].

2.2. Previous Work and Overview

Fluid simulation is a well-established area so, in addition to seminal works [35, 68, 34, 53], we refer readers to Bridson's book [28] and Ihmsen et al.'s STAR [41] for comprehensive surveys. Here, we focus primarily on areas most related to our work.

Fluid Up-res. Thuerey et al. [71] surveys recent fluid up-res methods which efficiently increase the apparent spatial resolution of a coarse simulation without altering the low-frequency behavior. As noted in Section 2.1, these methods have been most effective in smoke simulations [45, 55, 62, 58, 38]. Several works have also addressed the related problem of synthesizing frequency-controlled textures on moving surfaces [79].

Earlier attempts to apply up-res algorithms to liquids had limited success, both in Eulerian [55] and Lagrangian [80, 64] formulations, since they focus on increasing the resolution of the velocity field. As noted by Kim et al. [44], the turbulence on a free surface is only loosely

coupled to the fluid velocity field. Lab experiments show that surface waves tend to propagate much faster than the velocities of the underlying fluid would suggest. We thus choose to evolve a high-resolution simulation over the liquid surface to obtain more convincing results.

While CPT [44] works well for Eulerian liquids, no comprehensive Lagrangian up-res method exists. This is unfortunate, because the ad-hoc methods developed in industry [29] show that there is substantial interest in such techniques.

Several works have explored how to guide liquids to meet artistic goals [66, 60]. Nielsen and Bridson [57] use a low-frequency "guide shape", extracted from a coarse FLIP simulation, as the boundary condition for a thin, secondary, high-resolution simulation applied near the liquid boundary. Our work differs from this approach in two ways: first, we preserve the entire frequency content of the coarse simulation, including important quantities such as the silhouettes. Second, we add entirely new dynamics to the surface using a wave simulation. As such, our algorithm can complement such "guide shapes" approaches, especially since it is applied as a standalone post-process.

Surface Tracking. The importance of surface-only simulations has become increasingly clear in recent works on explicit surface tracking [72], and methods that use them [78]. Wojtan et al. [77] survey these recent developments. Instead of explicitly modeling the fluid surface and carefully incorporating it into the core simulation, we propose a post-process that can be applied directly to any coarse particle simulation, remaining *fully decoupled* from the simulator that generated the data. Maintaining a simulation mesh is cumbersome, often requiring external geometry processing tools. Our meshless method is self-contained, simplifying implementation. While we create a mesh for rendering, meshing artifacts do not degrade the stability of the simulation.

Inspired by optimization work for liquid surfacing [76, 26], we constrain our final surface to lie in a band around the input surface. We extend ideas from Eulerian level sets and meshes to particle-based surfaces. While these works focus on generating geometry for rendering, we improve their smoothness and temporal regularity to make them suitable for simulation.

Wave Simulation. Many works consider wave simulations. From the linear wave equation [43], and related shallow-water models [74], to bi-Laplacian variants [78] and the *iWave* [70]. We use the linear wave equation and rely on dispersion from stretching induced by the underlying advection. As with all these previous works, we manually set an average propagation speed for our surface waves.

On the other hand, *wave particles* represent surface waves [81, 33] with a dense set of advected points. These methods have difficulty with the complex, topologically-varying surfaces that we treat, requiring many more wave particles to represent the details we achieve with our approach.

Point-Based Simulation. Our work is related to point-based techniques, but unlike previous works that deal with static point sets [83, 22] or dynamic surfaces for rendering [36],

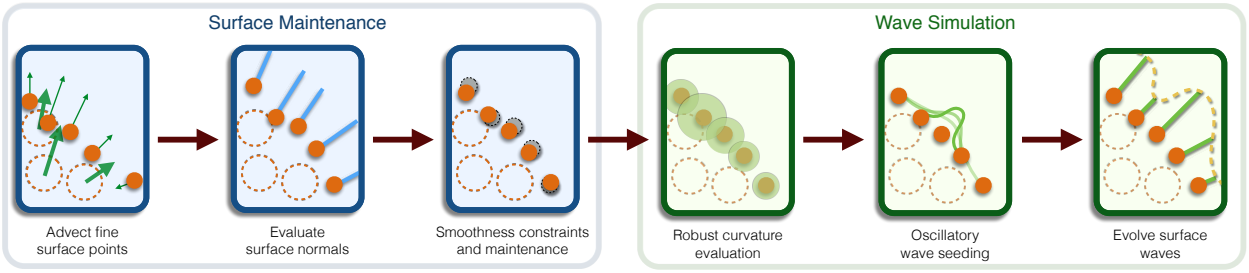


Figure 2.2. An overview of the different steps of our algorithm, performed for each frame of coarse input data. The fine surface points (solid circles) are evolved on the surface of the input coarse particles (dashed circles). The overview in Section 2.2 details each stage of this diagram.

our input particles represent a volume where every particle corresponds to a quantity of liquid. Point-based rendering treats each sample as a (possibly noisy) surface point, and previous point-based simulations [50, 42] operate on static point sets, and are thus not appropriate for dynamic particle sets from liquid simulations.

Overview. Figure 2.2 is a visual overview of our method. Given an input sequence of particles representing a liquid volume (*coarse particles*, dashed circles), we first construct a dense point set along the liquid interface (*fine surface points*, solid circles; Sections 2.3.2 & 2.3.3). We smooth and regularize these points to support point-based simulation (Section 2.3.4). Using per-point normals and displacement values we call *wave values* (green lines; Section 2.4.2), we perform a high-resolution wave simulation (green curve; Section 2.4.3) over the surface. We output the final detailed surface as a bump or height map over the high-density point set, or as a displaced point set. These points can be splatted directly or tessellated for rendering, which is easy given our regular surface point distribution.

2.3. Surface Construction and Maintenance

We construct a dense point set that represents the liquid's surface, and maintain a level of smoothness and regularity necessary for point-based wave simulation. We describe our novel smooth band constraint that controls the surface's behavior and ensures coherence between the surface points and underlying simulation. Unlike level set or mesh-based surface tracking [59, 77], our fluid surface is represented exclusively by oriented points i, each with a position $\mathbf{x}_i$ and normal $\mathbf{n}_i$.

2.3.1. Neighborhood Relationships

To ensure that our surface points behave as a unified manifold, we draw upon work in meshless simulation (e.g., [41]), taking advantage of the neighborhood relationships between point pairs. Our high-resolution surface points $\mathbf{x}_i$, and coarse input particles $\mathbf{X}_i$, will affect each other across spatial scales. Specifically:

- a coarse-scale length λ_c, obtained from the coarse particle simulator (e.g., the grid cell size in a FLIP solver), and
- a fine-scale length λ_f, a user parameter controlling the separation between points of the detail-enhanced surface.

We use λ_c for operations related to the underlying fluid, such as surface advection (Section 2.3.2) and curvature evaluation (Section 2.4.1), and λ_f for intrinsic surface operations, including point distribution regularization (Section 2.3.4) and wave evolution (Section 2.4.3). We use isotropic kernels to weight the effect of particles on their neighbors. Unless specified otherwise, we use a simple triangular kernel:

$$K_i^\delta(\mathbf{x}_j) = 1 - ||\mathbf{x}_i - \mathbf{x}_j|| \Big/ \delta, \text{ if } ||\mathbf{x}_i - \mathbf{x}_j|| < \delta \text{ ; 0, otherwise}, \tag{2.1}$$

where δ is the local neighborhood radius. We normalize the weighting kernel according to the local density around a point j,

$$\rho_j^\delta = \sum_{k \in \mathcal{F}} K_j^\delta(\mathbf{x}_k), \tag{2.2}$$

where $\mathcal{F}$ denotes the set of all surface points. This local normalization eliminates any bias introduced by potential non-uniformities across local point densities, and so the local weight is $w_i^\delta(\mathbf{x}_j) = K_i^\delta(\mathbf{x}_j)/\rho_j^\delta$. This density normalization is especially important at the finest scale, where point distribution variance is highest.

We also normalize the weighting so the contributions sum to unity, so our final weight of point j for point i is

$$W_i^\delta(\mathbf{x}_j) = w_i^\delta(\mathbf{x}_j) \Big/ \sum_{k \in \mathcal{F}} w_i^\delta(\mathbf{x}_k). \tag{2.3}$$

In practice, we only sum over particles in finite neighborhoods due to the local support of K. The weights also apply naturally to coarse particles by exchanging $\mathcal{F}$ for $\mathcal{C}$, the set of all coarse particles.

2.3.2. Surface Initialization and Advection

We create the initial set of fine-scale surface points in two steps. We first create an initial point set by centering spheres of radius λ_c around each coarse particle, sampling points uniformly on these spheres, and only retaining points that do not fall inside any other sphere's volume. This process results in a very rough, non-smooth, fine-scale surface point distribution that we regularize in a second step (see Section 2.3.4). For any subsequent frame n, fine surface points $\mathbf{x}_i^n$ are obtained by simply advecting points $\mathbf{x}_i^{n-1}$ from the previous frame. The updated surface point position is a weighted sum of the displacements of neighboring coarse particles $\mathbf{X}$,

$$\mathbf{x}_i^n \leftarrow \mathbf{x}_i^{n-1} + \sum_{k \in \mathcal{C}} W_i^{2\lambda_c}(\mathbf{X}_k)\,(\mathbf{X}_k^n - \mathbf{X}_k^{n-1}). \tag{2.4}$$

We can modify the neighborhood size used for advection according to how closely we want fine-scale surface points to track coarse particles; in practice, a neighborhood radius of $2\lambda_c$ yields smoothly advected fine-scale surfaces. We use this value for all of our results.

After advection, fine-scale points may no longer be located in the vicinity of the coarse particles, so the fine-scale surface behavior may deviate from the dynamics of the underlying coarse simulation. Maintaining a correspondence between the input (coarse) and output (fine) fluid behavior is essential for predictable artistic control, so we next devise a surface constraint that imposes this guarantee.

2.3.3. Surface Constraints

A fundamental component of our approach is the generation of a smooth, temporally coherent, high-resolution output surface that does not need to explicitly track manifold connectivity. To accomplish this, we devise an implicit representation that constrains the position of the fine-scale surface points.

Our method is motivated by Williams [76]: First, two concentric spheres of radius r and R are centered at each coarse particle, where R is the larger of the two. During surface evolution, the fine-scale surface points are constrained to remain in the volume between the union of all outer spheres and the union of all inner spheres. As depicted in Figure 2.3, this volume region forms "bands" around the coarse particles, delimiting the region where the advected fine scale surface is allowed to exist. This constrains surface points to regions not too far from, nor too close to, the existing coarse particles.

The r and R parameters affect the final output appearance. Small values create surfaces that more closely resemble the coarse simulation, but increase bumpiness. Large values can create surfaces that deviate significantly from the coarse simulation and exhibit over-smoothing. While they can be manually adjusted, we found that $R = \lambda_c$ results in a reasonably smooth surface relative to the underlying simulation and $r = R/2$ leaves sufficient space for fine particles to evolve without closely approaching the coarse particles.

Williams uses the spheres to constrain a thin-plate energy optimization to represent the surface; it is unclear how this can apply to meshless settings without costly intermediate meshing. Moreover, projecting onto the surface formed by the union of spheres is difficult due to discontinuities in the first derivatives, and an orthogonal projection creates a discontinuous surfaces unsuitable for wave simulation (Section 2.4.3).

To solve this problem, we instead project onto an implicit metaball-like formulation [27] that leads to smooth projection constraints better suited to the volumetric region between the inner- and outer-sphere unions. The efficacy of this strategy is shown in Figure 2.3. This novel *smooth band constraint* is constructed from an implicit function $\phi(\mathbf{y}) = g(f(\mathbf{y}))$ for an arbitrary point $\mathbf{y}$, where f is a standard metaball function and g is a rescaling function.

There are many possible choices for f, but we obtained good results with

$$f(\mathbf{y}) = \sum_{i \in \mathcal{C}} f_i(\mathbf{y})/\psi_i \quad \text{with} \quad f_i(\mathbf{y}) = \exp\left(-a|\mathbf{y} - \mathbf{X}_i|^2\right), \tag{2.5}$$

where ψ_i is the metaball density of the i-th coarse particle and a is a falloff parameter discussed below. The metaball density ψ_i differs from the local density ρ_i^δ in Equation 2.2, and is evaluated as

$$\psi_i = \sum_{j \in \mathcal{C}} D(||\mathbf{x}_i - \mathbf{x}_j||) \quad \text{with} \quad D(z) = \exp\left(-2\ (z/\lambda_c)^2\right). \tag{2.6}$$

Using different kernels for $f_i(\mathbf{y})$ and $D(z)$ provides the flexibility needed to design sufficient constraints. As in Equations 2.1 to 2.3, we sum over local surface point neighborhoods in Equations 2.5 and 2.6 due to the kernel's fall-off. Unlike before, truncation *does* introduce error, but only a negligible amount at a neighborhood radius of $2R$.

Finally, we impose an almost linear variation in ϕ, from 0 at the inner sphere to 1 at the outer sphere, as illustrated in Figure 2.3. We use the following scaling function, which is exact for a single particle:

$$g(z) = \left(\sqrt{-\ln(z)/a} - r\right) \Big/ (R - r). \tag{2.7}$$

We solve for a value of a that, given two coarse particle centers less than $\mu = 3/2 \times R$ units apart, will unify the inner sphere union constraint of the two particles as if their contribution resulted from the same component of the fluid surface:

$$a = \ln\left(2/[1 + D(\mu)]\right) \Big/ \left((\mu/2)^2 - r^2\right). \tag{2.8}$$

Figure 2.3 (bottom middle) is the case with particle centers exactly μ units apart. All our results use this value of a, and its associated μ.

Since our band constraint function ϕ is smooth, we can use its normalized gradient $\mathbf{g}$ to determine a reasonable projection direction when a particle exits the constraint region. Additionally, as the function is approximately linear with respect to the distance from the inner constraint, we can easily compute this projection as

$$\mathbf{x}_i \leftarrow \begin{cases} \mathbf{x}_i - (R - r) \cdot \phi(\mathbf{x}_i) \cdot \mathbf{g}_i & \text{if } \phi(\mathbf{x}_i) < 0 \\ \mathbf{x}_i - (R - r) \cdot (\phi(\mathbf{x}_i) - 1) \cdot \mathbf{g}_i & \text{if } \phi(\mathbf{x}_i) > 1 \\ \mathbf{x}_i & \text{otherwise}. \end{cases} \tag{2.9}$$

This introduces some approximation error, so a surface point may still lie outside the constraint region after projection. However, at this stage, it keeps points sufficiently close to the constraint region.

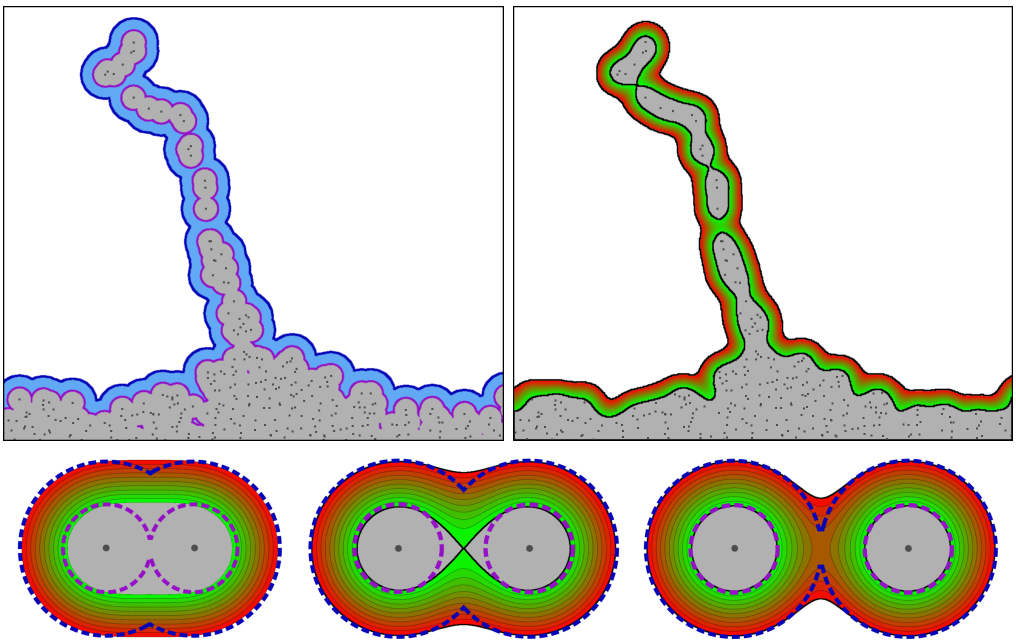


Figure 2.3. Williams' constraint circles (top left, bottom dashed circles) and our smooth band constraint (top right, bottom color gradient). Our constraints are smoother and define values in the interior that vary linearly in space, permitting simpler projections.

2.3.4. Surface Smoothing and Regularization

We now have a high-density set of surface points, but their distribution must be improved (i.e. regularized) before they are suitable for surface wave simulation. This regularization proceeds in four stages: normal computation, normal regularization, tangent regularization, and point insertion and deletion.

Normal Computation: A smooth, artifact-free normal field is essential for the maintenance of our surface structures. We compute the normal at each fine-scale surface point using an averaged least-squares planar fit to the local gradient of ϕ (see Algorithm 2.1).

Regularization Along the Normal: To improve the smoothness of our surface, we displace each surface point along its newly computed normal direction. Given a point i and one of its neighbors j, we consider the plane $\Pi_{i,j}$ spanned by vectors $\mathbf{n}_i$ and $(\mathbf{x}_j - \mathbf{x}_i)$. We find the unique circle in this plane that is equidistant to both points and orthogonal to both points' normals (see Figure 2.4). The projection of point i onto this circle is averaged over all its neighbors, and the point is then displaced along its averaged projected position.

This averaging is done in a neighborhood of radius λ_c, which pushes the points towards a surface that is consistent with the computed normal field. Since the normals vary smoothly, the resulting surface is also smooth. Denoting $\mathbf{n}_j^\star$ the normalized projection of $\mathbf{n}_j$ onto $\Pi_{i,j}$,

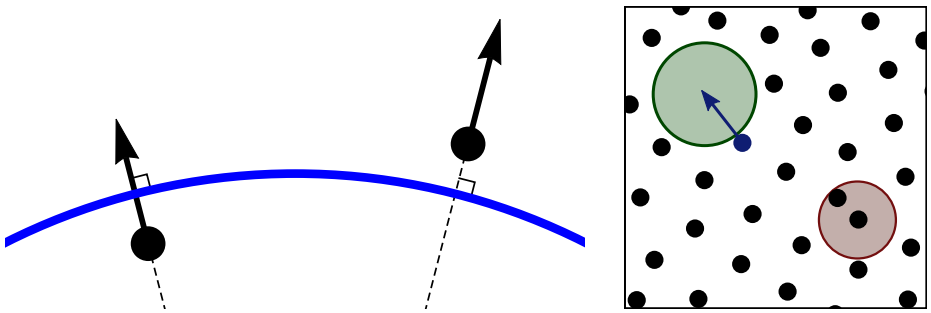


Figure 2.4. Left: Surface regularization shifts points along their normal towards circles consistent with the points' positions and normals, smoothing the surface. Right: A surface point added to a low density region (green) and deleted from a high density region (red).

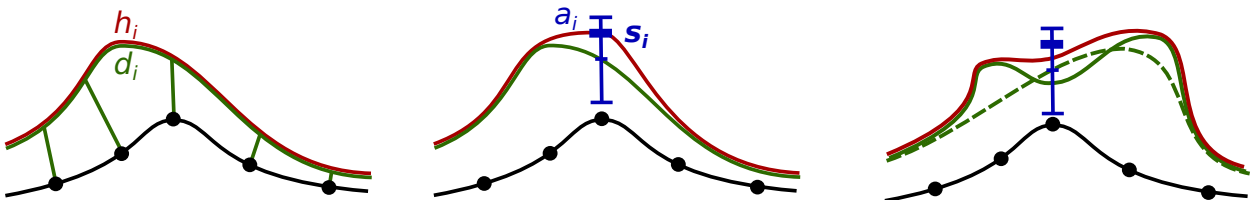


Figure 2.5. Our wave seeding strategy. A wave moving from the left side of the simulation reaches the highlight region of the surface (left); seeding has yet to occur here, so the displayed (d_i, green) and internal (h_i, red) waves match. The underlying surface has high curvature at the center surface point (middle). We increase the oscillator amplitude (a_i) here from 0 to Δa, and compute a wave seeding value (s_i) from a cosine oscillator with amplitude a_i. We evolve the wave simulation and subtract the seeding values from the computed wave to obtain the new displaced wave value (right). The dashed green line shows what the displayed wave would have been if no wave seeding had occurred.

the displacement of point i onto the circle is given by

$$\text{proj}_{i,j} = \mathbf{n}_i \frac{(\mathbf{n}_i + \mathbf{n}_j^\star) \cdot (\mathbf{x}_i - \mathbf{x}_j)}{2\,\mathbf{n}_i \cdot (\mathbf{n}_i + \mathbf{n}_j^\star)}\,, \tag{2.10}$$

so this regularization step is computed as

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \sum_{j \in \mathcal{F}} W_i^{\lambda_c}(\mathbf{x}_j)\,\text{proj}_{i,j}\,. \tag{2.11}$$

Regularization Along the Tangents: Similar to previous works, we insert repulsion forces [73] to improve the distribution of the surface points by driving them along their tangent directions. At each surface point, we compute the weighted direction *away* from its neighbors, and displace it in this direction. This averaging is performed in a local neighborhood λ_f around the point. Explicitly, this regularization step is computed at each surface point as

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + 0.5\,\lambda_f \sum_{j \in \mathcal{F}} W_i^{\lambda_f}(\mathbf{x}_j)\,\text{TN}_i(\mathbf{x}_j - \mathbf{x}_i)\,, \tag{2.12}$$

```
forall surface points i do
    n_i ← g_i;
    compute tangent t_i^1 and bi-tangent t_i^2, orthonormal to n_i;
    least square plane fitting in λ_c to the frame [t_i^1, t_i^2, n_i];
    n_i ← normal of plane;
forall surface points i do
    n_i ← averaged normal in neighborhood λ_c;
    n_i ← normalize (n_i);
```

Algorithm 2.1: Steps for computing the normals.

where TN_i projects onto the tangent plane of point i and then normalizes the result, and the 0.5 factor prevents two points from moving to the same location.

Insertion and Deletion: The last regularization step adds and deletes surface points according to changes in the underlying simulation. Points are deleted when the local point density is too high. We detect this by looking for pairs of points that are closer than $3/4\,\lambda_f$, in which case the most recently created point is deleted. Similarly, surface points are added when the local point density is too low, by Poisson disk sampling [32]. For each point, we recompute the tangential direction as it was computed in the previous tangent regularization step. This generally points in the direction of lowest density, so we place a sphere of radius λ_f at a distance λ_f in this direction and search for neighboring points that fall inside the sphere. If none are found, the point density is too low and we create a point at the displaced sphere's center (Figure 2.4).

The three regularization steps are global operations that influence the *entire* surface. However, as they each consist of spatially local computations, we apply them several times to each animation frame until convergence. We used 30 iterations for the first frame and between 3 and 10 iterations per subsequent frame in all of our simulations. The number of iterations used for each scene is shown in Table 2.1. The accompanying video shows the uniform point density maintained by our surface operations on a deforming fluid surface.

2.3.5. Interactions with Obstacles

We have so far focused on the management of surface points in unobstructed flow, but special care must be taken when surface points approach obstacles. The regularization in Section 2.3.4 works best if the point distribution around each surface point is approximately uniform, which is not the case near obstacle boundaries.

Motivated by ghost and boundary [63, 21] particles in SPH, we add ghost points for each surface point located close to an obstacle by reflecting the neighboring points of the current point w.r.t. the obstacle. We then simply apply the regularization steps to both "real" and ghost point sets.

2.4. Turbulence Creation and Evolution

Section 2.3 detailed the creation of high-density point sets that represent the underlying coarse fluid simulation surface. We can now add turbulent details atop this surface, simulating waves on its points.

2.4.1. Curvature Evaluation

Surface wave details should appear in areas of high activity, e.g., merging or separating regions. Mean curvature is a good indicator of these regions [44], as high curvature tends to denote under-resolved areas in the base simulation. Second-order neighborhood fitting is commonly used to compute curvature on point surfaces: this works well for smooth surfaces [75], but we deal with turbulent surfaces where quadratic fits can generate extreme and noisy curvatures, leading to instabilities over time.

We instead propose a new, robust alternative for evaluating curvature on point clouds. The curvature c_i at point i is defined as the signed distance of its neighbors to its tangent plane, easily obtained from the normal computed at each point (Section 2.3.4), which gives:

$$c_i = \sum_{j \in \mathcal{F}} W_i^{\lambda_c}(\mathbf{x}_j) \left(\mathbf{n}_i \cdot (\mathbf{x}_i - \mathbf{x}_j)\right). \tag{2.13}$$

While not identical to mean curvature, this measure is a very reliable criterion for wave generation. Moreover, thresholding any such criterion in a meaningful way across discretizations is very important. We derive practical thresholds $\{c_{\min}, c_{\max}\}$, evaluating Equation 2.13 at two extremal scenarios: single drops and thin sheets. For a surface of infinite point density at distance λ_c from the coarse particles, our measure evaluates to $c_{\max} = 0.15\,\lambda_c$ for a single drop and $c_{\min} \approx 0.077\,\lambda_c$ for a thin sheet. These thresholds (see Appendix A, Section 2.7 for derivations) are useful for parametrizing simulations.

2.4.2. Turbulence Creation

The simplest way to add turbulence to a surface is to add it directly to the wave heights. This can work well for grid-based methods [44] where surface curvature varies smoothly with respect to grid size, but particle-based methods often contain large, abrupt curvature variations. For instance, when a single particle falls onto a flat water surface, a discontinuous wave seeding causes an abrupt visual change in height. Curvature is always high on isolated droplets, causing uniform wave seeding over the droplet and unrealistic (non mass conserving) pulsing (see supplemental video).

A simple solution used in mesh-based simulations [78] is to turn off the wave seeding in regions of high curvature. This reduces artifacts from curvature discrepancies, but still produces abrupt changes in regions of near-maximum curvature. Moreover, it removes some

Data:

c_i	: surface curvature	c	: wave speed
$c_{\min,\max}$	: curvature thresholds	f_b	: base seeding frequency
a_i	: oscillator amplitude	f_o	: # of frequency octaves
Δa	: oscillator amplitude step size	W	: max. wave amplitude
A	: max. oscillator amplitude	F	: max. wave frequency
h_i	: internal wave height	t	: simulation time
v_i	: internal wave velocity	Δh_i	: laplacian
d_i	: displayed wave height	s_i	: seed value

```
1  forall surface points i do
2      a_tmp ← 2 smoothstep(|c_i|, c_min, c_max) − 1;
3      a_i ← clamp(a_i + a_tmp Δa,0,A);
4      s_i ← 0; f ← f_b; a ← a_i;
5      for λ = 1 .. f_o do
6          s_i ← s_i + a_i cos(t c f); f ← 2 f; a ← a/2;
7      h_i ← d_i + s_i;
8  forall surface points i do
9      v_i ← v_i + c² Δt Δh_i; h_i ← h_i + Δt v_i; d_i ← h_i − s_i;
10     d_i ← clamp(d_i, −W, W); v_i ← clamp(v_i, −W F, W F);
```

Algorithm 2.2: Pseudocode for wave evolution and seeding.

of the waves caused by fine splashes characteristic of particle-based fluids, and limits the amount of new added details.

We propose a different approach, outlined in Algorithm 2.2 and illustrated in Figure 2.5. In lines 2 and 3, we throttle seeding in high curvature regions based on the curvature thresholds of Section 2.4.1. We seed with time-varying cosine oscillators (line 6), but these oscillations are never directly visualized. Instead, we apply the wave equation to them and only new waves that have *propagated out* from the cosine oscillations are visualized. This avoids double accumulation of wave values in regions of high curvature: once from their own oscillator and once from the waves generated by neighboring oscillators. This also causes waves to appear at the boundary of the seeding regions: for an isolated droplet, the seeding region has no boundary and, as no wave is seeded, the pulsing artifact is removed.

Our seeding is easy to implement: in addition to the wave value h_i used to solve the wave equation, we store a *displayed wave value* d_i. We store cosine oscillators separately in *seed values* s_i. At each step, we first add seed values to the displayed wave values to obtain the wave values. We perform wave simulation on h_i and subtract seed values from h_i to recover d_i. The d_i thus only contain new features propagated out from the seed values, in addition to features that have evolved from previous timesteps. We never visualize h_i.

Finally, inspired by smoke up-res methods [45], our approach seeds features across multiple frequency octaves. To enable further control, the base seeding frequency f_b of the

cosine oscillators, and the number of additional octaves f_o, are exposed as user parameters. The accompanying video shows a comparison of our seeding method versus more direct approaches.

2.4.3. Turbulence Evolution

We evolve waves on the surface using the standard wave equation,

$$\partial_t^2 h = c^2 \, \Delta h \,, \tag{2.14}$$

where c is the wave speed and Δ is the Laplace-Beltrami operator. We solve this equation explicitly with a symplectic Euler scheme that also requires the wave velocity v_i to be stored at each surface point. The linear wave equation, due to its dispersive nature, only approximates capillary surface wave behavior. The wave speed c must thus be selected empirically, for instance by estimating the desired traveled distance of waves between two given frames. Despite this, the linear wave model is widely used [72, 31] and yields convincing results. Similar to these works, we add a small amount of diffusion to approximate wave dissipation, but this is not required to maintain stability.

Some previous works solve differential equations involving the Laplace-Beltrami operator on point surfaces [48, 49], requiring the evaluation of surface curvature as well as second derivatives of the embedded function. In our case, the curvature of the underlying surface is small compared to the length λ_f at which the wave equation operates, so we instead approximate the Laplace-Beltrami operator in Equation 2.14 with a simpler, flat Laplace operator. We avoid the metric tensor computations of Laplace-Beltrami, which are both costly and difficult to robustly evaluate on point surfaces. Figure 2.6 compares our flat

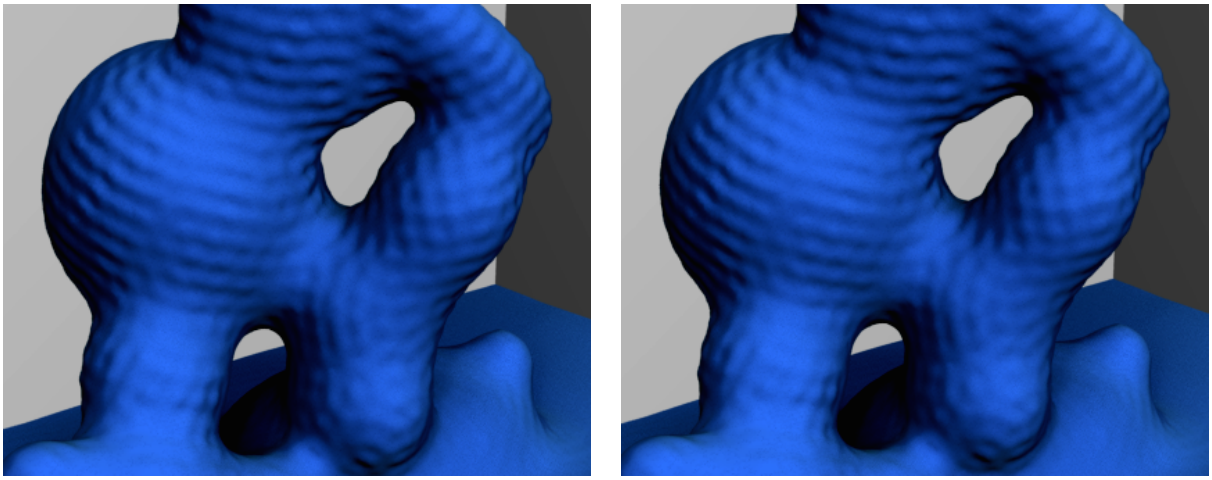


Figure 2.6. Wave propagation with Laplace-Beltrami (left) and our flat Laplace operator (right). The approximation error is negligible: no observable differences even after 1000 simulation steps.

Laplace operator to Laplace-Beltrami, showing results that are indistinguishable even after 1000 simulation steps. Here, the approximation was 6 times faster to compute and gave a maximum height difference of only 1.4%. Appendix B (Section 2.8) further validates our approximation.

A common method [49] for computing the flat Laplace operator on a point surface is to use the derivatives of a local quadratic least squares approximation of the function. Although this works with densely sampled surfaces, we generate waves at scales that can be of the same order as the distance between points. Only a few points can then be used for the quadratic approximation, which is imprecise and leads to instabilities over time. Our supplemental video and Figure 2.7 illustrate such instabilities.

We instead compute the Laplace operator by using the tangent plane, which was previously obtained during surface normal computation (section 2.3.4). By projecting nearby points onto the tangent plane, the displacement defined by the wave values h_i becomes a function defined on the tangent plane. In this coordinate system, an affine approximation P of the function is first computed using standard least-square minimization. Subtracting the values of P on each neighboring point eliminates the zeroth- and first-order derivatives of the function, so the Laplacian can be evaluated directly as a weighted sum of discrete directional second-order derivatives:

$$\Delta h_i = \sum_{j \in \mathcal{F}} W_i^{2\lambda_f}(\mathbf{x}_j) \, \frac{4\left((h_j - P(\mathbf{x}_j)) - (h_i - P(\mathbf{x}_i))\right)}{||\mathbf{x}_i - \mathbf{x}_j||^2} \, . \tag{2.15}$$

We have found that a neighborhood radius size of $2\lambda_f$ works well. Appendix B (Section 2.8) shows how our operator approximates the Laplacian. To our knowledge, we are the first to introduce this discrete operator for computing the Laplacian on a meshless set of points. Our supplemental video and Figure 2.7 show that it matches computations using the usual quadratic least squares approximation. Furthermore, since it uses an affine least squares fit instead of a quadratic fit, our operator remains stable even for waves at the highest frequency representable by the surface points. As such, it is particularly well suited to our meshless point representations, and we observed it was 2 times faster to evaluate than the quadratic least squares Laplacian.

We inject ghost points when surface points approach obstacles (Section 2.3.5) and also use these ghost points during wave computations, where we simply copy the wave value on a ghost point from its original "real" surface point. This gives Neumann boundary conditions on the obstacles and yields the expected wave reflections.

2.5. Results and Discussion

We apply our method to coarse simulations generated using Houdini 13's FLIP solver. We apply our method to a large 12.5 million particles input simulation (Figures 2.1 and

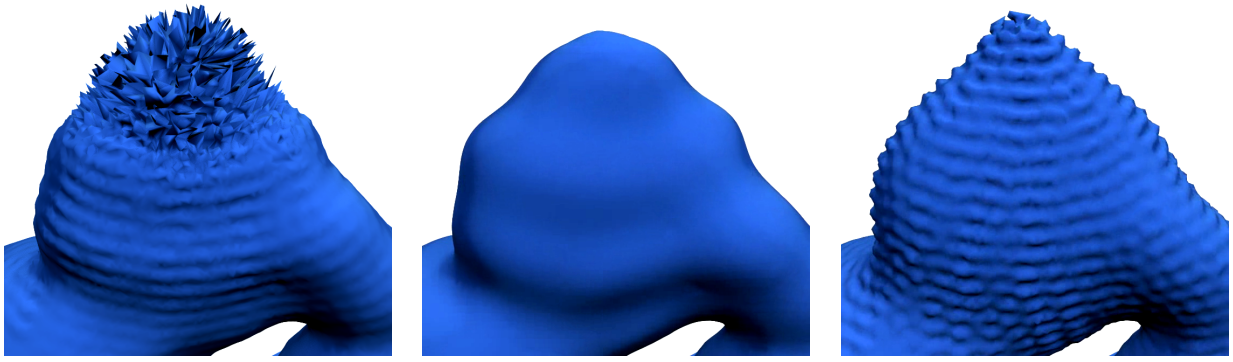


Figure 2.7. Comparing the Laplacian computed with a least squares fit to our new discrete operator. When generating waves at a scale near the point density limit, the least squares fit fails to isolate the desired wavelength and becomes unstable (i.e., undesirable small scale waves, left). At the same wave frequency, our discrete Laplace operator correctly treats the wave (middle) and even remains stable when pushed to the Nyquist limit of the discretization (right).

Scene	# particles	# surface	Total	Advection	Regularization	Curvature	Laplacian	Wave	Disk I/O
Dam Break	12500k	500k	85.4	8.6%	53.1% (10)	5.4%	2.9%	0.3%	2.0%
River	400k	280k	42.2	14.7%	48.7% (3)	9.9%	10.8%	0.8%	15.1%
Double Drop	1400k	350k	25.9	8.1%	57.0% (5)	5.4%	7.9%	1.0%	4.9%
Stir	390k	17k	1.5	15.0%	49.9% (5)	6.7%	4.1%	1.1%	4.6%
	390k	66k	5.4	10.8%	58.1% (5)	7.5%	6.0%	0.8%	4.8%
	390k	145k	15.2	8.7%	61.0% (5)	7.0%	9.8%	0.6%	3.9%

Table 2.1. Timings for the various steps of our algorithm. Total times are given in seconds per frame. All performance statistics were computed on an Intel i7 quad core running at 3.4 GHz with 32GB of RAM. In the regularization column, the parenthesis indicate the number of regularization steps used per frame.

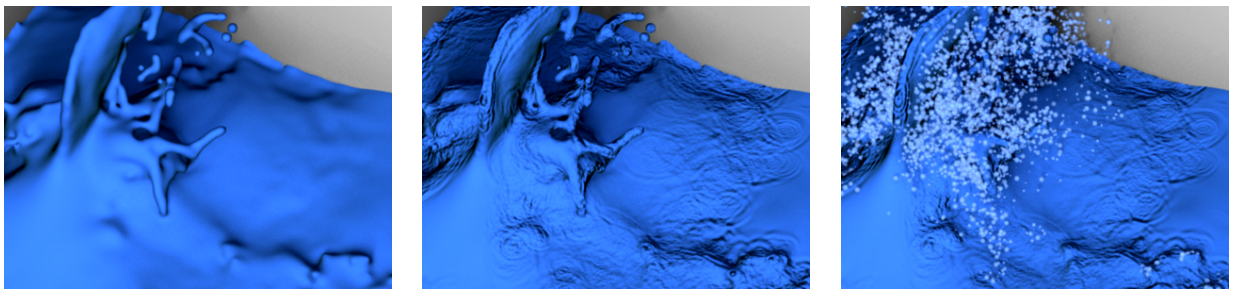


Figure 2.8. We upres an input simulation (left, 1.4 million particles) with 400K surface points (middle), augmenting details over the bulk of the surface. We can also combine our results with other methods (i.e., [40]) to further increase the visual fidelity (right).

2.12). Even at this high resolution, we enhance the visual quality by generating waves on a 500K surface point representation. Our complete post-process requires 85.4s/frame, compared to the 241s/frame used for the input simulation. This illustrates our scalability beyond resolutions achievable with regular fluid solvers.

Figure 2.9 features a complex moving obstacle [47], and demonstrates various levels of upresing: from a 390K input coarse particle simulation, we generate 17K, 66K and 145K surface points. Each successive point set is able to resolve higher frequency details corresponding to successively higher visual fidelity. In contrast to previous work [44], our method does not require any additional information apart from the points used to represent the final surface to simulate a similar amount of wave detail.

Figure 2.11 shows a complex, turbulent riverbed. Even at 400K particles, the coarse simulation cannot capture the intense turbulence that characterizes a river's flow. Our upresed output conveys this imagery, adding surface waves both where the water collides with rocks, as well as in stationary eddies behind these same obstacles.

Comparison with Full Resolution FLIP. Figure 2.10 compares the results of a high-resolution 4 million particles FLIP simulation with our method applied to a low resolution simulation comprising just 2500 particles. The 4 million particles are able to resolve certain fine structures, such as the splash, however, the final surface only contains rough, low frequency waves. In comparison, our method crisply resolves waves with many more frequencies. The 4 million particles scene requires 142s/frame while ours uses only 4.74s/frame, corresponding to a 30× speedup. Obtaining comparable waves with only a FLIP simulation would require even more particles, increasing our speedup for an equal-quality wave motion.

Implementation and Performance. Our method relies heavily on surface point lookups in small neighborhoods, so it is crucial to use an acceleration structure to store the surface point data. We used a hashing structure [69] to improve the efficiency of these operations, yielding a speedup in the range of 20× for 27K surface points to 200× for 290K surface points compared to brute-force lookups. Most of the computations are performed on individual surface points, so trivial parallelizations using OpenMP further accelerated these operations by another 4 to 8×. Full computation breakdowns for our four scenes are provided in Table 2.1.

Limitations. Since we are designing an upres technique, we deliberately leave the coarse dynamics of the simulation untouched. Consequently, we do not reduce the size of the smallest underlying simulation structures. While this does not affect the majority of a simulation, fine isolated structures (i.e., droplets) are limited by the size of the input simulation. As mentioned in Section 2.4.2, seeding waves on isolated particles leads to visible mass loss and undesirable behavior, so we leave them untouched. However, our method remains compatible with techniques that directly address this problem, such as Ihmsen et al.'s approach [40] (see Figure 2.8).

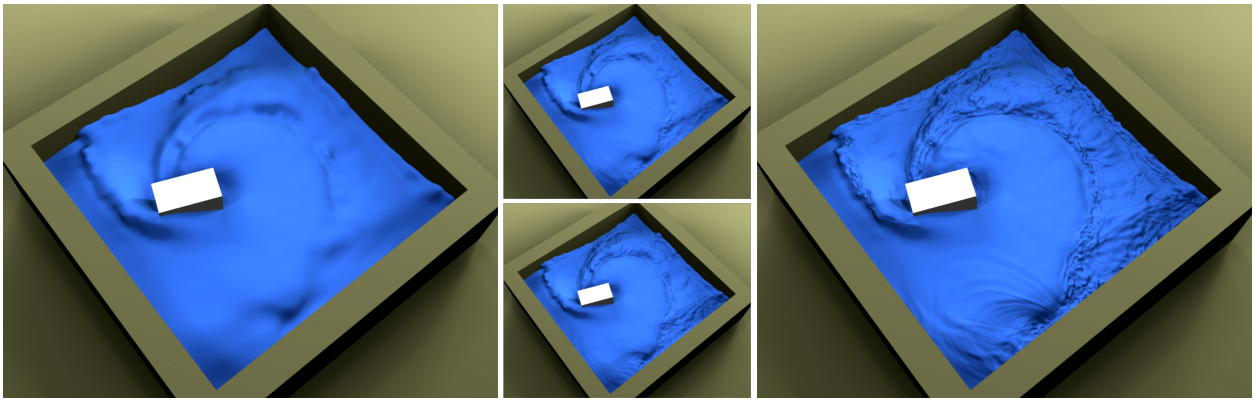


Figure 2.9. We upres a 380K FLIP simulation (left) with 17K (middle, top), 66K (middle, bottom), and 145K (right) surface points.

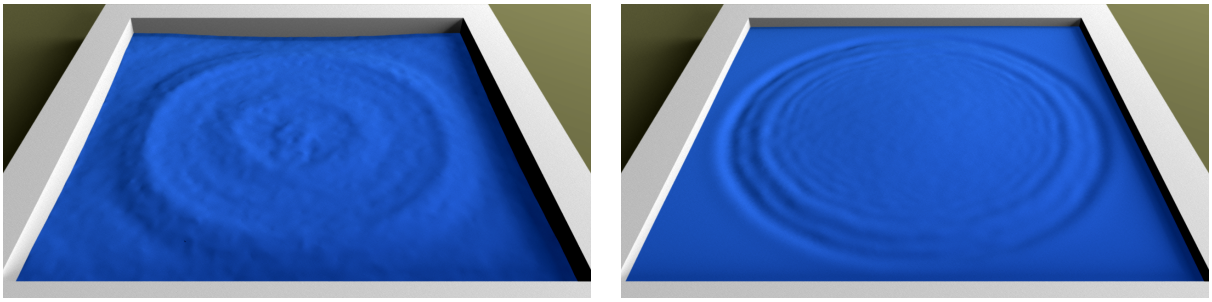


Figure 2.10. Comparing a very high resolution (left; 4 million particles) and low resolution FLIP simulation up-resed with our method (right; 2500 coarse particles). The high resolution simulation only contains shallow, low frequency surface waves. Our result is up-resed to 15500 surface points, yielding much crisper surface waves.

The timings in Table 2.1 show the majority of our computation is spent in surface regularization and neighborhood queries. This is as expected since the λ_c scale used for normal evaluation and regularization (Section 2.3.4), as well as for curvature computations (Section 2.4.1), does not decrease as the number of surface points increases. The number of neighbors thus grows quadratically with the total number of surface points. However, there are redundancies in these queries, since they all relate to the same underlying coarse simulation regardless of the final point count. A hierarchical method designed to instead select a constant-sized subset of representative neighbors for use in these queries would reduce the complexity of the regularization steps to that of all other steps. Designing such a selection method is a natural direction for future work.

2.6. Conclusion

We have presented a fully Lagrangian method for enhancing a particle-based liquid simulation using surface waves. This was made possible using a combination of several novel

techniques, including a robust method for point surface creation and maintenance and a stable discrete Laplace operator, both of which can apply more broadly in settings involving surface operations on animated point sets. We also proposed a novel wave injection strategy based on bands of oscillators, and we have demonstrated that our method can efficiently process coarse input simulations (of both low- and high-resolutions) into highly detailed and turbulent liquid surfaces.

In the future, we plan to explore more complex and expressive waves models, art-directable editing controls for the fine-scale details, closer coupling to secondary particle systems for drops and foam effects, and the application of our method to other types of secondary surface simulations for Lagrangian data.

2.7. Appendix A: Threshold Computations

To evaluate our curvature measure on a surface of infinite point density, we consider the integral form of (2.13) as $|\mathcal{S}| \to \infty$. Here, surface points all have equal density ρ, so we can replace W with K. We omit the neighborhood size λ_c from the kernel notation for brevity.

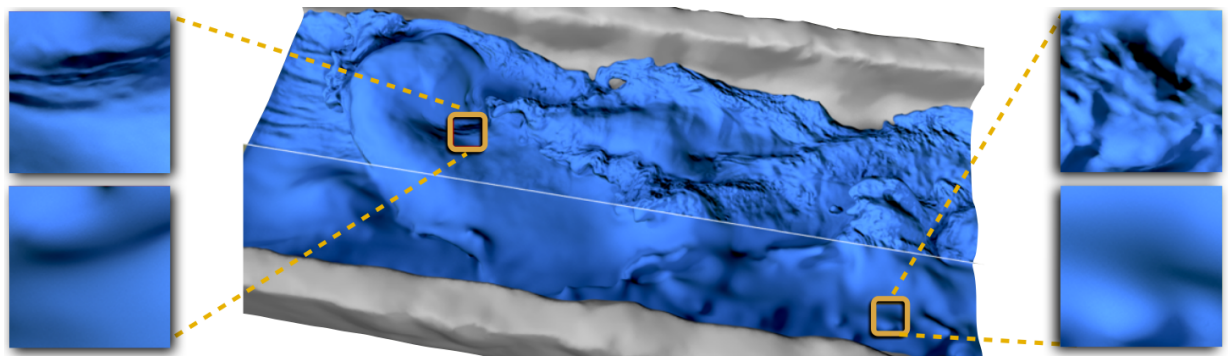


Figure 2.11. We upres an input 400K particle FLIP simulation (middle bottom half; zoom-ins bottom) with 280K surface points (middle top half; zoom-ins top). Our surface waves interact realistically with the turbulent flow over the rocks and the resulting stationary eddies.

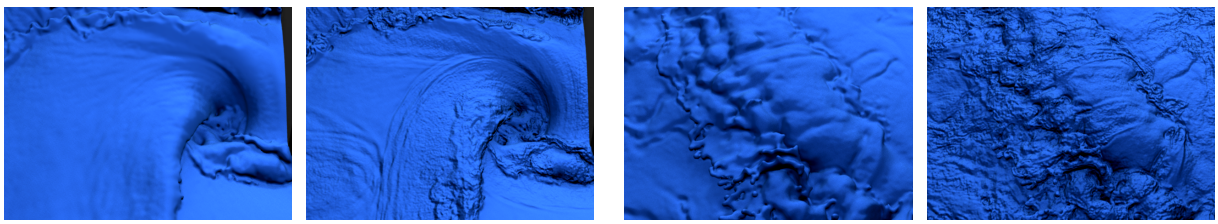


Figure 2.12. Two close-ups of the Dam Break scene (Figure 2.1). In each pair, the left image shows the input simulation and the right image shows our upresed output surface. Even with 12M input particles, the input simulation fails to resolve finer surface details, so our method is still capable of significantly increasing the visual quality of the result even with high-resolution inputs.

We work in a local frame with the surface point of interest at $(0, \lambda_c, 0)$ with normal (0,1,0) and xz-tangent plane, yielding

$$c_i = \int\limits_{\mathbf{x}\in S} K((0,\lambda_c,0) - \mathbf{x}\cdot\mathbf{e}_y)\,(\lambda_c - y)\,dS \Big/ \int\limits_{\mathbf{x}\in S} K((0,\lambda_c,0) - \mathbf{x})\,dS\,.$$

For the case of a single drop, we consider the surface $\{x^2+y^2+z^2 = \lambda_c^2\}$. Solving the integral analytically in spherical coordinates $p(\theta,\phi) = (\lambda_c \sin\theta\cos\phi, \lambda_c\cos\theta, \lambda_c\sin\theta\sin\phi)$ yields

$$c_i = \frac{\displaystyle\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/3} \lambda_c\,(1-\cos(\theta))\,K((0,\lambda_c,0) - p(\theta,\phi))\,d\theta\,d\phi}{\displaystyle\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi/3} K((0,\lambda_c,0) - p(\theta,\phi))\,d\theta\,d\phi} = \frac{3\lambda_c}{20}.$$

For the case of a thin sheet, we consider the surface

$$(\{y \geq 0\} \cap \{x^2+y^2 = \lambda_c^2\}) \cup (\{y \leq 0\} \cap \{|z| = \lambda_c\}).$$

Since K is non-zero only in the cylindrical part of the thin sheet, we can use cylindrical coordinates $p(x,\theta) = (x, \lambda_c\cos\theta, \lambda_c\sin\theta)$:

$$c_i = \frac{\displaystyle\int_{x=-\lambda_c}^{\lambda_c}\int_{\theta=-\pi/3}^{\pi/3} \lambda_c\,(1-\cos(\theta))\,K((0,\lambda_c,0) - p(x,\theta))\,\lambda_c\,dx\,d\theta}{\displaystyle\int_{x=-\lambda_c}^{\lambda_c}\int_{\theta=-\pi/3}^{\pi/3} K((0,\lambda_c,0) - p(x,\theta))\,\lambda_c\,dx\,d\theta}\,,$$

which evaluates numerically to $0.0771413\,\lambda_c$ (to double precision).

2.8. Appendix B: Laplace-Beltrami Approximation

We justify our Laplace approximation and detail computations used for Figures 2.6 and 2.7.

2.8.1. Flat Laplace Computation

First, we show that (2.15) indeed approximates the Laplace operator at surface point i. We orient the coordinate system to have origin at point i and tangent xy-plane. We express $W_i^{2\lambda_f}$ recentered at the origin as W, and override $\mathcal{F}$ as the neighboring points in those coordinates. We can rewrite the discrete operator as

$$\begin{aligned}
&\sum_{\mathbf{x}\in\mathcal{F}} 4\,W(\mathbf{x})((h(\mathbf{x}) - P(\mathbf{x})) - (h(\mathbf{0}) - P(\mathbf{0}))\Big/||\mathbf{x}||^2 \\
&= \sum_{\mathbf{x}\in\mathcal{F}} 4\,W(\mathbf{x})(h(\mathbf{x}) - P(\mathbf{0}) + \nabla P(\mathbf{0})\cdot\mathbf{x} - h(\mathbf{0}) + P(\mathbf{0}))\Big/||\mathbf{x}||^2 \\
&= \sum_{i\in\mathcal{F}} 4\,W(\mathbf{x})(h(\mathbf{x}) - \nabla h(\mathbf{0}) + O(\lambda_f^2)\cdot\mathbf{x} - h(\mathbf{0}))\Big/||\mathbf{x}||^2 \qquad (2.16)\\
&= \sum_{\mathbf{x}\in\mathcal{F}} \Big(4\,W(\mathbf{x})(h(\mathbf{x}) - \nabla h(\mathbf{0})\cdot\mathbf{x} - h(\mathbf{0}))\Big/||\mathbf{x}||^2\Big) + O(\lambda_f) \qquad (2.17)
\end{aligned}$$

where the error term in (2.16) results from the superconvergence demonstrated in Appendix A in Liang's work [49], and is possible due to the approximate point distribution symmetry maintained by our method (Section 2.3.4). As the point density approaches infinity, the operator converges to

$$\iint_{\mathcal{D}} 4\,K(\mathbf{x})(h(\mathbf{x}) - \nabla h(\mathbf{0}) \cdot \mathbf{x} - h(\mathbf{0}))\Big/||\mathbf{x}||^2\,d\mathbf{x} + O(\lambda_f) \tag{2.18}$$

where $\mathcal{D}$ is the disk $x^2 + y^2 \leq (2\lambda_f)^2$ and K is the kernel $K_i^{2\lambda_f}$ recentered at the origin and normalized to 1. Again, note that (2.18) is only correct if the point density of the surface is uniform, which we maintain in our method. Using a Taylor expansion of h in the direction $\mathbf{x}$, we have $h(\mathbf{x}) = h(\mathbf{0}) + ||\mathbf{x}||h'_{\mathbf{x}}(\mathbf{0}) + \frac{1}{2}||\mathbf{x}||^2 h''_{\mathbf{x}}(\mathbf{0}) + O(||\mathbf{x}||^3)$, where $h'_{\mathbf{x}}$ and $h''_{\mathbf{x}}$ are the first and second directional derivatives of h w.r.t. $\mathbf{x}$. Substituting the Taylor expansion into (2.18), we arrive at

$$\begin{aligned}
&= \iint_{\mathcal{D}} 4\,K(\mathbf{x})\left(\frac{1}{2}||\mathbf{x}||^2 h''_{\mathbf{x}}(\mathbf{0}) + O(||\mathbf{x}||^3)\right)\Big/||\mathbf{x}||^2\,d\mathbf{x} + O(\lambda_f) \\
&= \iint_{\mathcal{D}} 2\,K(\mathbf{x}) h''_{\mathbf{x}}(\mathbf{0})\,d\mathbf{x} + O(\lambda_f). \qquad (2.19)
\end{aligned}$$

The error term disappears since λ_f approaches zero as the point density increases. We split (2.19), effect a clockwise rotation of 90^{deg}, and use the symmetry of K to obtain

$$\begin{aligned}
&= \iint_{\mathcal{D}} K(\mathbf{x}) h''_{\mathbf{x}}(\mathbf{0})\,d\mathbf{x} + \iint_{\mathcal{D}} K(\mathbf{x}) h''_{\mathbf{x}}(\mathbf{0})\,d\mathbf{x} && (2.20) \\
&\approx \iint_{\mathcal{D}} K(x,y) h''_{(x,y)}(\mathbf{0})\,d\mathbf{x} + \iint_{\mathcal{D}} \underbrace{K(-y,x)}_{=K(x,y)} h''_{(-y,x)}(\mathbf{0})\,d\mathbf{x} && (2.21) \\
&= \iint_{\mathcal{D}} K(x,y)\left(h''_{(x,y)}(\mathbf{0}) + h''_{(-y,x)}(\mathbf{0})\right)d\mathbf{x} && (2.22) \\
&= \iint_{\mathcal{D}} K(x,y)\Delta h(\mathbf{0})\,d\mathbf{x} = \Delta h(\mathbf{0}) \iint_{\mathcal{D}} K(x,y)\,d\mathbf{x} = \Delta h(\mathbf{0}) && (2.23)
\end{aligned}$$

where the equality between (2.22) and the left most equation in (2.23) leverages the invariance of the Laplacian under rigid deformation.

2.8.2. Flat Surface Approximation

We can now justify our claim in Section 2.4.3 that locally approximating the curved surface with a flat surface yields negligible errors in the evaluation of differential quantities. We parameterize a quadratic surface q about a given surface point as

$$q(x,y) = q_{00} + q_{10}x + q_{01}y + q_{20}x^2 + q_{11}xy + q_{02}y^2. \tag{2.24}$$

Following [49], we compute a least square quadratic approximation centered at the surface point for both the surface (f) and the wave function (h) to approximate the Laplace-Beltrami

operator at this point. The full expression for the operator can be derived as

$$\frac{\begin{pmatrix} 2\left(1+f_{01}^4+2f_{01}^2+f_{10}^2+f_{10}^2f_{01}^2\right)h_{02} \\ +2\left(1+f_{10}^4+2f_{10}^2+f_{01}^2+f_{10}^2f_{01}^2\right)h_{20} \\ +\begin{pmatrix} f_{20}f_{01}^3+3f_{01}^3f_{02}+3f_{01}f_{02} \\ +2f_{10}^2f_{01}f_{02}+f_{01}f_{20}+2f_{10}^2f_{01}f_{20} \\ +2f_{10}f_{11}+f_{10}^3f_{11}+3f_{10}f_{01}^2f_{11} \end{pmatrix}h_{01} \\ +\begin{pmatrix} f_{02}f_{10}^3+3f_{10}^3f_{20}+3f_{10}f_{20} \\ +2f_{10}f_{01}^2f_{20}+f_{10}f_{02}+2f_{10}f_{01}^2f_{02} \\ +2f_{01}f_{11}+f_{01}^3f_{11}+3f_{10}^2f_{01}f_{11} \end{pmatrix}h_{10} \\ +\left(2f_{10}f_{01}+2f_{10}^3f_{01}+2f_{10}f_{01}^3\right)h_{11} \end{pmatrix}}{1+f_{01}^2+f_{10}^2} \tag{2.25}$$

Notice that if we align our local coordinate system with the surface, i.e., $f_{10} = f_{01} = 0$, (2.25) simplifies to $2h_{20} + 2h_{02}$, which is the flat Laplace operator. While our normal computation in Algorithm 2.1 attempts to get as close as possible to this perfect alignment, numerical errors will be present. Still, Figure 2.6 shows that f_{10} and f_{01} are small enough to permit our approximation with a flat Laplacian. Moreover, by only keeping the first-order terms of (2.25), i.e., ignoring terms that depend at least quadratically on f_{10} and f_{01}, we arrive at the first-order approximation

$$\begin{aligned} 2h_{02}+2h_{20}+&\left(3f_{01}f_{02}+f_{01}f_{20}+2f_{10}f_{11}\right)h_{01} \\ &+\left(3f_{10}f_{20}+f_{10}f_{02}+2f_{01}f_{11}\right)h_{10}. \end{aligned} \tag{2.26}$$

We apply this expression in Figure 2.6 when computing the Laplace-Beltrami operator, where $(2h_{02} + 2h_{20})$ is evaluated using (2.15) and the wave function derivatives h_{10} and h_{01} are evaluated using the affine approximation P already computed in (2.15).

2.C. Extension to Viscous Wrinkling

One extension of our method is the visual improvement of highly viscous fluid simulations, such as mud or tar. While the detail augmentation of liquid simulations presented in the paper mandates surface waves travelling on the surface, highly-viscous fluids do not naturally support such waves.

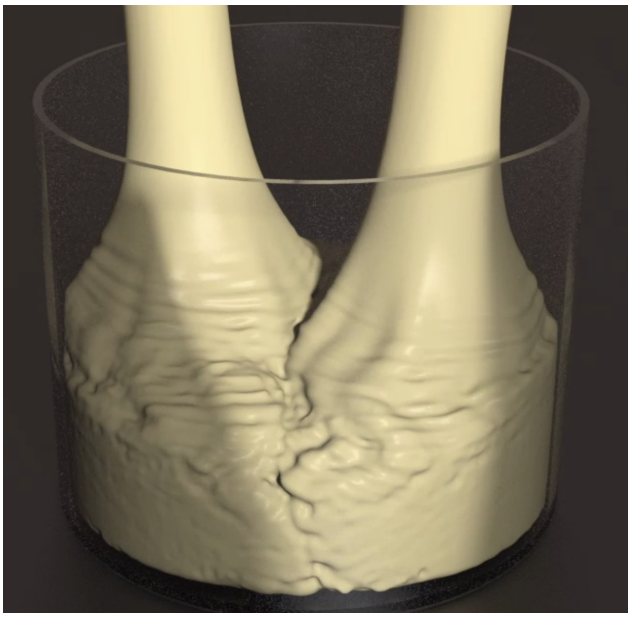


Figure 2.13. SPH simulation from Peer et al. [61]. This full-scale viscous simulation creates wrinkles as the fluid surface compresses. Our objective is to recreate this effect as a post-process.

Figure 2.13 shows results from a high-resolution particle-based viscous simulation form the work of Peer et al. [61], where a viscous fluid is poured into a cylindrical contained. As the fluid contacts with the boundaries of the container, it is compressed, and buckling effects create wrinkles at its surface. These wrinkles are key to the high resolution appearance of the fluid. They remain static mostly after creation and are only passively advected with the underlying simulation, slowly dissipating over time.

We investigate the use of our surface upres method to recreate the wrinkling behavior as a post process on a coarse viscous fluid simulation. We reuse the same surface construction method as described in Section 2.3, but devise a new surface displacement model that better fits the wrinkling behavior. To our knowledge, such a viscous upres method has never been successfully developed before. This work was done in collaboration with Theodore Kim of Pixar Animation Studios.

2.C.1. Wrinkle Model

Our viscous upres method is based on the model of Figure 2.14. As with the method presented in the paper, we assume the surface we construct around each input particle set is static, and add details by moving the high-resolution points in the directoin normal to the surface. On top of that surface, we add a thick viscous fluid layer covered by a flexible membrane. This construction is an active research area in physics [39, 37, 30, 46], since the complex interaction between the three layers (membrane, fluid, solid) leads to regular but difficult-to-predict wrinkling patterns. Simulating this system accurately leads to large systems of differential equations with high-order derivatives of the membrane height (sometimes up to order 6). Our surface construction can provide stable second order derivative evaluations (as was used in the wave simulation of Section 2.4), but higher order derivatives are difficult to robustly compute on our point-based surface. Moreover, most of these models allow for tangential displacements of the layers, e.g., to model compression, which our surface cannot represent since we only allow for normal displacements. Surface wrinkling has also been investigated in the graphics community, for instance for cloths [25, 54] or viscous fluid sheets [24], but these methods mostly work with mesh representations. The wrinkling mechanisms of these methods are also an integral part of the simulation, and are not necessarily applicable as a post-process.

For these reasons, we apply the layered model of Figure 2.14 as a reference but we greatly simplify its dynamics. We suppose the viscous layer is thick enough so the membrane and the solid surfaces do not intersect. We use the notation of Section 2.4.3 and denote h_i the normal displacement of the membrane on surface point i with respect to its rest position.

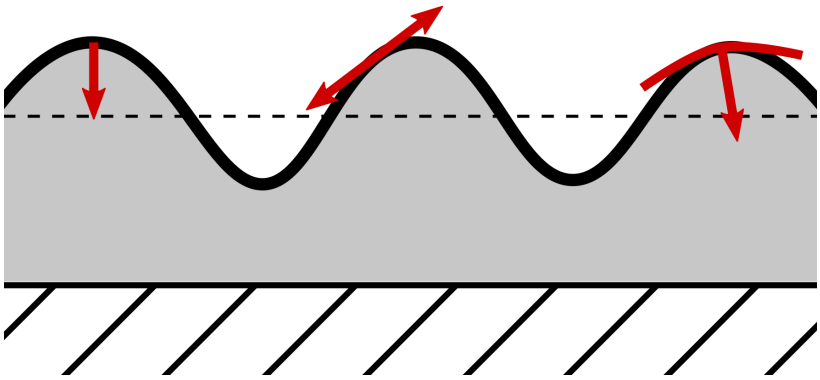


Figure 2.14. Viscous wrinkle model. The underlying simulation (dashed region) is assumed static for each given frame. Over the surface, a thick elastic membrane (curved line) is placed on top of a viscous fluid layer (gray region). The surface can be displaced vertically with respect to a rest state (dashed line), leading to three restitution forces (red arrows) caused by the deformation of the viscous layer (left), the stretching of the surface (middle) and the curvature of the surface (right).

Note that during rendering, we only apply the displacement h to the surface, ignoring the thick viscous layer.

From a given time step, the membrane is advected passively with the underlying simulation, and might be compressed or stretched at the following time step. In this case, we model three forces acting on the membrane, depicted as red arrows in Figure 2.14:

- an elastic force caused by the stretching of the viscous layer, which we assume reacts as a deformable solid under Hook's law,
- a stretching force caused by the compression or stretching of the membrane, and
- a force caused by the bending of the membrane.

We define the elastic and bending forces as $-\alpha_h h_i$ and $\alpha_\Delta \Delta h_i$, respectively, for some coefficients α_h and α_Δ.

The stretching force is more complicated to model. Our attempts at modelling these forces with laplacian or bilaplacian operators led to methods that were rather unstable or required prohibitively small time steps. We instead devise a method based on compressions and stretching of the surface. Let $A^{(t)}$ be the total area of the membrane at time step t. We define the stretching coefficient c as

$$A^{(t-1)} = (1 + c)A^{(t)}, \tag{2.27}$$

so c is the proportion by which the membrane must be stretched to recover the area of the previous time step. We use it to weight the strength of the stretching force on the membrane.

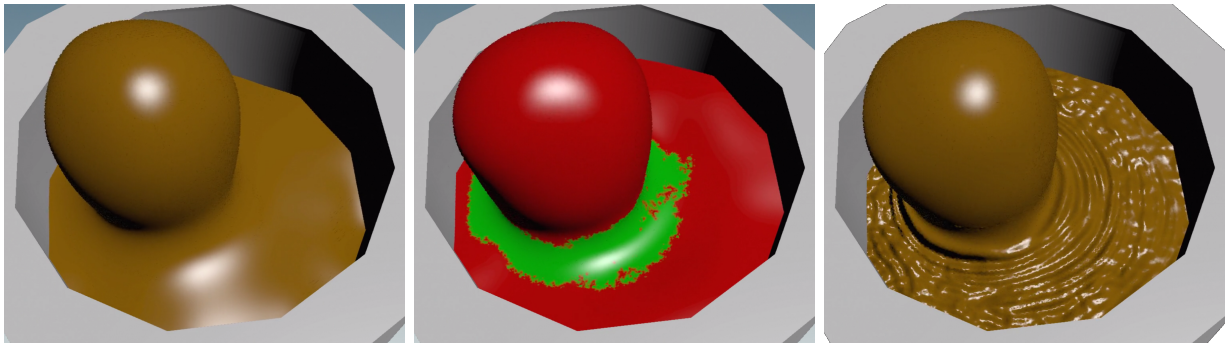
Instead of computing $A^{(t)}$ and c globally, we compute c_i locally at each surface point by measuring the distance to neighboring particles, using the same λ_f neighborhood size as in Section 2.3.4. We then fit a plane through neighboring displaced points, and define the local surface gradient ∇h_i as the slope of this plane [1]. We then scale the plane to recover the local area of the previous time step, and push neighboring points j towards the stretched plane, computed as

$$(1 + c_i)\sqrt{1 + ||\nabla h_i)||^2}\left((\nabla h_i)(\boldsymbol{x}_j - \boldsymbol{x}_i) + h_i\right). \tag{2.28}$$

This has the effect of stretching the surface locally by making it steeper.

After one stretching step, we compute the new value of c_i for each particle, and then average those values on λ_f neighborhoods. This allows the stretching of one particle to alleviate the stretching needs of neighboring particles, recovering the global nature of the c_i coefficients. We then apply the h and Δh forces. We iterate this process for a number of steps until within some user-defined tolerance.

1. If the slope is zero, we define ∇h_i as a random unit vector, which allows us to randomly seed the initial wrinkles



(a) (b) (c)

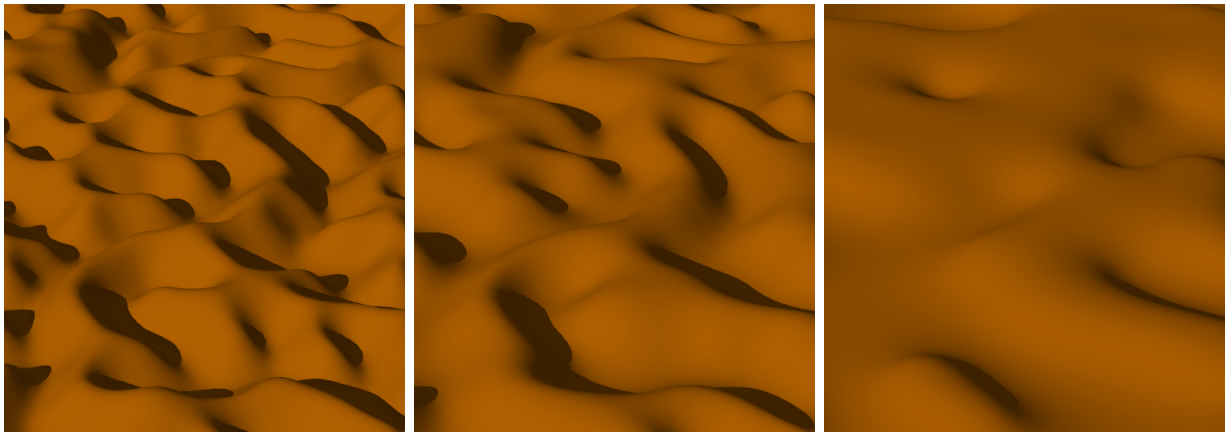
Figure 2.15. Wrinkles upres of a viscous mud poured into a container. a) The input surface is created around a coarse viscous fluid simulation. b) As the mud contacts with the pool of mud, surface compression occurs and generates surface wrinkles under our model. c) These wrinkles are passively advected with the fluid and are not advected on the surface, leading to a realistic, high-resolution appearance.

2.C.2. Preliminary Results

Our method is able to add wrinkles to coarse viscous fluid simulation, as shown in Figure 2.15 and in the accompanying video. These results are generated using Houdini [67]. By modifying the coefficients α_h and α_Δ, we can also control the frequency of the wrinkles, as shown in Figure 2.16.

However, our control over the frequency of the wrinkles is not intuitive, and the right parameters must be found by trial and error. Future work could investigate the model in more details and derive the right parameters for a prescribed wrinkle frequency. Furthermore, although the convergence slows significantly after a few iterations, it does not completely stop, even after thousands of steps. This is not a limiting factor, since different parameters still lead to different convergence speed and the process can be stopped after a few iterations, but a provably convergent model would provide a more sane method.

Future work could also investigate different models for generating wrinkles. For instance, reaction-diffusion models often have solutions converging to the similar types of wrinkle structures. Literature on such models is rich, and in many cases the dimensions and frequency of resulting structures can be predicted, for instance in the case of Hele-Shaw fingerings [52, 65].



(a) $\alpha_h = 0.01$, $\alpha_\Delta = 0.00025$ (b) $\alpha_h = 0.01$, $\alpha_\Delta = 0.001$ (c) $\alpha_h = 0.01$, $\alpha_\Delta = 0.004$

Figure 2.16. Wrinkles frequency control. Depending on the values of α_h and α_Δ used in our wrinkle model, the wrinkles evolve towards different frequency structures, significantly modifying the look of the final simulation.

References

[21] Akinci, N., M. Ihmsen, G. Akinci, B. Solenthaler and M. Teschner. 2012, Versatile rigid-fluid coupling for incompressible sph, *ACM Transactions on Graphics (TOG)*, vol. 31, nº 4, p. 62:1–62:8.

[22] Alexa, M., J. Behr, D. Cohen-Or, S. Fleishman, D. Levin and C. T. Silva. 2003, Computing and rendering point set surfaces, *IEEE Transactions on visualization and computer graphics*, vol. 9, nº 1, doi:10.1109/TVCG.2003.1175093, p. 3–15. URL `http://doi.ieeecomputersociety.org/10.1109/TVCG.2003.1175093`.

[23] Autodesk. 2014, Bifröst for Autodesk Maya, `http://www.autodesk.com/products/maya/overview`.

[24] Batty, C., A. Uribe, B. Audoly and E. Grinspun. 2012, Discrete viscous sheets, *ACM Transactions on Graphics (TOG)*, vol. 31, nº 4, p. 113.

[25] Bergou, M., M. Wardetzky, D. Harmon, D. Zorin and E. Grinspun. 2006, A quadratic bending model for inextensible surfaces, in *Symposium on Geometry Processing*, p. 227–230.

[26] Bhatacharya, H., Y. Gao and A. Bargteil. 2011, A level-set method for skinning animated particle data, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, p. 17–24.

[27] Blinn, J. F. 1982, A generalization of algebraic surface drawing, *ACM Transactions on Graphics (TOG)*, vol. 1, nº 3, p. 235–256.

[28] Bridson, R. 2008, Fluid simulation for computer graphics, AK Peters.

[29] Budsberg, J., M. Losure, K. Museth and M. Baer. 2013, Liquids in "The Croods", in *ACM SIGGRAPH Digital Production Symposium (DigiPro)*.

[30] Chatterjee, S., C. McDonald, J. Niu, S. S. Velankar, P. Wang and R. Huang. 2015, Wrinkling and folding of thin films by viscous stress, *Soft Matter*, vol. 11, nº 9, p. 1814–1827.

[31] Chentanez, N. and M. Müller. 2010, Real-time simulation of large bodies of water with small scale details, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

[32] Cook, R. L. 1986, Stochastic sampling in computer graphics, *ACM Transactions on Graphics (TOG)*, vol. 5, n° 1, doi:10.1145/7529.8927, p. 51–72, ISSN 0730-0301. URL `http://doi.acm.org/10.1145/7529.8927`.

[33] Cords, H. 2008, Moving with the flow: Wave particles in flowing liquids, in *Winter School of Computer Graphics (WSCG)*.

[34] Foster, N. and R. Fedkiw. 2001, Practical animation of liquids, in *Proceedings of SIGGRAPH*, p. 23–30.

[35] Foster, N. and D. Metaxas. 1996, Realistic animation of liquids, *Graphical models and image processing*, vol. 58.

[36] Guennebaud, G., M. Germann and M. H. Gross. 2008, Dynamic sampling and rendering of algebraic point set surfaces, *Computer Graphics Forum*, vol. 27, n° 2, doi:10.1111/j.1467-8659.2008.01163.x, p. 653–662. URL `http://dx.doi.org/10.1111/j.1467-8659.2008.01163.x`.

[37] Huang, R. and S. H. Im. 2006, Dynamics of wrinkle growth and coarsening in stressed thin films, *Physical Review E*, vol. 74, n° 2, p. 026 214.

[38] Huang, R., Z. Melek and J. Keyser. 2011, Preview-based sampling for controlling gaseous simulations, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, p. 177–186.

[39] Huang, R. and Z. Suo. 2002, Wrinkling of a compressed elastic film on a viscous layer, *Journal of Applied Physics*, vol. 91, n° 3, p. 1135–1142.

[40] Ihmsen, M., N. Akinci, G. Akinci and M. Teschner. 2012, Unified spray, foam and air bubbles for particle-based fluids, *The Visual Computer*, vol. 28, n° 6-8, p. 669–677.

[41] Ihmsen, M., J. Orthmann, B. Solenthaler, A. Kolb and M. Teschner. 2014, SPH fluids in computer graphics, in *Eurographics - State of the Art Reports*, p. 21–42.

[42] Jeong, S. and C. Kim. 2013, Combustion waves on the point set surface, *Computer Graphics Forum*, vol. 32, n° 7, doi:10.1111/cgf.12230, p. 225–234. URL `http://dx.doi.org/10.1111/cgf.12230`.

[43] Kass, M. and G. Miller. 1990, Rapid, stable fluid dynamics for computer graphics, in *Proceedings of ACM SIGGRAPH*.

[44] Kim, T., J. Tessendorf and N. Thuerey. 2013, Closest point turbulence for liquid surfaces, *ACM Transactions on Graphics (TOG)*, vol. 32, n° 2.

[45] Kim, T., N. Thuerey, D. James and M. Gross. 2008, Wavelet turbulence for fluid simulation, in *ACM Transactions on Graphics (TOG)*.

[46] Kodio, O., I. M. Griffiths and D. Vella. 2017, Lubricated wrinkles: Imposed constraints affect the dynamics of wrinkle coarsening, *Physical Review Fluids*, vol. 2, n° 1, p. 014 202.

[47] Lait, J. 2011, Correcting low frequency impulses in distributed simulations, in *ACM SIGGRAPH Talks*, p. 53:1–53:2.

[48] Liang, J., R. Lai, T. W. Wong and H. Zhao. 2012, Geometric understanding of point clouds using laplace-beltrami operator, in *IEEE Computer Vision and Pattern Recognition (CVPR)*.

[49] Liang, J. and H. Zhao. 2013, Solving partial differential equations on point clouds, *SIAM Journal on Scientific Computing*, vol. 35, nº 3.

[50] Macdonald, C. B., B. Merriman and S. J. Ruuth. 2013, Simple computation of reaction–diffusion processes on point clouds, *Proceedings of the National Academy of Sciences*, vol. 110, nº 23, p. 9209–9214.

[51] Macklin, M., M. Müller, N. Chentanez and T.-Y. Kim. 2014, Unified particle physics for real-time applications, *ACM Transactions on Graphics (TOG)*, vol. 33, nº 4, p. 153:1–153:12.

[52] Mineev-Weinstein, M. 1998, Selection of the saffman-taylor finger width in the absence of surface tension: an exact result, *Physical review letters*, vol. 80, nº 10, p. 2113.

[53] Müller, M., D. Charypar and M. Gross. 2003, Particle-based fluid simulation for interactive applications, in *ACM SIGGRAPH/Eurographics Symposium on Computer animation*, p. 154–159.

[54] Narain, R., T. Pfaff and J. F. O'Brien. 2013, Folding and crumpling adaptive sheets, *ACM Transactions on Graphics (TOG)*, vol. 32, nº 4, p. 51.

[55] Narain, R., J. Sewall, M. Carlson and M. C. Lin. 2008, Fast animation of turbulence using energy transport and procedural synthesis, *ACM Transactions on Graphics (TOG)*, vol. 27, p. 166:1–166:8.

[56] Next Limit Technologies. 2014, RealFlow, `http://www.realflow.com/`.

[57] Nielsen, M. B. and R. Bridson. 2011, Guide shapes for high resolution naturalistic liquid simulation, *ACM Transactions on Graphics (TOG)*.

[58] Nielsen, M. B., B. B. Christensen, N. B. Zafar, D. Roble and K. Museth. 2009, Guiding of smoke animations through variational coupling of simulations at different resolutions, in *ACM SIGGRAPH/Eurographics Symppsium on Computer Animation*.

[59] Osher, S. and R. Fedkiw. 2003, The level set method and dynamic implicit surfaces, Springer-Verlag, New York.

[60] Pan, Z., J. Huang, Y. Tong, C. Zheng and H. Bao. 2013, Interactive localized liquid motion editing, *ACM Transactions on Graphics (TOG)*.

[61] Peer, A., M. Ihmsen, J. Cornelis and M. Teschner. 2015, An implicit viscosity formulation for sph fluids, *ACM Transactions on Graphics (TOG)*, vol. 34, nº 4, p. 114.

[62] Schechter, H. and R. Bridson. 2008, Evolving sub-grid turbulence for smoke animation, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, p. 1–7.

[63] Schechter, H. and R. Bridson. 2012, Ghost sph for animating water, *ACM Transactions on Graphics (TOG)*, vol. 31, nº 4, p. 61:1–61:8.

[64] Shao, X., Z. Zhou, J. Zhang and W. Wu. 2014, Realistic and stable simulation of turbulent details behind objects in smoothed-particle hydrodynamics fluids, *Computer Animation and Virtual Worlds*, ISSN 1546-427X. URL `http://dx.doi.org/10.1002/cav.1607`.

[65] Shelley, M. J., F.-R. Tian and K. Wlodarski. 1997, Hele-shaw flow and pattern formation in a time-dependent gap, *Nonlinearity*, vol. 10, nº 6, p. 1471.

[66] Shi, L. and Y. Yu. 2005, Taming liquids for rapidly changing targets, *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

[67] Side Effects Software. 2012, Houdini 13, `http://www.sidefx.com`.

[68] Stam, J. 1999, Stable fluids, in *Proceedings of 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 121–128.

[69] Teschner, M., B. Heidelberger, M. Mueller, D. Pomeranets and M. Gross. 2003, Optimized spatial hashing for collision detection of deformable objects, in *Proceedings of Vision, Modeling, Visualization*, p. 47–54.

[70] Tessendorf, J. 2008, Vertical derivative math for iwave, .

[71] Thuerey, N., T. Kim and T. Pfaff. 2013, Turbulent fluids, in *ACM SIGGRAPH 2013 Courses*, ISBN 978-1-4503-2339-0, p. 6:1–6:1.

[72] Thuerey, N., C. Wojtan, M. Gross and G. Turk. 2010, A Multiscale Approach to Mesh-based Surface Tension Flows, *ACM Transactions on Graphics (TOG)*, vol. 29 (4), p. 10.

[73] Turk, G. 1991, Generating textures on arbitrary surfaces using reaction-diffusion, in *Proceedings of SIGGRAPH*, p. 289–298.

[74] Wang, H., G. Miller and G. Turk. 2007, Solving general shallow wave equations on surfaces, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

[75] Wang, R., Z. Yang, L. Liu and Q. Chen. 2013, Discretizing laplace–beltrami operator from differential quantities, *Communications in Mathematics and Statistics*, vol. 1, n° 3, p. 331–350.

[76] Williams, B. W. 2008, Fluid surface reconstruction from particles, *M.S. Thesis, The University of British Columbia, Canada*.

[77] Wojtan, C., M. Müller-Fischer and T. Brochu. 2011, Liquid simulation with mesh-based surface tracking, in *ACM SIGGRAPH 2011 Courses*, ISBN 978-1-4503-0967-7, p. 8:1–8:84.

[78] Yu, J., C. Wojtan, G. Turk and C. Yap. 2012, Explicit mesh surfaces for particle based fluids, *Computer Graphics Forum*.

[79] Yu, Q., F. Neyret, E. Bruneton and N. Holzschuch. 2009, Scalable real-time animation of rivers, *Computer Graphics Forum*, vol. 28, n° 2, p. 239–248.

[80] Yuan, Z., Y. Zhao and F. Chen. 2012, Incorporating stochastic turbulence in particle-based fluid simulation, *The Visual Computer*, vol. 28, n° 5, p. 435–444.

[81] Yuksel, C., D. H. House and J. Keyser. 2007, Wave particles, *ACM Transactions on Graphics (TOG)*, vol. 26, n° 3.

[82] Zhu, Y. and R. Bridson. 2005, Animating sand as a fluid, in *ACM Transactions on Graphics (TOG)*, vol. 24(3), ACM, p. 965–972.

[83] Zwicker, M., M. Pauly, O. Knoll and M. Gross. 2002, Pointshop 3d: an interactive system for point-based surface editing, in *ACM Transactions on Graphics (TOG)*, vol. 21(3), ACM, p. 322–329.

Chapter 3

LOCAL BASES FOR MODEL-REDUCED SMOKE SIMULATIONS

This second paper presents a method for model-reduced simulations of smoke. We construct basis vector fields from which we evolve a simulation. An important component of our method is the projection of vector fields onto our basis, which requires solving a linear system with an iterative algorithm. Our bases are constructed so as to sparsify this linear system, in order to improve the efficiency of the iterative method. Furthermore, our bases are designed to have enhanced orthogonality properties, which allows us to parallelize the iterative scheme while maintaining its convergence properties.

3.A. Publication

This paper has been submitted to ACM Transactions on Graphics (TOG) for the SIGGRAPH 2018 conference, and is under review at the time of writing. The paper has been reformatted to appropriately follow the format of this thesis.

Olivier Mercier is the principal author of this project. Mercier designed and implemented the method. He was in charge of the writing and presentation of the paper, including the generation of all the results.

Local Bases for Model-reduced Smoke Simulations

Olivier Mercier, *Université de Montréal*

Derek Nowrouzezahrai, *McGill University*

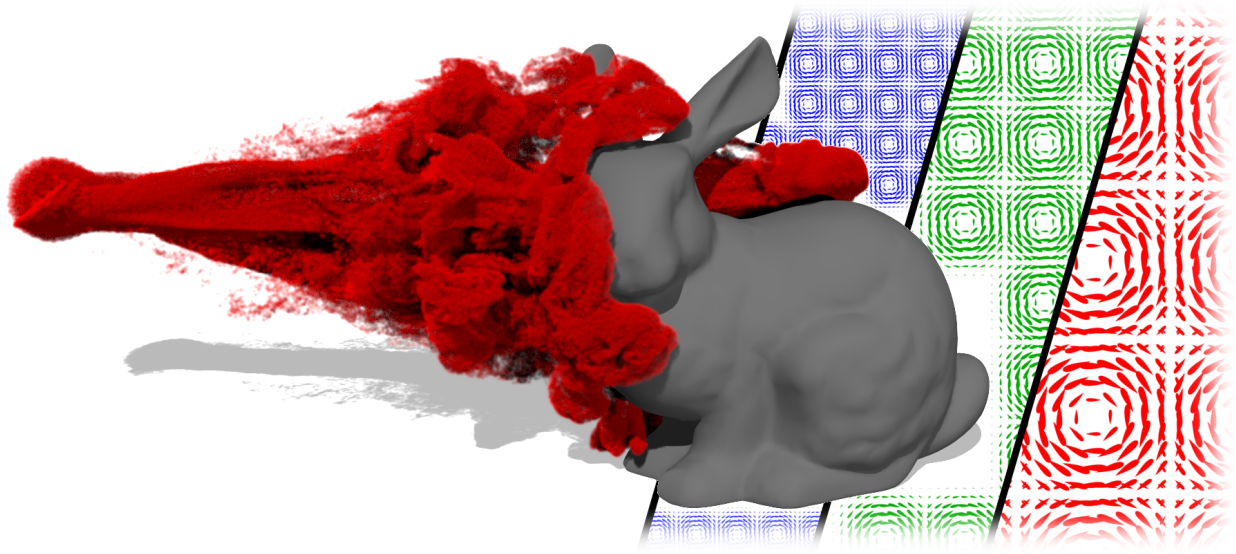


Figure 3.1. We advect smoke particles using our model-reduced, multiresolution representation of the underlying fluid dynamics. Each basis flow (visualized in simplified form, on the right) has local support and permits an adaptive representation of the fluid dynamics across scales over the simulation domain. Our basis is efficient to construct, apply and evaluate, it handles dynamic obstacles and curved boundaries, and it allows flexible user control.

Résumé

Nous présentons une méthode par réduction de modèle efficace et flexible pour la simulation de fluids incompressibles, dérivant une nouvelle base de champ de vecteurs qui capture la dynamique des fluides à plusieurs échelles. Nos bases ont une forme analytique simple et peuvent paver l'espace selon des arrangements réguliers, ce qui évite l'utilisation de structures de données complexes ou de recherches de voisins. Nous précalculons les interactions locales entre les bases et les réutilisons pour simuler des fluides sur n'importe quel domaine sans coûts additionnels. Nous projetons et résolvons les équations de Navier-Stokes sur notre base, en plus d'exposer des paramètres intuitifs pour le contrôle des transfers d'énergie entre les différentes échelles. Notre base peut être adaptée à des frontières courbes, peut être couplée à des obstacles dynamiques, et offre des compromis ajustables entre rapidité de calcul et précision.

Mots-clés : Simulation de fluide, Réduction de modèle

Abstract

We present an efficient and flexible model reduction method for simulating incompressible fluids, deriving a novel localized vector field basis that captures flow dynamics across scales. Our bases have simple analytic forms and can be tiled on regular lattices, avoiding the use of complicated data structures or neighborhood queries. We can precompute local basis interactions, and then warp and reuse them to simulate fluid dynamics on any simulation domain without additional overhead. We project and solve the Navier-Stokes equations onto our basis and expose intuitive parameters to control energy distribution across scales. Our basis can adapt to curved simulation boundaries, can be coupled with dynamic obstacles, and offers simple adjustable trade-offs between speed and accuracy.

Keywords: Fluid simulation, Model reduction

3.1. Introduction

Realistic fluid simulation remains a fundamental challenge in computer graphics. Complex and intricate fluid features appear across spatial scales, and reproducing these detailed dynamics on uniformly discretized domains requires prohibitively large resolutions. Multiresolution methods can reduce this limitation by adapting the spatial and temporal simulation resolution, focusing computation time on appropriate regions of interest, e.g., regions where a fluid evolves into finer-scale structures or regions where a viewer's attention is more likely to be drawn.

Fluid simulation on non-uniform grids, such as octrees, is one common multiresolution approach used in graphics: given a uniform simulation grid, each cell is repeatedly subdivided until the required simulation resolution is reached. Implementing these approaches in a robust manner requires care, as interactions between cells across simulation scales can become non-trivial. Wavelet-based methods are another alternative, relying on localized multi-scale representations of the underlying dynamics. While these methods result in more compact representations of the simulation quantities, converting between wavelet and primal-domain representations often creates a computational bottleneck.

Our work is based, instead, on a *model reduction* methodology: simulated quantities are represented as weighted combinations of basis vector fields and the underlying dynamics are reformulated as operators that act on this *reduced* representation. The utility of these methods relies heavily on the properties of the basis, and optimal basis design for model-reduced dynamics remains an open problem.

We present a novel basis suitable for multi-resolution fluid simulations, providing efficient algorithms to simulate fluid dynamics using our basis. Our contributions include:

- a simple method to construct anisotropic vector field bases with local support and important orthogonality properties,

- flexible tiling strategies to cover arbitrary simulation domains without any basis recomputation,
- a deformation method to adapt bases to curved boundaries and dynamic obstacles, and
- an efficient, stable simulation algorithm that uses localized basis interactions and provides control over turbulent energy cascades.

3.2. Previous Work

Foster and Metaxas [96] and Stam [115] pioneered efficient fluid simulation in computer graphics by solving a discretization of the dynamics on uniform grids. Modern high-fidelity fluid simulations, however, require resolutions that mandate representations that scale better than these uniform discretizations. We outline the various strategies used to accelerate fluid simulation in this context, below.

Model Reduction. The infinite-dimensional space of all possible vector fields can be reduced to a linear combination of specially-chosen basis fields. This high-level "model reduction" principle has been applied to many problems in computer graphics, including character animation [111, 105], cloth simulation [99], deformation [86] and global illumination [116].

Treuille et al. [119] introduced model reduction for fluids to graphics using vector field bases constructed from SVD decompositions of full-space simulation data. Their divergence-free basis satisfied boundary conditions, but the need for full-space simulation constrains its use to re-simulations in a single, fixed domain. Improvements allow for fluid parameter variations [104] and limited domain deformations [116], but the precomputed full-space simulation constraints remain.

Wicke et al. [121] improve *basis reusability* by precomputing modular, reconfigurable flows. Gerszewski et al. [97] instead enrich an set of existing bases for task adaptation. In both cases, however, the initial basis and any interactions with additional bases must still be precomputed from a costly full-space simulation.

Instead of precomputing bases, basis functions can be generated analytically. De Witt et al. [92] and Liu et al. [108] derive a basis using the eigenfunctions of the Laplacian, a construction akin to 1D Fourier bases. This method applies to any fixed domain shape and produces an arbitrary number of divergence-free bases, however bases must be completely recomputed if the simulation domain changes. Nevertheless, the advantage of isolating specific flow frequencies leads to an intuitive basis: every basis coefficient only influences a given flow frequency. The global support of the basis, however, precludes any local control in the final simulation.

We propose a multiresolution analytic basis that provides local control and that can be tiled and warped onto arbitrary domains, including domains with dynamic obstacles and curved boundaries.

Vorticity Methods. Several methods use the vorticity formulation of the Navier-Stokes equations to simulate fluid motion. Vorticity representations include point primitives [84], filaments [85, 120], and sheets [112]. These elements are advected by the flow and fully represent the fluid motion. Other methods use vorticity to add turbulent detail atop coarser simulations [98, 109]. In either case, fluid dynamics are transposed onto the vorticity elements, where they advect, rotate, and deform into elements of varying scale. Inspired by these approaches, we design a basis that can adapt to multiscale dynamics.

Since vorticity elements move freely (i.e., non-uniformly) in the simulation domain, fluid flow reconstruction requires neighborhood search and more complex data structures. While we do not rely explicitly on a vorticity formulation, our bases are similar to localized vortices, but defined entirely in the spatial domain (an in regular patterns): this allows us to avoid costly frequency conversions and use simple data structures to accelerate computation.

Wavelet Methods. Our bases are inspired by wavelets, which have a rich history in the fluid dynamics literature (e.g., Schneider and Vasilyev [114]). Wavelets are used to study the statistical properties of turbulent flow [94, 87] and for efficient simulation in vorticity formulations [89, 95] (albeit mostly limited to 2D domains). More general approaches, such as adaptive wavelet collocation methods (AWCM) [103, 110], are also popular in engineering applications since they provide a general and physically-accurate framework applicable to many classes of differential equations. The wavelets used in AWCM, however, are not divergence-free and require frequent conversions to-and-from the primal domain.

Divergence-free wavelets (DFWs) are also interesting, as they avoid the pressure and incompressibility computations of standard solvers. Lemarié-Rieusset [106] proposed DFWs with compact support and Deriaz et al. [93] applied them to simulation. These bases are not orthogonal and their construction mostly applies to periodic domains with grid discretizations, which limits their application to more complex, dynamic domains encountered in graphics. Recent extensions of DFWs onto square domains [117, 102] suffer from similar restrictions.

Our work is similar in principle to AWCM and divergence-free wavelets, but our bases are better adapted to high-performance fluid simulations on complex, potentially dynamic domains. While motivated by classical wavelet theory, we do not rely on it when constructing our orthogonal vector basis.

3.3. Notation and Model Reduction

We introduce our notation and model reduction in 2D, both of which extend naturally to 3D. We use lowercase script for scalars (a), bolded lowercase for vectors and vector-valued

functions ($\mathbf{a}$), and bold uppercase for matrices and tensors ($\mathbf{A}$). To simplify notation, we sometimes omit function parameters and integral differentials.

We rely on *quasimatrix* notation [118] to express linear combinations of functions, where "matrices" have one dimension of infinite size, i.e. columns are $\in \mathbb{R}^\infty$. In our (2D) case, quasimatrix columns represent vector-valued functions on the simulation domain $\Omega \subset \mathbb{R}^2$. Let $\odot_i$ denote a tensor-vector product, with the sum taken over the i^{th} collapsed tensor dimension. For instance, given a matrix $\mathbf{A}$ with columns $\mathbf{a}_j$, and a vector $\mathbf{v}$ with elements v_j, the product $\mathbf{A}\mathbf{v}$ can be written as $\mathbf{A} \odot_j \mathbf{v} = \sum_j v_j \, \mathbf{a}_j$.

We refer to the set of all continuous vector fields defined on Ω as the *full space* $\mathcal{F}$, and its elements are referred to as *flows*. The idea of model reduction is to operate on a *reduced* subspace $\mathcal{R} \subset \mathcal{F}$ of flows composed only of linear combinations of a given finite set of *basis flows* $\mathbf{b}_i \in \mathcal{F}, i \in \{1, \ldots, r\}$. Let $\mathbf{B} \in \mathbb{R}^{\infty \times r}$ be the quasimatrix whose columns are the r linearly-independent basis flows $\mathbf{b}_i$. Any flow $\mathbf{u} \in \mathcal{R}$ is represented by a set of coefficients $\tilde{\mathbf{u}} \in \mathbb{R}^r$ as $\mathbf{u} = \mathbf{B}\,\tilde{\mathbf{u}}$. Conversely, any flow $\mathbf{u} \in \mathcal{F}$ can be projected to the closest (in the least-squares sense) element of $\mathcal{R}$ using the *normal equation* $\tilde{\mathbf{u}} = \mathbf{B}^+\mathbf{u}$, where $\mathbf{B}^+$ is the *pseudoinverse* of $\mathbf{B}$, defined as

$$\mathbf{B}^+ = \mathbf{B}^- \, \mathbf{B}^T \qquad \text{where} \quad \mathbf{B}^- = \left(\mathbf{B}^T\mathbf{B}\right)^{-1} \tag{3.1}$$

and where matrix $(\mathbf{B}^T\mathbf{B}) \in \mathbb{R}^{r \times r}$ has entries $(\mathbf{B}^T\mathbf{B})_{ij} = \int_\Omega \mathbf{b}_i \cdot \mathbf{b}_j$.

Our goal is to solve the Navier-Stokes equations on Ω

$$\partial\mathbf{u} \big/ \partial t = -(\nabla\mathbf{u})\,\mathbf{u} - \nabla p + \nu\,\nabla^2\mathbf{u} + \mathbf{f} \;\text{ and }\; \nabla \cdot \mathbf{u} = 0 \;, \tag{3.2}$$

where $\mathbf{u}$, p and $\mathbf{f}$ are functions of space with an implicit dependence on time t. Specifically, $\mathbf{u}$ is the fluid velocity flow, p is the scalar pressure, ν is the viscosity, $\mathbf{f}$ is the external forces flow, ∇ is the gradient operator for scalars or the Jacobian operator for vectors, and ∇^2 is the Laplace operator. We use no-slip boundary conditions, i.e., the flow must match boundary velocities.

Model-reduced simulations commonly impose that all basis flows be divergence-free, making any combined flow $\mathbf{u} \in \mathcal{R}$ divergence-free by linearity. This reduces Equations 3.2 to a simpler system

$$\partial\mathbf{u} \big/ \partial t = -(\nabla\mathbf{u})\,\mathbf{u} + \nu\,\nabla^2\mathbf{u} + \mathbf{f} \tag{3.3}$$

and avoids the costly pressure solve of more traditional solvers [115]. Projecting both sides of Equations 3.3 onto $\mathcal{R}$ yields

$$\partial\tilde{\mathbf{u}} \big/ \partial t = -\mathbf{B}^- \left((\mathbf{A} \odot_i \tilde{\mathbf{u}}) \odot_j \tilde{\mathbf{u}}\right) + \nu\,\mathbf{B}^- \left(\mathbf{D} \odot_i \tilde{\mathbf{u}}\right) + \tilde{\mathbf{f}} \,, \tag{3.4}$$

where the *advection tensor* $\mathbf{A} \in \mathbb{R}^{r \times r \times r}$ and the *diffusion matrix* $\mathbf{D} \in \mathbb{R}^{r \times r}$ have elements

$$\mathbf{A}_{ijl} = \int_\Omega \mathbf{b}_l \cdot ((\nabla \mathbf{b}_i)\, \mathbf{b}_j) \quad \text{and} \quad \mathbf{D}_{il} = \int_\Omega \mathbf{b}_l \cdot \nabla^2 \mathbf{b}_i \,. \tag{3.5}$$

Equation 3.4 thus expresses the fluid dynamics in terms of only the reduced coefficients. This simplifies computational complexity at the cost of restricting the set of possible generated flows.

3.4. Basis Construction

Equation 3.4 suggests desirable properties for the basis functions $\mathbf{b}_i$:

1. **divergence-free** basis functions are essential in order to apply the simplified Navier-Stokes formulation in Equation 3.3,
2. an **orthogonal** basis, i.e., $\int_\Omega \mathbf{b}_i \cdot \mathbf{b}_j = 0 \, \forall i \neq j$, implies that $\mathbf{B}^-$ is diagonal, avoiding costly matrix inversions in Equation 3.4,
3. **local support** sparsifies the $\mathbf{A}$ and $\mathbf{D}$ tensors as elements corresponding to the combination of basis functions with non-overlapping support are zero; this facilitates basis manipulation, since modifying one basis' coefficient only affects flow locally in the domain,
4. and basis **completeness**, i.e., the ability to represent *any* flow in $\mathcal{F}$; while this is not generally achievable with a finite number of functions, it is desirable to have as large a set of linearly independent bases (and at as many *scales*) as possible.

As discussed in Section 3.2, methods based on full-space simulation snapshots generally only satisfy the divergence-free property, and previous work on Laplacian eigenvectors do not yield locally supported bases. Wavelet bases satisfy the last three properties, but it is mathematically impossible to create bases that satisfy all four properties using traditional wavelet theory [107].

We construct a *multiscale* basis that carefully compromises between these four properties: it is divergence-free, has bounded support, and can be made increasingly complete in a controllable manner. We devise a *relaxed* orthogonality property, where only basis functions at the same scale/frequency are orthogonal. We show in Section 3.5 how this last property can still be exploited to reduce computation cost. Finally, we detail a basis construction method that can be used to adapt a basis set to any new simulation domain and at any refinement scale, without any additional precomputations. Our exposition focusses on the 2D case, with particularities of extensions to 3D highlighted in Section 3.4.6.

3.4.1. Basis Scheme

Similarly to wavelet approaches, our basis set is built atop exemplar bases which are tiled over Ω at various scales. We first construct a *basis template* $\mathbf{b}^{\boldsymbol{k}}(\mathbf{x}) := \mathbf{b}^{(k_x,k_y)}(x,y)$ for each

frequency $\boldsymbol{k}$ in a predetermined integer set $\mathcal{K} \subset (\mathbb{N}_{\geq 1})^2$. Each basis template $\mathbf{b}^{\boldsymbol{k}}$ is centered at (0,0) and has a finite rectangular support $\mathcal{S}^{\boldsymbol{k}} = [-1/(2k_x),1/(2k_x)] \times [-1/(2k_y),1/(2k_y)]$, outside of which it has value (0,0). We tile Ω with copies of the basis templates and denote these translated bases $\mathbf{b}^{\boldsymbol{k}}_{\boldsymbol{c}}(\mathbf{x}) := \mathbf{b}^{(k_x,k_y)}_{(c_x,c_y)}(x,y) := \mathbf{b}^{\boldsymbol{k}}(x-c_x,y-c_y)$, where subscript $\boldsymbol{c} \in \mathcal{C}^{\boldsymbol{k}} \subset \mathbb{R}^2$ indexes the basis centers.

The choice of $\mathcal{K}$ and $\mathcal{C}^{\boldsymbol{k}}$ defines the overall coverage of the simulation domain, with larger sets improving coverage while also increasing computational cost. Throughout this section, we use $\mathcal{K} = \left\{(2^\alpha, 2^\beta) \,|\, \alpha,\beta \in \mathbb{N}_{\geq 0}\right\}$, which corresponds to a power-of-two basis refinement. Note that the refinement in each axis is independent, allowing for anisotropic bases. We tile the bases on regular lattices using $\mathcal{C}^{\boldsymbol{k}} = ((\phi/k_x)\mathbb{Z}) \times ((\phi/k_y)\mathbb{Z})$, where ϕ is the *tiling density*. We use $\phi = 1/2$, which means bases overlap by half of their support size. Figure 3.2 illustrates the basis coverage obtained with these parameter settings, visualizing only those bases with support inside the simulation domain. These $\mathcal{K}$ and $\mathcal{C}^{\boldsymbol{k}}$ choices adequately cover the simulation domain, but we detail other coverage options (e.g., a treatment more suitable for boundaries) in Section 3.6.

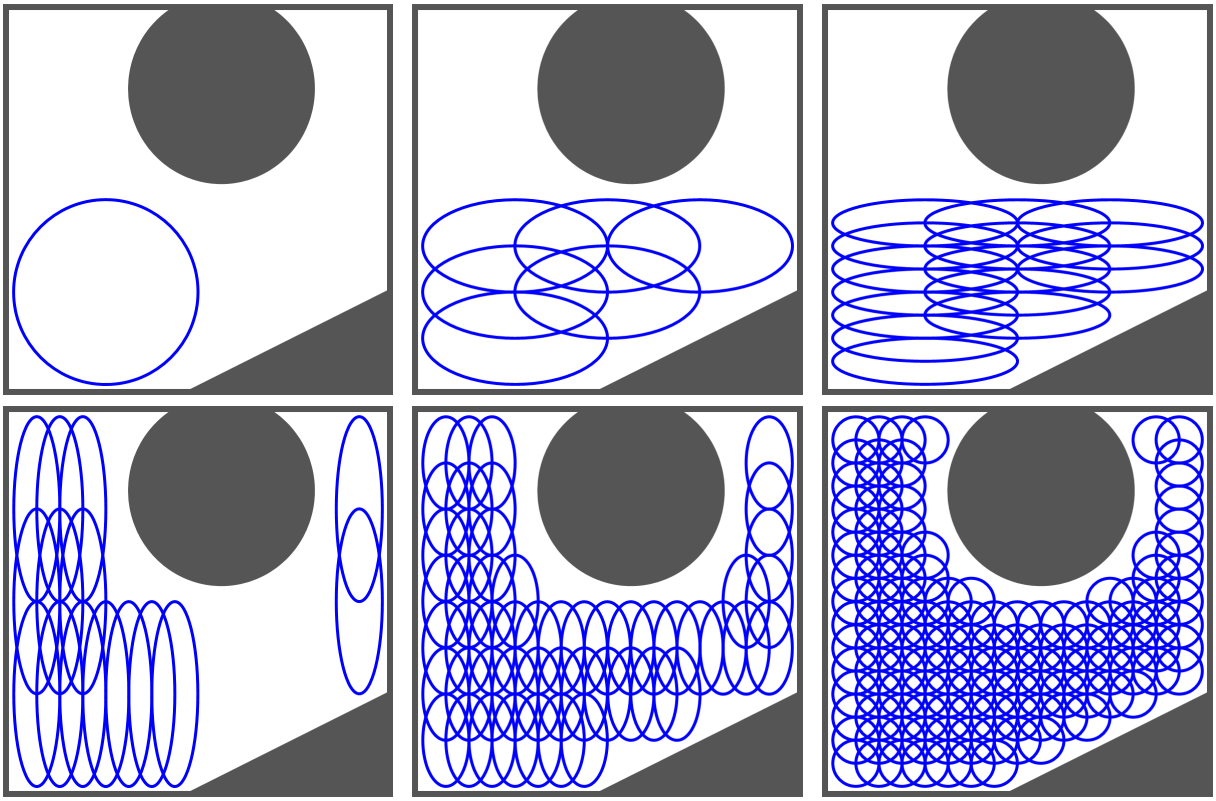


Figure 3.2. Visualization of our coverage of a simulation domain, where each ellipsoid represents a single basis. For each frequency layer, bases are aligned on a regular lattice, and cover as much of the simulation domain as possible. Bases with higher anisotropy or smaller scale naturally reach narrower regions, and better wrap around curved boundaries.

3.4.2. Divergence-Free, Continuity, and Locality Constraints

Inspired by De Witt et al. [92], our basis templates are constructed from the divergence-free vector-valued eigenfunctions of the Laplacian in the unit square, with free-slip boundary conditions. These eigenfunctions are more naturally described on the square $\mathcal{D}^{\mathbf{1}} = [0,1]^2$, so we will define the basis templates on $\mathcal{D}^{\boldsymbol{k}} = [0, 1/k_x] \times [0,1/k_y]$ and shift them back to $\mathcal{S}^{\boldsymbol{k}}$ in Section 3.4.5.

Vector eigenfunctions on $\mathcal{D}^{\mathbf{1}}$, which we call the *eigenflows*, are

$$\mathbf{e}^{\boldsymbol{k}}(\mathbf{x}) = \begin{pmatrix} k_y \, \sin(k_x \, \pi \, x) \, \cos(k_y \, \pi \, y) \\ -k_x \, \cos(k_x \, \pi \, x) \, \sin(k_y \, \pi \, y) \end{pmatrix} \in \mathbb{R}^2 \, , \tag{3.6}$$

where $\boldsymbol{k} \in (\mathbb{N}_{\geq 1})^2$ is the eigenflow's *frequency*. We consider a periodic extension of these eigenflows to $\mathbb{R}^2$, with period $2\,k_x$ in x and $2\,k_y$ in y (Figure 3.3.) This basis is akin to the Fourier basis, with functions of a single frequency and with infinite support.

We will use linear combinations of eigenflows to construct our bases, aiming for the simplest combination that satisfies our constraints. Ideally, we would need only use the single eigenflow $\mathbf{e}^{\boldsymbol{k}}$ to define a basis template $\mathbf{b}^{\boldsymbol{k}}$, since it isolates the coarsest frequency matching the size of $\mathcal{D}^{\boldsymbol{k}}$; however, we cannot simply clamp $\mathbf{e}^{\boldsymbol{k}}$ to zero outside $\mathcal{D}^{\boldsymbol{k}}$ as it would create a discontinuous flow.

To define basis template $\mathbf{b}^{\boldsymbol{k}}$, we therefore need to add some *harmonic eigenflows* with higher frequencies to the *fundamental eigenflow* $\mathbf{e}^{\boldsymbol{k}}$, where the frequencies of the harmonic eigenflows are integer multiples of $\boldsymbol{k}$. We denote them $\mathbf{e}^{\boldsymbol{ak}} = \mathbf{e}^{(a_x k_x, a_y k_y)}$ with $\boldsymbol{a} = (a_x, a_y) \in (\mathbb{N}_{\geq 1})^2$. In particular, harmonic $\boldsymbol{a} = (1,1)$ is the fundamental frequency. We consider linear combinations of harmonic eigenflows $\mathbf{h}^{\boldsymbol{k}}$, artificially restricting their support to $\mathcal{D}^{\boldsymbol{k}}$, yielding

$$\mathbf{h}^{\boldsymbol{k}}(\mathbf{x}) = \begin{cases} \sum\limits_{\boldsymbol{a} \in \mathcal{A}} w_{\boldsymbol{a}}^{\boldsymbol{k}} \, e^{\boldsymbol{ak}}(\mathbf{x}) & \text{if } \mathbf{x} \in \mathcal{D}^{\boldsymbol{k}} \\ 0 & \text{otherwise} \end{cases} , \tag{3.7}$$

where $\mathcal{A}$ is a given finite harmonic multiplier subset of $(\mathbb{N}_{\geq 1})^2$ containing at least (1,1), and $w_{\boldsymbol{a}}^{\boldsymbol{k}} \in \mathbb{R}$ are the scalar weights of the linear combination. Since we do not want bases to favor either direction, we impose $\mathcal{A}$ to be a Cartesian product of the same set of multipliers $\mathcal{A}_\star$ in x and y, i.e., $\mathcal{A} = \mathcal{A}_\star \times \mathcal{A}_\star$. We aim for $\mathcal{A}$ to be as small as possible, so that the linear combination is dominated by the fundamental eigenflow, and the structure of the basis is as simple as possible. For the same reasons, we impose

$$w_{\mathbf{1}}^{\boldsymbol{k}} = 1 \tag{3.8}$$

and aim for $|w_{\boldsymbol{a}}^{\boldsymbol{k}}|$ to be as small as possible $\forall \boldsymbol{a} \neq (1,1)$. To enforce continuity, we constrain $\mathbf{h}^{\boldsymbol{k}}$ to be zero on the boundary of its support, noted $\partial\mathcal{D}^{\boldsymbol{k}}$. Evaluating the eigenflows $\mathbf{e}^{\boldsymbol{ak}}$ on

the four sides of $\partial\mathcal{D}^{\boldsymbol{k}}$ results in

$$\begin{aligned}
e^{\boldsymbol{ak}}(\mathbf{x})\Big|_{x=0} &= (0\,,\, -a_x\,k_x\sin(a_y\,k_y\,\pi\,y)) \\
e^{\boldsymbol{ak}}(\mathbf{x})\Big|_{x=1/k_x} &= (0\,,\, -(-1)^{a_x}\,a_x\,k_x\sin(a_y\,k_y\,\pi\,y)) \\
e^{\boldsymbol{ak}}(\mathbf{x})\Big|_{y=0} &= (a_y\,k_y\,\sin(a_x\,k_x\,\pi\,x), 0) \\
e^{\boldsymbol{ak}}(\mathbf{x})\Big|_{y=1/k_y} &= ((-1)^{a_y}\,a_y\,k_y\,\sin(a_x\,k_x\,\pi\,x), 0).
\end{aligned} \tag{3.9}$$

Therefore,

$$\mathbf{h}^{\boldsymbol{k}}\Big|_{\partial\mathcal{D}^{\boldsymbol{k}}} = 0 \tag{3.10}$$

$$\Rightarrow \begin{cases}
\sum\limits_{a_y\in\mathcal{A}_\star}\left(\sum\limits_{a_x\in\mathcal{A}_\star} - w_{\boldsymbol{a}}^{\boldsymbol{k}}\,a_x\,k_x\right)\sin(a_y\,k_y\,\pi\,y) & = 0 \\
\sum\limits_{a_y\in\mathcal{A}_\star}\left(\sum\limits_{a_x\in\mathcal{A}_\star} - w_{\boldsymbol{a}}^{\boldsymbol{k}}\,(-1)^{a_x}\,a_x\,k_x\right)\sin(a_y\,k_y\,\pi\,y) & = 0 \\
\sum\limits_{a_x\in\mathcal{A}_\star}\left(\sum\limits_{a_y\in\mathcal{A}_\star} w_{\boldsymbol{a}}^{\boldsymbol{k}}\,a_y\,k_y\right)\sin(a_x\,k_x\pi\,x) & = 0 \\
\sum\limits_{a_x\in\mathcal{A}_\star}\left(\sum\limits_{a_y\in\mathcal{A}_\star} w_{\boldsymbol{a}}^{\boldsymbol{k}}\,(-1)^{a_y}\,a_y\,k_y\right)\sin(a_x\,k_x\,\pi\,x) & = 0
\end{cases}.$$

Since the functions $\sin(\beta\,\pi\,x)$ are linearly independent for different β, we can equate their coefficients in the sums to zero. This leads to

$$\Rightarrow \begin{cases}
\sum\limits_{a_x\in\mathcal{A}_\star} w_{\boldsymbol{a}}^{\boldsymbol{k}}\,a_x & = 0 \quad \forall a_y\in\mathcal{A}_\star \\
\sum\limits_{a_x\in\mathcal{A}_\star} w_{\boldsymbol{a}}^{\boldsymbol{k}}\,(-1)^{a_x}\,a_x & = 0 \quad \forall a_y\in\mathcal{A}_\star \\
\sum\limits_{a_y\in\mathcal{A}_\star} w_{\boldsymbol{a}}^{\boldsymbol{k}}\,a_y & = 0 \quad \forall a_x\in\mathcal{A}_\star \\
\sum\limits_{a_y\in\mathcal{A}_\star} w_{\boldsymbol{a}}^{\boldsymbol{k}}(-1)^{a_y}\,a_y & = 0 \quad \forall a_x\in\mathcal{A}_\star
\end{cases}, \tag{3.11}$$

which is a system of linear constraints for variables $w_{\boldsymbol{a}}^{\boldsymbol{k}}$. By choosing a sufficiently large $\mathcal{A}$, we can have enough variables to solve this system and obtain weights that zero $\mathbf{h}^{\boldsymbol{k}}$ on $\partial\mathcal{D}^{\boldsymbol{k}}$. Figure 3.3 provides some visual intuition on simple solutions to this system. With weights that satisfy Equations 3.11, $\mathbf{h}^{\boldsymbol{k}}$ is continuous and locally supported on $\mathcal{D}^{\boldsymbol{k}}$, as desired.

Interestingly, applying the same method to impose $\partial\mathbf{h}^{\boldsymbol{k}}/\partial x = 0$, $\partial\mathbf{h}^{\boldsymbol{k}}/\partial y = 0$ or $\partial^2\mathbf{h}^{\boldsymbol{k}}/(\partial x\partial y) = 0$ on $\partial\mathcal{D}^{\boldsymbol{k}}$ results in the exact same constraints. Meaning, by simply imposing continuity on the border, we also obtain continuous first-order derivatives. More importantly, it implies a well-defined divergence (of zero) along the border.

Equations 3.11 do not imply continuity in the higher-order derivatives, however these properties could be directly added to the constraint set, if desired. Since $\mathbf{h}^{\boldsymbol{k}}$ is composed exclusively of sinusoids, its derivatives are easily computed, leading to expressions similar to those in Equations 3.9. Any degree of smoothness, or more generally any homogeneous

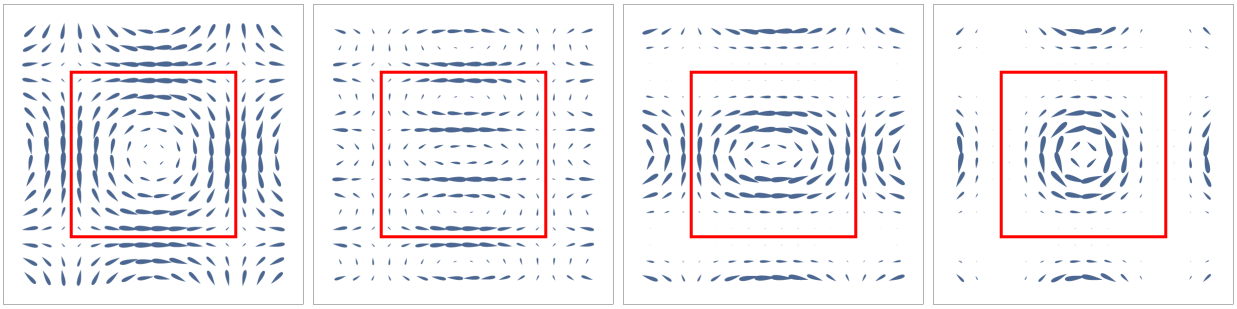


Figure 3.3. Combining eigenflows to zero-out flow along the boundary of $\mathcal{D}^{\boldsymbol{k}}$, shown in red. $\mathbf{e}^{(1,1)}$ exhibits the structure we desire for our basis (left), but has infinite support and is non-zero on the desired support boundary (and so clamping it would introduce discontinuities). $1/3\mathbf{e}^{(3,1)}$ has the same value along on the top and bottom boundaries (middle left), and subtracting it from $\mathbf{e}^{(1,1)}$ zeros the flow along the top and bottom (middle right). We repeat this step to zero-out flow along the entire boundary (right), and extend the entire process with more eigenflows to construct our basis templates.

linear boundary constraint on the derivatives of the basis, could thus be imposed on $\mathbf{h}^{\boldsymbol{k}}$. Doing so, however, would require a larger $\mathcal{A}$ to satisfy the added constraints, so we chose to not impose any additional smoothness constraints for our application.

3.4.3. Orthogonality per Frequency

Since $\mathcal{A}$ can be as large as necessary, Equations 3.11 can be solved with an arbitrarily large number of free coefficients. We exploit these extra coefficients to impose additional orthogonality properties.

If every basis in the domain were known in advance, and did not change during simulation, we could enforce full orthogonality between all basis pairs. This would result in a prohibitively large set of quadratic constraints for the coefficients $w_{\boldsymbol{a}}^{\boldsymbol{k}}$, mandating in turn the use of a very large $\mathcal{A}$: this is computationally impractical, creates bases with complex structures, and requires the computation of new bases each time the simulation domain changes.

To reduce the number of constraints, we compromise and only impose orthogonality between bases of the same frequency. Since two basis functions whose support do not intersect are trivially orthogonal, and our bases are tiled regularly with $\phi = 1/2$, we only need to locally impose orthogonality between any given basis and its eight neighbors (in 2D) of that same frequency. Due to symmetry, this reduces to only the three constraints:

$$\begin{cases} \int_{\cap_1} \mathbf{h}^{\boldsymbol{k}}(x,y) \cdot \mathbf{h}^{\boldsymbol{k}}(x,y-1/(2k_y)) & = 0 \\ \int_{\cap_2} \mathbf{h}^{\boldsymbol{k}}(x,y) \cdot \mathbf{h}^{\boldsymbol{k}}(x-1/(2k_x),y-1/(2k_y)) & = 0 \\ \int_{\cap_3} \mathbf{h}^{\boldsymbol{k}}(x,y) \cdot \mathbf{h}^{\boldsymbol{k}}(x-1/(2k_x),y) & = 0 \end{cases}, \tag{3.12}$$

where $\cap_1$, $\cap_2$ and $\cap_3$ are the support intersections of the two flows in each integrand. This system results in the quadratic constraints:

$$\begin{cases} \sum\limits_{\substack{\boldsymbol{a}_1\in\mathcal{A}\\ \boldsymbol{a}_2\in\mathcal{A}}} w^{\boldsymbol{k}}_{\boldsymbol{a}_1} w^{\boldsymbol{k}}_{\boldsymbol{a}_2} \int_{\cap_1} \mathbf{e}^{\boldsymbol{a}_1\boldsymbol{k}}(x,y)\cdot\mathbf{e}^{\boldsymbol{a}_2\boldsymbol{k}}\left(x,y-\frac{1}{2k_y}\right) & = 0 \\ \sum\limits_{\substack{\boldsymbol{a}_1\in\mathcal{A}\\ \boldsymbol{a}_2\in\mathcal{A}}} w^{\boldsymbol{k}}_{\boldsymbol{a}_1} w^{\boldsymbol{k}}_{\boldsymbol{a}_2} \int_{\cap_2} \mathbf{e}^{\boldsymbol{a}_1\boldsymbol{k}}(x,y)\cdot\mathbf{e}^{\boldsymbol{a}_2\boldsymbol{k}}\left(x-\frac{1}{2k_x},y-\frac{1}{2k_y}\right) & = 0 \\ \sum\limits_{\substack{\boldsymbol{a}_1\in\mathcal{A}\\ \boldsymbol{a}_2\in\mathcal{A}}} w^{\boldsymbol{k}}_{\boldsymbol{a}_1} w^{\boldsymbol{k}}_{\boldsymbol{a}_2} \int_{\cap_3} \mathbf{e}^{\boldsymbol{a}_1\boldsymbol{k}}(x,y)\cdot\mathbf{e}^{\boldsymbol{a}_2\boldsymbol{k}}\left(x-\frac{1}{2k_x},y\right) & = 0 \end{cases} . \tag{3.13}$$

We solve Equation 3.13's integrals analytically for each $\boldsymbol{k}$, obtaining a system of three quadratic equations for the coefficients $w^{\boldsymbol{k}}_{\boldsymbol{a}}$.

3.4.4. Solving the Constraints

From Sections 3.4.2 and 3.4.3, we search for the smallest set $\mathcal{A}$ that satisfies constraints 3.8, 3.11, and 3.13. In doing so, we arrive at an optimal harmonic set with $\mathcal{A}_\star = \{1,3,5\}$.

To solve (numerically) for the nine coefficients $w^{\boldsymbol{k}}_{\boldsymbol{a}}$ corresponding to this choice of $\mathcal{A}_\star$, we first solve the linear system (Equations 3.11 and 3.8) which expresses six of the $w^{\boldsymbol{k}}_{\boldsymbol{a}}$ as linear combinations of the other weights. Substituting this into the quadratic Equations 3.13, we are left with three quadratic equations of three unknowns. This cannot be solved exactly, as it requires solving roots of polynomials of degree eight, and so we obtain a numerical solution using the `NSolve` method from Mathematica[100] with the `"EndomorphismMatrix"` option.

From Bezout's theorem [90], there exist eight solutions to this system, leading us to a procedure for confirming whether our numerical process has found every solution. We discard solutions with complex coefficients and retain the (real) solution that minimizes $\|\mathbf{h}^{\boldsymbol{k}}\| := \sqrt{\int \mathbf{h}^{\boldsymbol{k}}\cdot\mathbf{h}^{\boldsymbol{k}}}$. This leads to a solution that minimizes the influence of higher harmonics, since we set $w^{\boldsymbol{k}}_{\mathbf{1}} = 1$.

3.4.5. Final Basis Templates

Having defined basis templates on $\mathcal{D}^{\boldsymbol{k}}$, we translate them back to their more convenient support $\mathcal{S}^{\boldsymbol{k}}$. We also normalize the bases to have unit norm. The final basis template definition is given by

$$\mathbf{b}^{\boldsymbol{k}}(x,y) = \left(1\Big/\|\mathbf{h}^{\boldsymbol{k}}\|\right)\ \mathbf{h}^{\boldsymbol{k}}(x-1/(2k_x),y-1/(2k_y)) \ . \tag{3.14}$$

A priori, it seems Equations 3.8, 3.11 and 3.13 must be solved for each fundamental frequency $\mathbf{k}$. However, if k_x and k_y have common factors, they can be factored out of the constraints, and the same solutions are obtained for fundamental frequencies (k_x,k_y) and $(\alpha\,k_x,\alpha\,k_y)\ \forall\alpha\in\mathbb{N}_{\geq 1}$. Therefore, we need only compute the harmonic coefficients once per *anisotropy ratio*, denoted $(k_x:k_y)$, and we only compute the templates for frequencies where

$\boldsymbol{a}$	Anisotropy Ratio		
	(1 : 1)	(2 : 1)	(4 : 1)
(1,1)	1.0000000000	1.0000000000	1.0000000000
(1,3)	-0.1107231463	0.0277351959	0.0336558844
(1,5)	-0.1335661122	-0.2166411175	-0.2201935306
(3,1)	-0.1107231463	-0.4866818264	-0.5578126029
(3,3)	0.1262767635	0.0543786840	-0.0036701213
(3,5)	-0.0536214289	0.0647091549	0.1137645934
(5,1)	-0.1335661122	0.0920090959	0.1346875617
(5,3)	-0.0536214289	-0.0381742496	-0.0045291041
(5,5)	0.0588860798	0.0045027306	-0.0242200499
$\|\mathbf{b}^{\widehat{\mathbf{k}}}\|$	0.9783644776	1.3121697019	1.7797075185

Table 3.1. Harmonic weights $w_{\boldsymbol{a}}^{\widehat{\boldsymbol{k}}}$ and scaling coefficent $\|\mathbf{b}^{\widehat{\mathbf{k}}}\|$ for anisotropy ratios $(1:1)$, $(2:1)$ and $(4:1)$ solving the constraints of Section 3.4.

$\min(\boldsymbol{k}) := \min(k_x,k_y) = 1$. Also, due to symmetry, coefficients need only be computed for ratios $(\alpha:\beta)$ with $\alpha \leq \beta$, since other basis templates can be obtained by rotation.

Finer bases with $\min(\boldsymbol{k}) > 1$ can be obtained by scaling. From the eigenflow definition in Equation 3.6, we have $\boldsymbol{e}^{\beta\boldsymbol{k}}(\boldsymbol{x}) = \beta \boldsymbol{e}^{\boldsymbol{k}}(\beta\boldsymbol{x})$ for any scalar β, which leads to the scaling relation

$$\mathbf{b}^{\boldsymbol{k}}(\boldsymbol{x}) = \min(\boldsymbol{k})\, \mathbf{b}^{\widehat{\boldsymbol{k}}}(\min(\boldsymbol{k})\, \boldsymbol{x}) \ , \tag{3.15}$$

where $\widehat{\boldsymbol{k}} = \boldsymbol{k}/\min(\boldsymbol{k})$. The few remaining basis templates that need to be explicitly evaluated are stored on a fine grid, and we evaluate these basis templates with tabulation (i.e., GPU texture lookups).

This ability to reuse basis templates is a major advantage of our method. As noted by Jones et al. [101], model reduction methods based on simulation snapshots usually store bases at the simulation grid resolution, incurring prohibitive memory costs. Our method only stores one basis template per anisotropy ratio, and only the centers and frequencies are needed to represent translated bases.

We provide the harmonic coefficients and normalization factors for frequency ratios $(1:1)$, $(2:1)$ and $(4:1)$ in Table 3.1, and the corresponding bases are illustrated in Figure 3.4. As desired, this basis set is divergence-free, has local support, and can be tiled at any scale while remaining orthogonal within each frequency layer.

3.4.6. Bases in 3D

We extend our construction method to 3D. The main difference is that 3D eigenflows are defined in three separate groups, aligned along each spatial axis. This falls from the fact that vorticity is a scalar value in 2D, but requires three components in 3D. We therefore construct

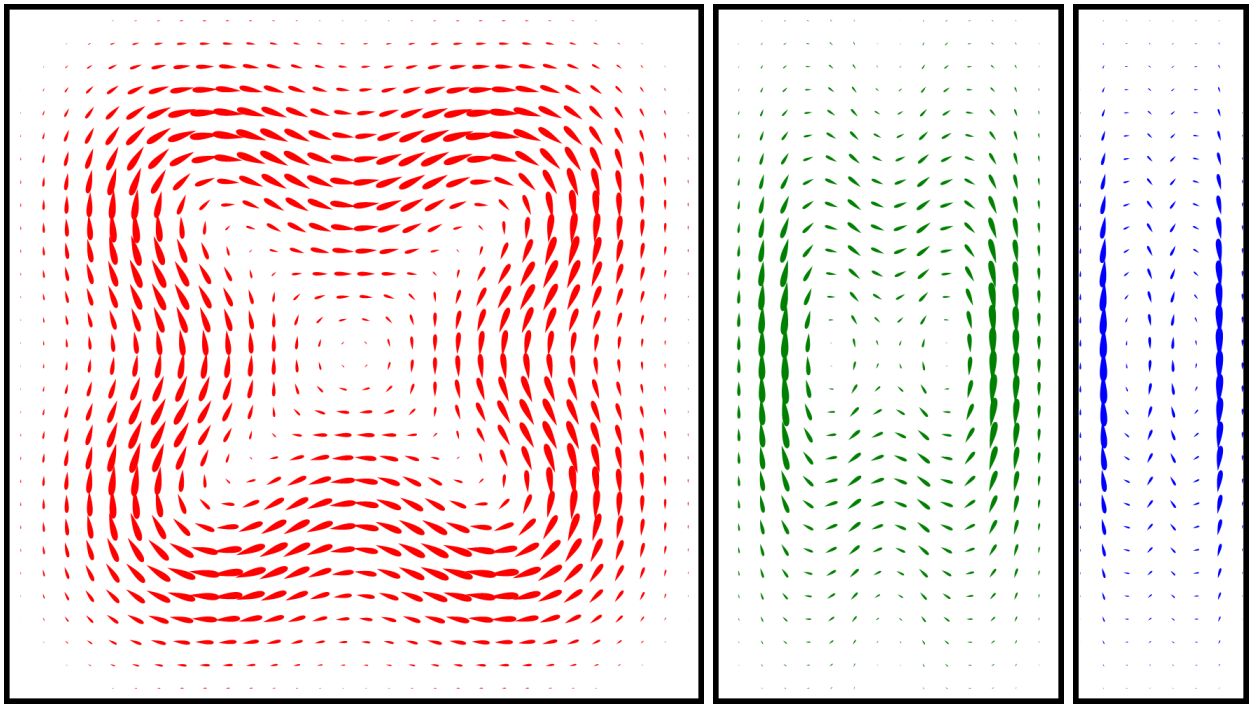


Figure 3.4. Basis flow templates for anisotropy ratios $(1:1)$, $(2:1)$ and $(4:1)$.

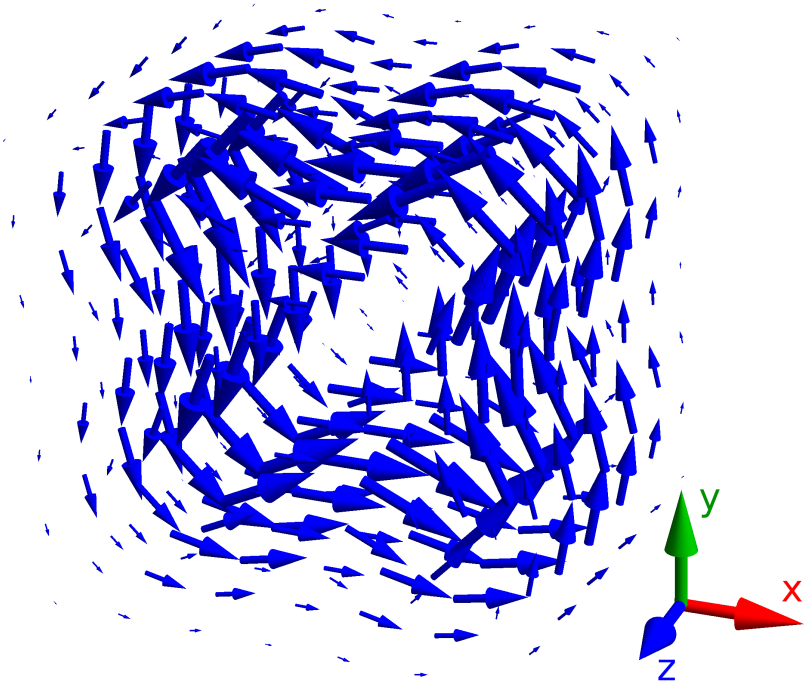


Figure 3.5. Z-aligned 3D basis flow with frequency (1,1,1), constructed from the extrusion of the 2D flows.

a basis template $\mathbf{b}_z^{\boldsymbol{k}}$ aligned along the z-axis, and rotate it to create basis templates $\mathbf{b}_x^{\boldsymbol{k}}$ and $\mathbf{b}_y^{\boldsymbol{k}}$ aligned along the x- and y-axes. We tile the simulation domain as before, but now each $\mathbf{c} \in \mathcal{C}^{\boldsymbol{k}}$ is the center of three collocated bases (one per axis).

Basis construction is more involved in 3D since not all the eigenflows are linearly independent. Fortunately, we arrive at a very simple solution, which we discuss on an intuitive level, below; we provide full mathematical derivations in Appendix A (Section 3.9).

To construct $\mathbf{b}_z^{\boldsymbol{k}}$ with frequency $\boldsymbol{k} = (k_x, k_y, k_z)$, we reuse the 2D basis template definition in (x,y) and extrude it along z. We define

$$\mathbf{b}_z^{\boldsymbol{k}}(x,y,z) = 2k_z \cos(k_z \pi z)^2 \begin{pmatrix} \left(\boldsymbol{b}^{(k_x,k_y)}(x,y)\right) \cdot (1,0) \\ \left(\boldsymbol{b}^{(k_x,k_y)}(x,y)\right) \cdot (0,1) \\ 0 \end{pmatrix} , \tag{3.16}$$

which is divergence-free by construction. The weighting in z is necessary to zero-out flow along the z boundaries of the basis' support. This choice of weighting function is obtained from the general basis construction method in Appendix A (Section 3.9). The $2k_z$ factor normalizes the basis, since

$$\iiint_{\mathcal{S}^{(k_x,k_y,k_z)}} \mathbf{b}_z^{(k_x,k_y,k_z)} \cdot \mathbf{b}_z^{(k_x,k_y,k_z)} \tag{3.17}$$

$$= \int_{-1/2k_z}^{1/2k_z} 2k_z \cos(k_z \pi z)^2 \iint_{\mathcal{S}^{(k_x,k_y)}} \mathbf{b}^{(k_x,k_y)} \cdot \mathbf{b}^{(k_x,k_y)} \tag{3.18}$$

$$= \int_{-1/2k_z}^{1/2k_z} 2k_z \cos(k_z \pi z)^2 = 1 \ . \tag{3.19}$$

We illustrate this basis in Figure 3.5 for frequency (1,1,1) and still need only compute bases where $\min(\boldsymbol{k}) = 1$, as detailed in Section 3.4.5. The scaling relation of Equation 3.15 is, however, replaced in 3D by

$$\mathbf{b}_z^{\boldsymbol{k}}(\boldsymbol{x}) = (\min(\boldsymbol{k}))^{3/2} \ \widehat{\mathbf{b}_z^{\boldsymbol{k}}}(\min(\boldsymbol{k}) \ \boldsymbol{x}) \ . \tag{3.20}$$

3.5. Model-Reduced Fluid Dynamics

We will now detail an efficient model-reduced fluid solver using our basis. In most model reduction methods, the dynamics are computed directly with Equation 3.4, which presents many challenges.

First, the advection tensor $\mathbf{A}$ can be very large. With our basis, however, basis locality introduces significant sparsity into $\mathbf{A}$, as elements of $\mathbf{A_{ijl}}$ are zero if the support of any two basis functions do not intersect. Still, $\mathbf{A}$ contains many non-zero entries, and computing interactions for each of the triplet of bases remains expensive.

Second, there is typically no guarantee that the dynamics equations will project well onto each basis flow, mainly due to Equation 3.4 modeling the linearized instantaneous behavior of the fluid: it is defined in a space decoupled from the simulation, and cannot necessarily be

accurately captured with a basis designed to represent the velocity. For instance, the term $\nabla \mathbf{b}_i$ in Equation 3.5 is not divergence-free, so projecting it onto a divergence-free basis is not likely to be accurate.

For these reasons, we avoid direct use of the advection tensor and diffusion matrix. Instead, we design a simulation method tailored to our basis, with a focus on synthesizing the key behaviors of realistic smoke. Specifically, Figure 3.6 illustrates the *energy cascade* [91] of a fluid undergoing motion, which in turn describes how a fluid's energy evolves in time; motivated by this cascade, we identify the following behaviors we wish to capture during simulation:

- energy is introduced into the system through, e.g., buoyancy forces or stirring motions, corresponding to $\widetilde{\boldsymbol{f}}$ in Equation 3.4,
- energy can be transported in the simulation domain without changing its frequency content, corresponding to rigid transformations of fluid structures in the flow; this component of transport is usually encoded as a part of the advection tensor $\mathbf{A}$,
- energy can also be transferred *between* energy levels, mostly from large structures and vortices decaying into smaller ones under deformation forces; this is also usually encoded as part of $\mathbf{A}$, and
- at the finest scale, energy is dissipated by viscosity; this is usually obtained by applying the dissipation tensor $\mathbf{D}$.

Our simulator represents velocity fields in our basis and simulates all four behaviors by evolving the reduced flow coefficients in time. Smoke particles are passively advected by the velocity field. We detail each of the simulation stages for the phenomena listed above in the following sections.

3.5.1. Projecting External Forces

As discussed in Section 3.3, any vector field $\mathbf{f} \in \mathcal{F}$ can be projected onto the basis $\mathcal{R}$ as $\widetilde{\mathbf{f}} = (\mathbf{B}^{\mathrm{T}}\mathbf{B})^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{f}$. First, computing $\mathbf{B}^{\mathrm{T}}\mathbf{f}$ involves computing $\int_\Omega \mathbf{b}_i \cdot \mathbf{f}$ for every basis function $\mathbf{b}_i$. Both $\mathbf{f}$ and $\mathbf{b}_i$ are defined as continuous functions on Ω, but we use a finite grid to numerically compute the integrals. We use a grid with (constant) resolution and size set to the support of each $\mathbf{b}_i$. This way, integrals involving bases of any scale require the same computational time. The grid resolution is set so as to accurately represent all basis features, assuming the force field has features at least as large as those of the bases. We found that an axial resolution of 32 is sufficient and note that, since bases of the same anisotropy have the same definition (up to a scale and offset), we can efficiently compute integrals involving these bases concurrently.

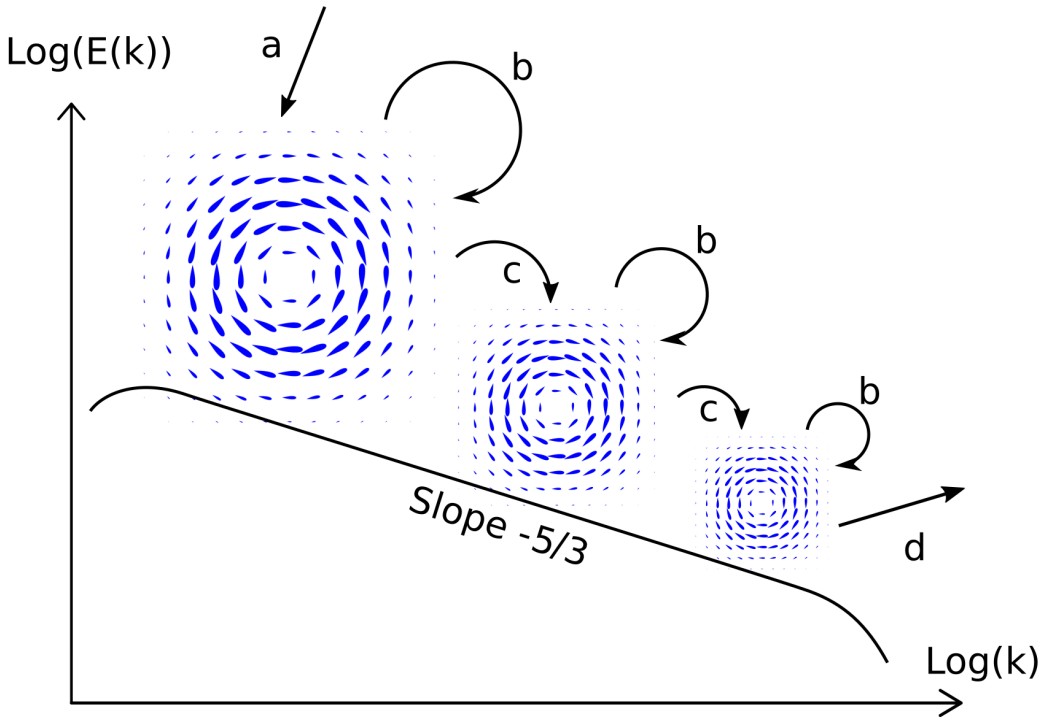


Figure 3.6. Energy cascade from Kolmogorov theory summarizing the key behaviors of a turbulent flow: (a) energy enters the system from external forces, it is transported within (b) and across (c) frequency bands, before dissipating at finer scales (d). This theory predicts an energy distribution proportional to $k^{-5/3}$, for a wavenumber k.

Second, we use Gauss-Seidel iterations solve the linear system to invert $(\mathbf{B}^{\mathrm{T}}\mathbf{B})$. This approach is often slower than other basic schemes, e.g., Jacobi iterations, since entries cannot generally be processed in parallel; however, Gauss-Seidel iterations are guaranteed to converge in this case since the matrix is symmetric positive definite, while Jacobi iterations offer no convergence guarantee and require smaller time steps [88].

Given our basis' local support, $(\mathbf{B}^{\mathrm{T}}\mathbf{B})$ is sparse, and the Gauss-Seidel iterations only need to be applied on non-orthogonal neighboring bases. Note that neighboring bases are easy to find since our tiling scheme is regular. Furthermore, the orthogonality properties we encode into our basis construction further accelerates computation by reducing the number of non-zero coefficients in $(\mathbf{B}^{\mathrm{T}}\mathbf{B})$. But, more importantly, our orthogonality property creates a *multicolor scheme* [113] where bases from the same orthogonality group can be iterated on in parallel. This is a powerful advantage of our basis structure, since it allows us to invert $(\mathbf{B}^{\mathrm{T}}\mathbf{B})$ with the stability of a Gauss-Seidel scheme and the parallelism of a Jacobi scheme.

Other iterative method could also be used to invert the system while still taking advantage of this multicolor property, for instance, when computing preconditioners for the conjugate gradient method [88]; however, we found that simple Gauss-Seidel iterations are sufficient and yield a stable and parallelizable method to solve the inversion. This is crucial

to the performance of any model reduced simulator, as force projection is one of the more computationally expensive steps. For every result (see Section 3.7), we use no more than 10 Gauss-Seidel iterations to accurately project external forces onto our basis.

3.5.2. Rigid Transport and Rotation

The terms $\mathbf{A}_{ijl}$ in Equation 3.5 can be interpreted as the influence of basis j on basis i, with the result projected onto basis l. We use this interpretation to simulate the transport and rotation of bases.

Although our bases have a fixed position, we can simulate their motion by interpreting them as rigid objects free to move about the simulation domain. Basis $\mathbf{b}_j$'s influence on basis $\mathbf{b}_i$ is evaluated as

$$\mathbf{t}_{ij} = \left(\int_{\mathcal{S}_i} \mathbf{b}_j(\boldsymbol{x})\, d\boldsymbol{x}\right) \Big/ \left(\int_{\mathcal{S}_i} d\boldsymbol{x}\right). \tag{3.21}$$

The total displacement of $\mathbf{b}_i$ can therefore be evaluated by summing the influence of all neighboring bases as $\sum_j \widetilde{u}_j\, \mathbf{t}_{ij}$. We obtain the new basis center by scaling this displacement by the time step Δt.

In general there are no bases centered at the new location, but we can represent the translated basis as an interpolation, using an important property of eigenflows: just as translated sinusoids can be expressed using a combinations of sinusoids spaced by $\pi/2$ as

$$\begin{aligned}\sin(x-\alpha) &= \cos(\alpha)\sin(x) - \sin(\alpha)\cos(x) \\ &= \sin(\pi/2-\alpha)\ \sin(x) + \sin(\alpha)\sin(x-\pi/2)\end{aligned}$$

for any scalar offset α, we express any translated eigenflow *exactly* as a combination of eigenbases on a regular lattice. For a lattice with edge length ϕ, we have (in 2D for scalars $\alpha, \beta \in [0,\phi]$),

$$\mathbf{e}_{(\alpha,\beta)} = \csc(\pi\phi)^2 \sum_{i,j\in\{0,1\}} \sin(\pi|i\phi-\alpha|)\sin(\pi|j\phi-\beta|)\, e_{(i\phi,j\phi)}. \tag{3.22}$$

A similar expression also holds in 3D. Since our bases are constructed from eigenflows, are arranged on a regular lattice, and the influence of the fundamental harmonic is made as prominent as possible, this shifting property also approximately holds for our basis. After updating the translated basis' center, we project it onto its four nearest bases on the lattice (eight, in 3D; see Figure 3.7.) Instead of the exact weights in Equation 3.22, we use linear weights to obtain

$$\mathbf{b}_{(\alpha,\beta)} \approx \sum_{i,j\in\{0,1\}} |i-\alpha/\phi|\,|j-\beta/\phi|\, b_{(i\phi,j\phi)}. \tag{3.23}$$

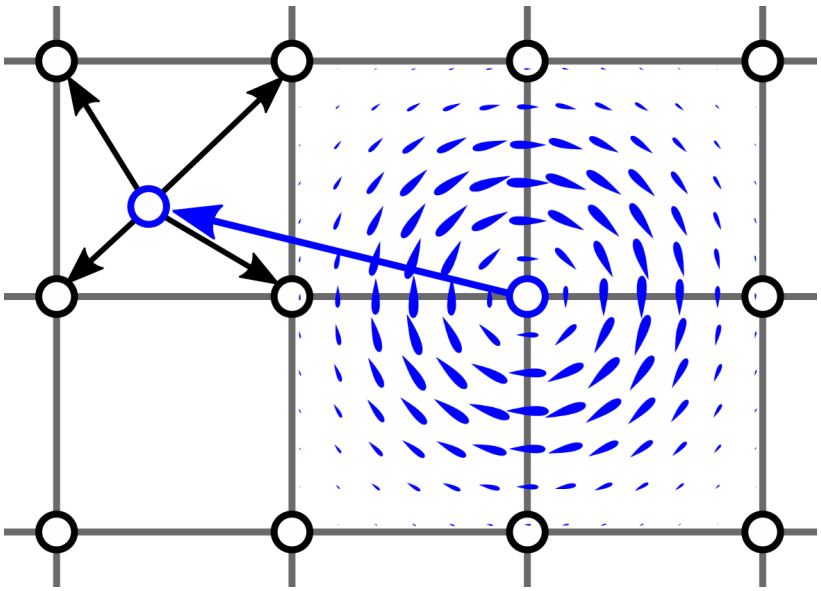


Figure 3.7. Basis transport method of Section 3.5.2. A basis is transported in the flow to a location where no basis of the same frequency exists. Since bases cannot actually move, its weight is distributed to its four (eight in 3D) neighbors, following the interpolation of Equation 3.23.

These weights are simpler to evaluate and are a good approximation to the trigonometric weights of Equation 3.22 (within 4% error when using the $\phi = 1/4$ modification of Section 3.6). Another added benefit of this weighting is that, while the trigonometric weights preserve the L^2 norm of the decomposition, linear weights preserve the L^1 norm: since we consider linear combinations of translated bases, the linear weights lead to an energy preserving method, while the trigonometric weights do not. For instance, if a single basis is transported in the domain, distributed to the four closest bases on the lattice (in 2D), and those four corners are then all transported back to the starting basis location, trigonometric weighting will increase the basis coefficient's magnitude while linear weighting will result **exactly** in the initial coefficient value.

We apply a similar strategy to rotate bases in 3D: by assuming that bases rotate about their center, we can compute the total rotation of basis $\mathbf{b}_i$ caused by the influence of all other bases as

$$\sum_j \widetilde{u}_j \, \mathbf{r}_{ij} \text{ with } \mathbf{r}_{ij} = \left(\int_{\mathcal{S}_i} \frac{(\boldsymbol{x} - \mathbf{c}_i) \times \mathbf{b}_j(\boldsymbol{x})}{\|\boldsymbol{x} - \mathbf{c}_i\|^2} d\boldsymbol{x} \right) \Big/ \left(\int_{\mathcal{S}_i} d\boldsymbol{x} \right). \tag{3.24}$$

We express the resulting rotation as a vector (rotation axis & amplitude) multiplied by the time step. Since each basis is collocated in a triplet of bases aligned along x, y and z, we have an orthogonal frame in which we can express the rotated axis. The coefficient of the rotated basis is thus transferred to the three collocated bases, and we obtain an energy preserving method by normalizing the L^1 norm of this coefficient transfer. Note that we

ignore rotations in 2D since bases comprise roughly circular vorticity profiles, and therefore do not significantly change under rotation.

3.5.3. Energy Transfer and Diffusion

So far, we have only treated energy transfers *within* a frequency layer. Next, we show how to also transfer energy *across* frequencies, which is fundamental for simulating complex, turbulent behaviors.

As illustrated for 2D in Figure 3.8, we arrange different frequency layers in a regular graph. To simulate forward energy cascades, each basis *receives* energy from its coarser neighboring bases and *transfers* energy to its finer neighboring bases. Not all bases are able to process this regular energy transfer: first, bases on the coarsest layer cannot receive energy from any coarser layers, however they instead receive energy from forces in the system (Section 3.5.1); second, bases at the highest frequency have no bases to transfer their energy to. Here, we simply remove this energy from the system, replicating the dissipation of energy at the finest scales and removing the need for an explicit application of the diffusion matrix $\mathbf{D}$ in Equation 3.4.

For all other bases (i.e, in the "middle" layers), we can control the rate at which energy is transferred: let $\|\boldsymbol{k}\| = \sqrt{k_x^2 + k_y^2}$ be the scalar *wavenumber* associated to a given frequency layer, let $\zeta(\boldsymbol{k})$ be the total energy in layer $\boldsymbol{k}$, and let $\tau(\boldsymbol{k})$ be the portion of the energy the layer transfers away at each time step. From Kolmogorov theory [91], we know that the physically-correct distribution of energy should be proportional to $\|\boldsymbol{k}\|^{-5/3}$, and we design a transfer mechanism with a steady state that follows this distribution.

Of the energy $\zeta(\boldsymbol{k}_0)\,\tau(\boldsymbol{k}_0)$ transferred by frequency $\boldsymbol{k}_0$, let $q(\boldsymbol{k}_0 \rightarrow \boldsymbol{k})$ be the proportion given to frequency $\boldsymbol{k}$. Particularly, $q(\boldsymbol{k}_0 \rightarrow \boldsymbol{k}) = 0$ if no energy is transferred between these layers. Note that, since our transfer mechanism is regular across all frequency layers, we have $\sum_{\boldsymbol{k}_0 \in \mathcal{K}} q(\boldsymbol{k}_0 \rightarrow \boldsymbol{k}) = 1$. We choose to distribute energy equally across all other layers of frequency at most double that of layer $\boldsymbol{k}_0$ (in any direction), as depicted in Figure 3.8. Each layer, thus, transfers energy to three other layers in 2D, or seven other layers in 3D.

Let $\tau(\boldsymbol{k}) = \lambda \|\boldsymbol{k}\|^{\epsilon}$ for scalars λ and ϵ. The energy transferred and received by a frequency layer should be equal at steady state, and so

$$\zeta(\boldsymbol{k})\,\tau(\boldsymbol{k}) = \sum_{\boldsymbol{k}_0 \in \mathcal{K}} \zeta(\boldsymbol{k}_0)\,\tau(\boldsymbol{k}_0)\,q(\boldsymbol{k}_0 \rightarrow \boldsymbol{k}) \tag{3.25}$$

$$\Leftrightarrow \|\boldsymbol{k}\|^{-5/3}\,\lambda\,\|\boldsymbol{k}\|^{\epsilon} = \sum_{\boldsymbol{k}_0 \in \mathcal{K}} \|\boldsymbol{k}_0\|^{-5/3} \lambda\,\|\boldsymbol{k}_0\|^{\epsilon}\,q(\boldsymbol{k}_0 \rightarrow \boldsymbol{k}) \tag{3.26}$$

and choosing $\epsilon = 5/3$ yields the desired steady state distribution. The λ parameter cancels out and does not affect the steady state, but it controls the rate at which the energy distribution approaches the steady state. Note that λ must be small enough so that $\tau(\boldsymbol{k}) < 1$ for

all frequencies, but the energy transfer can be performed in multiple passes if faster transfer rates are desired.

In practice, we transfer energy between bases, and not between entire frequency layers. We must therefore define the energy of a single basis, which is not trivial since not all bases are orthogonal. Furthermore, the natural choice of using $(\widetilde{u}_i)^2$ as the energy of a basis would require non-linear transfers between basis coefficients, and will not distinguish between clockwise and counter-clockwise swirls. We instead directly use $\widetilde{u}_i$ as a signed replacement for the energy of basis i. Given this, we withdraw energy from basis i by simply reducing its coefficient value by $\widetilde{u}_i\,\tau(\boldsymbol{k}_0)$.

This is a coarse approximation of the physically-correct behavior, but it leads to satisfactory energy cascades in practice. Note that, since energy measures are now linear instead of quadratic, the expected energy distribution exponent must be adjusted to $\epsilon = 5/6$. The steady-state of Equation 3.26 does not exactly hold with the signed energy, but $\epsilon = 5/6$ still gives a good default value about which other energy distributions can be explored.

We distribute each energy "packet" $\widetilde{u}_i\ \tau(\boldsymbol{k}_0)\ q(\boldsymbol{k}_0\ \rightarrow\ \boldsymbol{k})$ to all neighboring bases of frequency $\boldsymbol{k}$; let $\mathcal{B}^{\boldsymbol{k}}$ be the indices of all bases with frequency $\boldsymbol{k}$. We define the proportion given to each basis as

$$v(\boldsymbol{b}_i \rightarrow \boldsymbol{b}_j) = (B^T B)_{ij} \Big/ \textstyle\sum_{l \in \mathcal{B}^{\boldsymbol{k}}} |(B^T B)_{il}| \ . \tag{3.27}$$

With this weighting, a basis will transfer most of its energy to neighboring bases with similar structures (i.e., as their inner product approaches 1), leading to natural deformations. Note that the numerator above must be signed, otherwise a clockwise swirl could decay unnaturally into a counter-clockwise swirl.

Energy is finally transferred from basis i of frequency $\boldsymbol{k}_0$ to basis j of frequency $\boldsymbol{k}$ by adding $\widetilde{u}_i\,\tau(\boldsymbol{k}_0)\,q(\boldsymbol{k}_0 \rightarrow \boldsymbol{k})\,v(\boldsymbol{b}_i \rightarrow \boldsymbol{b}_j)$ to the coefficient of basis $\boldsymbol{b}_j$.

Figure 3.9 shows the effect of varying energy transfer parameters λ and ϵ in our energy transfer method. In general, smaller lambdas cause the simulation to be dominated by advection (Figure 3.9a). Larger lambdas instead dissipate energy more quickly, causing the simulation to be dominated by buoyancy forces (Figure 3.9d). Using a moderate λ (Figure 3.9b and c) leads to more interesting energy transfers. Using $\epsilon = 5/6$ as previously suggested then leads to a natural smoke plume, while using $\epsilon = -11/6$ leads to a more erratic, exaggerated behavior.

3.5.4. Reusing Dynamics Coefficients

The entire simulation dynamics are reduced to computing the *interaction coefficients* $\mathbf{t}_{ij}$, $\mathbf{r}_{ij}$, and $(\mathbf{B}^{\mathrm{T}}\mathbf{B})_{ij}$. Note that, contrary to the original advection tensor $\mathbf{A}$ that operates on basis triplets, our interaction coefficients only involve basis *pairs*, greatly reducing the number of coefficients to compute and store.

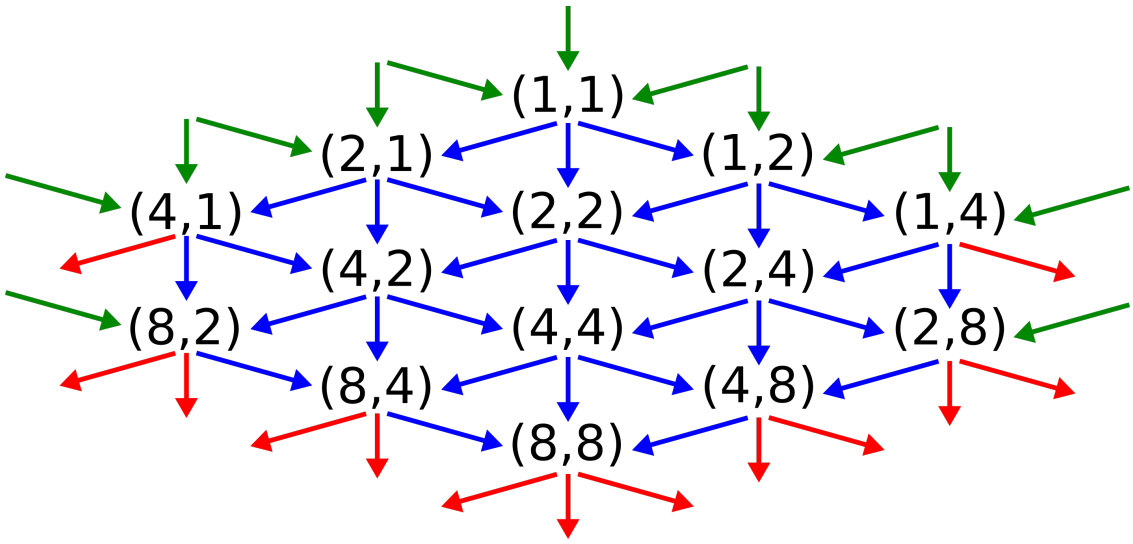


Figure 3.8. Our energy distribution graph (Section 3.5.3.) Energy enters the system at the coarsest frequencies (green arrows). At each time step, every basis distributes part of its energy to the neighbors indicated with blue arrows. At the finest scale, energy cannot be transferred to other bases, so we remove it from the system to simulate dissipation (red arrows).

Furthermore, we can reuse most of the interaction coefficients since the interaction of two bases only depends on their *relative* position, and our bases are distributed regularly throughout the domain. Similarly, interactions only depend on the relative scale of the bases involved, and so they can be computed once for coarse frequencies and reused for higher frequencies.

We apply a lazy evaluation scheme, re-scaling the two bases involved in an interaction so that their largest dimension has unit length, before translating them to center basis i at (0,0). We maintain a dictionary of all precomputed interaction coefficients, indexed by the relative sizes and positions of the resized bases. If the interaction coefficient for the resized bases is not present in the dictionary, it is computed (as detailed earlier) and added to the dictionary.

Each interaction coefficient scales to higher frequencies as a simple function of the scaling ratio. These relations are derived directly from Equations 3.15 and 3.20, and summarized in Table 3.2. The interaction coefficient for the resized bases is thus re-scaled back to the original basis size, and used to evolve the fluid's dynamics.

Note that no separate precomputation step is required before beginning the simulation, since the dictionary is naturally built during the first time step. This makes implementation simple, and ensures we only compute the interaction coefficients that are actually needed during simulation. Since basis interactions are local and do not depend on the simulation domain, we can save the dictionary and reuse it for any other simulation domain.

(a) $\epsilon = 0.6$, $\lambda = 0.01$ (b) $\epsilon = 0.8$, $\lambda = 0.05$ (c) $\epsilon = -1.8$, $\lambda = 0.05$ (d) $\epsilon = 0.6$, $\lambda = 0.2$

Figure 3.9. Smoke plumes with different energy transfer parameters.

Dictionary lookups can form a computational bottleneck, and so it is helpful to locally cache the coefficients that each basis requires during the first simulation step. If new bases are added during simulation and an interaction coefficient is not cached locally, a dictionary lookup is performed and added to the cache. This reduces computation time by a factor of $10 - 50\times$ depending on the scene, at the cost of a $1.5 - 4\times$ increase in memory footprint.

3.6. Improved Coverage and Obstacle Coupling

In standard basis construction (Section 3.4), we spaced every basis function on the same frequency layer by half its support size ($\phi = 1/2$), however this can lead to gaps in coverage: each layer will contain points where the flow is always zero (see Figure 3.10, left).One solution is to increase the number of bases in each layer (Figure 3.10, right): e.g., doubling the number of bases and separating them by a quarter their support ($\phi = 1/4$) would improve coverage,

Coefficient	Scaling Factor Exponent (α)	
	2D	3D
$\mathbf{B}^T\mathbf{B}$	0	0
$\boldsymbol{t}$	1	3/2
$\boldsymbol{r}$	—	5/2

Table 3.2. Factors for scaling interaction coefficients in Section 3.5.4. We obtain coefficients for bases with $\min(\boldsymbol{k}) > 1$ by multiplying the corresponding precomputed coefficient by $(\min(\boldsymbol{k}))^\alpha$, where α is given above.

but at the cost of additional computation. Note that this doesn't significantly affect the orthogonality properties of the basis; each frequency layer now has four orthogonal groups, but each group can still be processed in parallel during force projections (see Section 3.5.1).

Another solution is to maintain $\phi = 1/2$ spacing, but to offset basis layers independently with respect to each other, in each axis, by a quarter of the spacing of all coarser layers: i.e.,

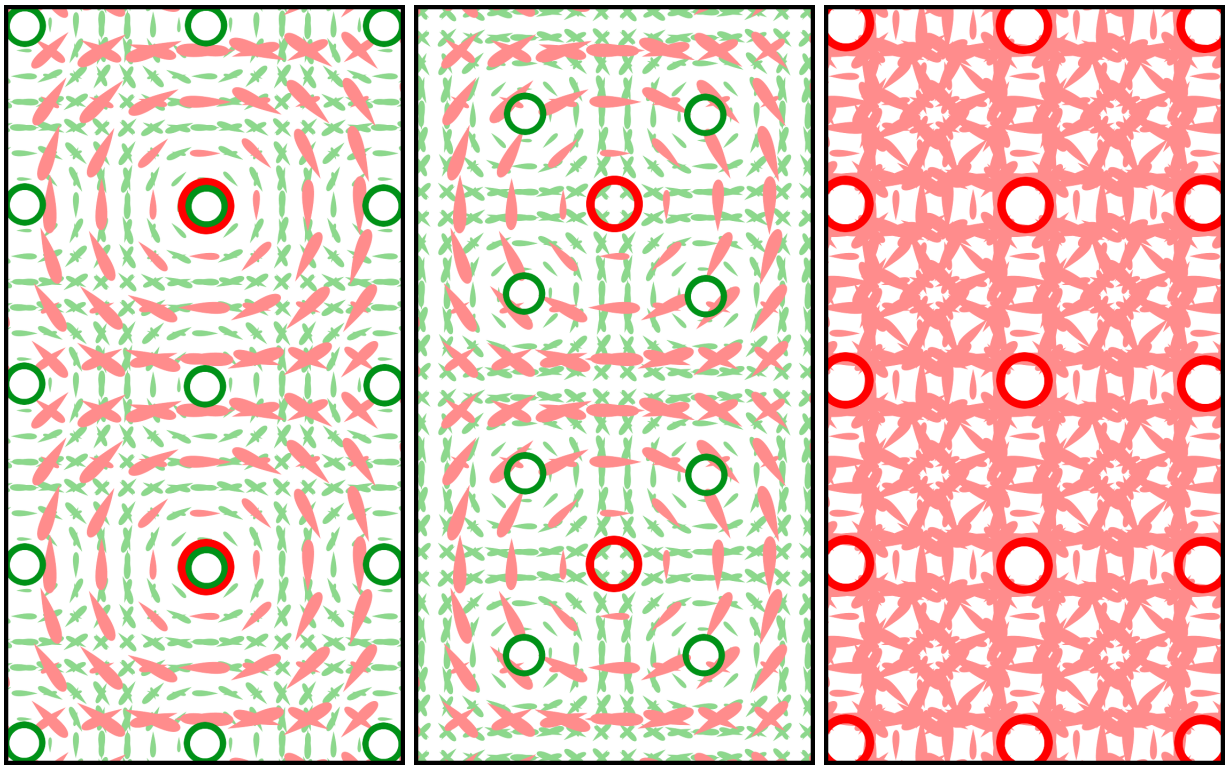


Figure 3.10. With $\phi = 1/2$ and no offset, the coverage across two layers ($\boldsymbol{k} = (1,1)$ in red and $\boldsymbol{k} = (2,2)$ in green) has points of zero flow (left, at the intersection of red and green centers). By offsetting bases in each layer, finer layers can cover the gaps left by coarser layers (middle). Alternatively, using $\phi = 1/4$ creates a much denser coverage (only $\boldsymbol{k} = (1,1)$ is shown; right).

an offset of ($o(k_x)$,$o(k_y)$), where

$$o(k) = \sum_{i=0}^{(\log_2 k)-1} \frac{\phi}{4} \left(\frac{1}{2}\right)^i = \frac{\phi\,(1 - 1/k)}{2}. \tag{3.28}$$

Figure 3.10 (middle) illustrates how using this approach improves coverage without increasing the number of bases (or computation cost). Note also that this offsetting scheme maintains the regular structure of bases across frequency layers, and so all basis reusability advantages remain (Section 3.5.4). Unfortunately, this offsetting strategy is incompatible with anisotropic bases in 3D: since we apply the offset independently for each axis, bases with different alignments get a different offset, and we lose the property required in Section 3.5.2 for bases to be grouped as triplets of collocated bases.

For all results in this paper, we use $\phi = 1/4$ as it provides significantly better coverage than half-support separation. The improvement in coverage is even more significant in 3D because of the three collocated bases at each point. For that reason, we did not find necessary to use anisotropic bases in 3D with the $\phi = 1/4$ modification. We therefore also use the offset strategy for all our results, which improves coverage even more at no additional cost.

User-driven Coverage. We can additionally apply a spatially varying basis placement tailored, e.g., to scene complexity or artist-driven simulation constraints. For example:

- if the simulation domain has both large and narrow regions, it may be sufficient to place finer-scale bases exclusively in the narrow regions and near boundaries, or
- if we desire view-dependent simulation accuracy/refinement, we can place higher-frequency bases closer to the camera (i.e., where a viewer is most likely to notice finer-scale details), or
- we can place bases only where smoke particles are present.

These three strategies are illustrated in Figure 3.11 and used in the results of Figures 3.17 and 3.14.

Note that some strategies may require adding and/or removing bases during simulation; since we precompute all local interactions once (Section 3.5.4), doing so does not incur any significant overhead.

3.6.1. Curved Boundaries

We adapt our simulator to new domain shapes by tiling the domain with our bases. Given the square (cubical, in 3D) basis support, however, and their regular lattice positioning, our basis does not naturally adapt to curved boundaries or obstacles: Figure 3.12 (bottom left, dotted line) illustrates "staircasing" artifacts due to poor coverage near boundaries for coarser simulations.

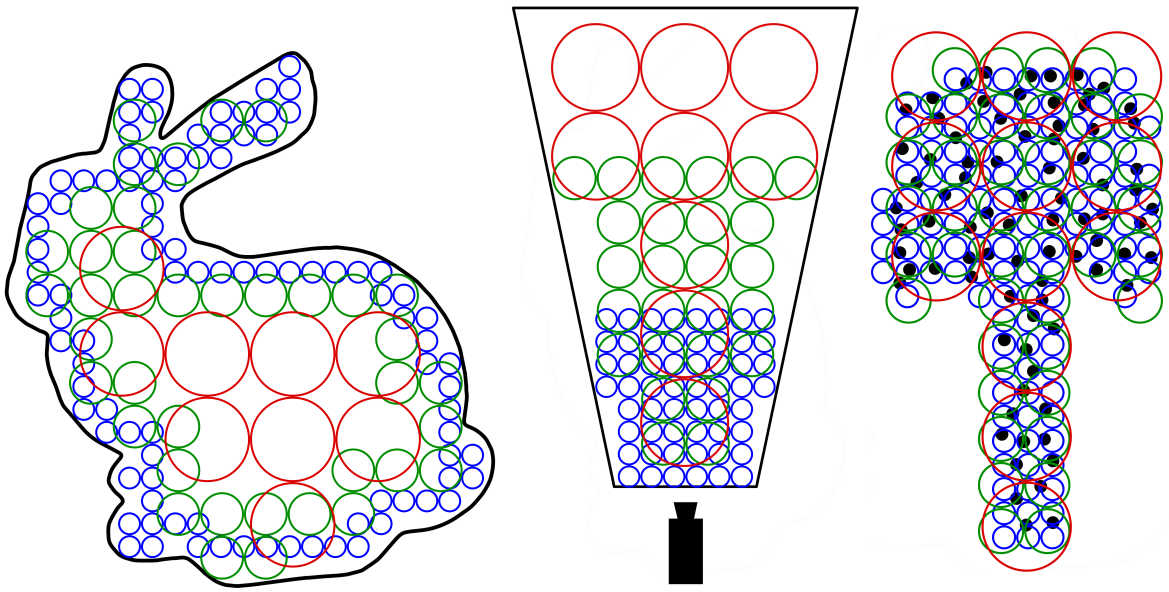


Figure 3.11. Spatially-varying coverage strategies. Left to right: placing finer-scale bases closer to boundaries; view-dependent coverage generates richer dynamics closer to the viewer; basis placement biased to regions with particles, since velocities further from particles contribute less to their behavior.

We propose a basis deformation scheme to solve this problem: instead of only tiling bases in the simulation domain Ω, we allow bases to overlap boundaries, as long as the distance from their center to the nearest boundary exceeds 1/4 their support.

We represent boundaries and obstacles with signed distance functions (SDF), before displacing and warping bases that overlap boundaries, as follows: For each corner of the basis located inside the obstacle, we uniformly squash the basis along the direction of the SDF gradient, leaving the opposite corner fixed, until the initial corner is out of the obstacle. We then uniformly stretch the basis in the direction perpendicular to the SDF gradient to recover the original area (volume in 3D) of the basis. The deformed basis, even after area preservation, may no longer be divergence-free [1], but it nevertheless helps reduce staircase aliasing (Figure 3.12, bottom left dotted line) and results in a more visually pleasing vector field (i.e., one where particles do not seem to be "compressed" in free-space for no apparent reason.) Note that, in stretching the basis, we sometimes force another corner out of the simulation bounds. In practice, we conduct a squashing pass to try to move each corner out with post-stretching, and then a second pass that displaces them without post-stretching, to ensure all final corners are inside of the simulation domain.

1. If only one corner is displaced, the deformation is a linear stretch, and so the deformed basis remains divergence-free, but this is not necessarily true if more than one corner is moved.

Given the deformed basis support, we map the original flow $\boldsymbol{b}$ to a point $\boldsymbol{x}$ of the new support as $\boldsymbol{M}\left(\boldsymbol{b}\left(\boldsymbol{M}^{-1}\left(\boldsymbol{x}\right)\right)\right)$, where $\boldsymbol{M}$ is the bilinear transform from the original axis-aligned support to the deformed support. We compute $\boldsymbol{M}^{-1}$ using Newton iterations, where no more than five iterations are required in all our tests.

In principle, new interaction coefficients should be computed for each of the deformed bases. This would however be computationally expensive, and we instead simply use the coefficients of the original square bases. This further approximates the fluid behavior, but still yields satisfactory results in practice.

3.6.2. Obstacle Coupling

In order for moving obstacles to influence our simulations, we project the normal velocity at the boundary of dynamic obstacles onto our bases. Figure 3.13 illustrates this process for a downward-moving circular obstacle.

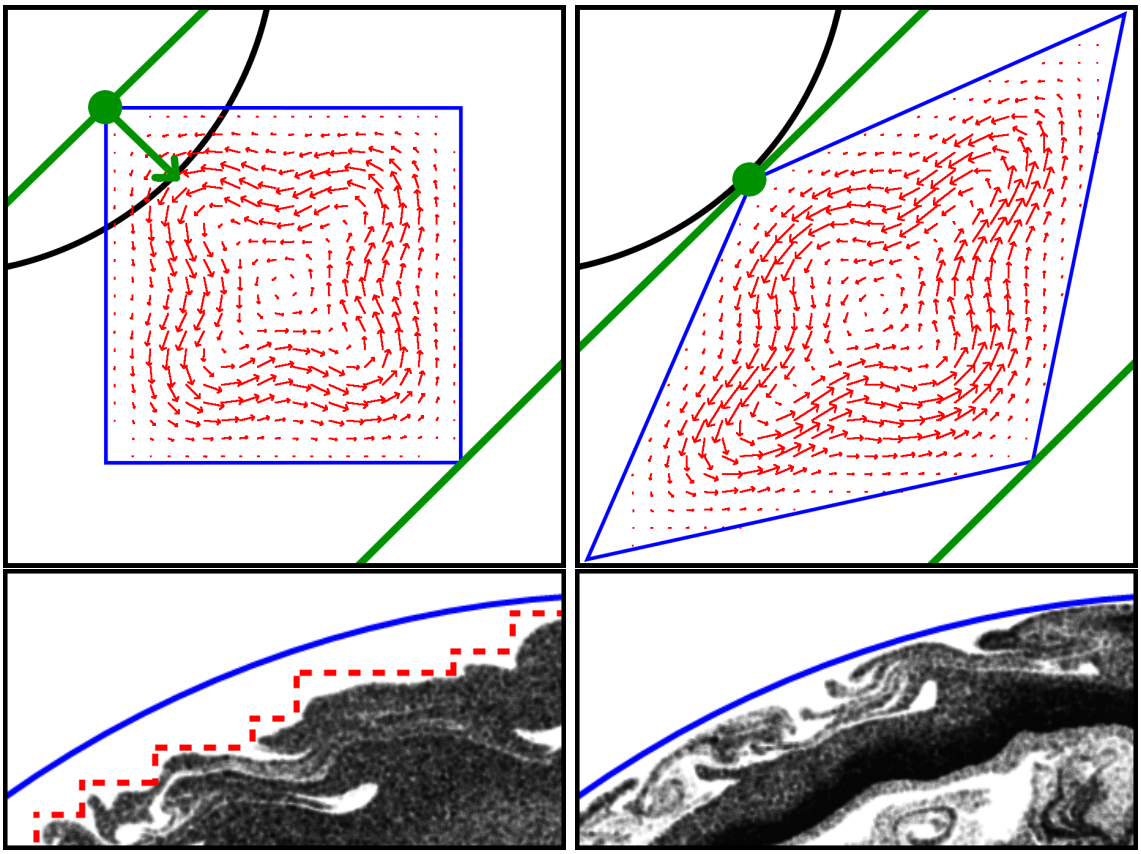


Figure 3.12. Top: we displace bases (blue) that overlap a boundary (black) along the SDF gradient (green arrow) and deform them (right) to maintain their area. Bottom: without basis deformation, the extend of the bases (dotted line) does not properly cover the domain, and coarse simulations can exhibit staircase artifacts. Our deformation eliminates these issues (right).

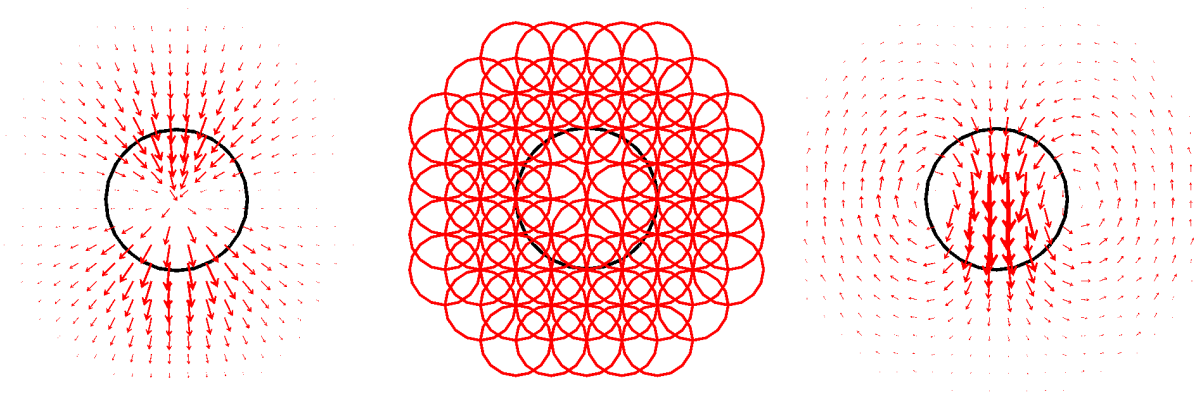


Figure 3.13. Obstacle coupling. Left to right: we evaluate the obstacle's normal velocity near its boundary; the velocity is projected onto bases near the boundary, allowing bases to intersect the object; the resulting divergence-free flow approximates the normal velocity of the object.

We evaluate the normal flow near obstacle boundaries using the difference in the dynamic obstacle's SDF at two consecutive time steps, $\mathrm{SDF}^{(t-1)}$ and $\mathrm{SDF}^{(t)}$, as

$$\left((\mathrm{SDF}^{(t-1)} - \mathrm{SDF}^{(t)})\big/\Delta t\right) \nabla \mathrm{SDF}^{(t)} \max\left(1 - |\mathrm{SDF}^{(t)}|\big/\sigma, 0\right) \tag{3.29}$$

where $\nabla \mathrm{SDF}^{(t)}$ is the normalized gradient of the current SDF, Δt is the time step, and $\sigma > 0$. The last term reduces the strength of the normal flow as we move away from the boundary, which makes the normal flow continuous in Ω, and σ effectively controls the distance at which moving obstacles influence the simulation.

We then project this normal flow onto only the bases that fall within a neighborhood band of width σ around the boundary, including bases that intersect the interior of the obstacle. Here, we do not apply basis deformation (Section 3.6.1) during projection. The projected normal flow is divergence-free, which makes it behavior more natural around the object. For instance, in Figure 3.13, the projected normal flow pushes and drags smoke particles in the direction of obstacle motion, as expected. It also "rolls" particles around the side of the obstacle, which is a desirable behavior not present in the normal flow itself prior to projection.

This additional projected boundary flow is then added to the velocity field when advecting smoke particles. We also add the boundary flow coefficients to the original coefficients when computing the fluid interactions (as per Section 3.5), and so obstacle movement can generate secondary motions.

3.7. Results and Discussion

We apply our method to various smoke simulation scenarios and we refer readers to the supplemental video for full animation sequences.

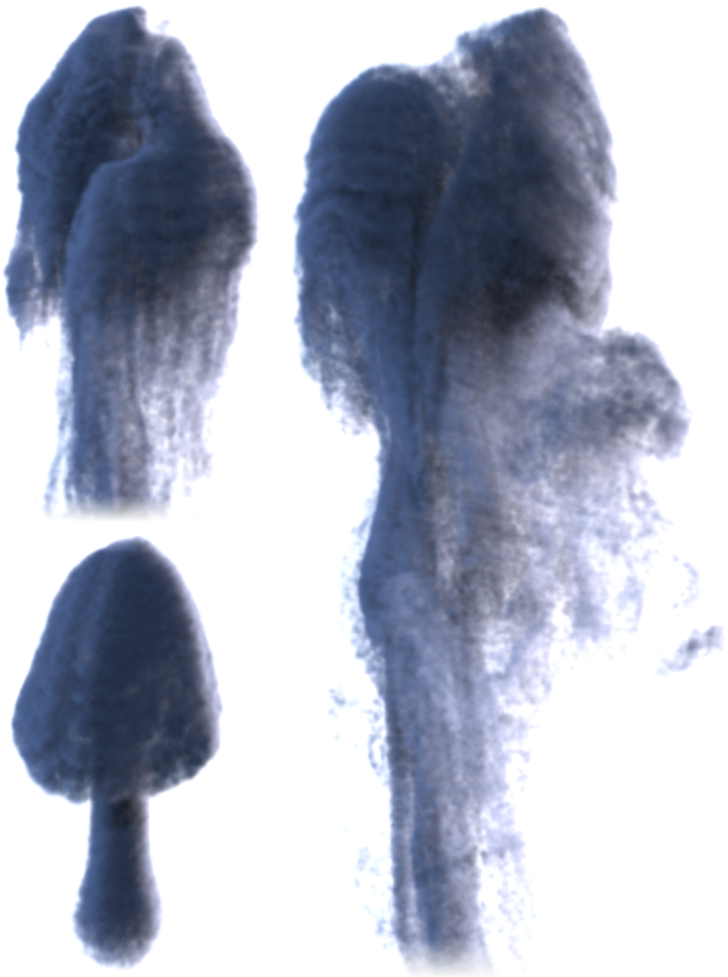


Figure 3.14. 3D smoke plume. Buoyancy and advection initially lead to the characteristic mushroom shape, after which energy transfers introduce turbulent behavior. We use only two frequency levels in this simple simulation, and a coverage that activates bases only around smoke particles, as in Figure 3.10c.

Figure 3.14 illustrates a simple 3D smoke plume. Energy enters the system from external buoyancy forces, which we model as small vertical vectors projected onto our bases at each particle location. This scene applies the coverage strategy in Figure 3.11c where bases are only activated around particles. This significantly reduces computation costs, especially early in the simulation, since bases at the top of the domain are only utilized after hundreds of frames.

Figure 3.15 shows a smoke plume interacting with two static sphere obstacles (which admit simple analytic SDF expressions). Our basis deformation method (Section 3.6.1) causes the plume to more closely wrap around each sphere.

Figure 3.1 demonstrates interactions with a more complex triangle mesh obstacle. Here, the boundary SDF is computed by using an explicit search for the closest distance between a given point and a point on the mesh. We accelerate this computation by discretizing the

simulation domain on a coarse regular grid and precomputing the list of mesh triangles in each cell. This allows us to accelerate both the SDF query and projection.

Figure 3.16 shows a hand moving through a smoke cloud. No buoyancy is used in this scene, and energy is only created from the hand's movement. We again use a triangle mesh, and the same coarse grid SDF acceleration as in Figure 3.1, to represent the hand. Since the hand only undergoes translation, we can reuse the acceleration structure between frames. For generic mesh deformations, the acceleration may need to be recomputed (or deformed) at each time step.

Our obstacle coupling method (Section 3.6.2) is approximate, and particles can still collide with moving objects (e.g., see the 2D moving obstacles in our accompanying video). While unavoidable in general, since our bases are not tailored to specific obstacle geometries,

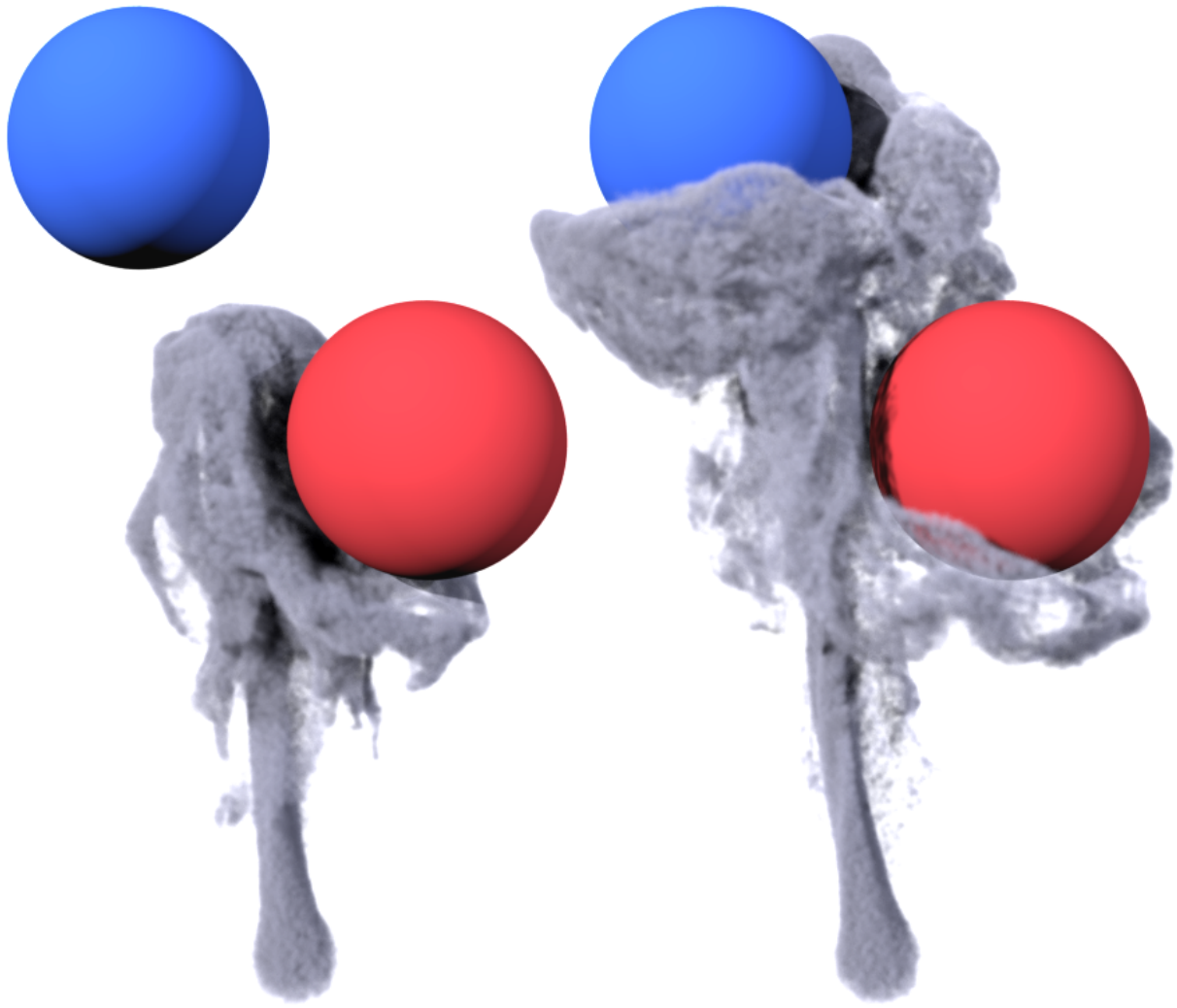


Figure 3.15. Smoke plume interacting with two spheres. Our basis deformation scheme allows particles to more closely wrap around obstacles. This simulation also uses only two frequency levels.

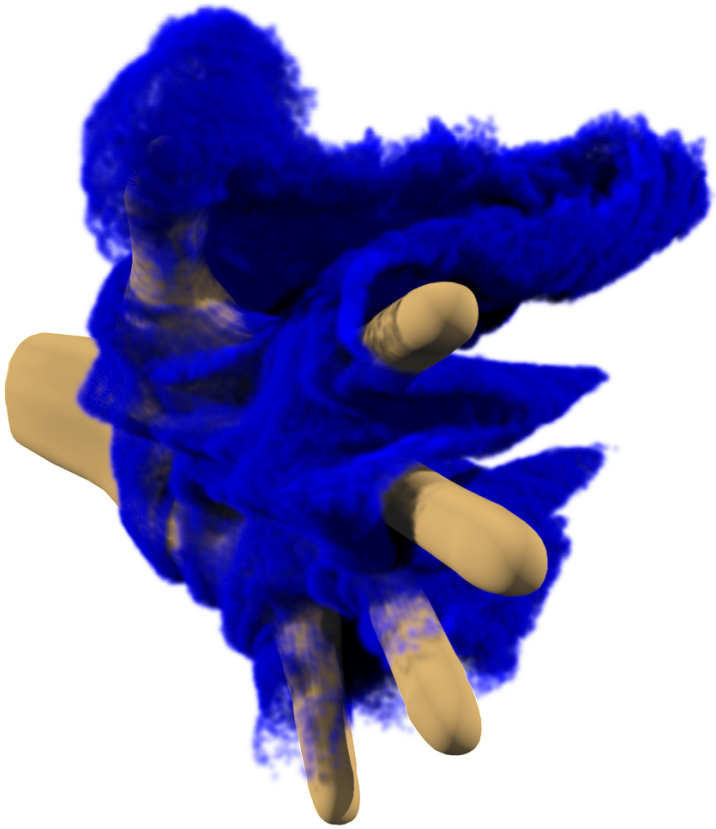


Figure 3.16. Hand pushing through a smoke cloud. No buoyancy forces are used.

the moving hand in Figure 3.16 is illustrative of the effectiveness of the approximation with complex obstacles.

Finally, Figure 3.17 shows a smoke simulation inside of a glass bunny, illustrating our method's ability to adapt to complex simulation domains without any need for customized basis precomputation. We apply the tiling strategy in Figure 3.11a here, where finer bases are only placed in narrow regions in the domain (e.g. the bunny's ears) and near boundaries. Tiling the entire domain with bases at all scales would require 412K basis functions, whereas our adaptive placement uses only 41K basis functions (i.e., 10× fewer bases). Note how particles reach and fill tiny cavities, and the more costly computations are only incurred where needed.

3.7.1. Computation Times

We present a breakdown of computational costs for all our 3D scenes in Table 3.3. All results are computed on an Intel i7 quad core running at 3.4 GHz with 32GB of RAM.

It is clear that the actual dynamics computations are never the bottleneck of our method (Basis Advection and Energy Transfer columns). This confirms the benefits of only having to compute interactions with a few bases in a local neighborhood.

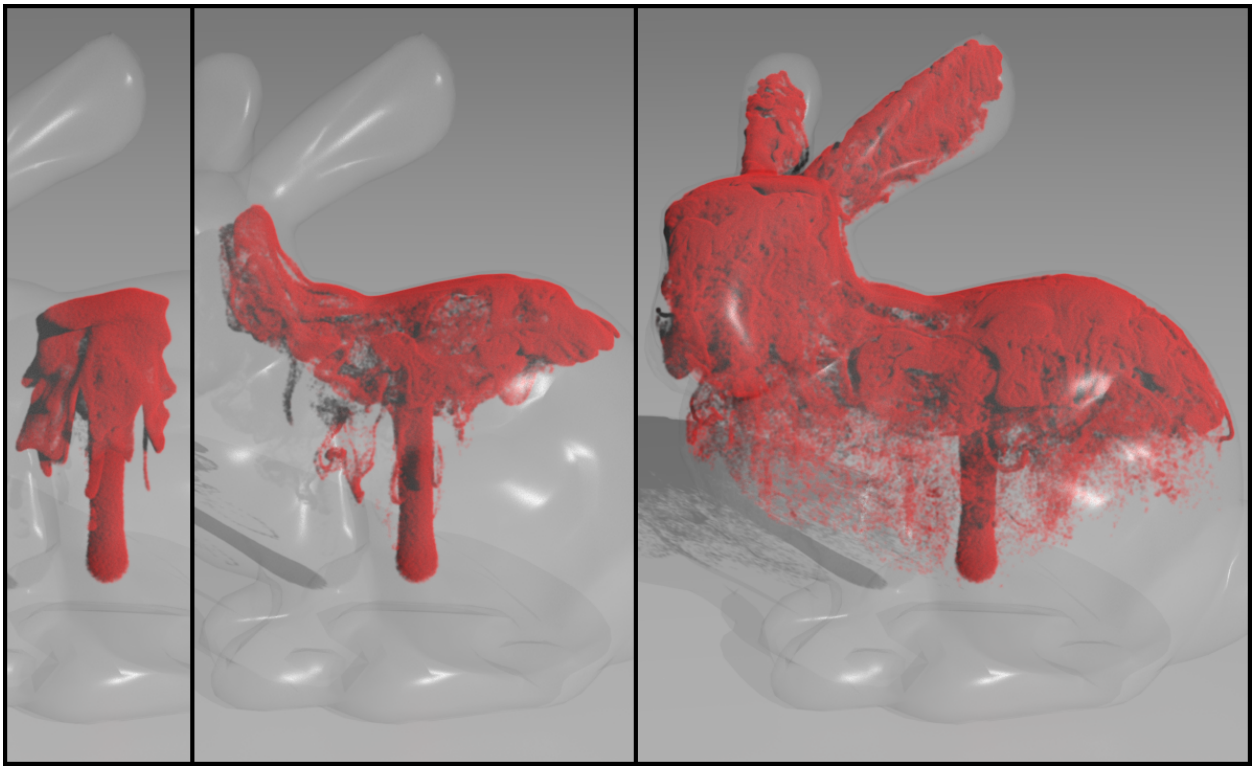


Figure 3.17. Smoke plume inside a glass bunny. Three frequency levels are used, which is made possible by using the coverage strategy of Figure 3.11a. The finest bases are only placed where necessary (i.e., in the ears), reducing the number of required bases by 10×.

Scene	#particles	#bases	Computation Time (seconds)						Memory
			Buoy.	Stretch	Boundary	Basis Adv.	Energy	Particle Adv.	
Two Spheres	145K	345K	20.11	0.32	—	0.91	0.84	67.54	13.4
Bunny Face	738K	277K	38.67	41.68	—	1.37	1.64	168.78	24.8
Hand	151K	565K	—	62.24	27.69	0.90	1.61	274.95	14.5
Glass Bunny	807K	41K	4.74	221.96	—	0.09	0.06	320.18	1.2
3D Plume	300K	99K	12.79	—	—	0.48	0.56	28.48	8.7

Table 3.3. Scene statistics for 3D results (Section 3.7.) Every value, including particle count and active basis count, are averages over all simulation frames. Memory load is given in Gigabytes, "buoy." stands for "buoyancy", and "adv." stands for "advection".

The cost of the basis warping computations depends greatly on the scene, and understandably so: warping is cheaper when SDFs are faster to compute (i.e., Figure 3.15). All other scenes (except Figure 3.14, which has no obstacles) use mesh obstacles, where SDF computation is costly, even with our acceleration structure. This is most problematic in the "Glass Bunny" (Figure 3.17): given the low number of bases, the relative cost of basis stretching/warping (which depends heavily on the number of particles) increases.

The two most costly operations are external force computation and particle advection, as expected: these are the only operations that require conversions between the reduced- and full-spaces.

Our particle advection implementation is not performance-optimized: we parallelize inner loops on the CPU using OpenMP. An end-to-end GPU-accelerated implementation is likely to improve performance measurably; our current implementation only leverages the GPU to accelerate external force projection. Of note, particle and basis data are both well-suited for representation in textures, and velocity fields can be directly rasterized into output buffers. We leave such accelerations to future work.

Finally, our method is usually memory bound, especially in Figure 3.1. Recall that our interaction coefficient cache exaggerates this constraint, trading memory for performance. Still, we find the improvements in compute cost significant enough to justify the additional memory it requires.

3.8. Conclusion and Future Work

We presented an end-to-end model reduction method for adaptive multiscale smoke simulations. We detail a novel vector field basis construction method that yields divergence-free bases with localized support and beneficial orthogonality properties. We show how to locally precompute basis interactions for reuse across simulation domains and with dynamic obstacles.

In the future we would like to investigate new coverage schemes during basis construction: for instance, even if our power-of-two refinement scheme is a natural choice, we may be able to obtain better coverage from, e.g., a power-of-$\sqrt{2}$ refinements. We are also interested in extending our technique to other simulation problems, such as the simulation of free-surface behavior, which could lead to efficient model-reduced liquid simulations.

3.9. Appendix A: Basis Construction in 3D

This section completes the basis construction method of Section 3.4.6. As explained in that section, the 3D bases used in the paper can be derived in a very simple manner from the 2D bases directly. However, if the bases are to be used for other applications, different constraints could be imposed on the bases, and the simple construction of Section 3.4.6 might not work. The method described here is more general.

Eigenflows aligned along the z axis are of the form

$$\mathbf{e}_z^{\boldsymbol{k}}(\mathbf{x}) = \begin{pmatrix} -k_x\, k_z\, \sin(k_x\,\pi\,x)\, \cos(k_y\,\pi\,y)\, \cos(k_z\,\pi\,z) \\ -k_y\, k_z\, \cos(k_x\,\pi\,x)\, \sin(k_y\,\pi\,y)\, \cos(k_z\,\pi\,z) \\ (k_x^2 + k_y^2)\, \cos(k_x\,\pi\,x)\, \cos(k_y\,\pi\,y)\, \sin(k_z\,\pi\,z) \end{pmatrix} \in \mathbb{R}^3\,, \tag{3.30}$$

and $\mathbf{e}_x^{\boldsymbol{k}}$ and $\mathbf{e}_y^{\boldsymbol{k}}$ can be obtained by rotation. Note that even if we construct z-aligned templates, we combine together eigenflows aligned in x, y and z, using coefficients $w_{x,\mathbf{a}}^{\mathbf{k}}$, $w_{y,\mathbf{a}}^{\mathbf{k}}$, and $w_{z,\mathbf{a}}^{\mathbf{k}}$, respectively.

The 3D constraint system is larger than in 2D, but it has the same structure, and can be solved in the same way. Equations 3.9 become a set of six constaints (one for each face of the 3D basis support). For the orthogonality constraints, we do not impose orthogonality between a basis and each of its neighbors of the same frequency, as this would create too many constraints. We instead only impose orthogonality among bases of the same axis alignment and same frequency located on the same axis-aligned slice. For instance, we impose that each z-aligned basis is orthogonal to each other z-aligned basis of the same frequency centered at the same z coordinate. This creates individual planes of orthogonal bases, which is sufficient for our use, as described in Section 3.5. These orthogonality constraints are therefore similar to the 2D case, and also reduce to three quadratic equations.

While using harmonic 0 is invalid in 2D since it creates an identically zero flow, it is a valid harmonic in 3D. It however creates problematic cases where eigenflows with different parameters are not linearly independent. For instance,

$$k_y \, \mathbf{e}_y^{(kx,ky,0)} = -k_x \, \mathbf{e}_x^{(k_x,k_y,0)}. \tag{3.31}$$

These eigenflows cannot be avoided, since they portray the desired behavior for basis $\mathbf{b}_z^{\mathbf{k}}$, i.e., a swirling motion around the z axis, at the scale of the basis support. Equation 3.8 is therefore replaced in 3D by

$$k_y \, w_{y,(1,1,0)}^{\boldsymbol{k}} - k_x \, w_{x,(1,1,0)}^{\boldsymbol{k}} = 1. \tag{3.32}$$

To remove linearly dependent bases, we first look at each pair of eigenflows, and remove one of them if one is a scaled version of the other. Then, after the linear constraints of the systems are resolved, we look at each remaining free coefficient, and compare the solutions obtained by setting the coefficient to 1 or 0. If both solutions are the same, the coefficient represents a null subspace, and can be discarded.

The harmonic set in 3D must be larger to account for the additional constraints, and we have found $\mathcal{A}_\star = \{0,1,2,3,5\}$ to be the smallest set to yield solutions. It however still gives us too many free variables, so we additionally impose that the basis be symmetric in the z direction with respect to its center. This reduces to linear constraints on the weights similar to Equations 3.9, and brings the number of free coefficients down to three. Finally, as in 2D, the three orthogonality constraints can be solved with exactly eight solutions, and we keep the real solution minimizing $\|\mathbf{h}^{\boldsymbol{k}}\|$.

Because of all the removed linearly dependent eigenflows, only 30 of the 375 coefficients are non-zero. They are given in Table 3.4 for the most useful anisotropy ratios. Figure 3.5

axis, $\boldsymbol{a}$	Anisotropy Ratio		
	(1 : 1 : 1)	(2 : 2 : 1)	(1 : 1 : 2)
y,(1,1,0)	1	0.5	1
y,(1,1,2)	-0.33333333	-0.33333333	-0.11111111
z,(1,1,2)	-0.33333333	-0.16666666	-0.22222222
y,(1,3,0)	-0.11072314	-0.05536157	-0.11072314
y,(1,3,2)	0.07908796	0.05032870	0.04258582
z,(1,3,2)	0.04745277	0.01509861	0.05110299
y,(1,5,0)	-0.13356611	-0.06678305	-0.13356611
y,(1,5,2)	0.11575729	0.06430960	0.08268378
z,(1,5,2)	0.04452203	0.01236723	0.06360291
y,(3,1,0)	-0.03690771	-0.01845385	-0.03690771
y,(3,1,2)	0.02636265	0.01677623	0.01419527
z,(3,1,2)	0.00527253	0.00167762	0.00567811
y,(3,3,0)	0.04209225	0.02104612	0.04209225
y,(3,3,2)	-0.03443911	-0.01993843	-0.02228413
z,(3,3,2)	-0.01147970	-0.00332307	-0.01485608
y,(3,5,0)	-0.01787380	-0.00893690	-0.01787380
y,(3,5,2)	0.01599235	0.00868156	0.01215419
z,(3,5,2)	0.00470363	0.00127670	0.00714952
y,(5,1,0)	-0.02671322	-0.01335661	-0.02671322
x,(5,1,2)	1.41909610	1.51847588	0.40189539
y,(5,1,2)	0.30697067	0.31655709	0.09691583
z,(5,1,2)	0.56941932	0.30418986	0.32406043
y,(5,3,0)	-0.01072428	-0.00536214	-0.01072428
x,(5,3,2)	-0.09040709	-0.05933200	-0.04599034
y,(5,3,2)	-0.04464884	-0.03039026	-0.02030169
z,(5,3,2)	-0.03446953	-0.01140678	-0.03421844
y,(5,5,0)	0.01177721	0.00588860	0.01177721
x,(5,5,2)	-0.12084482	-0.06721802	-0.08591584
y,(5,5,2)	-0.13174965	-0.07299117	-0.09483797
z,(5,5,2)	-0.05051889	-0.01402092	-0.07230152
$\|\mathbf{b}^{\widehat{\mathbf{k}}}\|$	1.19824688	1.19824688	0.84728849

Table 3.4. Non-zero coefficients of eigenflows to construct our 3D bases.

shows the structure of the 3D flows: z-aligned basis have no z component, and their xy-cut has the same structure as the basis we constructed in 2D.

Direct computations can confirm numerically that this linear combination is the same as the basis defined in Equation 3.16.

References

[84] Angelidis, A. 2017, Multi-scale vorticle fluids, *ACM Transactions on Graphics (TOG)*, vol. 36, n° 4, p. 104.

[85] Angelidis, A., F. Neyret, K. Singh and D. Nowrouzezahrai. 2006, A controllable, fast and stable basis for vortex based smoke simulation, in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, ISBN 3-905673-34-7, p. 25–32.

[86] Barbič, J. and D. L. James. 2005, Real-time subspace integration for st. venant-kirchhoff deformable models, in *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, p. 982–990.

[87] Berkooz, G., P. Holmes and J. L. Lumley. 1993, The proper orthogonal decomposition in the analysis of turbulent flows, *Annual review of fluid mechanics*, vol. 25, n° 1, p. 539–575.

[88] Burden, R. L. and J. D. Faires. 2011, Numerical analysis, .

[89] Charton, P. and V. Perrier. 1996, A pseudo-wavelet scheme for the two-dimensional navier-stokes equation, *Computational and Applied Mathematics*, vol. 15, p. 139–160.

[90] Coolidge, J. L. 1932, A treatise on algebraic plane curves, *Bulletin of the American Mathematical Society 38 (1932), 163-165*, p. 0002–9904.

[91] Davidson, P. 2015, Turbulence: an introduction for scientists and engineers, Oxford University Press, USA.

[92] De Witt, T., C. Lessig and E. Fiume. 2012, Fluid simulation using laplacian eigenfunctions, *ACM Transactions on Graphics (TOG)*, vol. 31, n° 1, p. 10.

[93] Deriaz, E. and V. Perrier. 2006, Divergence-free and curl-free wavelets in two dimensions and three dimensions: application to turbulent flows, *Journal of Turbulence*, , n° 7, p. N3.

[94] Farge, M. and G. Rabreau. 1988, Transformée en ondelettes pour détecter et analyser les structures cohérentes dans les écoulements turbulents bidimensionnels, *Comptes Rendus de l'Académie des Sciences de Paris Série II b*, vol. 307, p. 433–440.

[95] Farge, M., K. Schneider and N. Kevlahan. 1999, Non-gaussianity and coherent vortex simulation for two-dimensional turbulence using an adaptive orthogonal wavelet basis, *Physics of Fluids (1994-present)*, vol. 11, n° 8, p. 2187–2201.

[96] Foster, N. and D. Metaxas. 1996, Realistic animation of liquids, *Graphical models and image processing*, vol. 58, n° 5, p. 471–483.

[97] Gerszewski, D., L. Kavan, P.-P. Sloan and A. W. Bargteil. 2015, Basis enrichment and solid–fluid coupling for model-reduced fluid simulation, *Computer Animation and Virtual Worlds*, vol. 26, n° 2, p. 109–117.

[98] Golas, A., R. Narain, J. Sewall, P. Krajcevski, P. Dubey and M. Lin. 2012, Large-scale fluid simulation using velocity-vorticity domain decomposition, *ACM Transactions on Graphics (TOG)*, vol. 31, n° 6, p. 148.

[99] Hahn, F., B. Thomaszewski, S. Coros, R. W. Sumner, F. Cole, M. Meyer, T. DeRose and M. Gross. 2014, Subspace clothing simulation using adaptive bases, *ACM Transactions on Graphics (TOG)*, vol. 33, n° 4, p. 105.

[100] Inc., W. R. 2017, Mathematica, Version 10.3, Champaign, IL.

[101] Jones, A. D., P. Sen and T. Kim. 2016, Compressing fluid subspaces, in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, p. 77–84.

[102] Kadri Harouna, S. and V. Perrier. 2015, Divergence-free wavelet projection method for incompressible viscous flow on the square, *Multiscale Modeling & Simulation*, vol. 13, n° 1, p. 399–422.

[103] Kevlahan, N. K. and O. V. Vasilyev. 2005, An adaptive wavelet collocation method for fluid-structure interaction at high reynolds numbers, *SIAM Journal on Scientific Computing*, vol. 26, n° 6, p. 1894–1915.

[104] Kim, T. and J. Delaney. 2013, Subspace fluid re-simulation, *ACM Transactions on Graphics (TOG)*, vol. 32, n° 4, p. 62.

[105] Kry, P. G., L. Revéret, F. Faure and M.-P. Cani. 2009, Modal locomotion: Animating virtual characters with natural vibrations, in *Computer Graphics Forum*, vol. 28, Wiley Online Library, p. 289–298.

[106] Lemarie-Rieusset, P. G. 1992, Analyses multi-résolutions non orthogonales, commutation entre projecteurs et dérivation et ondelettes vecteurs à divergence nulle, *Revista Matemática Iberoamericana*, vol. 8, n° 2, p. 221–238.

[107] Lemarié-Rieusset, P.-G. 1994, Un théorème d'inexistence pour les ondelettes vecteurs à divergence nulle, *Comptes rendus de l'Académie des sciences. Série 1, Mathématique*, vol. 319, n° 8, p. 811–813.

[108] Liu, B., G. Mason, J. Hodgson, Y. Tong and M. Desbrun. 2015, Model-reduced variational fluid simulation, *ACM TOG*, vol. 34, n° 6, p. 244.

[109] Narain, R., J. Sewall, M. Carlson and M. C. Lin. 2008, Fast animation of turbulence using energy transport and procedural synthesis, in *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, p. 166.

[110] Nejadmalayeri, A., A. Vezolainen, E. Brown-Dymkoski and O. V. Vasilyev. 2015, Parallel adaptive wavelet collocation method for pdes, *Journal of Computational Physics*, vol. 298, p. 237–253.

[111] Pentland, A. and J. Williams. 1989, Good vibrations: Modal dynamics for graphics and animation, vol. 23, ACM.

[112] Pfaff, T., N. Thuerey and M. Gross. 2012, Lagrangian vortex sheets for animating fluids, *ACM Transactions on Graphics (TOG)*, vol. 31, n° 4, p. 112.

[113] Saad, Y. and M. Sosonkina. 1999, Enhanced parallel multicolor preconditioning techniques for linear systems, in *PPSC*.

[114] Schneider, K. and O. V. Vasilyev. 2010, Wavelet methods in computational fluid dynamics*, *Annual Review of Fluid Mechanics*, vol. 42, p. 473–503.

[115] Stam, J. 1999, Stable fluids, in *Proceedings of 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., p. 121–128.

[116] Stanton, M., Y. Sheng, M. Wicke, F. Perazzi, A. Yuen, S. Narasimhan and A. Treuille. 2013, Non-polynomial galerkin projection on deforming meshes, *ACM Transactions on Graphics (TOG)*, vol. 32, nº 4, p. 86.

[117] Stevenson, R. 2011, Divergence-free wavelet bases on the hypercube: Free-slip boundary conditions, and applications for solving the instationary stokes equations, *Mathematics of Computation*, vol. 80, nº 275, p. 1499–1523.

[118] Stewart, G. W. 1998, Afternotes goes to graduate school: lectures on advanced numerical analysis, vol. 58, Siam.

[119] Treuille, A., A. Lewis and Z. Popović. 2006, Model reduction for real-time fluids, in *ACM Transactions on Graphics (TOG)*, vol. 25, ACM, p. 826–834.

[120] Weißmann, S. and U. Pinkall. 2010, Filament-based smoke with vortex shedding and variational reconnection, in *ACM Transactions on Graphics (TOG)*, vol. 29, ACM, p. 115.

[121] Wicke, M., M. Stanton and A. Treuille. 2009, Modular bases for fluid dynamics, in *ACM Transactions on Graphics (TOG)*, vol. 28, ACM, p. 39.

Chapter 4

FAST GAZE-CONTINGENT OPTIMAL DECOMPOSITIONS FOR MULTIFOCAL DISPLAYS

This third paper describes a virtual scene decomposition algorithm for presenting images on multifocal displays at interactive rates. We formulate this decomposition as a constrained minimization problem, and solve it iteratively. Previous methods for computing such decompositions require multiple minutes per frame, while our method can do so in 200 milliseconds per frame. This significant improvement is due to our reformulation of the problem in terms of simple image operations, leading to a simpler iterative scheme amenable to efficient GPU implementations.

4.A. Publication

This paper was published in ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2017, in November 2017. It has been reformatted to appropriately follow the format of this thesis.

Olivier Mercier is the principal author of this project. Mercier was primarily involved in the design and numerical analysis of the decomposition scheme presented in the paper. He was also in charge of implementing the algorithms on a physical testbed, as well as the writing and presentation of the paper. He was in charge of all result generation, except for the preliminary user study in Section 4.6.4.

Fast Gaze-Contingent Optimal Decompositions for Multifocal Displays

Olivier Mercier, *Université de Montréal and Oculus Research*

Yusufu Sulai, *Oculus Research*

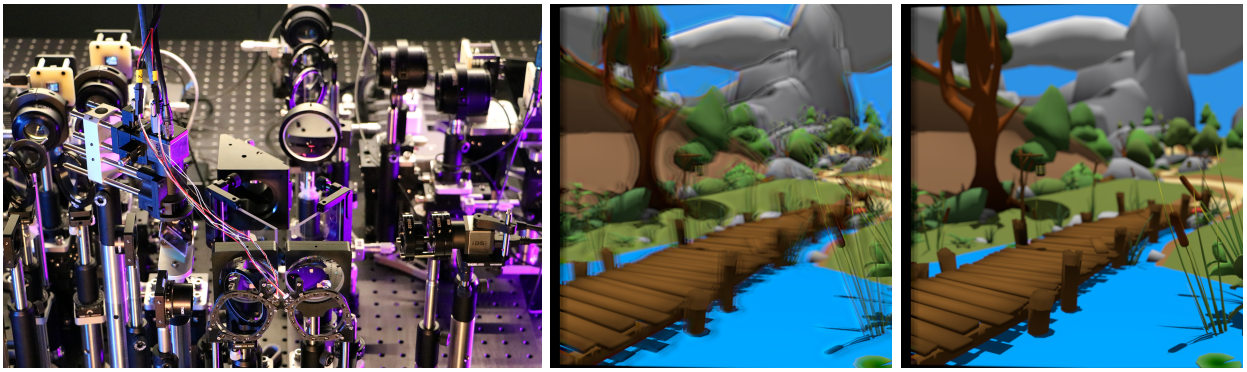
Kevin MacKenzie, *Oculus Research*

Marina Zannoli, *Oculus Research*

James Hillis, *Oculus Research*

Derek Nowrouzezahrai, *McGill University*

Douglas Lanman, *Oculus Research*



(a) Multifocal Testbed with Eye and Accommodation Tracking (b) Eye Movement without Correction (c) Eye Movement with Correction

Figure 4.1. Multifocal displays require a decomposition of the scene onto the display planes, which often assumes perfect alignment of the viewer with the system. Otherwise, parallax introduced by eye rotation and head offsets relative to the display may result in misregistration between the images, creating halos and increasing blurriness. (a) In this paper, we present the first multifocal display with eye tracking and accommodation measurement. (b-c) We also introduce the first computationally efficient optimal decomposition algorithm, enabling interactive content that utilizes eye tracking to directly maintain image alignment. Both our hardware and algorithmic contributions are necessary steps towards better understanding the practical requirements of multifocal displays.

Résumé

Les casques de réalité virtuelle utilisent majoritairement un seul écran à une distance fixe de chaque oeil, ce qui peut créer un conflit entre la vergence et l'accommodation des utilisateurs. Les systèmes multicouches ont été investigués comme une solution potentielle où plusieurs

écrans couvrent l'étendue de l'accommodation d'un utilisateur. Ces dispositifs nécessitent la décomposition d'une scène virtuelle pour distribuer son contenu sur les divers écrans, et il a été démontré que des algorithmes simples peuvent accomplir cette tâche en temps réel. Par contre, le récent développement d'une décomposition optimale améliore la qualité des image décomposées, surtout pour des scènes au contenu complexe. Ces décompositions sont plus exigeante en temps de calcul et nécessitent potentiellement un meilleur alignement des multiples écrans avec la position de l'utilisateur, ce qui nuit à leur utilisation dans des applications pratiques.

Notre objectif est de rendre possible des décompositions optimales interactives capables de faire fonctionner un banc de test mesurant la vergence et l'accommodation. Ultimement, un tel banc de test est nécessaire pour établir les exigences requises pour l'utilisation d'écrans multicouches dans des applications pratiques en termes de demande de calcul et de matériel. Pour ce faire, nous présentons un algorithme efficace pour calculer les décompositions optimales qui inclus des idées de science de la vision. Notre méthode est applicable à des implémentations sur carte graphique et permet d'accélérer les méthodes de décomposition existantes par trois ordres de grandeur. Nous démontrons également que des mesures oculométriques peuvent être combinées à des déformations en espace image pour adéquatement aligner les écrans, compensant à la fois les rotations des yeux et les mouvement de tête de l'utilisateur. De plus, nous construisons le premier banc de test qui inclus des mesures de position et d'accommodation oculaires, ouvrant la voie à l'établissement des exigences oculométriques pour cette classe de dispositifs. Finalement, nous présentons les résultats préliminaires d'une étude pilote utilisant notre banc de test qui étudie les réponses d'accommodation d'utilisateurs visionant du contenu dynamique présenté avec une décomposition optimale.

Mots-clés : écrans multicouches, rendu multidirectionnel, conflit vergence-accommodation

Abstract

As head-mounted displays (HMDs) commonly present a single, fixed-focus display plane, a conflict can be created between the vergence and accommodation responses of the viewer. Multifocal HMDs have long been investigated as a potential solution in which multiple image planes span the viewer's accommodation range. Such displays require a scene decomposition algorithm to distribute the depiction of objects across image planes, and previous work has shown that simple decompositions can be achieved in real-time. However, recent optimal decompositions further improve image quality, particularly with complex content. Such decompositions are more computationally involved and likely require better alignment of the image planes with the viewer's eyes, which are potential barriers to practical applications.

Our goal is to enable interactive optimal decomposition algorithms capable of driving a vergence- and accommodation-tracked multifocal testbed. Ultimately, such a testbed is necessary to establish the requirements for the practical use of multifocal displays, in terms of computational demand and hardware accuracy. To this end, we present an efficient algorithm for optimal decompositions, incorporating insights from vision science. Our method is amenable to GPU implementations and achieves a three-orders-of-magnitude speedup over previous work. We further show that eye tracking can be used for adequate plane alignment with efficient image-based deformations, adjusting for both eye rotation and head movement relative to the display. We also build the first binocular multifocal testbed with integrated eye tracking and accommodation measurement, paving the way to establish practical eye tracking and rendering requirements for this promising class of display. Finally, we report preliminary results from a pilot user study utilizing our testbed, investigating the accommodation response of users to dynamic stimuli presented under optimal decomposition.

Keywords: computational displays, multifocal displays, multiview rendering, vergence-accommodation conflict.

4.1. Introduction

More than a century of research into stereoscopic and multiscopic displays has worked toward an accurate reproduction of the three-dimensional world [166]. Today's binocular head-mounted displays (HMDs) offer an accessible means to resolve persistent deficiencies of 3D displays, achieving accurate reproduction of motion parallax, as well as depicting 360-degree imagery enveloping the viewer. However, modern HMDs do not correctly reproduce all natural depth cues available to the human visual system. In particular, due to the fixed optical focus of current HMDs, the retinal blur created by out-of-focus scene components is synthesized inaccurately. Correspondingly, the use of HMDs may lead to vergence-accommodation conflict (VAC), which biases perceived depth [169], and has been linked to visual fatigue and visual discomfort [134, 163].

Volumetric displays are one solution to alleviate the issues associated with VAC. This widely studied class of glasses-free 3D display can depict accurate retinal defocus blur by synthesizing an additive volume of modulated light sources [125]. Rolland et al. [162] were among the first to propose a *multifocal* volumetric HMD, capable of generating multiple virtual image planes spanning a range of depths. By incorporating an eyepiece, Rolland et al. demonstrated that a compact multifocal HMD can reproduce a volume of light sources extending throughout a viewer's accommodative range.

As first described by Akeley et al. [123], a *scene decomposition* must be performed to distribute virtual objects across the various image planes to produce near-correct retinal defocus blur. Specifically, they introduced a *linear blending* algorithm to divide the depiction

of objects across the nearest enclosing image planes. In this paper, we significantly expand upon the capabilities and practicality of the more recent *optimized blending* algorithm of Narain et al. [156], which is better suited for depicting occlusions and reflections, as well as accurate retinal defocus blur.

Despite nearly two decades of investigation, multifocal displays remain potentially unsuitable for practical applications, primarily due to two unresolved issues. First, computing high-quality scene decompositions is inefficient, as evidenced by the minutes-long run times reported by Narain et al. and other more complex decomposition approaches [154]. Second, all existing multifocal display decompositions assume a single, fixed viewpoint. As shown in Figure 4.1, this can cause the projections of the image planes to be misregistered on the retina if the position and direction of the viewer's eye are not exactly the same as those assumed during the scene decomposition, which can significantly reduce image quality.

In this paper, we present solutions to these long-standing challenges. First, we show that high-quality scene decompositions can be computed at interactive frame rates, leveraging insights from numerical methods and perceptual science. Second, we demonstrate how eye tracking measurements can be efficiently used to correct for eye movements. We apply these methods to drive the first multifocal testbed with integrated eye tracking and accommodation measurement, demonstrating the feasibility of gaze tracking within a multifocal display. Our hardware and algorithmic contributions enable the use and study of multifocal displays with dynamic content, and open the way to a better understanding of practical requirements for multifocal displays.

4.1.1. Contributions

- We achieve a three-orders-of-magnitude improvement in computation time relative to state-of-the-art optimal scene decompositions, reaching interactive performance through a different numerical method that is provably stable and amenable to GPU implementations;
- From prior perceptual studies, we derive a modified decomposition algorithm to optimize the retinal defocus blur gradient, as well as the retinal defocus blur itself, further accelerating computations;
- We develop an efficient algorithm to correct for eye movements detected after scene decomposition, showing eye tracking can be used to solve the misalignments due to eye rotation and head movements relative to the display;
- We develop the first adaptive multifocal system with integrated vergence and accommodation eye tracking, supporting three adjustable-focus displays per eye. This system is the first to support dynamic content, leveraging our efficient decomposition method and eye movement correction;

- We report preliminary results from empirically measured accommodation responses that show, for the first time, that optimal decomposition correctly drives accommodation.

4.2. Related Work

4.2.1. Driving Accommodation with HMDs

Volumetric displays, through their evolution into multifocal HMDs, are not the only means to address the vergence-accommodation conflict. As reviewed by Kramida [144], there exists a broad spectrum of such designs, spanning comparatively modest modifications (e.g., varifocal HMDs) to nearly complete overhauls (e.g., near-eye light field displays). In this context, multifocal displays present a moderate, but technically challenging, progression, adding display elements and computational complexity in exchange for extending the supported accommodation range.

With any HMD, the viewer's pupil must remain within the designed eye box. Correspondingly, to mitigate VAC, the stimulus to accommodation should be depicted correctly over this limited region. A direct solution is offered by near-eye light field displays, faithfully reproducing wavefronts of natural scenes for perspectives within the eye box. Lanman and Luebke [145], Hua and Javidi [136], and Song et al. [164] demonstrate microlens-based architectures for this purpose, whereas Huang et al. [137] apply multilayer LCDs; however, in all these examples, resolution remains limited with current display technologies. Similarly, Konrad et al. [141] recently showed that accommodation-invariant displays can be used to alleviate the VAC problem, but again at the cost of a tradeoff in resolution.

Another approach to mitigate VAC is offered by varifocal HMDs in which the virtual image distance is varied to match the vergence distance reported by an eye tracking system. This concept has been explored using electronically-tunable lenses, in part, by Liu et al. [148], Johnson et al. [138], Konrad et al. [140] and Padmanaban et al. [158]. Relative to near-eye light field displays, varifocal HMDs can offer higher resolutions and larger field of views [129], but require tunable optics that must rely on accurate eye tracking. In addition, retinal defocus blur can only be rendered synthetically. Although eye tracking can improve the rendered blur [139], it cannot be properly reproduced optically as the viewer accommodates.

In contrast to light field displays, volumetric displays utilize an additive superposition of display elements located at different depths. This construction raises natural questions regarding the density of planes required for accurate depictions. Early research by Rolland et al. [162] suggests as many as 14 layers would be required to support one arcminute resolution (i.e., 20/20 visual acuity) over an accommodation range of two diopters (e.g., from 50 cm to optical infinity). More recently, MacKenzie et al. [152, 151] established that a coarser separation between layers, as wide as 0.6 to 1.0 diopters, is sufficient to correctly

drive accommodation, requiring only four planes for a two-diopter accommodation range. With this reduced requirement on the number of planes, recent research has focused on identifying practical hardware to support a limited number of planes. For example, Love et al. [150] apply fast-switching birefringent optics, Hu et al. [135] investigate deformable membrane mirrors, and Llull et al. [149] incorporate electronically-tunable lenses. Matsuda et al. [154] use a spatial light modulator to create non-planar focal surfaces, which more closely adapts the few layers to the scene content but increases processing times. In contrast to these works, our efforts are focused on unresolved, yet fundamental, questions of maintaining image quality under natural eye movements and reducing algorithmic complexity; as a result, our system is optimized as a perceptual testbed, with these prior works showing potential paths toward compact form factors.

4.2.2. Multifocal Displays, Blur, and Accommodation

Similar to binocular disparity, the magnitude of retinal defocus blur varies monotonically with the separation between an object and the point of focus. Therefore, this retinal blur provides another cue to perceived depth [132]. Based on statistics of natural scenes and the properties of the human visual system, Burge and Geisler [128] found that reliable estimates of depth could be obtained from retinal defocus blur alone. This result is consistent with a growing body of psychophysical work showing the importance of retinal defocus blur for depth perception. The tilt-shift illusion provides a convincing example: artificial blur, as added to a photograph or a computer-generated image, can dramatically affect perceived scale [133, 167]. Moreover, recent studies have employed multifocal displays to show that retinal defocus blur, in isolation, is sufficient to recover depth ordering. Critically, this finding was supported only when retinal defocus blur was created by the optics of the eye, as opposed to synthetically rendered on a conventional display [171].

As discussed above, multiple HMD architectures have been proposed to depict retinal defocus blur. We emphasize that those relying on rendered defocus blur alone, such as varifocal displays, may not appear correctly or respond quickly enough to changes in the viewer's accommodative state due to unmodeled aspects of the eye or system latency, respectively. Multifocal displays avoid these concerns by creating retinal defocus blur through optical means (i.e, resulting from physiological changes within the eye). MacKenzie et al. [152] confirmed that the linear blending algorithm of Akeley et al. [123] approximates retinal defocus blur. Specifically, the accommodative state producing maximum retinal contrast occurs when focusing at the correct depth.

Others have investigated alternative decomposition algorithms. Wu et al. [170] use a saliency map to optimize the display plane locations for linear blending. Liu and Hua [147] advocate a nonlinear weighting to maximize the modulation transfer function (MTF) for objects within the display volume. Subsequent analysis by Ravikumar et al. [161] reported a

preference for linear blending over nonlinear weighting when considering biologically plausible metrics of image quality and properties of natural scenes. In later work, Narain et al. [156] demonstrated that, despite subtle differences between these methods, no prior scene decomposition suppresses salient artifacts at occlusion boundaries and reflections. This deficiency motivated the development of their optimized decomposition, directly using the reconstructed focal stack to compute the displayed images. We build on their work: to be practical, multifocal displays must support artifact-free viewing, but must also demonstrate real-time frame rates with unconstrained eye movements.

As described above, current evaluations of retinal defocus blur depictions have focused on the maximization of retinal contrast. Yet, as with rendered blur, it is not enough to correctly replicate this retinal blur itself, but also its variation as the eye accommodates (i.e., the retinal defocus blur gradient). Current evidence suggests that the accommodative system may exploit the temporal change of contrast that is induced through accommodative microfluctuations. This signal may be applied to resolve the direction of an accommodative stimulus (i.e., whether it is closer or further than the plane of focus) [152, 155]. Similarly, others have identified the retinal defocus blur gradient as a critical feedback signal to the accommodative response [124, 142, 157]. To our knowledge, we are the first to directly use the retinal defocus blur gradient produced by multifocal displays within the optimization formulation of the scene decomposition.

4.3. Interactive Scene Decomposition

Any practical application of a multifocal display requires decomposing a virtual scene onto the layers of the display. In order to do so both accurately and efficiently, we formulate the scene decomposition as an optimization problem with an efficient numerical solution. We begin with a simplified formulation of the problem in the theoretical case of a fixed eye, and only later generalize the formulation to support eye movements (Section 4.4.1). We assume monochromatic (i.e., grayscale) images, but the same formulation can be independently applied to any number of color channels.

We write scalars in math italic (e.g. x, D), n-d points/vectors in boldface italic (e.g. $\mathbf{b}$), and matrices/sub-matrices in sans serif (e.g. K). Depending on context, vector and matrix subscripts may either refer to individual scalar entries or to contained sub-vector/sub-matrices.

4.3.1. Optimal Decomposition

Figure 4.2 provides a schematic of a three-plane multifocal display: a stack of optically additive display planes are positioned at distances d_i from the eye, for $i \in \{1, \ldots, D\}$. The optical axis of the system is defined so as to intersect the displays orthogonally at their centers. We additionally assume, for simplicity and without loss of generality, that each

display has a square resolution of $N \times N$ pixels. We compute *focal slices* to form a *focal stack*, modeling the retinal defocus blur when a viewer is accommodated at focal distances f_i, for $i \in \{1, \ldots, F\}$. All distances are in diopters, and we refer to the display planes and focal slices by their distance from the eye.

When the viewer accommodates at f_i, the superposition of the display images should be as close as possible to the corresponding focal slice. This requirement for multifocal display image formation can be cast as a minimization problem [156]. Motivated by this formulation, we propose a novel solution that differs significantly from previous work by its interpretation, efficiency, and stability.

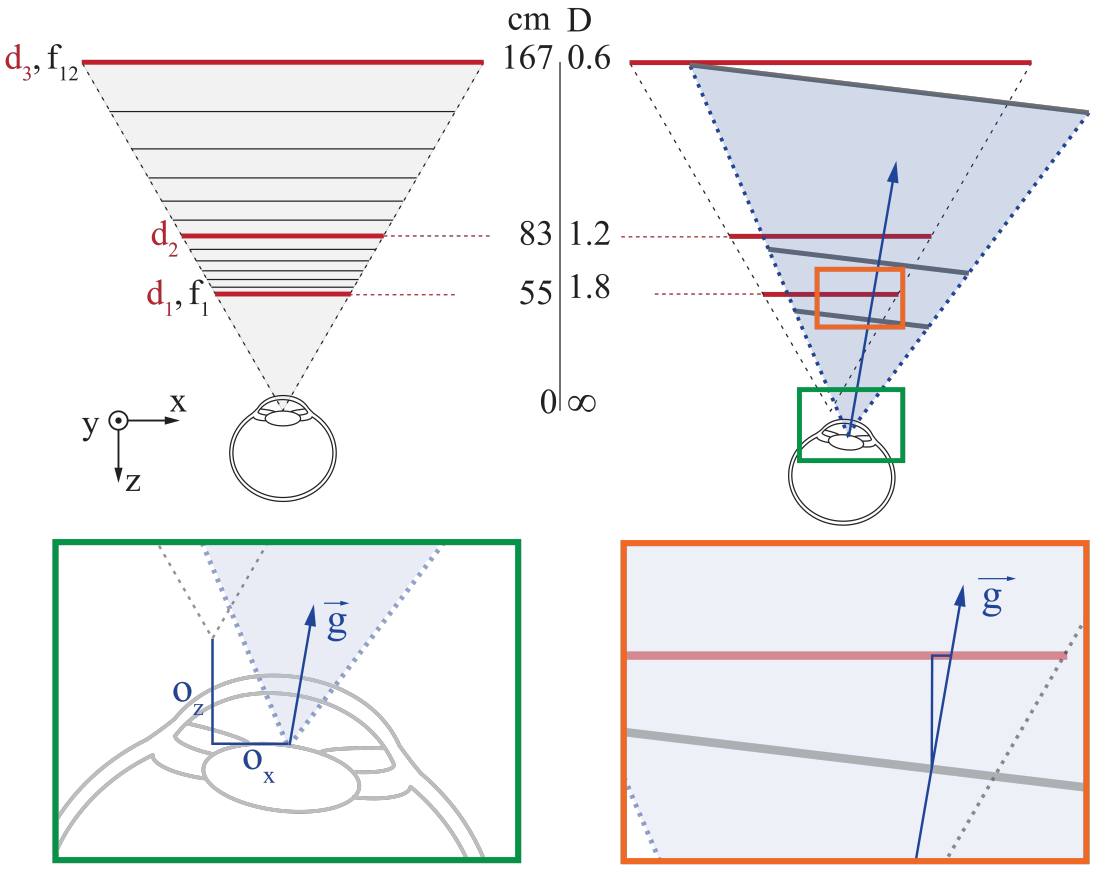


Figure 4.2. (*Top left*) Multifocal display diagram, showing the idealized configuration of an eye perfectly aligned with the display view frustum, for $D = 3$ displays and $F = 12$ focal slices. Distances d_i and f_i are in diopters. (*Top right and insets*) In practice, eye rotations and head movements relative to the display cause the entrance pupil, at position $\mathbf{o}$ and with unit gaze direction $\mathbf{g}$, to become misaligned with the displays, distorting the perceived images.

We formalize the optimal decomposition of a scene onto D display planes as the solution of the following constrained block-matrix system:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\| \overbrace{\begin{pmatrix} \mathsf{K}_{11} & \dots & \mathsf{K}_{1D} \\ \vdots & \ddots & \vdots \\ \mathsf{K}_{F1} & \dots & \mathsf{K}_{FD} \end{pmatrix}}^{\mathsf{K}} \overbrace{\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_D \end{pmatrix}}^{\mathbf{x}} - \overbrace{\begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_F \end{pmatrix}}^{\mathbf{b}} \right\| \tag{4.1}$$

$$\text{such that } 0 \leq \mathbf{x}_i \leq 1, \ \ \forall i. \tag{4.2}$$

Here, $\| \cdot \|$ is the Euclidean norm and we define:

- $\mathsf{K}_{fd} \in \mathbb{R}^{N^2 \times N^2}$ as the *kernel sub-matrix* for focal slice f and display d (see below),
- $\mathbf{x}_d \in \mathbb{R}^{N^2}$ as the unknown optimal pixel intensities for display d, and
- $\mathbf{b}_f \in \mathbb{R}^{N^2}$ as the known pixel values of focal slice f.

The pixels of every display and focal slice are linearized to form vectors $\mathbf{x}$ and $\mathbf{b}$. Each column of a kernel sub-matrix K_{fd} corresponds to the discretized point spread function (PSF) of a given pixel on display d viewed while focusing at distance f, which we refer to as the *kernel* of this pixel. Each column of the entire kernel matrix K therefore comprises the kernels of a displayed pixel as focus spans that of the whole focal stack. The constraints in Equation 4.2 are necessary to model the finite, nonnegative range of display intensities.

The system in Equation 4.1 can be solved using the normal equations

$$(\mathsf{K}^\top \mathsf{K})\, \mathbf{x} = \mathsf{K}^\top \mathbf{b} \ . \tag{4.3}$$

Solving directly for $\mathbf{x}$ in Equation 4.3 will not generally give a solution that satisfies the constraints, but it provides a way of approaching the constrained solution. We thus study the unconstrained normal equations here, and will discuss constraints further in Section 4.3.3.

It is useful to expand the left-hand side of Equation 4.3 as

$$\begin{pmatrix} \sum \mathsf{K}_{i1}^\top \mathsf{K}_{i1} & \dots & \sum \mathsf{K}_{i1}^\top \mathsf{K}_{iD} \\ \vdots & \ddots & \vdots \\ \sum \mathsf{K}_{iD}^\top \mathsf{K}_{i1} & \dots & \sum \mathsf{K}_{iD}^\top \mathsf{K}_{iD} \end{pmatrix} \equiv \begin{pmatrix} \mathsf{C}_{11} & \dots & \mathsf{C}_{1D} \\ \vdots & \ddots & \vdots \\ \mathsf{C}_{D1} & \dots & \mathsf{C}_{DD} \end{pmatrix}.$$

We can similarly expand the right-hand side as:

$$\mathsf{K}^\top \mathbf{b} = \begin{pmatrix} \sum_{i=1}^{F} \mathsf{K}_{i1}^\top \mathbf{b}_i \\ \vdots \\ \sum_{i=1}^{F} \mathsf{K}_{iD}^\top \mathbf{b}_i \end{pmatrix} \equiv \begin{pmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_D \end{pmatrix} \ .$$

This allows us to re-write Equation 4.3 more concisely as

$$\mathsf{C}\, \mathbf{x} = \mathbf{r} \tag{4.4}$$

with $\mathsf{C} \in \mathbb{R}^{DN^2 \times DN^2}$ and $\mathbf{r} \in \mathbb{R}^{DN^2}$.

Recalling that columns of K_{ij} are pixel kernels for a single focal slice, the $(a,b)^{\text{th}}$ element of C_{ij} corresponds to the sum of correlations of pixel a's kernel in display i and pixel b's kernel in display j. Similarly, the a^{th} element of $\mathbf{r}_i$ is the sum of the correlations of pixel a's kernel in display i with each focal slice.

Rewritten this way, we see that $\mathsf{C}\,\mathbf{x}$ is a discrete convolution of the displayed images with summed cross-correlated kernels, reducing the optimal decomposition problem to that of discrete deconvolution.

We can compute the kernel for any displayed pixel directly and accurately with a virtual scene consisting of a plane positioned at the same distance as the physical display of that pixel. If we discretize the plane geometry with an $N \times N$ grid, "activate" (i.e., render with unit intensity) the grid element aligned with the pixel of interest, and then render the resulting focal stack, we can compute the pixel's kernels across the focal stack.

If we represented the eye with a physically accurate model, the kernel of each pixel would need to be computed independently since the PSF of a human eye depends non-linearly on position and accommodation distance. All the matrix and vector elements in C and $\mathbf{r}$ would therefore need to be computed independently, requiring significant storage and processing: for example, for 8-bit monochromatic images with $N = 1024$ and $D = 3$, matrix C would require 9 TB of memory. As such, we adopt several carefully chosen approximations to afford a practical, yet accurate, formulation.

4.3.2. A Thin Lens Approximation of Defocus Blur

To simplify computations, we approximate the optics of the human eye as an ideal thin lens system. This approximation is common in graphics [160] and has also been adopted in the vision science community, particularly for multifocal displays [156]. Our derivation applies an additional small angle (paraxial) approximation, which is valid since display and focal distances $1/d_i$ and $1/f_i$ are large compared to the pupil diameter ϕ (given in meters). With these approximations, kernels obtained from the thin lens model are spatially invariant and constant over a circular support. We can also express the image formation in terms of tangent angles, i.e., a point in focus on focal slice f positioned $\mathbf{p}$ meters away from the optical axis maps to $\mathbf{p}/(1/f)$ in image space.

For focal slice f, the kernel $k(\mathbf{p})$ of a pixel at position $\mathbf{p}_0$ on display d is

$$k(\mathbf{p}) = \text{circ}\left(\frac{(\mathbf{p}-\mathbf{p}_0)}{(\phi/2)\,|d-f|}\right), \tag{4.5}$$

where $\mathbf{p}$ and $\mathbf{p}_0$ are given in tangent angles, and $\text{circ}(x)$ is 1 inside a unit disk and 0 elsewhere. After rasterization, kernels are normalized to have unit area.

To avoid complications at image boundaries, we add a band of black pixels around the image so that pixels near the boundary can use the same kernels as inner pixels. The

necessary width of this band is easily evaluated from the maximum kernel radius. The value of these black pixels is never changed, and they are removed after optimization. Note that this approach is not applicable to the method of Narain et al., since it changes the frequency information of the images. In our implementation of their method, which we require later for comparison, we use a band of replicated edge pixels around the images with a smooth falloff, as described in their paper.

These approximations drastically simplify the computation of our matrix system and allow us to recast Equation 4.3 in terms of simple image operations. Columns of each sub-matrix K_{fd} now all have the same structure, differing only by a translation. As such, each sub-matrix K_{fd} can be replaced by a single image $\overline{\mathsf{K}}_{fd} \in \mathbb{R}^{N \times N}$ of the kernel for the display's central pixel. All subsequent matrix operations can also be written in terms of kernel images, instead of using large, impractical kernel matrices. We arrive at

$$\overline{\mathsf{C}}_{ij} = \sum_{f=1}^{F} \overline{\mathsf{K}}_{fi} * \overline{\mathsf{K}}_{fj} \quad \text{and} \quad \overline{\mathbf{r}}_i = \left(\sum_{f=1}^{F} \overline{\mathsf{K}}_{fi} \right) * \mathbf{b}_i \ , \tag{4.6}$$

and we need to solve the system $\overline{\mathsf{C}} \star \overline{\mathbf{x}} = \overline{\mathbf{r}}$, or explicitly

$$\begin{pmatrix} \overline{\mathsf{C}}_{11} & \dots & \overline{\mathsf{C}}_{1D} \\ \vdots & \ddots & \vdots \\ \overline{\mathsf{C}}_{D1} & \dots & \overline{\mathsf{C}}_{DD} \end{pmatrix} \star \begin{pmatrix} \overline{\mathbf{x}}_1 \\ \vdots \\ \overline{\mathbf{x}}_D \end{pmatrix} = \begin{pmatrix} \overline{\mathbf{r}}_1 \\ \vdots \\ \overline{\mathbf{r}}_D \end{pmatrix} \ , \tag{4.7}$$

where the correlation ($*$) and the convolution ($\star$) of a matrix of images with a vector of images is defined as a regular scalar matrix multiplication, replacing multiplications of scalars by the corresponding image operation. The addition of images is computed pixelwise. Note that since the circular kernels of Equation 4.5 are symmetric, the correlations are convolutions.

Although conceptually similar to the scalar matrix formulation in Equation 4.4, the image formulation we obtain in Equation 4.7 is significantly more compact. The terms $\overline{\mathsf{C}}_{ij}$ and $\sum_{f=1}^{F} \overline{\mathsf{K}}_{fi}$ can be precomputed once as images and easily fit in memory; the matrix of images $\overline{\mathsf{C}}$ now only requires 9 MB of memory.

4.3.3. Solving the Constrained Minimization

Even if unusable in practice, the full scalar matrix formulation in Equation 4.4 allows us to reason about using numerical linear algebra to solve the system more efficiently than in previous work. We detail our optimal decomposition solver, relying on over-relaxed Jacobi iterations [127].

Let λ_d be the scalar value of the central pixel of image $\overline{\mathsf{C}}_{dd}$, and let $\boldsymbol{\lambda}^{-1} = (1/\lambda_1, ..., 1/\lambda_D)^\top$. Given the approximate solution vector $\overline{\mathbf{x}}^{(k)}$ obtained during the k^{th} Jacobi iteration, we can write the next Jacobi iteration $\overline{\mathbf{x}}^{(k+1)}$ of the image matrix

system (Equation 4.7) in the compact image matrix notation of Section 4.3.2 as

$$\overline{\mathbf{x}}^{(k+1)} = (1-\alpha)\,\overline{\mathbf{x}}^{(k)} + \alpha\,\boldsymbol{\lambda}^{-1}\left(\overline{\mathbf{r}} + \boldsymbol{\lambda}\,\overline{\mathbf{x}}^{(k)} - \overline{\mathbf{C}}\,\overline{\mathbf{x}}^{(k)}\right)\,, \tag{4.8}$$

where α is a positive scalar, and the product of scalars $\boldsymbol{\lambda}^{-1}$ with a vector of images simply scales each image by the corresponding scalar entry. We leverage the fact that kernels are non-negative to prove (see Appendix A, Section 4.8) that this iterative process is guaranteed to converge when

$$0 < \alpha < \underbrace{1 \Big/ \left(\sum_{d=1}^{D}\left(\frac{F}{\lambda_d}\right)\right)}_{\hat{\alpha}}\,. \tag{4.9}$$

Empirically, $\alpha = 0.75\,\hat{\alpha}$ yields good results in all of our tests, but a more comprehensive analysis of the system could lead to more insights on optimal α settings.

The only remaining step is to deal with constraints (Equation 4.2). To do so, we simply clamp the pixels of images $\overline{\mathbf{x}}$ to 0 and 1 after each Jacobi iteration. Although simplistic, this projection step does not impact the convergence guarantee (Appendix A, Section 4.8), is easy to implement, and yields consistently good results in practice (see Section 4.6).

4.4. Practical Considerations

The Jacobi iterations of Section 4.3 provide an efficient way of computing the scene decomposition, but do not fully solve the two main issues of current HMDs discussed at the beginning of this paper. This section discusses the correction of errors caused by eye movements, the modification of the objective function to further improve convergence speed, and GPU implementation details. The final algorithm including these modifications is summarized in Algorithm 4.1.

4.4.1. Eye Tracker Deformation

The assumption that the pupil of the user is always at the exact position assumed by the scene decomposition is not easily maintained in practice. This may create plane misalignments that can significantly impair the quality of the perceived images, as shown in Figure 4.3 and in the accompanying video. This problem is still present in very recent multifocal systems [154], and diminishes the impact of new advances in multifocal display technology on practical applications.

Even when trying to constrain the position of a user in a display system, for instance using a bite bar, eye rotations move the position of the pupil relative to the displays because the center of rotation of the eye is located behind the pupil. This type of misalignment is predictable (using eye dimensions from an average viewer) and can be alleviated geometrically without eye tracking. In effect, plane misalignment errors are most noticeable near the region

(a) Ground Truth (b) Linear Blending (c) Optimal Decomposition

Figure 4.3. Small eye offsets have significant effects for all multifocal decomposition methods, most noticeably near large depth discontinuities. For linear blending (b), the errors often appear as salient dark bands across edges. For optimal decomposition (c), the errors show more subtly as additional blur at the edges. The supplementary video also shows the effects of misalignments.

fixated by the user. As proposed by Akeley [122], we can maintain a localized plane alignment by rendering each pixel on given lines of sight from the assumed viewpoint of a rotated pupil. However, this approach requires rendering the scene from multiple viewpoints (one per line of sight), which breaks the assumptions of Section 4.3.2, prevents us from leveraging modern single-viewpoint oriented hardware, and ultimately hinders an efficient implementation of our decomposition. Note that an approximation of this behavior can be obtained by using the center of rotation of the eye as the center of projection of the camera [122], but we do not use this approximation.

More importantly, a local alignment strategy that only corrects for eye rotations still assumes the head of the viewer is perfectly static within the display device. This requirement may be too constraining for practical applications, since users constantly move their head slightly when looking into benchtop systems, and HMDs cannot be perfectly fixed to a user's head. Therefore, because of the higher dimensionality and possibly larger amplitude of viewpoint displacements in practical applications, misalignments are not easily predictable.

We show here how eye tracking can be used to correct for such eye movements. The eye tracker gives the position of the pupil and gaze direction of the user relative to the origin and direction assumed by the decomposition. First, we offset the virtual camera of the renderer to match the eye-tracked position and gaze. The scene decomposition is then carried out normally, but since the displays are now tilted and shifted with respect to the new frame of reference of the virtual camera (Figure 4.2), we cannot simply show the decomposed images on the displays.

The display misalignments can be corrected with a simple image-space deformation of the image computed by the decomposition. This transformation is obtained by directly computing the mapping between the physical pixels of the displays and the pixels of the virtual images used for the decomposition (respectively red and blue planes in Figure 4.2). Let $\mathbf{n} = (n_x,n_y)^\top \in [-1,1]^2$ be the normalized coordinate of a pixel, $\mathbf{g} = (g_x, g_y, g_z)^\top$ and $\mathbf{o} = (o_x, o_y, o_z)^\top$ be respectively the measured gaze direction and eye offset, and let $\mathbf{t} = (t_x, t_y) = (\tan(\text{fov}_x/2), \tan(\text{fov}_y/2))$. The mapping from a physical pixel to a decomposed image is given explicitly by

$$\mathbf{n} \mapsto \frac{\mathsf{M}_1\,\mathbf{n} + \mathbf{v}_1}{\mathsf{M}_2\,\mathbf{n} + \mathbf{v}_2} \tag{4.10}$$

where $\mathsf{M}_1, \mathsf{M}_2 \in \mathbb{R}^{2x2}$ and $\mathbf{v}_1,\mathbf{v}_2 \in \mathbb{R}^2$ are defined as

$$\begin{aligned}
\mathsf{M}_1 &= \begin{pmatrix} d\,g_z\,t_x & 0 \\ -d\,g_x\,g_y\,t_x & d\,t_y\,(1-g_y^2) \end{pmatrix} \\
\mathsf{M}_2 &= \begin{pmatrix} -t_x^2\sqrt{1-g_y^2}\,d\,g_x & -t_x\,t_y\,\sqrt{1-g_y^2}\,d\,g_y \\ -t_y\,t_x\,\sqrt{1-g_y^2}\,d\,g_x & -t_y^2\,\sqrt{1-g_y^2}\,d\,g_y \end{pmatrix} \\
\mathbf{v}_1 &= \begin{pmatrix} (d+o_z)\,g_x - g_z\,o_x \\ g_x\,g_y\,o_x - o_y\,(1-g_y^2)) + (d+o_z)\,g_y\,g_z \end{pmatrix} \\
\mathbf{v}_2 &= \begin{pmatrix} t_x\,\sqrt{1-g_y^2}\,(d\,g_z + o\cdot g) \\ t_y\,\sqrt{1-g_y^2}\,(d\,g_z + o\cdot g) \end{pmatrix}.
\end{aligned} \tag{4.11}$$

This mapping strategy is easy to implement as operations on images, and its exactness is only limited by the precision of the eye tracker. Furthermore, it is completely decoupled from the decomposition strategy, so it can be applied directly to any other decomposition method, including linear blending.

As shown in Figure 4.1(c), displaying the decomposed images deformed by Equation 4.10 exactly solves the display misalignment problem. This is also demonstrated in the accompanying video. Notice that black bands appear at the edges of the displays since the offset virtual view frustum is not entirely contained in the original view frustum formed by the displays. This can be solved by artificially reducing the field of view of the renderer, so the offset virtual view frustum remains within the display frustum for reasonable eye rotations and translations. In practice, we have not found the outside edge artifacts to be disturbing, and we prefer to ignore them in order to maximize the field of view of the system.

4.4.2. Blur Gradient Heuristic

Our optimal decomposition solver can be further improved by investigating the behavior of our Jacobi iterations. As seen in Figure 4.5, the solution of the decomposition after a large

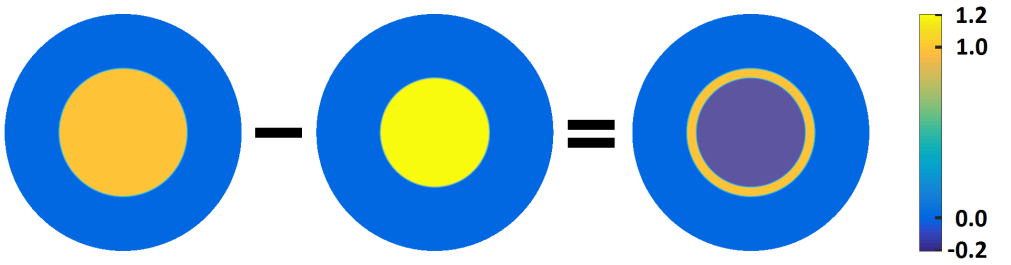


Figure 4.4. The blur gradient kernels are obtained by subtracting the kernels of adjacent focal slices, creating desirable ring structures.

(a) (b) (c)

Figure 4.5. Front plane of the optimal scene decomposition. (a) The converged solution features ring structures around occlusion boundaries. (b) The ring features take a large number of iterations to appear when using our method without the blur gradient modification, and are not visible after only 10 regular Jacobi iterations. (c) Using our blur gradient modification pushes the solution toward the optimal image more aggressively, and the ring structures already start to appear after a single iteration. As verified in Section 4.6.3, this consequently improves the convergence speed of our method.

number of iterations features ring structures around depth discontinuities. These structures appear in the optimal decomposition of most scenes, and therefore seem to be important for the accurate reconstruction of the focal stack, but our algorithm requires many iterations before these patterns emerge.

As mentioned in Section 4.2.2, current perceptual science research suggests the gradient of the blur with respect to changes in depth is key to driving accommodation. To try to force the ring features to appear more quickly, we modify the minimization formulation of Equation 4.1 in order to explicitly include the blur gradient. Since the scene is densely sampled in depth by the focal stack, a gradient in depth can be approximated as finite differences by subtracting adjacent focal slices. We can thus include the gradient term in

the minimization as

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left\|\mathsf{K}\,\mathbf{x}-\mathbf{b}\right\|^2+\beta\left\|\mathsf{K}'\,\mathbf{x}-\mathbf{b}'\right\|^2 \tag{4.12}$$

where β weights the contributions of the reconstructed images and their gradient, and

$$\mathsf{K}' := \begin{pmatrix} \mathsf{K}'_{11} & \dots & \mathsf{K}'_{1D} \\ \vdots & \ddots & \vdots \\ \mathsf{K}'_{F1} & \dots & \mathsf{K}'_{FD} \end{pmatrix}, \quad \mathsf{K}'_{f,d} := \mathsf{K}_{f+1,d} - \mathsf{K}_{f,d} \tag{4.13}$$

$$\mathbf{b}' := \begin{pmatrix} \mathbf{b}'_1 \\ \vdots \\ \mathbf{b}'_F \end{pmatrix}, \quad \mathbf{b}'_f := \mathbf{b}_{f+1} - \mathbf{b}_f\,. \tag{4.14}$$

Note that the new terms in Equation 4.12 are obtained by reusing the already-computed focal stack and blur kernels, so this modification does not incur any significant additional cost. The new minimization can be solved through the normal equations

$$(\mathsf{K}^\top\mathsf{K}+\beta\mathsf{K}'^\top\mathsf{K}')\,x = (\mathsf{K}^\top\mathbf{b}+\beta\mathsf{K}'^\top\mathbf{b}'). \tag{4.15}$$

This system is processed similarly to the original normal equations to obtain an efficient formulation in the image matrix notation of Section 4.3.2, and likewise reduces to simple operations on the blur gradient kernel images $\overline{\mathsf{K}'}_{fd}$ and blur kernel images $\overline{\mathsf{K}}_{fd}$.

The blur gradient formulation, despite its structural similarity to the original minimization, cannot be solved directly with our Jacobi iterations. Since the blur gradient kernels are composed of differences of the original kernels, they are not non-negative kernels, and the convergence criterion of Equation 4.9 does not hold. This results in divergent instabilities in the decomposition after a large number of iterations. Still, we use the blur gradient formulation for the first few iterations of the decomposition, and then revert back to the original formulation to maintain stability. We have found using a single blur-gradient-augmented Jacobi step with $\beta \approx 250$ to be sufficient in our experiments to increase convergence speed, but the optimal values for the weights and the number of blur-gradient-augmented steps remain to be investigated.

Some intuition on the effects of the blur gradient term can be gained by looking at the structure of the blur gradient kernels $\overline{\mathsf{K}'}_{fd}$, shown in Figure 4.4. The new kernels possess ring structures akin to the structures we observe in the converged optimal decomposition in Figure 4.5, which might explain why they improve the convergence of the decomposition. The computational benefits of using the blur gradient are verified in Section 4.6.3.

4.4.3. GPU Implementation

The Jacobi iterations of Equation 4.8 are mostly composed of per-pixel operations, which are implemented in a pixel shader on the GPU. Only the term $\overline{\mathsf{C}}\,\overline{\mathbf{x}}^{(k)}$ requires more attention,

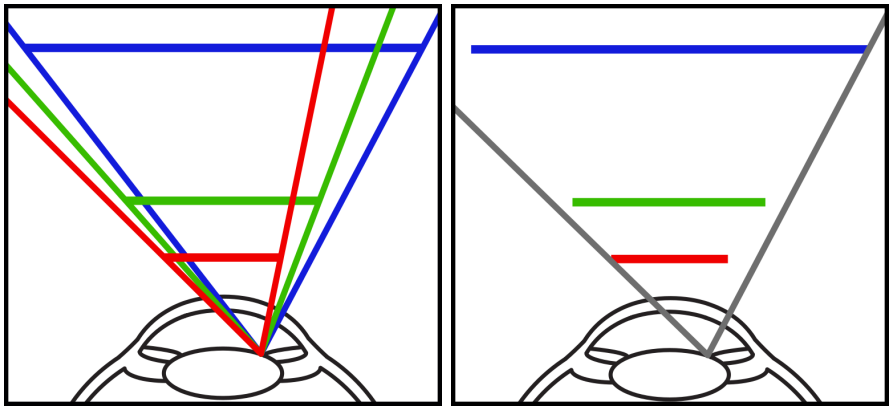


Figure 4.6. To generate the focal stack, we accumulate samples on the pupil of the virtual camera. (*left*) This would usually require one render per sample per focal slice, as shown here for a single pupil sample and $F = 3$ focal slices. (*right*) As explained in Section 4.4.3, we instead use a single render per eye sample which envelops the rendering frustums required by all focal slices, greatly reducing the cost of focal stack generation.

as it corresponds to a convolution. However, for the parameters used throughout this paper, the cross-correlated kernels are small and only convolutions over a few pixels are required. Even if standard GPU convolution techniques can be used [159], a naive implementation summing over neighboring pixels in a pixel shader outperformed all other methods we have tested. Approximate downsampled approaches could also be used, but they would introduce errors in the decomposition and would require further analysis.

Generating the focal stack $\bar{\mathbf{r}}$ in Equation 4.8 is also a challenging part of the decomposition, as it requires accurately rendering depth of field blur for each focal slice. We compute the focal stack by accumulating images over 64 samples on the virtual pupil. As depicted in Figure 4.6, this would usually require $64 \times F$ renders using a standard pinhole rendering pipeline. We instead approximate this process by using a single view frustum per pupil sample which envelops all focal slices, reducing the number of required renders to 64. The enveloping images are rendered at higher resolution, and the image for each focal slice is extracted by cropping the enveloping images. This greatly improves the efficiency of focal stack generation, and we have found the resulting sampling errors to be negligible. More approximate but faster focal stack rendering methods might give superior results, but we prefer to use a slower but accurate focal stack generation method in this paper, so that no additional error is introduced in our analysis.

4.5. Eye-Tracked Multifocal Display Testbed

We build a multifocal display testbed driven by the methods of Sections 4.3 and 4.4, leveraging a combination of off-the-shelf and custom components. Our testbed is purpose-built to explore open questions regarding the accommodation response to, and visual perception of,

multifocal displays, as enabled by our efficient gaze compensation and scene decomposition algorithms. As such, three subsystems are necessary: a binocular multifocal display with a reasonably large field of view to reproduce natural viewing conditions; an eye tracker to account for parallax caused by eye rotations and head movements, following Section 4.4.1; and a way of measuring accommodation to experimentally verify the reaction of subjects under various viewing conditions. This section describes the components selected for our testbed, their use, and their calibration. Figure 4.7 details the construction, and a detailed view of the system is also presented in the accompanying video.

for *each eye* **do**
- update virtual camera to match measured eye position
- render focal stack $\overline{\mathbf{r}}$ (Section 4.4.3)
- initialize $\overline{\mathbf{x}}$ to zero
- do one Jacobi step with blur gradient (Section 4.4.2)
- do S regular Jacobi steps (Equation 4.8)
- apply eye tracker deformation (Equation 4.10)

Algorithm 4.1: Scene decomposition (computed each frame)

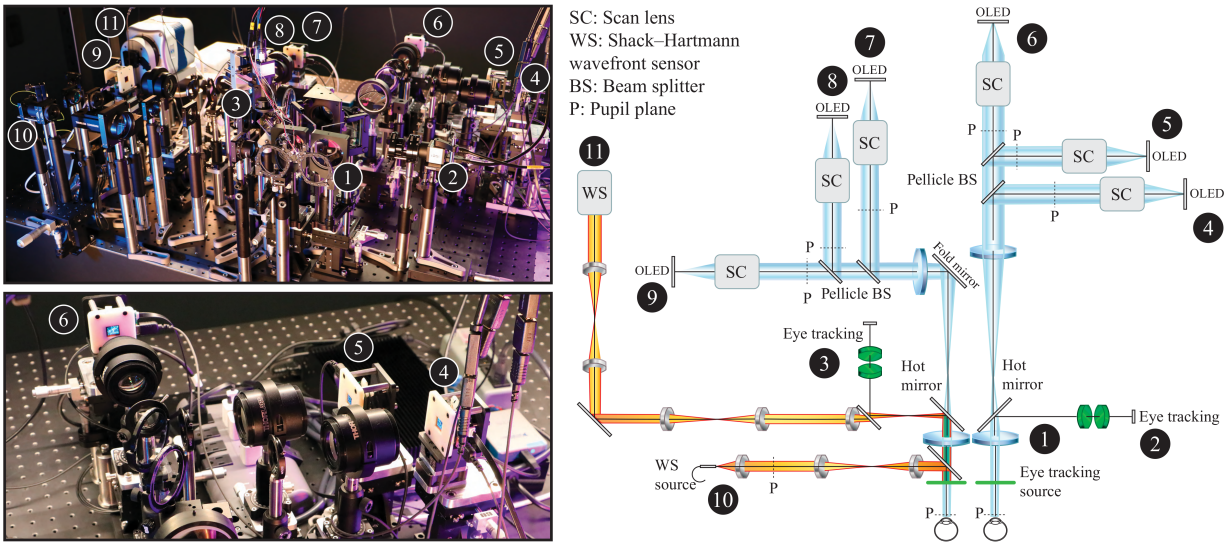


Figure 4.7. Our multifocal testbed includes three primary subsystems, as denoted in the photographs (left) and the optical diagram (right). First, the display subsystem uses three OLED panels per eye to achieve a 17-diopter accommodation range over a 20-degree field of view. Lenses in this subsystem are shaded blue, and blue optical rays are traced from display pixels to the eye box. Second, the eye tracking subsystem comprises a pair of 250 Hz cameras and a set of near-infrared LEDs. Lenses in this subsystem are shaded green. Third, the accommodation measurement subsystem uses a Shack-Hartmann wavefront sensor. Lenses in this subsystem are shaded gray and red optical rays are traced from the illumination source, to the eye, and back to the wavefront sensor. See the supplementary video for additional details.

4.5.1. Displays

The testbed employs six full-color organic light-emitting diode (OLED) display panels (MicroOLED MDP02), each supporting 1280×1024 resolution at a 60 Hz refresh rate. The panels are mounted on motorized translation stages to create three variable-focus virtual image planes per eye. Light from the displays is combined using pellicle beamsplitters and relayed to the eye through a pupil-forming optical system. This pupil-forming system affords a 20-degree field of view, a 10-mm-diameter eye box, and is designed to be telecentric in the virtual image space, so as to maintain image resolution of one arcminute per pixel (i.e, 20/20 visual acuity). Each display panel is independently actuated to address a 17-diopter depth of focus (DOF), and these ranges are staggered to address a total DOF spanning from -5 to $+12$ diopters. This extended range allows for the correction of the spherical component of the viewer's prescription, eliminating the need to use corrective eyewear when viewing the testbed, thereby assisting eye tracking and accommodation measurements. Viewers are positioned, relative to the viewing optics, using a bite bar. Note that the bite bar helps stabilize the user's head, but does not eliminate head movements, so the corrections of Section 4.4.1 are still required in this system. A manual translation stage controls the interaxial distance (IAD) by altering the separation between the right-eye display subsystem and the remainder of the testbed to adjust to the user's interpupillary distance, if necessary.

Because a different display synthesizes each virtual image plane, the system requires accurate radiometric and color calibrations. These calibrations are obtained from measurements of the gamma curves and primary spectra, as recorded with a Photo Research PR-745 SpectraScan Spectroradiometer. A look-up table converts target sRGB image values to color-corrected, display-specific RGB values, following the method of Brainard [126]. The focus of each display was measured using a SID4 wavefront sensor from Phasics Corp. Optical distortions and alignment between the virtual images are measured using a method akin to Gilson et al. [130]. Similar to Watson and Hodges [168], distortions and alignment are corrected by pre-warping imagery on the GPU.

4.5.2. Eye Tracker

The use of eye tracking for multifocal displays has been discussed before, for instance in the early design of Rolland et al. [162]. However, to our knowledge, our testbed is the first to incorporate such eye tracking. We employ a conventional model-based eye tracking algorithm, as surveyed by Hansen and Ji [131], wherein the position and pose of the eyes are estimated by tracking the boundary of the viewer's pupil and the bright reflections of point light sources from the anterior surface of the cornea. The point light sources consist of an array of near-infrared light-emitting diodes (LEDs) placed into a structure in front of the designed eye box. A pair of infrared-sensitive cameras record images focused over a

Method	Time 100 iterations	Precomp.	Per Iteration
Narain CPU	-	-	1.8
Narain GPU	20	1.5	0.185
Ours	2.38	0.5	0.025

Table 4.1. Time comparison (in seconds) of the original CPU implementation of Narain et al. [156], our GPU implementation of their work, and our method. The total and precomputation times for Narain et al. were not reported. We report the time required to compute 100 iterations for image resolution 512×512, and break down timings into precomputations and the iterations themselves.

25-mm-diameter region centered on this eye box at a sampling rate of 250 Hz. Dichroic "hot" mirrors combine the eye tracking and display paths. We emphasize that our development of this eye tracking system closely follows prior constructions, with extended implementation details provided for a similar design by Stengel et al. [165].

4.5.3. Accommodation Measurement

The accommodative state of the viewer's left eye is measured at 67 Hz using the well documented Shack-Hartmann wavefront sensing technique [146]. Our system employs near-infrared light created with a Thorlabs SLD830S-A10 superluminescent diode (SLD) that is coupled to the eye using a weakly reflecting beamsplitter. Light passing through the viewer's eye and reflecting from the retina is separated from the display path using another "hot" mirror and relayed to an Imagine Optic HASO wavefront sensing camera with a 34×34 microlens array achieving a 294 μm pitch at the system entrance pupil.

4.6. Results and Discussion

In this section, we show that our method is an efficient way of solving the optimal decomposition formulation of Equation 4.1, outperforming previous work, and that it unlocks the interactive use of high-quality decompositions for practical multifocal display applications.

4.6.1. Efficiency Versus Previous Work

In their original paper, Narain et al. [156] describe a CPU implementation of their method. They report a computation time of 180 seconds for 100 iterations, or 1.8 seconds per iteration. Note that this time does not appear to include the computation of the ground truth focal stack and other various images (e.g., the Fourier transforms of the PSFs required by their method). To obtain a fair comparison between our method and theirs, we first implement their method more efficiently on the GPU. Doing so, we report a computation time of about 0.185 seconds per iteration for similar conditions, which is an order of magnitude faster. Times are reported in Table 4.1. All computations are done on a 12-core

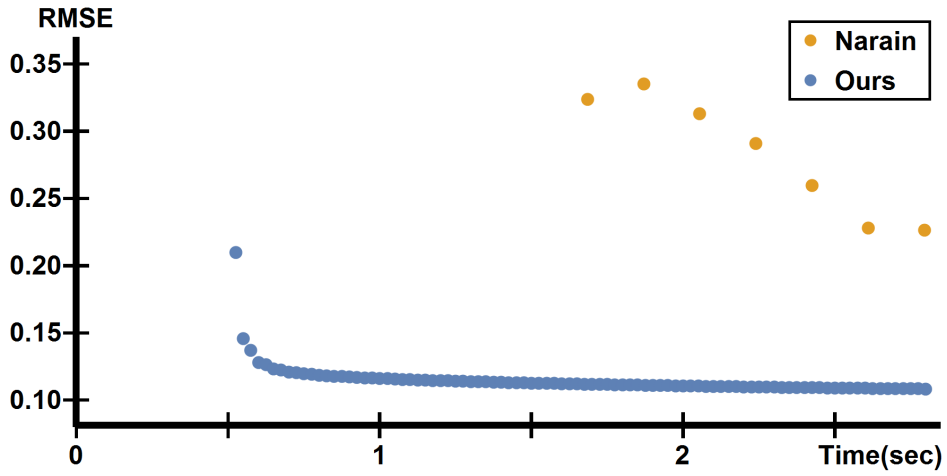


Figure 4.8. Comparing the residual mean square error (RMSE) over time of our GPU implementation of Narain et al. [156] against our method. We can compute 47 Jacobi iterations, which yields a solution close to optimal, before a single iteration of Narain et al. is computed.

3.5 GHz processor and an NVidia TitanX Pascal graphics card. Note that because of possible small differences between our benchmark test and that originally used by Narain et al., the improvement of our GPU implementation could be slightly lower than 10x, but this uncertainty is taken into account later in this section.

The implementation of our Jacobi iterations is also done on the GPU, following Algorithm 4.1. We report a time of 0.025 seconds per frame, which is an order of magnitude faster than our GPU implementation of Narain et al. The reason for this significant speed up in our implementation is due to fact that the method of Narain et al. solves the deconvolution problem in Fourier space, but applies the constraints projection in the primal domain, which requires two Fourier transforms per iteration. Convolutions in Fourier space become pointwise multiplications, which is efficient for very large kernels. However, in our case, the radii of the kernels are fairly small, especially for a plane spacing of 0.6 diopters. We found it much faster to compute the convolutions directly in the primal domain, as described in Section 4.4.3.

Figure 4.8 compares our Jacobi method with our GPU implementation of Narain et al. We use scene A (shown in Figure 4.10), and image resolution 512×512. We use the residual mean square error (RMSE) to compare each reconstructed focal slice to a ground truth image rendered with correct defocus blur. The errors are averaged over the focal range at twice the frequency used by the decomposition, i.e., we average the errors over 23 focal slices whereas the decomposition uses $F = 12$ for all results in this paper. Doing so verifies that no large error appears between the focal slices used by the decomposition.

Our method converges to the optimal solution faster than previous work, both in terms of number of iterations and computation time. We can use at least 10 times fewer iterations

with our method compared to Narain et al. to reach the same image quality. As such, the computational time can be further divided by 10, which gives a total of **three orders of magnitude** improvement in computational time for our method compared to the original implementation of Narain et al. Note that, according to Figure 4.8, the improvement in number of iterations for our Jacobi method over our GPU implementation of Narain et al. is actually significantly more than 10 fold. We report this conservative value to account for the uncertainty, described earlier in this section, related to the comparison between the CPU and GPU implementations of Narain et al.

With our current implementation, we can thus run the optimal decomposition of scenes at 5 frames per second (FPS) for a 512×512 image resolution. As indicated by the steep slope of the error curve in Figure 4.8 for a low number of iterations, we begin to notice errors if we reduce this number of iterations further. Note that this timing does not include precomputation times, which are mostly comprised of the focal stack generation. As seen in Table 4.1, we clock our focal stack generation at roughly 2 FPS, but we emphasize that this is highly dependent on the renderer, scene complexity, and focal stack generation method. We use 64 pupil samples and 12 focal slices throughout this paper, which generates a high-quality focal stack and allows us to avoid the effects of focal stack errors in our analysis. However, it is very likely that much more efficient focal stack generation methods can be employed. For instance, a simple reduction in the number of pupil samples would directly reduce precomputation times. Furthermore, using well-known approximations, such as a reverse-mapped z-buffer, would trivially bring focal stack generation to real-time rates. Determining whether such fast focal stack generation methods are perceptually sufficient is an interesting research avenue that our system enables.

This performance allows us to compute the optimal decomposition of dynamic content with good quality at interactive frame rates. Figure 4.9 shows images captured within our system, and the accompanying video shows a sequence with dynamic content captured in real-time in our testbed. However, faster framerates are desirable, and the resolution of the images (stretched to fill the displays vertically) is only half the maximal resolution of our displays. The performance of our method is highly sensitive to differences in equipment and implementation details, and we believe that the significant improvements we report in comparisons with previous work, both in equal time and equal number of iterations, confirm the fundamental benefits of our method. By decreasing optimal decomposition times from the order of minutes to milliseconds, we believe the path to true real-time performances becomes a manageable problem of hardware and implementation efficiency.

4.6.2. Equal Time Comparison

We use our Jacobi iterations with the blur gradient modification to solve the optimal decomposition for a variety of different scenes, shown in Figure 4.10. We test our method for

a display spacing of 0.6 diopters, as recommended by current research [151], but also for larger display distances of up to 2 diopters to test the possibility of covering larger accommodation ranges. We use $F = 12$ focal slices, and image resolution 512×512. Since the goal of scene decomposition is to reproduce the focal stack as closely as possible, Figure 4.10 uses the popular perception-based HDR-VDP-2 metric [153] to compare reconstructed focal stacks with ground truth images rendered with correct defocus blur. The HDR-VDP-2 metric gives the probability for an average user to detect differences between two images, which we use to compare reconstructed and reference focal slices. The probability is then averaged over the whole focal range, sampled at twice the frequency used by the decomposition (similarly

Figure 4.9. Captures from our testbed with a camera focused at 0.6 diopters. The accompanying video also shows captures of focal stacks for dynamic content.

Metric	Method	Scene				
		A	B	C	D	E
Q	Converged	72.46	67.70	73.59	74.93	74.84
	Linear	66.85	61.70	60.78	66.52	57.92
	Narain	63.66	62.91	64.09	65.28	62.99
	Ours	71.80	66.52	72.92	74.24	72.25
RMSE	Converged	0.0974	0.0794	0.0441	0.0913	0.0413
	Linear	0.1324	0.1760	0.1259	0.1297	0.1506
	Narain	0.3237	0.3910	0.2965	0.4275	0.1792
	Ours	0.1210	0.1170	0.0700	0.1295	0.0515

Table 4.2. Quantification of the error for the decomposition methods and scenes of Figure 4.10. We use an equal-time comparison at 5 frames per second, which corresponds to 1 iteration of Narain et al. and 8 iterations of our method. The HDR-VDP-2 Q metric (higher is better) and the RMSE (lower is better) are averaged over the entire focal stack. In all scenes, our method beats both linear blending and Narain et al. in both metrics.

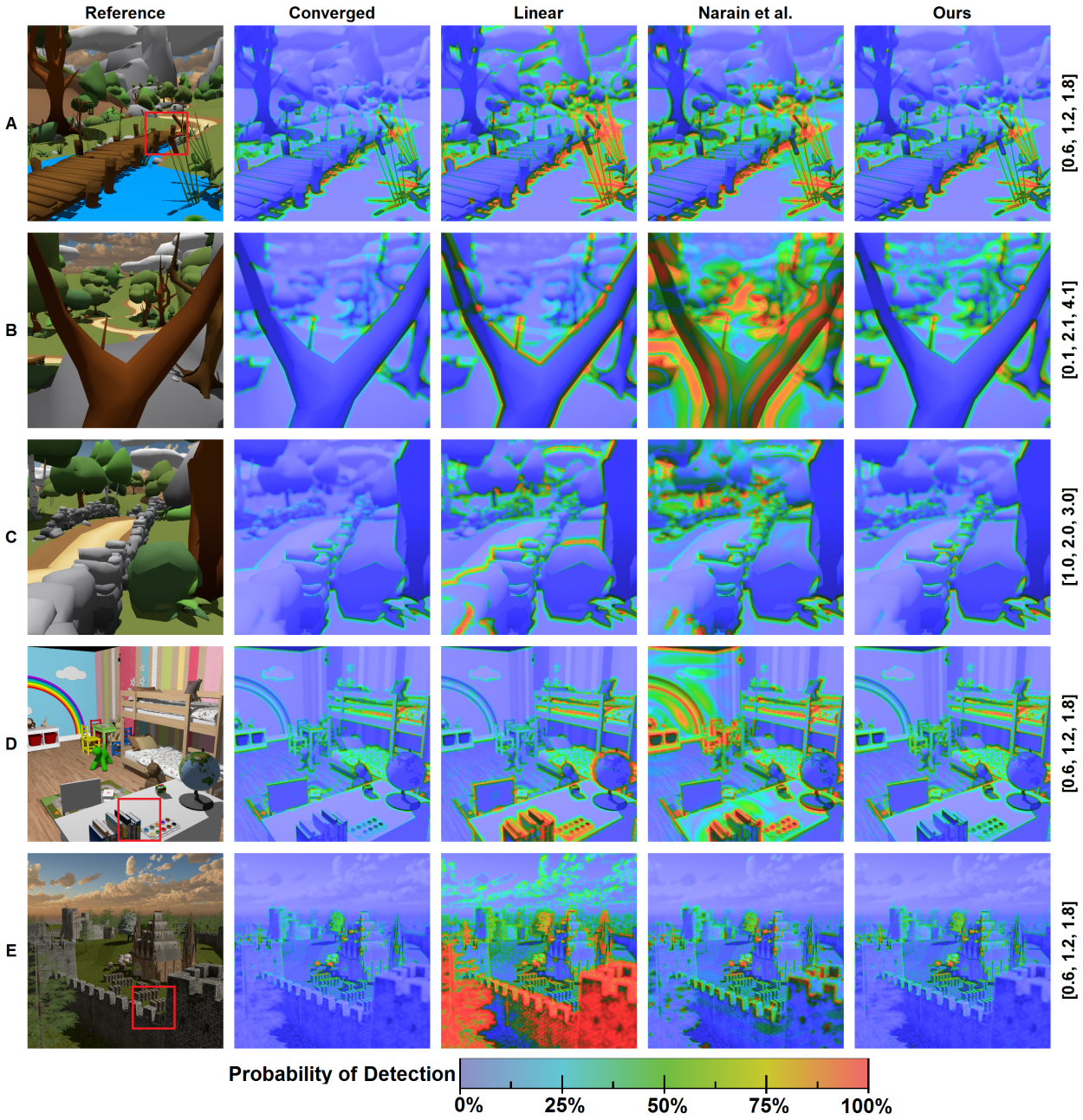


Figure 4.10. Comparison of the HDR-VDP-2 metric for different decomposition methods applied to various scenes, compared to the ground truth focal stack. We average the metric over the depth range spanned by the displays, so the colors can be interpreted as the probability of detection of differences between the reconstructed and original focal stacks. The reference images (column 1) show the scenes (identified A to E) viewed from a pinhole camera, without any defocus blur. The insets in the reference images for scenes A, D and E are used in Figure 4.11. To compare Narain et al. (column 4) with our method (column 5), we use an equal-time comparison at 5 frames per second, ignoring precomputation time. In this time, we can afford 1 iteration of Narain et al. and 8 iterations of our Jacobi method. We also compare both methods to linear blending (column 3) and to the converged solution of the optimal decomposition (column 2), which we compute using 10,000 iterations of our method. The numbers on the right give the display plane positions used for each scene. For all scenes, our method gives better results than both Narain et al. and linear blending at this interactive frame rate. These results are also quantified in Table 4.2.

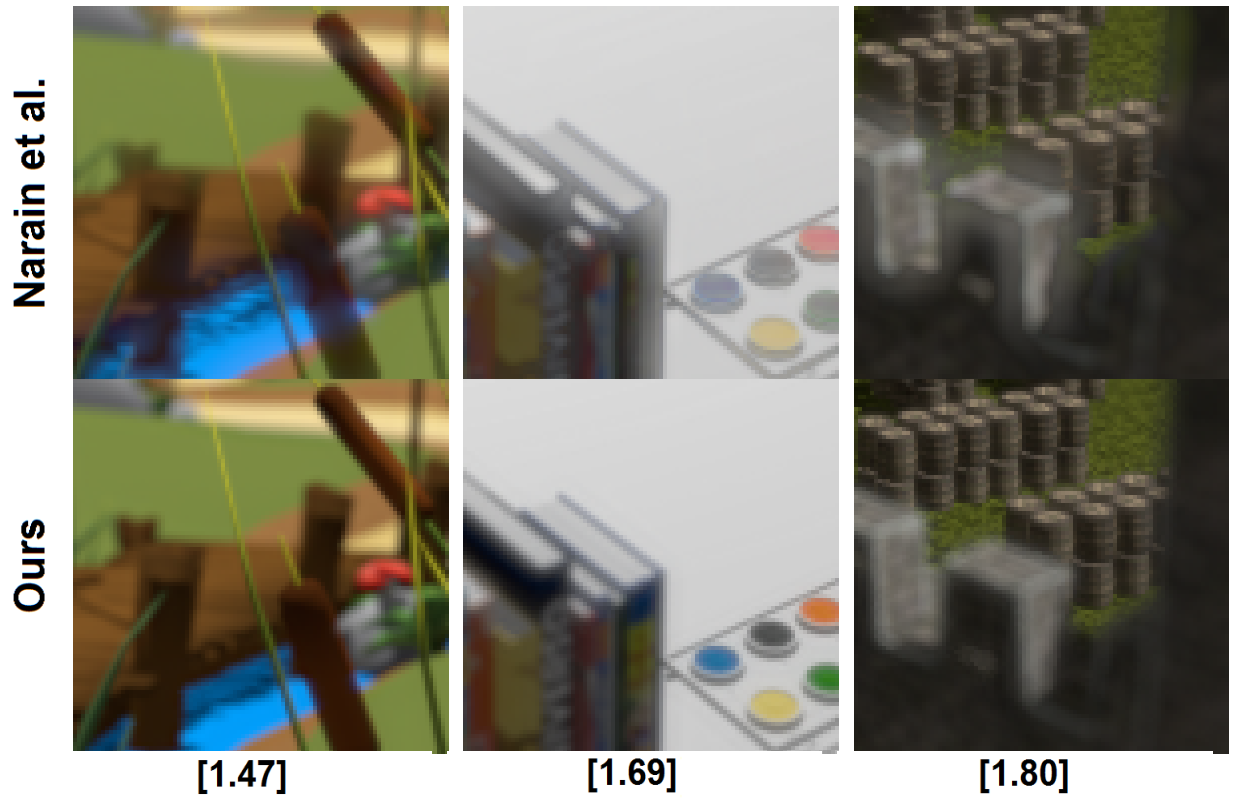


Figure 4.11. Insets of scenes A, D and E from Figure 4.10, using an equal-time comparison of Narain et al. and our method at 5 frames per second. Images are taken at a single focal depth indicated in square brackets. This shows how the errors detected by the HDR-VDP-2 metric of Figure 4.10 translate to visible artifacts in the reconstructed images. Note how Narain et al. washes out the colors and makes parts of the scene bleed into each other, for instance on the left where the river is visible through the cattail, while our method gives a sharper reconstruction.

to Section 4.6.1). Furthermore, as discussed in Section 4.3.2, the different decomposition methods treat boundary pixels differently. We therefore remove an additional small band of pixels around the images before comparing them to reduce the possible effect of boundary treatment on the image quality metrics.

Since our Jacobi approach and that of Narain et al. are based on the same objective function, they will ultimately converge to similar solutions after a large number of iterations. Focusing on interactive applications, we use an equal time comparison at 5 FPS, without counting precomputation time. In this period, we can compute one iteration of our GPU implementation of Narain et al., and eight iterations of our Jacobi method. Both methods are also compared to linear blending [123], and to the converged solution of the optimal decomposition, computed using 10,000 iterations of our Jacobi method. Note that 5 FPS is the fastest frame rate we can use for the equal-time comparison since we need to compute at least one step of Narain et al. Slower frame rates could be used, but this would only

improve the advantage of our method compared to linear blending, and would reduce the gap between our method and Narain et al., making the analysis less clear.

Table 4.2 also compares the methods and scenes of Figure 4.10 quantitatively. HDR-VDP-2 provides a global image quality metric Q, which we use to compare reconstructed and reference focal slices, again averaging over the whole focal stack. We also give the same comparison using the RMSE metric, which is proportional to the quantity minimized by the optimal decomposition formulation of Equation 4.1.

For all five test scenes and all metrics, our method performs better than both Narain et al. and linear blending, reaching decomposition results that are perceptually close to the converged solution, at interactive frame rates. Note that even though the RMSE metric sometimes gives similar values for our method and linear blending (e.g. in Scene D), the Q metric and the images in Figure 4.10 distinctly highlight the advantages of our method. Note also that using the RMSE metric in Figure 4.10, or using the maximum error over the focal range instead of the average, gave similar results in all cases.

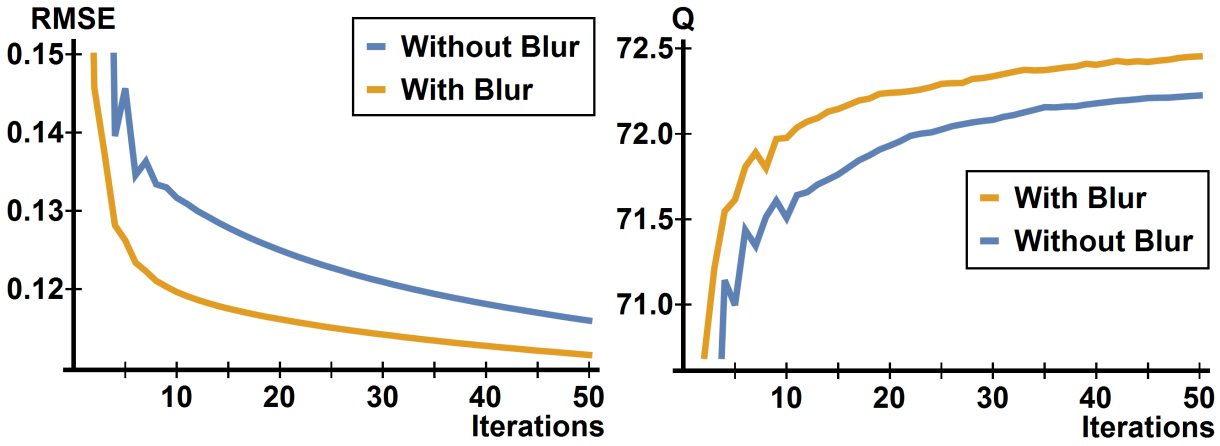
Figure 4.11 shows insets for three of the scenes presented in Figure 4.10, as indicated by red squares in the reference images of that figure. These insets compare our method to Narain et al. for a given focal slice and show that the differences identified by the HDR-VDP-2 metric do indeed correspond to perceivable differences in the reconstructed images. In general, for a low number of iterations, the method of Narain et al. tends to create halos around objects, and generates blurrier images with colors from different objects bleeding into one another. This is also visible in the equal time comparison present in the accompanying video.

4.6.3. Blur Gradient Evaluation

All results presented in Sections 4.6.1 and 4.6.2 use the blur gradient modification described in Section 4.4.2. Figure 4.12 compares the errors obtained with and without this blur modification, using both the HDR-VDP-2 Q and RMSE metrics described in Section 4.6.2. As done in previous sections, the metrics are computed by averaging over the whole focal stack, sampled at twice the depth frequency used by the decomposition. From this figure, we see that the blur modification does indeed improve the performance of our method, reducing the number of iterations (and therefore the computation time) required to reach a given error by roughly 2 to 4 times.

4.6.4. Accommodation of Human Subjects

We tested the capabilities of our system in a pilot user study where we compared the accommodation responses of users looking at dynamic content decomposed using linear blending and our optimal decomposition method. We collected accommodation responses from four observers to a target oscillating sinusoidally in depth between 0.6 and 1.8 diopters at



(a) (b)

Figure 4.12. Comparison of our method with and without the blur gradient modification of Section 4.4.2. Both in the RMSE (lower is better) and HDR-VDP-2 Q (higher is better) metrics, our method reaches a given error roughly 2 to 4 times faster when using the blur gradient.

a rate of 0.1 hertz. Observers were asked to maintain fixation on the target, and the image deformation of Section 4.4.1 was used to adjust to the user's pupil location. The target consisted of a Snellen eye chart embedded into scene C of Figure 4.10. The target size was held constant (2.4 degrees wide, letter size 0.2-0.7 degrees) in order to remove looming as a potential cue to accommodation. Three repetitions of the movement were collected over 30 seconds for each of the four observers. Observers viewed the scene monocularly to avoid the influence of binocular cues (e.g., vergence distance), and to ensure the changes in accommodation were driven by retinal blur alone.

The results of this study are shown in Figure 4.13. Individual observer responses were shifted relative to the stimulus position in order to align responses while accounting for subtle shifts in the accommodation response unique to each observer's optics. This was done by computing the average accommodative position through the captured sequence, and then shifting the responses by an amount equal to the difference between that average and the average stimulus position (1.2 diopters).

The results shown in Figure 4.13 indicate that both decomposition methods provide a stimulus that drives the accommodation response. For linear blending, we replicate the findings expected from literature [152], with an accommodative gain of about 0.61. Our Jacobi algorithm also drove changes in the accommodation response, but with a significantly lower gain than linear blending, as confirmed with a repeated measures t-test (0.28, $t(3) = 5.08$, $p=0.015$).

Measuring the modulation transfer function (MTF) under both decompositions can help explain the results of the user study. Figure 4.14(a) shows the MTF measured with a camera looking at a point stimulus in the system, decomposed with either linear blending

or optimal decomposition. In all cases, the stimulus is placed between two display planes at 1.5 diopters, and the camera is focused at this same depth. We see that linear blending has a higher MTF, notably in the 4-8 cycles per degree range which maximizes the signal to accommodation [151]. Figure 4.14(b) shows the MTF for the same stimulus, but captured virtually in software. Since the stimulus is a point, and the virtual system is not diffraction limited, the ground truth MTF is constant. Again, we see that linear blending gives a better MTF than optimal decomposition. The method itself is therefore responsible for at least part of the drop in relative contrast observed in the real MTF of Figure 4.14(a).

Many factors could explain the lower MTF and accommodation gain of optimal decomposition. For instance, optimal decomposition reduces high spatial frequencies at the display planes [156], which can thus reduce the strength of the accommodative signal and the MTF, even with a virtual camera. This is particularly important for the scene we used because of the large depth discontinuity at the edges of the eye chart. Furthermore, the display alignment appears to be more critical for optimal decomposition since it distributes light across all three planes, while linear blending distributes light to the two nearest ones. Small calibration errors could therefore increase image blur for optimal decomposition, further reducing the strength of the accommodative signal and the captured MTFs of Figure 4.14.

These results and their explanation require a more in depth investigation, but the study illustrates that optimal decomposition does provide a stimulus that can drive accommodation. We reiterate that this user study is preliminary and only serves to demonstrate the capabilities of our system. This simple user study already raises many questions and possible research avenues, such as the possibility of modifying the objective function to optimize for the MTF directly, which shows the potential and usefulness of our testbed.

4.7. Discussion and Conclusion

We have presented significant, necessary improvements over the current state-of-the-art in multifocal displays. Our efficient scene decomposition method unlocks the use of optimal decomposition for high-quality interactive applications. We have also demonstrated how eye tracking can be used to efficiently maintain plane alignment in multifocal displays.

The way current display technologies drive accommodation is still under active investigation [143], but accommodation in multifocal displays has so far been difficult to study due to impractical decomposition times, misalignment issues, and the difficulty of integrating measurement paths to a multifocal system. By combining eye tracking and accommodation measurement with our interactive decomposition algorithm, our multifocal testbed is the first to fully enable the investigation of many open questions regarding multifocal displays and the human visual system.

Many of these open questions are intricately coupled with the design of our system. For instance, we hope to investigate the required precision and latency of eye tracking, the

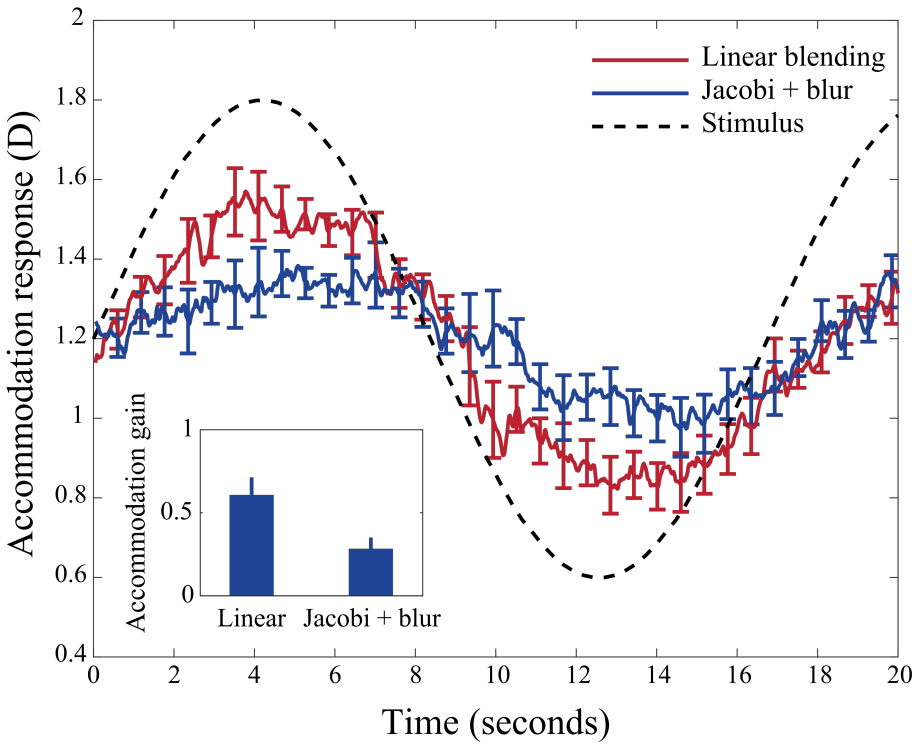


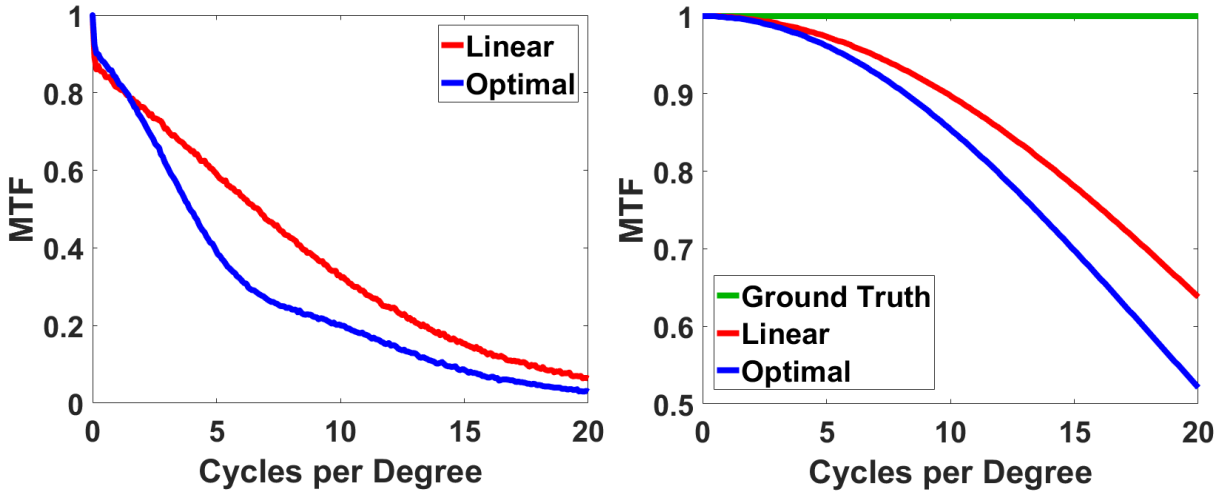
Figure 4.13. Results of accommodation measurement to both linear blending (red), and our optimal decomposition method (blue). The black dashed curve shows the stimulus profile. Error bars represent +/-1 standard error of the mean. Accommodative gains (inset) were obtained by computing the difference between the maximum and minimum responses during the stimulus movement, and scaling the difference by the amplitude of the stimulus movement. These results show, for the first time, that optimal decomposition does indeed drive accommodation, albeit with a lower gain than linear blending.

effects of our blur gradient heuristic and better optimization functions on accommodation, and the relation between the error metrics and the perceived realism, quality and comfort in multifocal displays. By its significant form-factor, our testbed is however limited to investigating the accommodation of static users, and cannot be used to study such open questions relating to the interactions of more depth cues as a user moves freely in a virtual environment. We hope that a better understanding of these questions will allow us to improve our testbed, and in turn guide the design of future multifocal displays.

4.8. Appendix A: Convergence Proof

We prove the convergence criterion of Equation 4.9. Because the sub-matrices of $\mathsf{K}^\top\mathsf{K}$ are definite-positive, C only has positive eigenvalues, and the convergence criterion for over-relaxed Jacobi iterations [127] is

$$\alpha < \hat{\alpha} := \frac{2}{\rho(\Lambda^{-1}\,\mathsf{C})} \tag{4.16}$$



(a) (b)

Figure 4.14. MTF measurements for a point stimulus placed at 1.5 diopters and decomposed using either linear blending or optimal decomposition. The camera is also focused at 1.5 diopters. (a) MTF using a camera looking into our system. (b) MTF computed in software using a virtual camera with a 5mm aperture.

where Λ is the diagonal matrix of C and ρ denotes the spectral radius. The proof thus reduces to computing the largest eigenvalue of $\Lambda^{-1}\,\mathsf{C}$.

For a single display d and a single focal slice f, the action of $\Lambda^{-1}\,\mathsf{C}$ is a convolution by the kernel image $\frac{1}{\lambda_d}\overline{\mathsf{K}}_{fd}^{\top} * \overline{\mathsf{K}}_{fd}$. The largest eigenvalue for this case is obtained by finding the eigenvector image $\overline{\Omega}$ with largest norm after convolution.

By Parseval's identity, the problem can be solved equivalently in Fourier space. Denoting the Fourier transform by $\mathcal{F}$, we can decompose the convolution into a pixelwise multiplication as

$$\mathcal{F}\left(\left(\frac{1}{\lambda_d}\overline{\mathsf{K}}_{fd}^{\top} * \overline{\mathsf{K}}_{fd}\right) \star \overline{\Omega}\right) = \mathcal{F}\left(\frac{1}{\lambda_d}\overline{\mathsf{K}}_{fd}^{\top} * \overline{\mathsf{K}}_{fd}\right) \cdot \mathcal{F}(\overline{\Omega})\ . \tag{4.17}$$

The largest norm after convolution is thus obtained by using the image $\overline{\Omega}$ which only contains the frequency with the largest amplitude in $\mathcal{F}(\overline{\mathsf{K}}_{fd}^{\top} * \overline{\mathsf{K}}_{fd})$. Because our kernels are positive, the largest amplitude is located at frequency (0,0). The image with the largest norm after convolution is thus a constant image, which is also trivially an eigenvector image under convolution.

Because the kernels are normalized, the convolution of a constant image $\overline{\Omega}_i$ on display i with kernel $\frac{1}{\lambda_d}\overline{\mathsf{K}}_{fd}^{\top} * \overline{\mathsf{K}}_{fd}$ results in the uniform image $\frac{1}{\lambda_d}\overline{\Omega}_i \quad \forall f$. Combining all displays and all focal slices then yields the eigensystem

$$\sum_{f=1}^{F}\sum_{i=1}^{D}\frac{1}{\lambda_d}\overline{\Omega}_i = \frac{F}{\lambda_d}\sum_{i=1}^{D}\overline{\Omega}_i = \gamma\overline{\Omega}_d \quad d \in \{1, ..., D\} \tag{4.18}$$

for eigenvalue γ, whose solution is

$$\gamma = \sum_{i=1}^{D} \frac{F}{\lambda_i} \,. \tag{4.19}$$

Combining this result with Equation 4.16 gives the convergence criterion of Equation 4.9.

References

[122] Akeley, K. 2004, Achieving near-correct focus cues using multiple image planes, Ph.D. Thesis, Stanford University.

[123] Akeley, K., S. J. Watt, A. R. Girshick and M. S. Banks. 2004, A stereo display prototype with multiple focal distances, *ACM Transactions on Graphics (TOG)*, vol. 23, n° 3, p. 804–813, ISSN 0730-0301.

[124] Alpern, M. 1958, Variability of accommodation during steady fixation at various levels of illuminance, *Journal of the Optical Society of America*, vol. 48, p. 193–197.

[125] Blundell, B. and A. Schwartz. 1999, Volumetric three-dimensional display systems, Wiley-IEEE Press.

[126] Brainard, D. H. 1989, Calibration of a computer controlled color monitor, *Color Research and Application*, vol. 14, n° 1, p. 23–34.

[127] Burden, R. L. and J. D. Faires. 2011, Numerical analysis, 9th international edition, *Brooks/Cole, Cencag Learning*.

[128] Burge, J. and W. S. Geisler. 2011, Optimal defocus estimation in individual natural images, *PNAS*, vol. 108, p. 16 849–16 854.

[129] Dunn, D., C. Tippets, K. Torell, P. Kellnhofer, K. Akşit, P. Didyk, K. Myszkowski, D. Luebke and H. Fuchs. 2017, Wide field of view varifocal near-eye display using see-through deformable membrane mirrors, *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, n° 4, p. 1322–1331.

[130] Gilson, S. J., A. W. Fitzgibbon and A. Glennerster. 2011, An automated calibration method for non-see-through head mounted displays, *Journal of Neuroscience Methods*, vol. 199, n° 2, p. 328–335.

[131] Hansen, D. W. and Q. Ji. 2010, In the eye of the beholder: A survey of models for eyes and gaze, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, n° 3, p. 478–500.

[132] Held, R. T., E. A. Cooper and M. S. Banks. 2012, Blur and disparity are complementary cues to depth, *Current Biology*, vol. 22.

[133] Held, R. T., E. A. Cooper, J. F. O'Brien and M. S. Banks. 2010, Using blur to affect perceived distance and size, *ACM Transactions on Graphics (TOG)*.

[134] Hoffman, D. M., A. R. Girshick, K. Akeley and M. S. Banks. 2008, Vergence-accommodation conflicts hinder visual performance and cause visual fatigue, *Journal of Vision*, vol. 8, n° 3, p. 33.

[135] Hu, X. and H. Hua. 2014, High-resolution optical see-through multi-focal-plane head-mounted display using freeform optics, *Optics Express*, vol. 22, n° 11.

[136] Hua, H. and B. Javidi. 2014, A 3D integral imaging optical see-through head-mounted display, *Optics Express*, vol. 22, n° 11.

[137] Huang, F.-C., K. Chen and G. Wetzstein. 2015, The light field stereoscope: Immersive computer graphics via factored near-eye light field displays with focus cues, *ACM Transactions on Graphics (TOG)*, vol. 34, n° 4, ISSN 0730-0301.

[138] Johnson, P. V., J. A. Parnell, J. Kim, C. D. Saunter, G. D. Love and M. S. Banks. 2016, Dynamic lens and monovision 3D displays to improve viewer comfort, *Optics Express*, vol. 24, n° 11.

[139] Kellnhofer, P., P. Didyk, K. Myszkowski, M. M. Hefeeda, H.-P. Seidel and W. Matusik. 2016, Gazestereo3d: seamless disparity manipulations, *ACM Transactions on Graphics (TOG)*, vol. 35, n° 4, p. 68.

[140] Konrad, R., E. A. Cooper and G. Wetzstein. 2016, Novel optical configurations for virtual reality: Evaluating user preference and performance with focus-tunable and monovision near-eye displays, *ACM Conference on Human Factors in Computing Systems (CHI)*, p. 1211–1220.

[141] Konrad, R., N. Padmanaban, K. Molner, E. A. Cooper and G. Wetzstein. 2017, Accommodation-invariant Computational Near-eye Displays, *ACM Transactions on Graphics (TOG)*, , n° 4.

[142] Kotulak, J. C. and C. M. Schor. 1986, A computational model of the error detector of human visual accommodation, *Biological Cybernetics*, vol. 54, p. 189–194.

[143] Koulieris, G.-A., B. Bui, M. S. Banks and G. Drettakis. 2017, Accommodation and comfort in head-mounted displays, *ACM Transactions on Graphics (TOG)*, vol. 36, n° 4, p. 11.

[144] Kramida, G. 2016, Resolving the vergence-accommodation conflict in head-mounted displays, *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, n° 7, p. 1912–1931.

[145] Lanman, D. and D. Luebke. 2013, Near-eye light field displays, *ACM Transactions on Graphics (TOG)*, vol. 32, n° 6, ISSN 0730-0301.

[146] Liang, J., B. Grimm, S. Goelz and J. F. Bille. 1994, Objective measurement of wave aberrations of the human eye with the use of a hartmann-shack wave-front sensor, *Journal of the Optical Society of America A*, vol. 11, n° 7, p. 1949–1957.

[147] Liu, S. and H. Hua. 2010, A systematic method for designing depth-fused multi-focal plane three-dimensional displays, *Optics express*, vol. 18, n° 11, p. 11 562–11 573.

[148] Liu, S., H. Hua and D. Cheng. 2010, A novel prototype for an optical see-through head-mounted display with addressable focus cues, *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, n° 3, p. 381–393.

[149] Llull, P., N. Bedard, W. Wu, I. Tošić, K. Berkner and N. Balram. 2015, Design and optimization of a near-eye multifocal display system for augmented reality, in *Imaging and Applied Optics*, Optical Society of America.

[150] Love, G. D., D. M. Hoffman, P. J. Hands, J. Gao, A. K. Kirby and M. S. Banks. 2009, High-speed switchable lens enables the development of a volumetric stereoscopic display, *Optics Express*, vol. 17, n° 18.

[151] MacKenzie, K. J., R. A. Dickson and S. J. Watt. 2012, Vergence and accommodation to multiple-image-plane stereoscopic displays: "real world" responses with practical image-plane separations?, *Journal of Electronic Imaging*, vol. 21, n° 1.

[152] MacKenzie, K. J., D. M. Hoffman and S. J. Watt. 2010, Accommodation to multiple-focal-plane displays: Implications for improving stereoscopic displays and for accommodation control, *Journal of Vision*, vol. 10, n° 8, p. 22–22.

[153] Mantiuk, R., K. J. Kim, A. G. Rempel and W. Heidrich. 2011, Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions, in *ACM Transactions on Graphics (TOG)*, vol. 30, ACM, p. 40.

[154] Matsuda, N., A. Fix and D. Lanman. 2017, Focal surface displays, *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, vol. 36, nº 4, p. 14.

[155] Metlapally, S., J. L. Tong, H. J. Tahir and C. M. Schor. 2014, The impact of higher-order aberrations on the strength of directional signals produced by accommodative microfluctuations, *Journal of vision*, vol. 14, nº 12, p. 25–25.

[156] Narain, R., R. A. Albert, A. Bulbul, G. J. Ward, M. S. Banks and J. F. O'Brien. 2015, Optimal presentation of imagery with focus cues on multi-plane displays, *ACM Transactions on Graphics (TOG)*, vol. 34, nº 4, ISSN 0730-0301.

[157] Owens, D. A. 1980, A comparison of accommodation responses and contrast sensitivity for sinusoidal gratings, *Vision Research*, vol. 29, p. 159–167.

[158] Padmanaban, N., R. Konrad, T. Stramer, E. A. Cooper and G. Wetzstein. 2017, Optimizing virtual reality for all users through gaze-contingent and adaptive focus displays, *Proceedings of the National Academy of Sciences*, p. 201617 251.

[159] Podlozhnyuk, V. 2007, Image convolution with cuda, *NVIDIA Corporation white paper, June*, vol. 2097, nº 3.

[160] Potmesil, M. and I. Chakravarty. 1981, A lens and aperture camera model for synthetic image generation, *Computer Graphics*, vol. 15, nº 3, p. 297–305, ISSN 0097-8930.

[161] Ravikumar, S., K. Akeley and M. S. Banks. 2011, Creating effective focus cues in multi-plane 3D displays, *Optics Express*, vol. 19, nº 21.

[162] Rolland, J. P., M. W. Krueger and A. Goon. 2000, Multifocal planes head-mounted displays, *Applied Optics*, vol. 39, p. 3209–3215.

[163] Shibata, T., J. Kim, D. M. Hoffman and M. S. Banks. 2011, The zone of comfort: Predicting visual discomfort with stereo displays, *Journal of Vision*, vol. 11, nº 8, p. 11.

[164] Song, W., Y. Wang, D. Cheng and Y. Liu. 2014, Light field head-mounted display with correct focus cue using micro structure array, *Chinese Optics Letters*, vol. 12, nº 6, p. 060 010.

[165] Stengel, M., S. Grogorick, M. Eisemann, E. Eisemann and M. Magnor. 2015, An affordable solution for binocular eye tracking and calibration in head-mounted displays, in *Proceedings ACM Multimedia*, p. 15–24.

[166] Urey, H., K. V. Chellappan, E. Erden and P. Surman. 2011, State of the art in stereoscopic and autostereoscopic displays, *Proceedings of the IEEE*, vol. 99, nº 4, p. 540–555.

[167] Vishwanath, D. and E. Blaser. 2010, Retinal blur and the perception of egocentric distance, *Journal of Vision*, vol. 10.

[168] Watson, B. A. and L. F. Hodges. 1995, Using texture maps to correct for optical distortion in head-mounted displays, in *Virtual Reality Annual International Symposium*, p. 172–178.

[169] Watt, S. J., K. J. MacKenzie and L. Ryan. 2005, Real-world stereoscopic performance in multiple-focal-plane displays: How far apart should the image planes be?, in *SPIE Stereoscopic Displays And Applications*, vol. 8288.

[170] Wu, W., P. Llull, I. Tošić, N. Bedard, K. Berkner and N. Balram. 2016, Content-adaptive focus configuration for near-eye multi-focal displays, in *IEEE Multimedia and Expo*.

[171] Zannoli, M., G. D. Love, R. Narain and M. S. Banks. 2016, Blur and the perception of depth at occlusions, *Journal of Vision*, vol. 16, n° 6, p. 17.

Chapter 5

CONCLUSION

This thesis has presented three papers where the model used for simulation or rendering has been designed specifically to enable the efficient use of iterative solvers. The first paper uses an iterative method to create a robust high-resolution point-based surface around a fluid simulation, in order to add realistic details to its surface. Since the point-based surface we create is robust and regular, we could investigate its application to other surface phenomena, similarly to the viscous wrinkling presented in Section 2.C. The addition of wrinkles to cloths and garments is another possible application, where coarse meshes could drive the surface contractions leading to high-resolution wrinkles. Such post-processing approaches for cloth have been investigated before for mesh representations [177, 176], but to our knowledge, a point-based cloth upres method would be a novel contribution. Our surface approach could also be extended to dynamic waves on cloths, as seen for instance on a waving flag, which might require the developement of different wave seeding and evolution strategies to adapt to the specific behavior of cloth. Furthermore, the wave model used for our fluid upres could be improved, for instance based on recent work of Canabal et al. [172], which more accurately simulates the dispersion behavior of surface waves.

The second paper designs and uses a vector field basis for fluid simulations. The basis is specifically designed to facilitate force projections, improving the efficiency of the method. Although the paper focuses on single-phase fluids, it would be interesting to investigate the application of our basis method to other scenarios. As mentioned in the paper, the extension to two-phase liquid simulations is a natural research direction, where the interactions of the surface with the fluid would need to be projected onto our basis. This might impede the efficiency of our method if it involves too many projections, but approximations could be obtained through the basis deformation method of Section 3.6.1. It would be interesting to then combine such a liquid simulation method with the upres technique of Chapter 2. Our basis construction method could also be used to design new bases for other simulation problems, such as stratified fluids [174] or magnetohydrodynamics simulations [175].

The last paper presents an optimal decomposition method modelled entirely in image space, allowing for the use of simple iterative solvers amenable to GPU implementations. As discussed in the paper, many other perceptual and optical phenomenons could be added to our minimization formulation. Recently, the work of Cholewiak et al [173] showed that taking into account the deformations specific to each color channel gives significantly better results in terms of accommodation, perception of depth, and realism. Including such effects into our multifocal decomposition could therefore improve the quality of our system, and these effects could be analysed further in the context of multifocal displays using our testbed.

References

[172] Canabal, J. A., D. Miraut, N. Thuerey, T. Kim, J. Portilla and M. A. Otaduy. 2016, Dispersion kernels for water wave simulation, *ACM Transactions on Graphics (TOG)*, vol. 35, n° 6, p. 202.

[173] Cholewiak, S. A., G. D. Love, P. P. Srinivasan, R. Ng and M. S. Banks. 2017, Chromablur: rendering chromatic eye aberration improves accommodation and realism., *ACM transactions on graphics (TOG)*, vol. 36, n° 6, p. 210.

[174] Desbrun, M., E. S. Gawlik, F. Gay-Balmaz and V. Zeitlin. 2013, Variational discretization for rotating stratified fluids, *Discrete and Continuous Dynamical Systems*, vol. 34, n° 2, p. 477–509.

[175] Gawlik, E. S., P. Mullen, D. Pavlov, J. E. Marsden and M. Desbrun. 2011, Geometric, variational discretization of continuum theories, *Physica D: Nonlinear Phenomena*, vol. 240, n° 21, p. 1724–1760.

[176] Gillette, R., C. Peters, N. Vining, E. Edwards and A. Sheffer. 2015, Real-time dynamic wrinkling of coarse animated cloth, in *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, p. 17–26.

[177] Rohmer, D., T. Popa, M.-P. Cani, S. Hahmann and A. Sheffer. 2010, Animation wrinkling: augmenting coarse cloth simulations with realistic-looking wrinkles, in *ACM Transactions on Graphics (TOG)*, vol. 29, ACM, p. 157.