Low-dimensional Representations for the Simulation and Control of Complex Dynamics

Adrien Treuille

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

Doctor of Philosophy

University of Washington

2008

Program Authorized to Offer Degree: Computer Science & Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Adrien Treuille

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Zoran Popović

Reading Committee:

_____

Zoran Popović

_____

Dieter Fox

_____

Steven M. Seitz

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Low-dimensional Representations for the
Simulation and Control of Complex Dynamics

Adrien Treuille

Chair of the Supervisory Committee:
Associate Professor Zoran Popović
Computer Science & Engineering

Most work in computer simulation has focused on off-line algorithms where each step can take arbitrarily long to compute. This thesis focuses instead on *interactive simulations* that run in real-time. Besides running quickly, such simulations must allow interaction, remain stable under unexpected user input, and remain "correct" (in some sense) for unbounded periods of time. Further, biomechanical simulation requires near-instantaneous planning without simulating the future. These requirements motivate the development of fundamentally new simulation algorithms.

This thesis approaches the interactive simulation problem by developing new, low-dimensional *representations* of the phenomena being simulated. These representations perform several functions at once. First, they can correlate many degrees of freedom of the underlying phenomenon, allowing us to represent the system with fewer variables. Second, the reduction is constructed so as to allow rapid simulation or control. Finally, the representations allow us to express correctness constraints.

Three examples of such simulations are presented, covering fluids, crowds, and human animation. The fluid model enables large, real-time, detailed flows with continuous user interaction, and can handle moving objects immersed in the flow. The crowd model is based on a fluid-like continuum representation, and naturally exhibits emergent phenomena that have been observed in real crowds. Finally, the human model can automatically

compute near-optimal human animations using a low-dimensional basis representation of the planning space.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables vi

Chapter 1: Introduction 1
1.1 Approach 3

Chapter 2: Related Work 5
2.1 Fluid Dynamics 5
2.2 Crowd Dynamics 8
2.3 Animal Flocking 11
2.4 Human Animation 12
2.5 Model Reduction 13
2.6 Reinforcement Learning 15

Chapter 3: Fluid Dynamics 17
3.1 Galerkin Projection 19
3.2 Galerkin Projection of Fluids 20
3.3 Simulation 21
3.4 Rendering 27
3.5 Boundary Conditions 27
3.6 Results 35
3.7 Implementation Details 40
3.8 Discussion and Future Work 43

Chapter 4: Crowd Dynamics 45
4.1 The Governing Equations 47
4.2 Implementation 54
4.3 Results 59
4.4 Parameter Settings 62

4.5 Discussion and Future Work 64

Chapter 5: Human Animation 68
5.1 Motion Model 69
5.2 Control 71
5.3 Policies 74
5.4 Results 79
5.5 Conclusion and Future Work 83

Chapter 6: Conclusion 86
6.1 Future Horizons 87

Bibliography 89

LIST OF FIGURES

Figure Number — Page

1.1 Overview of simulation techniques presented in this thesis: (a) fluids (chapter 3), (b) crowds (chapter 4), and (c) human motion (chapter 5). 1

3.1 An interactive fluid simulation of a car driving through 4000 leaves runs at 24 frames/second on a $256\times64\times128$ grid. (Car model courtesy of Digimation.) 17

3.2 Bases 1-35 for a 512×512 fluid simulation with a fix yin boundary. (Grids have been subsampled for display.) 22

3.3 Bases 36-64 for a 512×512 fluid simulation with a fix yin boundary. (Grids have been subsampled for display.) 23

3.4 Energy dissipation in a 128×128 simulation. My simulator exactly preserves kinetic energy in the absence of diffusion, $\nu = 0.0$. A diffusion coefficient of $\nu = 0.2$ (blue) most accurately matches the numerical viscosity inherent in my full-dimensional simulator (green). 26

3.5 (1)-(23): The 23 boundary basis grids for a c-shaped object. These form a basis for the expected velocities at the surface, and are divergence-free everywhere else. $(\dot{x})$, $(\dot{y})$, and $(\dot{\theta})$: The additional translational and rotational bases (section 3.5.5). 28

3.6 The resampling operator $R_{\Phi,\mathbf{t}}$ changes a velocity field's coordinate system, while the surface speed operator S_i measures surface normal velocity. 30

3.7 Per-timestep pseudocode. The global field is simulated first (1)-(3), then boundaries are handled (4)-(6). 31

3.8 2D Fluid Simulations. (a) *2D Boundaries:* User-manipulated boundaries in a 128×128 domain. (b) *Forces:* User-added forces on a 512×512 domain running at 41 frames/second. See Tables 3.1, 3.2, 3.3 for more details. 35

3.9 Real-time fluid-object interaction in a $64\times64\times64$ cube. 38

3.10 Simulation error for the model reduction of fluids. Error is measured against a full 128×128 semi-Lagrangian simulation of my *2D Boundaries* example. Green lines represent results of our reduced model at varying numbers of bases. Blue lines show errors under full simulation at varying resolutions. . 39

3.11 Further error metrics for the model reduction of fluids. (a) Observed force directions (0° and 90°) have lower error than unobserved force directions ($0° < \theta < 90°$). (b) Using fewer precomputed feedforward matrices (spacing the samples further apart) yields lower boundary fidelity. 40

3.12 A MAC-style grid. Velocities are defined as fluxes at the cell faces. 41

4.1 A variety of city scenarios simulated with our model. 45

4.2 (a) Two groups pass, naturally forming lanes. (b) A vortex forms as four groups cross. 47

4.3 General algorithm overview. 54

4.4 Discretized grid structure. 55

4.5 People in yellow shirts evacuate a building. 59

4.6 A continuum crowd reacts to a user-driven flying saucer agent in real-time. 61

4.7 A continuum crowds group (blue) interacts with a group of Reynolds agents (red). 61

4.8 A large army chases a smaller one through mountainous terrain in real-time. 62

5.1 A clip. The first subsequence C_{in} runs from (a) to (c), while the second C_{out} comprises (c) through (e). The constraint frames are (b) and (d). 70

5.2 Clip blending. Blending the sequence $A, B, C, D, E \subseteq \mathcal{C}$. Constraint frames (shown as solid vertical bars) are aligned across successive clips. Blending occurs where two clips overlap. 71

5.3 State variables for a digital character. Root orientation θ is measured relative to the desired path (along the positive x-axis). The torso orientation τ and actual motion direction ρ are measured relative to the root. 72

5.4 Planning vs. greedy controllers. This graph compares the cost $c_s + c_t$ of a greedy navigational controller (equation (5.6)) with my planning controller. The characters act almost identically until the user reverses the intended motion direction (A). The planning controller then chooses a high cost step (B) to enable a sharp and realistic turn one step later. 75

5.5 Moving obstacle avoidance. Moving obstacle avoidance models short-horizon crowd dynamics: each character who is viewed as an obstacle by another is marked by a cylinder. Thus, characters automatically avoid one another. . . 80

5.6 Basis functions for fixed obstacle avoidance. (1)-(15) are the polynomial bases for fixed obstacle avoidance (FOA). (16)-(18) are three of the four Gaussian bases. All bases are shown as slices through the u-v plane, holding θ, C, and $x + u$ constant. My method solves for the linear superposition of basis functions shown on the right. 82

5.7 Value function convergence. This graph varies the degree d of the root orientation polynomial $\mathcal{P}_d(\theta)$ for a simple navigational controller. Errors are measured relative to the 10th-degree approximation as the sum of squared differences over a set of samples along the θ axis. 82

LIST OF TABLES

Table Number Page

3.1 Summary of fluid results. The "Type" column indicates the reduced simulation's processor architecture. 36

3.2 Timing summary. All times are measured in milliseconds per frame. "GPU" and "CPU" are the milliseconds per timestep of my full-dimensional simulations on the graphics and central processing units, respectively. "Vels," "Bound," and "Media" are the time taken by the velocity, boundary, and immersed media (density or particle) steps in the reduced simulator. "Total" is the full cost of a timestep in the reduced simulator. The final two columns measure the speedup ratio of model reduction compared to a standard (full) simulation. 36

3.3 Precomputation time. The "Velocities" column shows the time to compress the simulation basis and project the physics, while "Boundaries" shows the time to compress the boundary basis and compute reduced coupling forces for each object. All times are in hours. Note that these are *total* times which I reduced in practice by parallelizing over a 16 processor cluster. 37

4.1 Parameter settings for the crowd results. The left-most column lists simulation names, while the rest of the columns give parameter values. "*Lanes*" and "*Vortex*" are the linear and vortical lane formation examples. "*Armies*" is the result where one army chases another. "*Quad*" is the school quad animation, and "*City*" is the city animation. Space is measured in units of grid cell length, while time is measured in timesteps. Section 4.4 gives advice on choosing parameter values. 63

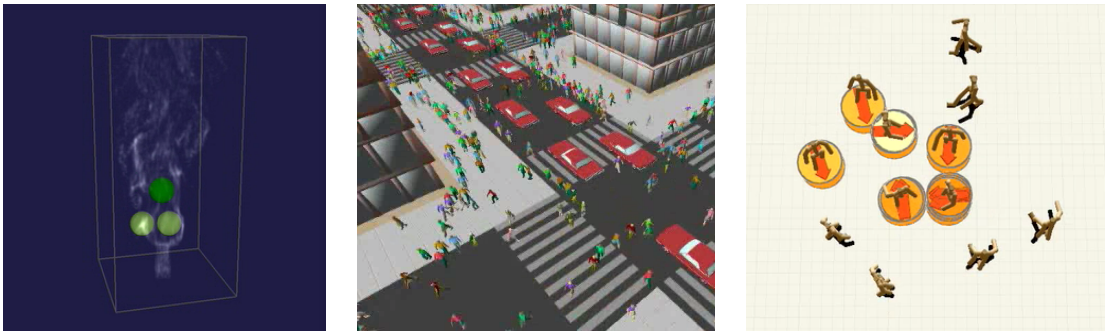
5.1 Motion tasks. Columns 2-4: Active state variables. Columns 5-10: Nonzero cost terms are indicated by bullets. 79

ACKNOWLEDGMENTS

I would like to thank all my friends and the University of Washington computer science department for providing such a simulating and supportive environment here in Seattle. I am especially grateful to my advisor, Zoran Popović for his energy, his instruction, and most of all, for believing in me. Finally, I'd like to acknowledge Grandpère for his scientific inspiration, and my family whom I love so much: Beverly, Antoine, Geneviève, Philippe, and Isabelle. Or should I say: Mamoushka, Pops, Jon, Fildok, and Blinks.

Chapter 1

INTRODUCTION



(a) (b) (c)

Figure 1.1: Overview of simulation techniques presented in this thesis: (a) fluids (chapter 3), (b) crowds (chapter 4), and (c) human motion (chapter 5).

Physics-based animation is one of the most beautiful areas of computer graphics. It pervades our society through the special effects and computer game industries, and it plays a central role in virtual reality with implications for training and medicine. Physics-based animation also has enormous potential for education, immersing students in exotic virtual worlds or allowing them to "edit" the laws of physics. The field presents scientific challenges as well: physics-based animation is "the physics of everyday things," and as such, forces us to scale our abstract physical knowledge to the true complexities of the world around us, from wet hair emerging from a pool, to beach sand squishing between our toes, to animal locomotion and crowd dynamics. *Interactive* animation also adds rigorous new requirements to simulation algorithms which are not usually included in traditional numerical analysis. Finally, on a personal and aesthetic level, the delight of this field is that it asks us to rebuild the world around us anew using computers. This almost insurmountable task cannot but open our eyes to the richness, complexity, and beauty of our

world.

Physics-based animation research in the past has mostly used *off-line* simulation where each simulation step can take arbitrarily long to compute. In contrast, this thesis looks at a much less explored domain: *interactive simulation*. I will focus specifically on systems with many degrees of freedom and highly nonlinear dynamics. Such simulations require much more than applying existing techniques more quickly. They requires fundamentally new algorithms and correctness metrics. In particular, interactive simulation presents the following challenges:

1. **Speed.** Simulation steps must occur in fraction of a second. To reach such rates, methods must be developed with asymptotically better time complexity than traditional algorithms. For example, this thesis will present a fluid simulator that correlates millions of variables, freeing us from updating every variable at each timestep. Researchers have also looked at parallel computing architectures such as Graphics Processing Units (GPUs), although these hardware-based methods will not be the focus of this thesis.

2. **Correctness.** Traditionally, the accuracy of simulations has been measured for asymptotically small step sizes. Graphics applications, by contrast, may use large timesteps for which such metrics do not apply. Further, interactive simulations do not have preordained length, implying that error should be measured over the long term, rather than for short timesteps. For example, it would be useful to have controlled long-term energy dissipation or provable volume preservation, as needed. [1]

3. **Interaction.** Real-time simulation must handle continuous user input. Unlike off-line applications, one cannot roll back the system if the simulation becomes unstable or otherwise incorrect because typically the user is still interacting with it. For this reason, video game developers often employ play-testers who try to test every possible game interaction for correctness. Clearly, this approach cannot scale to increasingly

[1] Many traditional water simulation techniques cause all the water to disappear over time–creating accidental numerical "evaporation."

complex worlds with many interlocking systems. Better interaction formalisms will be required.

4. **Control.** Biomechanical systems including humans and crowds have *control* mechanisms that determine free variables such as muscle impulses. Real-time simulation of biomechanics requires that control be determined instantaneously without the benefit of simulating the future. In this case, the realtime requirement mirrors that of biology, giving us an extra impetus to study such systems.

1.1 Approach

As described above, interactive simulation provides significant challenges beyond traditional off-line simulation. The general approach of my thesis is to present *new, compact representations* of these systems. These representations–often called model reductions–perform several functions. First, they can correlate many degrees of freedom of the underlying phenomenon, allowing us to represent the system with fewer variables. Second, the reduction is constructed so as to allow rapid simulation or control. Finally, the representations allow us to express correctness constraints.

This thesis presents three techniques that embody this approach. The first demonstrates huge speedups in fluid simulation (Figure 1.1(a)) with little cost in terms of accuracy by reducing the dimensions of a simulation from millions of variables to fewer than 100. I show that moving objects can be integrated into the flow by endowing them with their own reduced models. Moreover, the coupling forces between the individual models can be computed in the reduced space as well. This treatment of moving objects overcomes one of the principle hurdles to adapting model reduction to interactive fluid simulation. In addition, my model can precisely control energy dissipation throughout the simulation.

The next technique focuses on real-time crowd motion synthesis for thousands of individuals with intersecting paths (Figure 1.1(b)). I adopt a new continuum perspective where individuals are treated like particles in a fluid. This approach effectively amortizes control calculations over an entire crowd, yielding a continuum formulation radically different from the agent models historically used in the graphics and crowd literatures. This model

unifies global planning and collision avoidance. Individuals in my simulations do not face conflicting requirements between local collision avoidance and global planning, and they exhibit smoother motion than has been previously reported. This brings my model closer to real crowds, which are characterized by surprisingly few sudden changes in direction.

The final technique (Figure 1.1(c)) looks at the interactive control of a single digital character. Capturing the fine nuances of human motion requires a high-dimensional motion repertoire and a multivariate space of input parameters which can change continuously, often unpredictably. For example, a human might fluidly move to avoid unexpected moving obstacles while maintaining a direction and walking next to another person. Naive optimal control algorithms can be intractable even for 6 or 7 dimensions. By this standard, a human model with 80 variables is insurmountably complex. I address these issues through two different forms of model reduction. First, I reduce the motion space to a graph of prerecorded motions, greatly diminishing the dimensionality of the control space. Second and more importantly, I focus on low-dimensional linear representations of the control space.

The remainder of this thesis is structured as follows. I will first describe related work in interactive simulation and control (chapter 2). Then I will present each simulation system in more technical detail (chapters 3-5), explaining how I used new representations to simulate complex, high-dimensional systems. After describing these three projects, this thesis will conclude with some thoughts about future directions (chapter 6).

Chapter 2
RELATED WORK

This chapter presents previous work related to the three phenomena studied in this thesis: fluid dynamics (section 2.1), crowds (sections 2.2-2.3), and human motion (section 2.4). The final two sections present an overview of general work on model reduction (section 2.5) and reinforcement learning (section 2.6) as these ideas apply to the thesis presented here.

2.1 Fluid Dynamics

The Navier-Stokes equations are the widely accepted mathematical model of incompressible fluid flow. The numerical simulation of these equations has been the subject of decades of research in physics and applied mathematics. This section will focus primarily on the study of fluids for computer graphics applications. Readers interested in general computational fluid dynamics (CFD) are encouraged to read the good overviews by Anderson [2] and Fletcher [38]; the former serves as a better introduction, while the latter provides more breadth.

Computer graphics researchers have made extensive use of computational fluid dynamics techniques to create digital water, fire, smoke, and other fluids. The first to do this were Kajiya and Von Herzen [65] who showed that physical simulation could produce more visually faithful flows than non-physical methods. Since then, increasingly sophisticated CFD techniques have been used in graphics from MAC discretizations (introduced by Foster and Metaxas [42] based on the seminal paper by Harlow and Welch [51]), to semi-Lagrangian advection [124], to vorticity confinement [35], and many others. Occasionally, although less frequently, graphics techniques have transferred back to computational fluid dynamics. For example, the work of Enright *et al.* on capturing water surfaces for graphics [33] lead to the most accurate existent level-set algorithms according to widely accepted

benchmarks [34]. In general, computational fluid dynamics and computer graphics use different evaluation criteria leading to different simulation algorithms. Research in the former generally focuses on physical accuracy, especially for asymptotically small timesteps and find grid discretizations. Computer graphics, by contrast, is interested more in the visual fidelity, speed, and controllability. This gives graphics practitioners considerably more leeway to work with coarser grids and longer timesteps. Indeed much work in computer graphics concerns methods that admit long timesteps while maintaining a certain baseline of physical correctness, such as stability [124]. In chapter 3, I will argue that the goal of fluid simulation in graphics should not be asymptotic accuracy (as it often is in CFD), but rather long-term statistical correctness. For example, a glass of digital water when stirred should show the correct energy dissipation profile, neither diffusing too quickly, nor remaining in an excited state for too long. Similarly the volume of water should remain constant. Moreover these quantities should be measured as $t \to \infty$, not as $t \to 0$. These long-term properties better match the requirements of realtime graphics for computationally inexpensive simulations that can run indefinitely without visual degradation.

Subsequent to the work Kajiya and Von Herzen described above, Kass and Miller approximated the two-dimensional (2D) shallow water equations to simulate water surfaces [67], while Chen and Lobo used a different 2D approximation which allowed fluid-object interaction [16]. The next major advances were the works of Foster and Metaxas on modeling water and gases [43]. They showed that impressive animations could be created even on relatively coarse grids, and were the first in graphics to use MAC grid discretizations which form the basis for our discretization (section 3.7). Fluid models related to that of Foster and Metaxas were proposed in [17] and [141]. These methods all require a strict bound on the time step for their simulations to remain stable, resulting in larger computation times. Shortly thereafter, Stam [124] introduced the Stable Fluids algorithm to address these limitations, using a semi-Lagrangian treatment of advection combined with an implicit solver for viscosity. Fluid solvers have been coupled with level-set methods to create impressive simulations of water [41, 33] and fire [99]. For smoke, to combat the numerical dissipation inherent in the stable solution of the equations, Fedkiw *et al.* [35] proposed to reinject the lost energy in the small vortices of the flow using a confinement force. For ex-

plosions, Yngve *et al.* modeled compressible flow [143], which had not been necessary for standard smoke simulation.

The fluid simulator of Stam [124] is considered the standard in graphics, and forms the basis for my own full-dimensional fluid simulations in chapter 3. Much work has aimed at speeding up Stam's algorithm through hierarchical space decomposition [86] or computation on non-uniform meshes [36, 32]. But these efforts have not achieved real-time simulation of high-resolution fluids. Solving the fluid equations on the hierarchical spatial subdivision [86] has yielded impressively high-resolution results, but the high per-step computation cost prevents this method from being used in real-time contexts. Under very specific conditions with strictly periodic boundaries, Fourier methods [125] can produce very fast performance. While hierarchical, non-uniform, and Fourier methods can provide substantial speedups, they still require computation proportional to the size of the domain.

Pure vortex methods [102] are not restricted to a particular domain, but the effective resolution coarsens as the particles are spaced farther apart. Vortex methods can also present difficulties handling boundaries and diffusion. Hybrid grid and vortex methods have also been proposed [117] although the intent is to produce more faithful dynamics, not faster simulation. Both grid and vortex-based methods are at best linear in the velocity sampling.

Various fluid simulation methods have been mapped to the GPU as a stream processor including a Lattice Boltzman Method [84], Coupled Map Lattice [52], semi-Lagrangian solver with boundaries [142], and a Poisson solver [73, 13, 48]. These algorithms have the same time complexity as their CPU versions and do not offer resolution-independent simulation. They can, however, achieve large constant factor speedup over CPU algorithms, often producing real-time performance for medium-size problems.

Recently, fluid work in graphics has focused on small-scale structure such as bubbles [20] and water droplets [139], and exotic phenomena such as the interaction between heterogeneous types of fluids [62, 87]. Of particular interest is a recent trend towards model reduction techniques in fluids. Besides the algorithm presented in chapter 3, Yuksel *et al.* presented a highly interesting Lagrangian model reduction of the 2D wave equations [144]. Looking forward, one of the principle challenges in fluid dynamics for graphics will be to extend these fast interactive techniques to the increasingly sophisticated phenomena cur-

rently being studied for off-line simulation.

2.2 Crowd Dynamics

Crowd dynamics has been much less extensively studied than fluid dynamics. The most natural way to model locomotion of large human crowds is with agent-based methods in which each person plans individually. This is primarily because people have distinct characteristics and make decisions based on personal goals. Such dynamics have reached perhaps their most sophisticated form in the work of Funge *et al.* [45], which models not only behavioral dynamics such as stimulus response but also cognitive aspects such as knowledge and learning. This model was further expanded to include pedestrian visibility and path planning in [118]. Massive Software, founded by Stephen Regelous, publishes a production-quality crowd simulator [91] for cinematic effects that gives the animator the freedom to author details describing each individual agent within the crowd. Though each of the above mentioned systems could be used to create ultra-realistic crowds, considerable expertise is required to design each individual. It has recently been observed that no existing methods can cover a comprehensive range of scenarios with accuracy suitable for safety engineering purposes [126]. More importantly, in order to accurately model human reasoning and congestion planning, the computation cost for each individual is large. For this reason the most accurate methods are impractical for real-time simulations of large crowds.

To improve on the performance characteristics of agent-based methods, and to avoid the complexities of constructing a cognitive model for each agent, researchers have explored a number of simplifications to crowd models, including local methods, precomputed static path plans, and global path planning on a coarse environment graph. Local agent methods in graphics can be traced back to the seminal work of Reynolds [109], who demonstrated that emergent flocking behavior can be generated from simple local rules. Since his work, a considerable corpus of research has emerged focused on modeling human crowds with locally controlled agents, and Reynolds [108] has expanded his set of possible behaviors. Other work takes into account sociological issues [97], psychological

effects [105], geographically-based direction [127], social forces [56, 23], and nontrivial motion dynamics [14], to name just a few.

Unlike real people, locally controlled agents do not perform any global or mid-range planning to avoid non-local congestion or cross-flowing pedestrians. Collisions are prevented locally by changing direction when other people get too close. Many interesting collision avoidance methods have been proposed including geometric models [37], grid-based rules [88], density-dependent techniques [98], behavioral models [137], particle force interaction models [53], and Bayesian decision processes [94].

Ultimately, local collision avoidance can only go so far in capturing the inherently long-term planning process that humans employ to smoothly avoid dynamic congestion. Because people continuously plan ahead, real pedestrians exhibit virtually no near-misses or "visual bumps" as they navigate through congestion. Further, collision avoidance alone cannot model crowds where people have an overall goal or objective. Consequently, many models have combined collision avoidance with global navigation. In general, global planning has taken the form of graph-based techniques or static potential fields. Lamarche and Donikian [75] used sophisticated topological precomputations to enable real-time per-agent global path planning which takes into account visibility. They employed a separate collision avoidance process, so their agents were susceptible to "getting stuck" due to local minima under congestion. Bayazit *et al.* [10] created impressive results using coarse graph-based roadmaps for global planning together with collision avoidance. Probabilistic maps and decoupled planners were also combined with motion graphs and constraints to compute detailed locomotion of pedestrians [128]. Pettré [106] uses navigation graphs of static scenes to find pedestrian paths in real-time, but does not account for collisions or congestion. Kamphuis and Overmars [66] extended path planning so that agents could stay together. Li *et al.* [83] assumed a leader/follower model; they reduced the expense of global planning by only planning for the leader at the cost of a less general system. Goldenstein *et al.* [47] introduced a sophisticated three-layer technique to handle local and global navigation, using a static potential field for maneuvering around fixed obstacles. Precomputed, static potential functions have also been used to study evacuation [70]. In general, static potential fields do not handle changing environments, such as changing traffic lights

and exit congestion.

Agent-based crowd modeling has also been studied in transportation research for urban planning, path design, and evacuation [54]. Helbing's social force model has been influential in this field, and has proven capable of reproducing specific crowd phenomena [55]. A number of high-fidelity crowd models are commercially available. For example, the EGRESS system [68] employs a cell-based approach to simulating macro behavior of crowds; Crowd Dynamics [126] uses a calibrated agent-based framework to accurately model a number of observed crowd phenomena.

Multiple-agent path planning has also been studied extensively in robotics, primarily for cooperative tasks of multiple robots. "Centralized" planners choreograph the motion of all agents through spacetime [82]. These methods can be exponential in the number of robots, and are not appropriate for crowds where agents optimize for personal goals. Closer to my work are "decoupled" planners where agents plan individually. Such algorithms require priority schemes to resolve conflicting plans [11]. Parker [103] explores the trade-offs between local and global planning. Potential fields for navigation have also been explored in robotics, as by Arkin [6].

Finally, an alternative approach to crowd synthesis has emerged from the fluid dynamics community. Hughes [60] has developed a model which represents pedestrians as a continuous density field and presents a pair of elegant partial differential equations describing the crowd dynamics. Most importantly, this system is driven by an evolving potential function, defined so as to guide the density field optimally toward its goal. A full derivation can be found in [59], and in a subsequent application to a medieval battle analysis [21]. The 1D version of this model was also confirmed with real crowd data [58]. Alternative continuum models of 1D crowd flow have also been analyzed [22]. In the graphics community, Chenney's flow tiles [18] demonstrated that desirable crowd properties such as congestion avoidance can be achieved with divergence-free flows, although they do not address many of the other important aspects of crowds.

The model which I present in chapter 4 was directly inspired by Hughes's work. I use a similar potential function to guide pedestrians toward their goal. I also adopt his technique of combining pedestrians into groups, and the use of "discomfort fields" to handle

geographical preferences. However, Hughes only investigates analytic properties of his equations and does not discuss simulation. I address simulation by transforming Hughes's continuous crowd field into a particle representation which is better suited to rendering and which forms the basis for my stable, efficient simulator. The particle representation is fully integrated throughout the dynamics, from the theoretical underpinnings to my grid discretization scheme, which is tailored for the difficult case of fine grid resolutions relative to the size of the people. I have also made numerous modifications to Hughes's model itself, especially to take into account crowd flow when computing individual speed. This allows my simulator to exhibit a number of visually interesting and empirically-confirmed phenomena such as *lane formation* where people cross in contrary directions. Also, I have changed the definition of path optimality to include a distance term, which inhibits the oscillations found in Hughes's dynamics.

2.3 Animal Flocking

Related to the work in crowd simulation, there has also been considerable work in the biology and applied math literature on finding mathematical models of animal flocking, using both agent-based and continuum dynamics. Agent-based simulations go at least back to Aoki's 1982 work on fish schooling [3]. Czirók and Vicsek proposed simple rules for individuals to align their velocity with that of local neighbors [28]. The model's simplicity enabled detailed analysis using techniques from non-equilibrium statistical physics. Buhl *et al.* later empirically confirmed aspects of this model for locusts [15]. More complex models for fish schooling and bird flocking that account for group attraction and repulsion have been proposed by Couzin *et al.* [26], and Parrish *et al.* [104]. There has also been extensive agent-based work on self-organization in ants, with particular attention to the effects of pheromone trails [30, 44]. Couzin and Franks used pheromone trails to explain lane-formation in ants [25]. The model proposed in this thesis predicts lane-formation in human crowds through path planning rather than trail following. A comprehensive overview of work on agent-based flocking can be found in [27].

Several researchers have worked on the intersection of agent-based and continuum

models of flocking. Flierl *et al.* combined agent-based fish schooling with an Eulerian fluid simulation, and studied the bridge between individual and continuum models of flocking [39]. Levine *et al.* studied self-organization from the agent-based perspective, then "coarse-grained" the models into a set of partial differential equations (PDEs), showing good agreement between these dual approaches [81]. Pure continuum models to describe animation motion go at least back to Okubo's 1980 book on diffusion in biology [100]. Gueron and Levin showed that wildebeest migration wave patterns could be well modeled by one-dimensional wave equations [49]. Further PDE models of flocking have been proposed by Edelstein-Keshet *et al.* [31], and Toner and Tu [129]. Mogilner and Edelstein-Keshet highlighted the importance of non-local interactions in capturing flocking behavior [95]. Topaz and Bertozzi presented a general 2D continuum approach that applies to swarming in many different animals, and presented an analytic investigation of the range of phenomena their model can generate [130]. The article by Topaz and Bertozzi [130] and that by Flierl *et al.* [39] present good overviews of work in PDE approaches to animal motion.

Unlike the model proposed in this thesis, the work on animal flocking does not take into account global planning other than through static potential fields. Because of the higher cognitive capabilities of humans, I believe that realistic crowd behavior is better modeled with global path planning, as proposed in chapter 4.

2.4 Human Animation

There have been several approaches to synthesizing realistic full-body human motion. Variational methods [140, 85] solve for a digital character's degrees of freedom for an entire motion sequence at once. These approaches are most useful for off-line applications (such as special effects) where the character's initial pose and goals are all known *a priori*. For real-time applications, researchers have also looked into blending between prerecorded clips of motion [107, 72]. These techniques require very little computation per timestep, but do not generalize readily beyond the input motion capture data. In many video games, human motion is based on pre-computed motion cycles. This approach gives the game designer more control over the motion, but traditionally produces less realistic animation

than pure motion capture. Most recently, Sok *et al.* have opened up a new path in human animation by showing that a full dynamical simulation of a human could be controlled to follow motion capture data [123].

The technique I present in chapter 5 is based on the motion capture blending approach. Early work in this field includes that of Pullen and Bregler [107] which segmented motion data into small pieces and rearranged them to meet user-specified keyframes, and that of Kovar *et al.* [72] which generated a graph structure from motion data allowing characters to follow sketched paths using a branch and bound algorithm. Arikan and Forsyth [4] created a hierarchy of graphs and employed a randomized search algorithm for synthesizing a new motion subject to temporal and position constraints. These algorithms employ complex graph search and optimization techniques to produce animations and are not well suited for interactive control. Lee *et al.* [78] used a local on-line search algorithm which is demonstrated to generate 5 to 8 frames of motion per second when the search horizon is limited. The graph search can be made more efficient by using a simple cyclic graph structure [46], by composing the world from smaller environment-specific graphs [80], by precomputing search trees [76], and by grouping similar motion segments into parametrized graph nodes and building sparse hierarchical motion graphs [74]. Choi *et al.* [19] generated environment-specific graph representations that enable biped locomotion through a complex environment with obstacles. Arikan *et al.* [5] showed that motion synthesis from a graph structure can be cast as dynamic programming. Researchers have also generalized transition graphs to form richer, more complete motion spaces [119, 114].

2.5 Model Reduction

In the applied mathematics literature, as well as in computational fluid dynamics (CFD), dimensional model reduction is well studied [89]. These methods use Galerkin projection to yield systems of differential equations involving fewer equations and fewer unknown variables. The reduced equations can be solved much more quickly than the original problem, at some accuracy cost. These methods also appear in literature under the names of proper orthogonal decomposition (POD), Karhunen-Loéve decomposition and subspace

integration. In CFD, these methods are primarily used to study organized spatial characteristics in a flow, i.e. coherent structures [57], and have been used to design flow control systems, since they provide a more controllable description of the system dynamics [112, 8]. Sirovich [122] introduced snapshot POD, which essentially constructs a reduced model from a sequence of instantaneous spatial fields acquired at different times. To increase accuracy, these models can be subsequently calibrated to the observed data [24]. After a long-time integration of dissipative flows, POD-based simulation methods exhibit drifting inaccuracies that have been handled with non-linear Galerkin projection [90] and viscosity method correction [121]. Extended POD methods [92] were developed to study correlated events in turbulent flow that are not directly related to energy.

Treatment of moving boundary conditions is not as extensively studied for model reduction. A straightforward way to treat moving boundaries is to construct a separate set of bases for each possible boundary configuration [115]. This approach leads to an explosion of the required bases unless the boundary movement is restricted to a small subspace. For flows with periodic boundaries, and with inherent symmetries within the flow dynamics, it is possible remove uniform translation modes [110, 111]. In a restricted case when analyzing a single boundary in free flow, one can form the reduced basis in the frame of reference of the boundary as it is moved through various angles of flow attack [8]. Similarly, when a single moving boundary moves along a single dimension such as pistons within the engine cylinder, stretching and aligning of the flow basis can allow for a reduced model [40]. Prior to my work with Lewis and Popović [133] (presented in chapter 3), model reduction had not been applied to fluid simulation in graphics, and no work in any field treated multiple moving boundaries, or boundaries that move in unpredictable ways.

For phenomena other than fluids, model reduction is an increasingly important technique in computer graphics. It has been used to reduce a wide range of problems, ranging from global illumination to elastostatics and dynamics. James and Fatahalian [63] precompute nonlinear deformation responses to a finite set of user impulses and apply dimension reduction using PCA. Their approach restricts the range of possible runtime interactions to a small discrete set of pre-selected impulses. Barbič and James [9] present a more gen-

eral approach for finite element models that allows general runtime forcing within the reduced-dimensional subspace. Multilinear model reduction [138] has also been used as a compression tool for non-interactive time-sequences.

2.6 Reinforcement Learning

My controller uses *reinforcement learning* methods which are well studied in the control community. Kaelbling *et al.* [64] provide a good overview of the discrete reinforcement learning problem. Bertsekas [12] presents in-depth coverage of the continuous case, as well as of the delayed rewards model which I use. LaValle [77] presents a good overview with many applications to robotics and motion planning. Atkeson *et al.* [7] frame reinforcement learning in the broader context of function approximation. However, these techniques suffer acutely from dimensionality, making control difficult or impossible in high-dimensional spaces. Moore and Atkeson [96] address this problem specifically with a multi-resolution reinforcement learning algorithm; however this method relies heavily on storing samples, which can become intractable for the large-scale problems I want to solve. Researchers have explored linear programming to efficiently solve reinforcement learning problems [120, 134]. In this vein, Pucci de Farias presented a linear programming approach that uses basis functions to compactly approximate the value function without storing samples [29]. This algorithm forms the basis for my approach to character animation because it greatly mitigates the cost of dimensionality by avoiding runtime sampling. In chapter 5, I also explain how to reduce the dimensionality burden by learning optimal controllers only within certain subspaces, while parallelizing precomputation along other dimensions.

Reinforcement learning is increasingly appearing in the graphics literature, for example to create video textures [116] or for local character navigation [61]. The closest work to my own is that of Lee and Lee [79] which used value iteration to construct a sample-based value function for boxing. Relative to this method, my precomputations are 7$\times$ faster on comparably sized problems, but require more memory. At runtime, however, my basis approximation requires 30-90$\times$ *less* memory than Lee and Lee's sample representation. To be similarly compact, sample-based methods such as Lee and Lee's would require very coarse

discretizations, incurring significant risk of missing minima and other important features of the value function. McCann and Pollard [93] have integrated a model of user behavior into reinforcement learning, enabling highly responsive realtime character animation.

Chapter 3

FLUID DYNAMICS

Figure 3.1: An interactive fluid simulation of a car driving through 4000 leaves runs at 24 frames/second on a $256 \times 64 \times 128$ grid. (Car model courtesy of Digimation.)

Computer graphics researchers have developed strikingly realistic models of fluid phenomena such as curling smoke and splashing water. As described in chapter 1, a much less explored domain is real-time interactive fluid simulation, which opens new avenues for training, computer games, and interactive media. Simulation steps must occur in fraction of a second, handle fluid-object interaction, and produce visually convincing fluid flows

[1]The work described in this chapter was originally presented as a paper [133] at SIGGRAPH 2006.

that capture important coherent features such as turbulence. To achieve such rates, simulation time should ideally be proportional to the rendering complexity, not the size of the computational domain. Further, since interactive simulations do not have preordained length, it would be important to have unconditional stability and controlled energy dissipation.

This chapter presents fluid simulator that was the first to satisfy all the constraints above. The method begins with an accurate off-line solver [124] that produces a set of high-resolution fluid simulations representative of the expected user input. These velocity fields are distilled into a small basis of size proportional to the system's principle modes of variation. From there, one can project the fluid equations onto this low-dimensional basis. After these precomputations, my method simulates the velocity field very quickly through the subspace, achieving simulation runtime costs polynomial in the number of basis states, but *independent* of the number of simulation voxels. This algorithm also can preserve the energy in the basis, allowing for controlled energy dissipation. Finally, multiple moving objects may be immersed in the flow at runtime, and fluid-object interaction can be handled entirely in the reduced space.

This approach has unique properties not present in full-dimensional simulations. Complex phenomena can be observed on large grids (e.g., the wake of a speeding car) and the number of variables is essentially proportional to the underlying degrees of freedom (e.g., the position and direction of the cars). As such, each model represents a problem-specific subspace of all simulations and is inherently less accurate or general than full dynamics. However, my technique has very different time complexity as well: the reduced algorithm would remain just as fast even if one doubled the spatial resolution. In fact, the method is not bound by the size of the computational domain, but by the computer memory and cost of rendering.

Depending on the rendering choice, I couple the reduced simulator with either immersed particles (textured smoke sprites, dust, leaves), or a grid-based density simulation. Immersed particles, which are used extensively in games in order to reduce rendering costs, are especially well suited to model reduction, as they need only sparsely query the velocity field. Density simulation and rendering both require a pass through the entire

grid, and therefore dominate any fluid computations in the reduced space. Importantly, for both particles and densities, my algorithm is *output-sensitive:* the cost of simulation is proportional to the cost of rendering.

3.1 Galerkin Projection

Model reduction consists of simulating a system with a compact, low-dimensional representation. My approach to fluid simulation uses a particular form of model reduction known as *Galerkin projection*. This section presents a general overview of linear dimension reduction and its connection to Galerkin projection.

Dimension reduction means representing a vector in high dimensional space $\mathbf{u} \in \mathbf{R}^n$ with a corresponding vector in a much lower dimensional space $\mathbf{r} \in \mathbf{R}^m$. While necessarily approximate, such a representation can be quite useful for compressing redundant information in $\mathbf{u}$. To move between the two spaces one needs a *projection operator* $P : \mathbf{u} \mapsto \mathbf{r}$ and its inverse $P^{-1} : \mathbf{r} \mapsto \mathbf{u}$. The former cannot be one-to-one because $m < n$ by assumption.

For some applications, dimension reduction alone is a useful technique. In physical simulation, however one is interested not only in the state $\mathbf{u}$, but also in its time evolution, typically described by an ordinary differential equation: $\dot{\mathbf{u}} = F(\mathbf{u})$. (The dot indicates a time derivative.)

The link between dimension reduction and physics is *model reduction*. One seeks an analogous evolution equation for the reduced state: $\dot{\mathbf{r}} = \hat{F}(\mathbf{r})$. Ideally, this equation should be computable in closed form without resorting to the high-dimensional space. The standard solution is called the *Galerkin projection* of F onto the reduced dimensional space

$$\hat{F} = P \circ F \circ P^{-1}. \tag{3.1}$$

Now consider an important special case: suppose the reduced vectors cover an m-dimensional *linear* subspace of $\mathbf{R}^n$. That is, there exists an $n \times m$ matrix B such that $\mathbf{u} = B\mathbf{r}$. Further, if B is orthonormal then $\mathbf{r} \approx B^T\mathbf{u}$, with equality when $\mathbf{u}$ lies in the subspace spanned by B. Armed with this linear subspace, consider the case of linear differential

equation $\dot{\mathbf{u}} = M\mathbf{u}$. The Galerkin projection of M yields another linear differential equation $\dot{\mathbf{r}} = B^T MB\mathbf{r}$. Moreover, one can *precompute* the $m \times m$ matrix product $B^T MB$ to simulate without resorting to the high-dimensional space. More generally, if F is an n-dimensional polynomial, then $\hat{F}$ is an m-dimensional polynomial of the same degree which can be computed in closed form without resorting to the high-dimensional space [9]. I will use this important fact several times.

3.2 Galerkin Projection of Fluids

This section applies Galerkin projection specifically to create a high-speed fluid simulator. The first task is to create a low-dimensional basis B for the simulation. Assume that the domain has been spatially discretized so that all velocity fields can be represented as n-dimensional vectors. In practice, I use the standard MAC-style discretization [51, 42], which is illustrated in Figure 3.12. The technique, however, is general enough to handle arbitrary arrangements such as irregular tetrahedral meshes [36, 32].

The technique begins with a set of example velocity fields $\{\mathbf{u}_i\}$, forming a representative basis U of the interactions expected in the system. These example states typically are snapshots drawn from a set of off-line, full-dimensional fluid simulations. The method assumes that the velocity fields in U satisfy two important properties:

1. The states are *divergence free*: $\nabla \cdot \mathbf{u}_i = 0$.[2]
2. The states satisfy *free-slip* boundary conditions: for all fixed surface points $\mathbf{x}$ with normal $\mathbf{n}$, the constraint $\mathbf{u}_i(\mathbf{x}) \cdot \mathbf{n} = 0$ holds.

The next step is computing a low-dimensional orthonormal basis $B = [\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_m]$ ($m \ll n$) that minimizes the square reconstruction error

$$||U - BB^T U||_F^2, \tag{3.2}$$

subject to the basis states $\hat{\mathbf{u}}_i$ preserving the two constraints above. Here $|| \cdot ||_F^2$ denotes the sum of squared matrix entries, known as the Frobenius norm.

[2]Divergence-free in the discrete sense: every voxel has zero net flux.

Equation (3.2) can be minimized by setting B to the first m eigenvectors of the matrix UU^T, effectively performing a Principal Component Analysis (PCA). Further, the vectors produced by PCA satisfy the above constraints without modification. To see why, note that both constraints are linear. That is, for each constraint there exists a *constraint matrix* C whose null space is exactly the set of vectors satisfying the constraint (in particular $CU = 0$). Now suppose that $\hat{\mathbf{u}}_i$ is a vector in B. Then by definition $UU^T\hat{\mathbf{u}}_i = \breve{}_i\hat{\mathbf{u}}_i$, which implies that $CUU^T\hat{\mathbf{u}}_i = \breve{}_iC\hat{\mathbf{u}}_i = 0$. So the basis vector $\hat{\mathbf{u}}_i$ will satisfy the constraints as long as the eigenvalue $\breve{}_i$ is nonzero, as it will be in general. Figure 3.3 shows the basis constructed for a domain with a fixed yin-yang boundary. Notice that all basis states satisfy both the divergence and boundary constraints.

3.3 Simulation

Given the reduced velocity basis B, I now present an algorithm capable of simulating the incompressible fluid equations through the reduced subspace. The incompressible Navier-Stokes equations can be written as:

$$\underset{\text{(A)}}{\dot{\mathbf{u}}} = \underset{\text{(A)}}{-(\mathbf{u}\cdot\nabla)\mathbf{u}} - \underset{\text{(B)}}{\nu\nabla^2\mathbf{u}} + \underset{\text{(C)}}{\nabla p} + \underset{\text{(D)}}{\mathbf{f}}, \quad \text{s.t.} \quad \underset{\text{(C)}}{\nabla\cdot\mathbf{u} = 0}, \tag{3.3}$$

where $\mathbf{u}$ denotes the velocity field, p the pressure, ν the viscosity, and $\mathbf{f}$ the external forces. Following Stam [124], my method decomposes the simulation into a series of separate steps, each one tackling a different term in equation (3.3). This technique is known as *operator splitting*. The steps are: (A) Advection, (B) Diffusion, (C) Projection, and (D) External Forces. The remainder of this section explains how each of these four steps can be simulated on the subspace B, with a time complexity per timestep *that depends only on the basis dimensionality*.

3.3.1 Advection

To simulate advection, split the velocity field into its three components $\mathbf{u} = [u^x, u^y, u^z]^T$, and write the velocity advection of each component as $\dot{u}^\star = -(\mathbf{u}\cdot\nabla)u^\star = -\nabla\cdot(\mathbf{u}u^\star)$, where $\star$

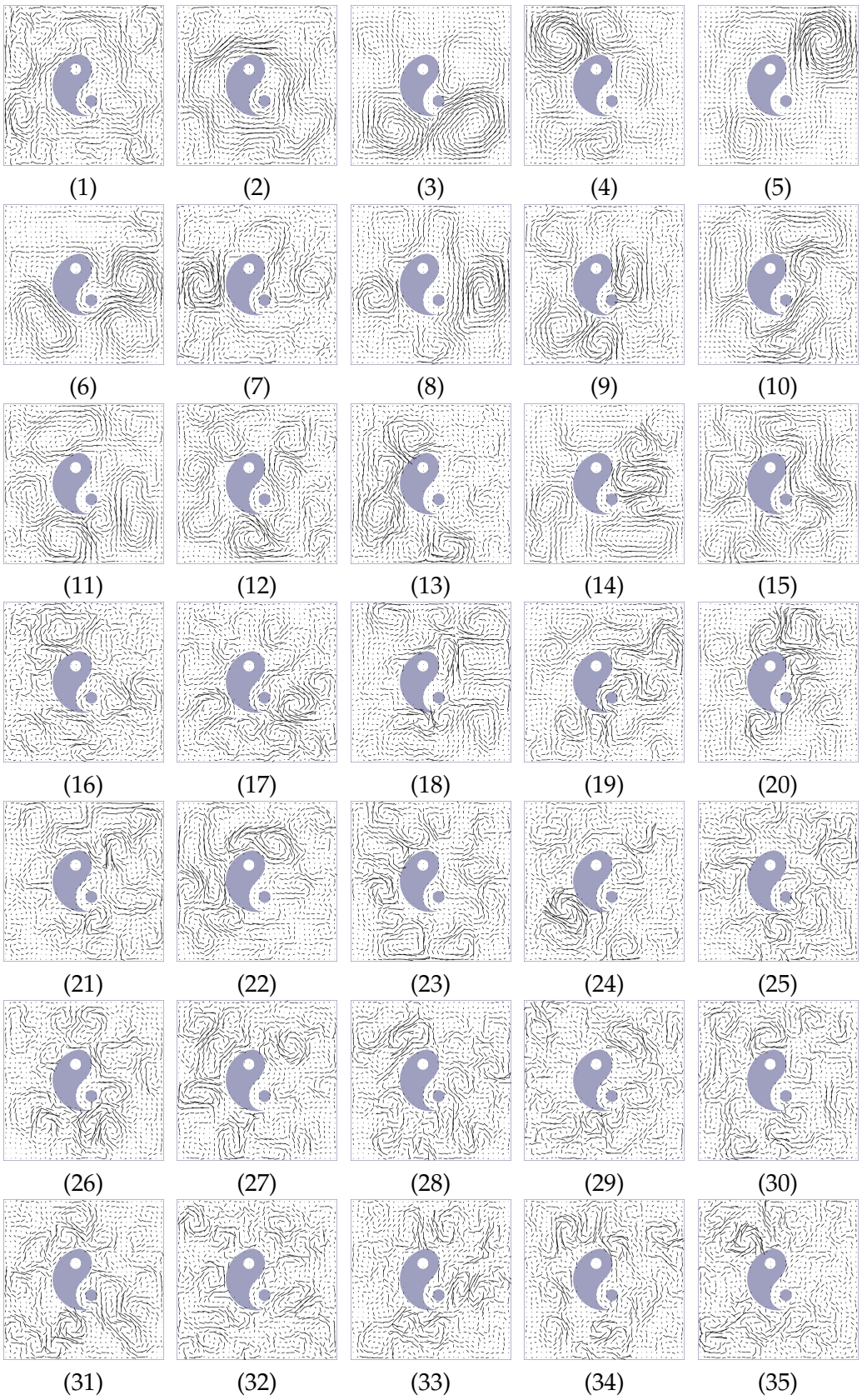


Figure 3.2: Bases 1-35 for a 512×512 fluid simulation with a fix yin boundary. (Grids have been subsampled for display.)

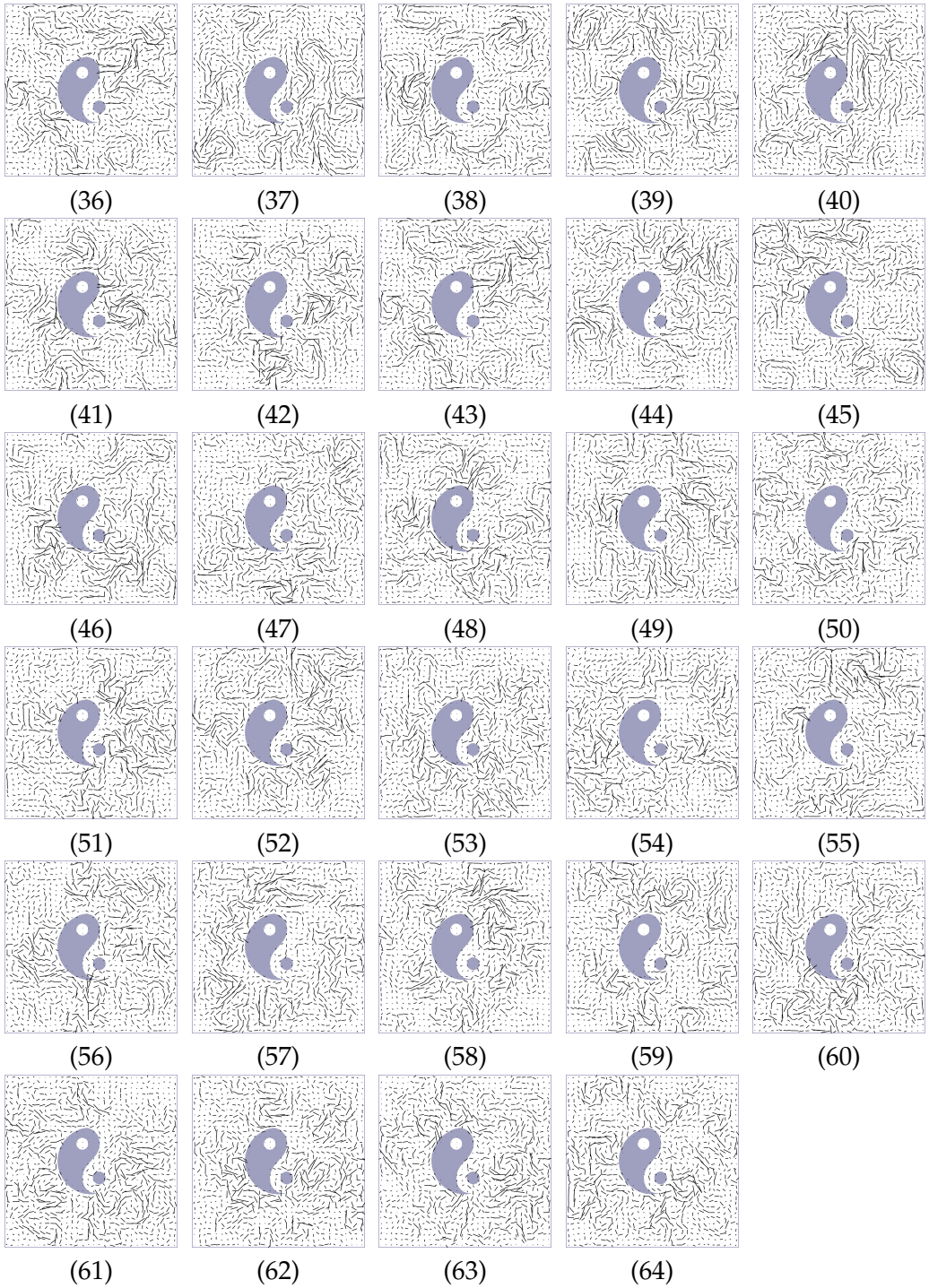


Figure 3.3: Bases 36-64 for a 512×512 fluid simulation with a fix yin boundary. (Grids have been subsampled for display.)

stands for x, y, or z. The right-hand side of this equation is called the *conservation form* of the advection equation, and equality follows from the zero divergence of the velocity field. The conservation form can be discretized about each grid face (i,j,k) to form the partial differential equation:

$$\begin{aligned} \dot{u}^{\star}_{i,j,k} = \tfrac{1}{\Delta h}\big(&(u^x u^\star)_{i-1/2,j,k} - (u^x u^\star)_{i+1/2,j,k} + \\ &(u^y u^\star)_{i,j-1/2,k} - (u^y u^\star)_{i,j+1/2,k} + \\ &(u^z u^\star)_{i,j,k-1/2} - (u^z u^\star)_{i,j,k+1/2}\big), \end{aligned} \tag{3.4}$$

where Δh is the grid spacing. (Section 3.7 provides more details.) Fixing the velocity fluxes and recombining the three grids yields a linear differential equation:

$$\dot{\mathbf{u}} = A_{\mathbf{u}}\mathbf{u} \tag{3.5}$$

which describes how the velocity vector $\mathbf{u}$ would change if advected through a fixed velocity field defined by the advection matrix $A_{\mathbf{u}}$.

Now project this equation onto the reduced subspace B. Let $\mathbf{r} = [r_1, \ldots, r_m]^T$ represent the reduced state so that $\mathbf{u} = B\mathbf{r}$. Then, the advection matrix in equation (3.5) can be expressed as a linear combination of advection matrices for each velocity basis state:

$$\begin{aligned} \dot{\mathbf{u}} &= \left(A_{r_1\hat{\mathbf{u}}_1 + \cdots + r_m\hat{\mathbf{u}}_m}\right)\mathbf{u} \\ &= \left(r_1 A_{\hat{\mathbf{u}}_1} + \cdots + r_m A_{\hat{\mathbf{u}}_m}\right)\mathbf{u}. \end{aligned} \tag{3.6}$$

The second equality follows from the bilinearity of advection. This equation can, in turn, be orthogonally projected onto the B basis as:

$$\dot{\mathbf{r}} = \left(r_1 B^T A_{\hat{\mathbf{u}}_1} B + \cdots + r_m B^T A_{\hat{\mathbf{u}}_m} B\right)\mathbf{r} = \hat{A}\mathbf{r}. \tag{3.7}$$

Hence, at each timestep, take a linear combination of the m precomputed reduced advection matrices $B^T A_{\hat{\mathbf{u}}_i} B$ to form the reduced advection matrix $\hat{A}$ as in equation (3.7). The reduced state then can be advanced to the next timestep by exactly solving the linear ODE using the eigen decomposition method:

$$\mathbf{r}(t + \Delta t) = Ee^{\Delta t \Lambda} E^{-1}\mathbf{r}, \tag{3.8}$$

where $E\Lambda E^{-1} = \hat{A}$ is the eigen decomposition of $\hat{A}$. All of these operations take time $O(m^3)$ and are independent of n, the number of simulation voxels. Implementation details for the advection algorithm are given in section 3.7.

The advection algorithm described above naturally preserves kinetic energy, a highly useful property for which I now give a proof sketch. It can be shown that the discretization (3.4) always yields a zero-definite advection matrix ($\forall \mathbf{x}\ \ \mathbf{x}^T A_{\mathbf{u}} \mathbf{x} = 0$) given a divergence-free field $\nabla \cdot \mathbf{u} = 0$. The total kinetic energy is defined as $E = \frac{1}{2}\mathbf{u}^T\mathbf{u}$ (where $\mathbf{u}$ is the vector over all grid velocities). By the chain rule, $\dot{E} = \mathbf{u}^T\dot{\mathbf{u}} = \mathbf{r}^T B^T B\dot{\mathbf{r}} = \mathbf{r}^T\dot{\mathbf{r}}$. Substituting equation (3.7) for $\dot{\mathbf{r}}$ gives $\dot{E} = \sum_i \mathbf{r}^T B^T A_{\hat{\mathbf{u}}_i} B\mathbf{r}$. Because the advection matrices are all zero-definite, the terms $\mathbf{r}^T B^T A_{\hat{\mathbf{u}}_i} B\mathbf{r}$ all vanish, implying that $\dot{E} = 0$. Hence, total energy remains unchanged for exact solutions to the linear advection ODE, as in equation (3.8).

I emphasize that energy is preserved *only within the reduced subspace*; velocity modes not present in the subspace have zero energy, and represent this algorithm's primary source of error. Moreover, in practice I intentionally dissipate energy through diffusion (section 3.3.2) because completely inviscid flows are not seen in nature and will gradually depart from the observed basis. Also, diffusion adds stability during user interaction. Nonetheless, energy preservation allows the user to control the energy dissipation profile precisely (Figure 3.4).

3.3.2 Diffusion

The diffusion step accounts for viscosity. Simulation proceeds by discretizing the diffusion operator $\nu\nabla^2$ into a diffusion matrix νD. In an unreduced simulation, the next step would be to solve the linear ODE either explicitly or implicitly, each of which has problems. Explicit methods become unstable for high kinematic viscosities ν, while implicit solvers are stable, but require solving a linear system. The technique described here resolves both of these problems: it simulates more quickly than an explicit solver and more accurately than an implicit method. The algorithm is as follows: orthogonally project the diffusion matrix

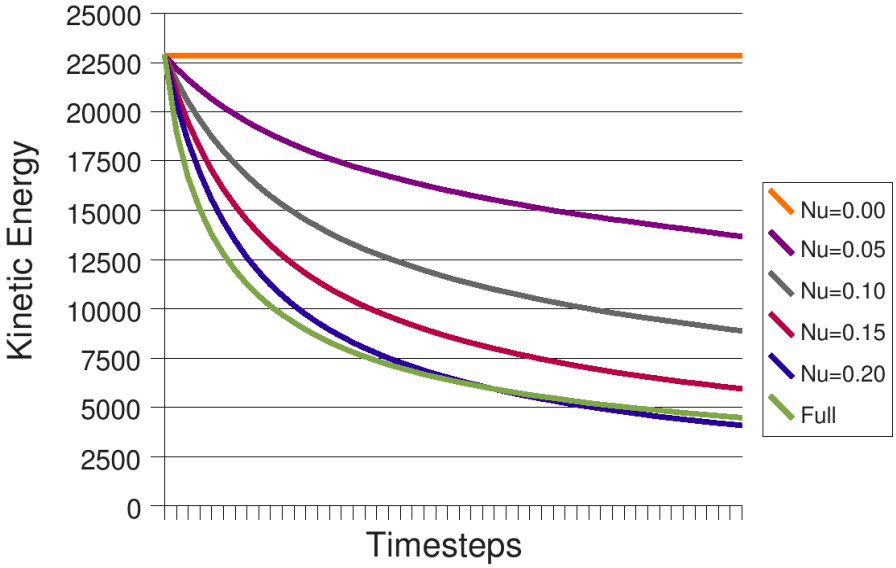


Figure 3.4: Energy dissipation in a 128×128 simulation. My simulator exactly preserves kinetic energy in the absence of diffusion, $\nu = 0.0$. A diffusion coefficient of $\nu = 0.2$ (blue) most accurately matches the numerical viscosity inherent in my full-dimensional simulator (green).

D to produce the reduced diffusion matrix $\hat{D} = B^T D B$. Then, *exactly* solve the linear ordinary differential equation defined by the reduced diffusion matrix $\mathbf{r}(t + \Delta t) = E e^{\Delta t \Lambda} E^{-1} \mathbf{r}$ (where $E \Lambda E^{-1} = \hat{D}$ now denotes the eigen decomposition of the precomputed *diffusion* matrix). Unlike advection, this transition matrix is constant throughout the simulation and therefore can be precomputed. This enables stable, accurate diffusion with a single $m \times m$ matrix multiply. Note that, as with advection, accuracy is only within the reduced subspace; the system cannot diffuse into configurations outside the subspace.

3.3.3 *Projection*

Projection involves adding a pressure force ∇p to enforce the flow's incompressibility and boundary conditions. As noted in section 3.2, PCA preserves these constraints automatically for fixed boundaries. Thus $p = 0$, and the projection step can be skipped. This yields substantial speed gains because projection is, in general, the most costly step of grid-based simulators.[3] Section 3.5 extends this technique to *moving* objects by using a local approxi-

[3]Even with symmetric operators on hierarchical grids, this step can take 25% of each timestep [86].

mation of the pressure forces around each object.

3.3.4 External Forces

External forces can be added by projecting the forces $\mathbf{f}$ onto the reduced subspace $\hat{\mathbf{f}} = B^T\mathbf{f}$, and adding those onto the reduced velocity vector: $\mathbf{r} += \hat{\mathbf{f}}$. Note that because the basis is global, even sparse forces $\mathbf{f}$ project to non-sparse reduced force vectors $\hat{\mathbf{f}}$.

3.4 Rendering

For the flow to be visible, pure velocity simulations must be coupled with immersed media, such as particles (including dust, leaves, and texture sprites) or volumetric smoke densities. Particles are particularly well suited to real-time reduced fluid simulation: particle rendering is fast and simulation only requires reconstructing velocities in the neighborhood of each particle. Thus, one sees particular speedups when marrying model reduction with particle simulation. To simulate volumetric densities, I perform semi-Lagrangian advection [124] through the full velocity field $\mathbf{u}$ at each timestep. While the $O(mn)$ velocity reconstruction step introduces a performance dependency on the number of voxels n, this approach is justified because the cost of rendering is also proportional to the number of voxels, and is generally much higher than that of simulation. For both particles and densities my fluid simulator is *output-sensitive*: the costs of simulation and rendering are proportional to one another.

3.5 Boundary Conditions

The above algorithm simulates the complete incompressible Navier-Stokes equations on a reduced dimensional basis. This algorithm is self-contained, and generally much faster than traditional techniques. The only means of user interaction, however, is the external force term (section 3.3.4). While forces can be useful, a more natural form of user interaction would be to *move* objects through the flow. I now explain how to extend the above simulator to do just that.

Boundaries are handled via the pressure term (3.3C), which adds the minimal force so

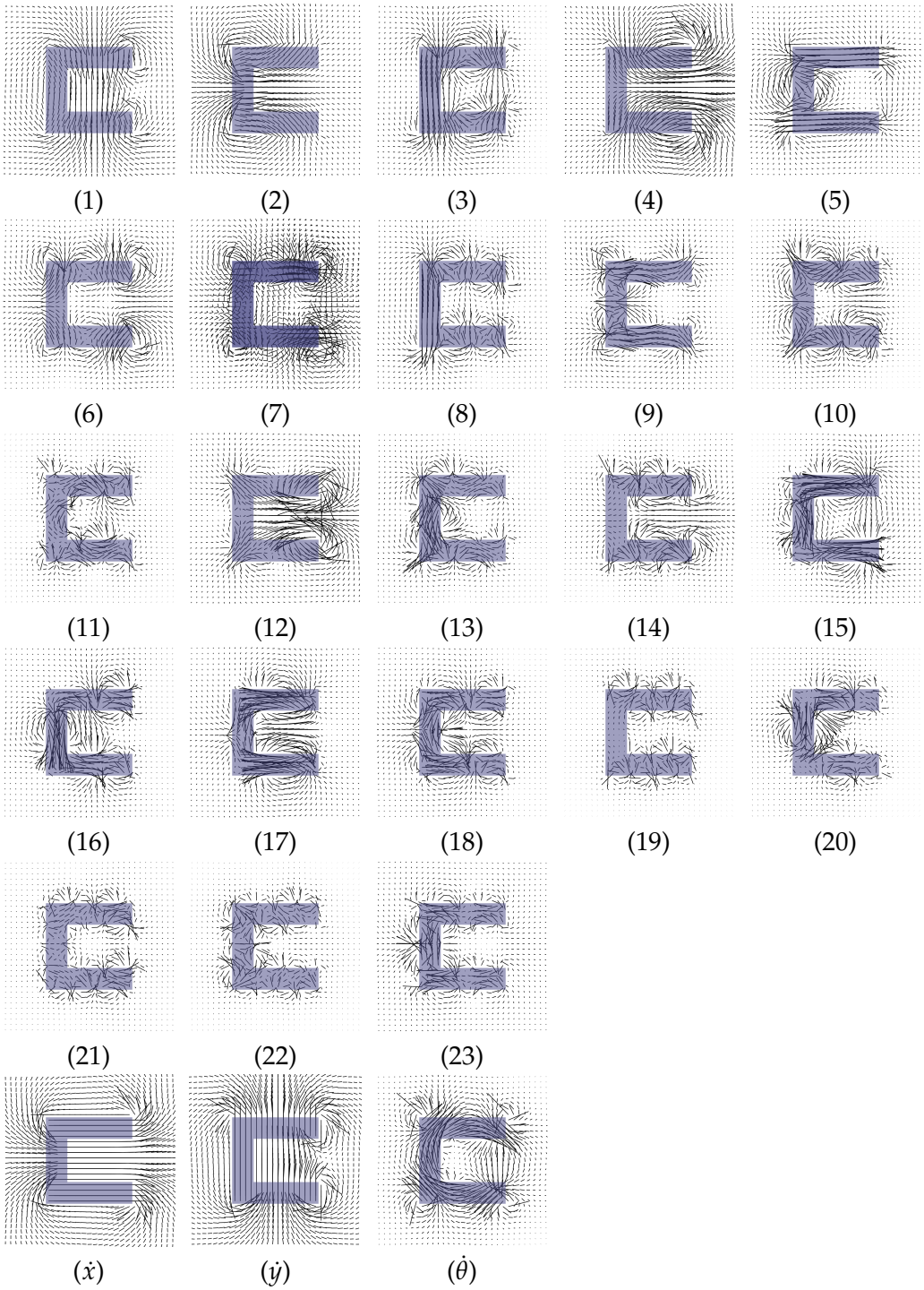


Figure 3.5: (1)-(23): The 23 boundary basis grids for a c-shaped object. These form a basis for the expected velocities at the surface, and are divergence-free everywhere else. $(\dot{x})$, $(\dot{y})$, and $(\dot{\theta})$: The additional translational and rotational bases (section 3.5.5).

that the fluid velocity field $\mathbf{u}$ equals the speed of the object $\mathbf{v}$ along all surface normals. That is, at surface point $\mathbf{x}$ with normal $\mathbf{n}$, the constraint $\mathbf{u}(\mathbf{x}) \cdot \mathbf{n} = \mathbf{v}(\mathbf{x}) \cdot \mathbf{n}$ holds (tangential fluid movement is allowed). This *free-slip* condition cannot be satisfied by the low-dimensional physics alone because the model does not have sufficient degrees of freedom. Therefore, the reduced model must somehow be extended. Ideally, the solution would allow high-frequency fidelity along the surface: for example, smoke would appear to curl around a surface rather than penetrate it. It should also be possible to handle multiple objects moving independently. Most importantly, the boundary model should be low-dimensional (possibly approximate), and admit all computations in the reduced-dimensional space.

3.5.1 The Boundary Basis

The solution is to add a set of forces in the local neighborhood of each object which enforces the boundary conditions at the surface. I use a linear representation, so that these forces consist of a boundary basis $\tilde{B} = [\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_p]$ combined with a reduced state vector $\tilde{\mathbf{r}} = [\tilde{r}_1, \ldots, \tilde{r}_p]^T$. The power of the boundary bases is that they track the location of their objects. For an object with rotation matrix Φ and position vector $\mathbf{t}$, let the *resampling operator* $R_{\Phi,\mathbf{t}}$ convert a velocity field from the object's frame of reference into the global coordinate system: $[R_{\Phi,\mathbf{t}}\mathbf{u}](\mathbf{x}) \equiv \Phi\mathbf{u}(\Phi^{-1}\mathbf{x} - \mathbf{t})$, as illustrated in Figure 3.6. The full velocity field $\mathbf{u}'$ combines both simulation basis and all boundary bases (properly oriented in the global coordinate system):

$$\mathbf{u}' = B\mathbf{r} + \sum_i R_i \tilde{B}_i \tilde{\mathbf{r}}_i, \tag{3.9}$$

where $R_i = R_{\Phi_i,\mathbf{t}_i}$ is shorthand for object i's resampling operator. By combining object tracking with a linear representation, boundary bases allow the quick and compact enforcement of the free-slip conditions.

Since the free-slip boundary conditions constrain surface speed, it will be useful to define an operator which measures these velocities. To do so, discretize object i's surface into a set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_q\}$ with corresponding surface normals $\{\mathbf{n}_1, \ldots, \mathbf{n}_q\}$. Now define the *surface speed operator* $S_i\mathbf{u} \equiv \left[\mathbf{u}(\mathbf{x}_1) \cdot \mathbf{n}_1, \ldots, \mathbf{u}(\mathbf{x}_q) \cdot \mathbf{n}_q\right]^T$. Intuitively, the indices into this

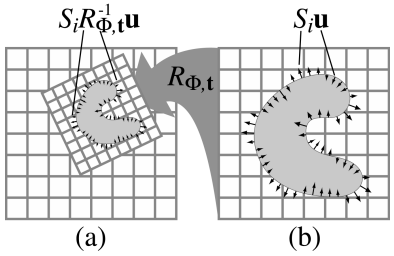


Figure 3.6: The resampling operator $R_{\Phi,\mathbf{t}}$ changes a velocity field's coordinate system, while the surface speed operator S_i measures surface normal velocity.

vector correspond to points along the surface, and the entries are the normal speeds at these points. Graphically, these are the lengths of the spiky surface vectors in Figure 3.6(b). Surface speeds can be computed for other object orientations $(\Phi, \mathbf{t})$ by first resampling the velocity field $S_i R^{-1}_{\Phi,\mathbf{t}}\mathbf{u}$ as in Figure 3.6(a). (The inverse R^{-1} is required to convert into the *object*'s frame of reference.)

I now explain how the boundary basis is constructed. For simplicity, provisionally assume that each object is at rest. Hence, the boundary basis should cancel all velocity normal to the object surface: $S_i R_i^{-1}\mathbf{u} = \mathbf{0}$, meaning that the basis should match the velocity basis as closely as possible (with opposite sign). Therefore, the goal is to construct the boundary basis most representative of the expected surface velocities. To do so, perform a double loop over the velocity fields in the simulation basis $\mathbf{u} \in B$ and a representative set of orientations $(\Phi, \mathbf{t}) \in \mathcal{P}$. For each velocity field / orientation pair, measure the surface speed assuming object i were thus situated, and collect these vectors into a large set $\mathcal{S}_i = \{S_i R^{-1}_{\Phi,\mathbf{t}}\mathbf{u} \mid (\Phi, \mathbf{t}) \in \mathcal{P}, \mathbf{u} \in B\}$. Running PCA yields the most representative p-dimensional basis for $\mathcal{S}_i$. In fact, I set the boundary basis $\tilde{B}_i$ to be the set of minimum energy divergence-free velocity fields whose surface speeds are those PCA eigenvectors: $S_i\tilde{B}_i = PCA_p[\mathcal{S}_i]$. Figure 3.5(1)-(23) shows the resulting boundary basis for a c-shaped object. Note that because PCA always returns an orthonormal set of vectors, the following useful property holds:

$$\left(S_i\tilde{B}_i\right)^T S_i\tilde{B}_i = I. \tag{3.10}$$

3.5.2 Coupling

Taking the boundary bases into account requires a modest extension to the algorithm. At each timestep, after the reduced state $\mathbf{r}$ has been simulated but *before* immersed media have been advected, the reduced state vector $\tilde{\mathbf{r}}$ is computed for each object. Immersed media can then be advected through the *complete* velocity field $\mathbf{u}'$ (3.9). At the end of each timestep, the algorithm Galerkin projects these boundary forces back into the reduced simulation basis. Figure 3.7 gives more precise pseudocode. For now, let us take a closer look at how information is propagated between the simulation and boundary bases, which I call the *feedforward* $F_i : \mathbf{r} \mapsto \tilde{\mathbf{r}}_i$ and *feedback* $G_i : \tilde{\mathbf{r}}_i \mapsto \mathbf{r}$ operators, respectively.

1.	Advect the velocities.	$O(m^3)$	(3.3.1)
2.	Diffuse the velocities.	$O(m^2)$	(3.3.2)
3.	Add forces.	$O(m)$	(3.3.4)
4.	**for each object** i:		
	// Feedforward.		
	$\tilde{\mathbf{r}}_i \leftarrow F_i\mathbf{r}$	$O(pm)$	(3.5.3)
5.	Advect immersed media in $\mathbf{u}'$.		(3.4)
6.	**for each object** i:		
	// Feedback.		
	$\mathbf{r} \mathrel{+}= G_i\tilde{\mathbf{r}}_i$	$O(pm)$	(3.5.4)

Figure 3.7: Per-timestep pseudocode. The global field is simulated first (1)-(3), then boundaries are handled (4)-(6).

3.5.3 Forward Coupling

For simplicity, assume for now that object i is at rest (this assumption is relaxed in section 3.5.5). As mentioned above, this implies no velocity along surface normals: $S_iR_i^{-1}\mathbf{u} = \mathbf{0}$. Therefore, premultiplying equation (3.9) by $S_iR_i^{-1}$ and ignoring all objects but i yields:

$$\mathbf{0} = S_iR_i^{-1}B\mathbf{r} + S_i\tilde{B}_i\tilde{\mathbf{r}}_i.$$

Ignoring these objects is justified because coupling effects are negligible unless two objects are very close. Using the orthonormality of $S_i\tilde{B}_i$ (3.10), the above equation can be solved

as $\tilde{\mathbf{r}}_i = -\left(S_i\tilde{B}_i\right)^T S_i R_i^{-1} B\mathbf{r}$. Therefore the feedforward operations:

$$F_i \equiv -\left(S_i\tilde{B}_i\right)^T S_i R_i^{-1} B. \tag{3.11}$$

Intuitively, the velocity at the object surface $S_i R_i^{-1} B$ has been projected into the space of boundary basis fields $S_i\tilde{B}_i$ to get the reduced boundary vector $\tilde{\mathbf{r}}$ which best cancels the surface velocities. Computationally, this operation requires first iterating over all object faces and accumulating velocities, and then projecting the surface speeds onto the boundary basis. Performing these two steps takes time $O(qp + qm)$, where q is the number of faces, and m and p are the sizes of the simulation and boundary bases, respectively. Section 3.5.6 describes how this complexity may be greatly reduced through precomputation.

3.5.4 Backward Coupling

After the reduced boundary vectors have been computed for each object, and immersed media have been advected, the simulator projects the effect of the boundary forces back onto the reduced simulation basis. Equation (3.9) can be Galerkin-projected (premultiplied by B^T) to read:

$$\mathbf{r}' = \mathbf{r} + \sum_i B^T R_i \tilde{B}_i \tilde{\mathbf{r}}_i.$$

Therefore, the feedback operation

$$G_i \equiv B^T R_i \tilde{B}_i \tag{3.12}$$

requires iterating over the voxels in the boundary basis, accumulating boundary forces, and then projecting these forces onto the reduced simulation basis. This two step process takes time $O(bp + bm)$, where b is the number of voxels in the boundary basis. Like forward coupling, the feedback operator may be reduced, as described in section 3.5.6.

3.5.5 *Extension to Moving Objects*

The above algorithm assumes that the object is at rest. However, a simple extension handles moving objects. First, the boundary basis $\tilde{B}$ must be extended to include local forces due to translation and rotation. In 2D there are two translational forces, $\dot{\mathbf{x}}$ and $\dot{\mathbf{y}}$, and one rotational force $\dot{\theta}$, as shown in Figure 3.5. Therefore the basis is expanded to become $\tilde{B}' = \left[\tilde{B}, \dot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\theta}\right]$. In 3D there are three translational forces and three rotational forces. At runtime, the feedforward operator is computed as before (section 3.5.3), except that after surface velocities have been canceled, the extra forces are added equal to the translation and rotation of the object. The feedback operation (section 3.5.4) remains identical, except that the full boundary basis, including the extra forces, must be projected back into the reduced basis.

3.5.6 *Precomputed Coupling*

Sections 3.5.3 and 3.5.4 defined the feedforward and feedback operators in the full space: feedforward requires iterating over all object faces, while feedback requires iterating over all voxels in the boundary basis. Notice however that these operations (3.11) and (3.12) are *linear* in their operands (although nonlinear in the object location). Therefore, the matrices F and G can be precomputed for a discretized set of boundary orientations $(\Phi, \mathbf{t})$. This allows feedforward and feedback simulation entirely in the reduced space. Both operations become virtually instantaneous matrix multiplies taking time $O(pm)$ per object, where p and m are the dimensions of the boundary and simulation bases respectively.

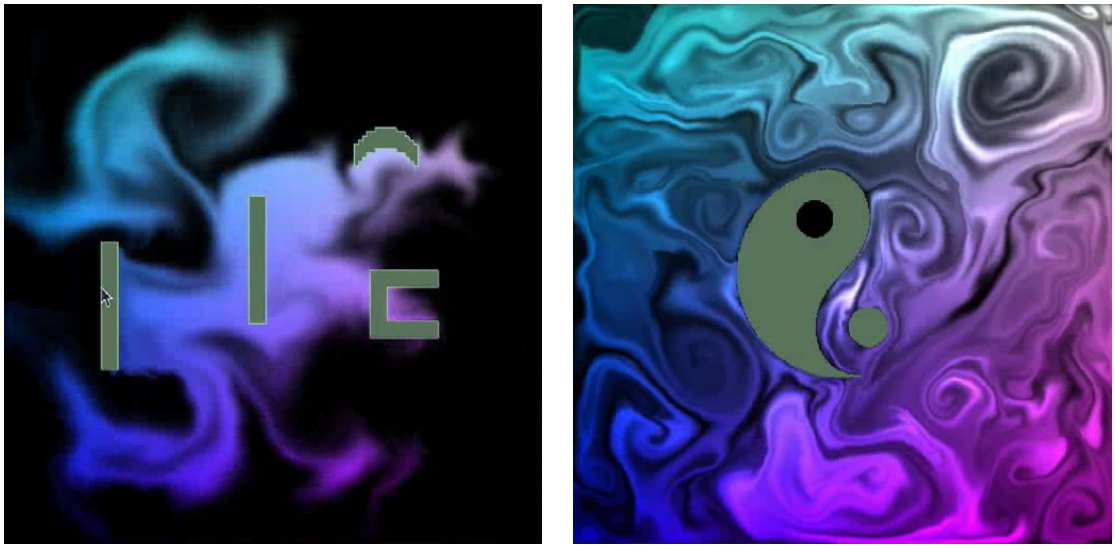
Precomputing the feedback and feedforward operators dramatically improves simulation time. However, it can be very costly in terms of memory, particularly if matrices are being stored along both translational and rotational axes. High-order tensor compression offers the most promising solution to this problem. For my results, I used PCA to compress the feedback matrices, storing only the coefficients at each grid cell. The technique of Wang *et al.* [138] and related ideas might yield even greater memory savings. There are also a number of other strategies one can employ to reduce the memory burden. For example, if an object were restricted to a certain part of the domain or certain orientations, then

only the corresponding matrices need be stored. A balance must also be sought between finely discretized boundary orientations and memory. Doubling the spacing between precomputed matrix samples reduces memory requirements by 75% in 2D, but also decreases simulation accuracy, a tradeoff which is explored in Figure 3.11. In some cases, boundary operations may not be the bottleneck, and precomputation is not necessary. However especially in particle simulations, feedforward and feedback precomputation allows simulation completely in the reduced space, yielding large time savings.

3.5.7 The Reduced Model's Relation to the Physics

The pressure term of the Navier-Stokes equations (3.3C) ensures both that the flow is divergence-free and that surface boundary conditions are met. In my model, the former constraint is implicitly inherited from the basis grids. Time-varying boundary conditions, however, must be enforced explicitly using boundary bases. In fact, the boundary basis model can be viewed as an approximation of the pressure term: both add forces to the flow to enforce boundary conditions. However, the boundary basis technique relies on a number of physical approximations that I now make explicit.

Most importantly, the algorithm presented does not exactly satisfy the free-slip conditions unless the number of boundary basis states equals the number of object faces. However, PCA does find the optimal truncated set of boundary basis grids. In practice, visually accurate simulations can be achieved with 64 basis states, the number used for the examples in section 3.6. Figure 3.11 quantifies the success of this approach. Second, the technique assumes that the *instantaneous* effect of an object on the flow is limited to the extent of that object's basis. While the pressure equation is theoretically global, in reality the effects outside the boundary basis are vanishing small. Note that non-local time effects are handled by the feedback term. Third, every object is computed independently, ignoring the visually negligible coupling between constraints. Finally, the pressure equation requires that the field be minimally disturbed to accommodate boundaries, but no such assurance is possible because the boundary basis may not be orthogonal to the underlying simulation basis. Compression of the feedback matrices introduces yet another layer of



(a) (b)

Figure 3.8: 2D Fluid Simulations. (a) *2D Boundaries:* User-manipulated boundaries in a 128×128 domain. (b) *Forces:* User-added forces on a 512×512 domain running at 41 frames/second. See Tables 3.1, 3.2, 3.3 for more details.

approximation as described in section 3.5.6.

3.6 Results

I have produced a number of simulations with my system (Table 3.2).[4] Precomputation began by compressing a set of high-resolution fluid snapshots into the simulation basis, then compressing a representative set of surface speed vectors into the boundary basis for each object. For dimension reduction, I used the output-sensitive PCA algorithm described in a technical report by James and Fatahalian [63]. Precomputation runtimes are summarized in Table 3.3. All precomputations fit in 3.5GB of main memory, except for the 2 million voxel *Leaves* example for which out-of-core PCA was used. After precomputation, my system performed real-time simulation and rendering on a 3.6GHz Intel Xeon computer with an NVIDIA Quadro FX 3450 graphics card. All fluid and particle simulation was performed on the Central Processing Unit (CPU), while density simulation and rendering

[4]Videos of these results can be seen on the webpage: http://grail.cs.washington.edu/projects/model-reduction/

Table 3.1: Summary of fluid results. The "Type" column indicates the reduced simulation's processor architecture.

Name	Shape	Bases	Type	Figure
2D Boundaries	128×128	64	CPU/GPU	3.8(a)
Forces	512×512	64	CPU/GPU	3.8(b)
3D Boundaries	$64\times64\times64$	64	CPU/GPU	3.9
Leaves	$256\times64\times128$	32	CPU	3.1

Table 3.2: Timing summary. All times are measured in milliseconds per frame. "GPU" and "CPU" are the milliseconds per timestep of my full-dimensional simulations on the graphics and central processing units, respectively. "Vels," "Bound," and "Media" are the time taken by the velocity, boundary, and immersed media (density or particle) steps in the reduced simulator. "Total" is the full cost of a timestep in the reduced simulator. The final two columns measure the speedup ratio of model reduction compared to a standard (full) simulation.

Info	Full Runtime		Reduced Runtime				Speedup (Slowdown)	
Name	GPU	CPU	Vels	Bound	Media	Total	vs. GPU	vs. CPU
2D Boundaries	6.9	107	8.8	5.7	2.4	16.9	**(2)**	**6**
Forces	139	3595	8.8	0.0	14.5	23.3	**6**	**154**
3D Boundaries	521	3381	8.8	8.9	15.6	33.3	**16**	**102**
Leaves	–	26893	1.2	0.3	0.2	1.7	**–**	**15819**

were accelerated by the Graphics Processing Unit (GPU).

I now explain in more detail how the results were created. The *2D Boundaries* example was trained on 144 simulations of a moving c-shaped object immersed in the flow (Figure 3.5). I then introduced two additional objects, maintaining visually plausibility although the simulator had not trained on these objects. The *Forces* example (Figure 3.8(a)) was trained with 15 examples of impulse forces which the user can interactively mix at runtime. In both of the 2D examples, the density coloration is purely aesthetic. For the *3D Boundaries* example (Figure 3.9), I trained the system with a flat boundary object at three locations in a turbulent rising velocity field. At runtime, the user can move both this, and a similar object containing an aperture. I cause the smoke to rise by immersing it in a uniform updraft that unfortunately produces a rather laminar flow. A more principled approach would use a model-reduced temperature field in conjunction with the velocities; much of

Table 3.3: Precomputation time. The "Velocities" column shows the time to compress the simulation basis and project the physics, while "Boundaries" shows the time to compress the boundary basis and compute reduced coupling forces for each object. All times are in hours. Note that these are *total* times which I reduced in practice by parallelizing over a 16 processor cluster.

Name	Snapshots	Velocities	Boundaries
2D Boundaries	8208	8	2
Forces	1024	11	0
3D Boundaries	400	10	113
Leaves	364	17	50

the turbulence of rising smoke is caused by forces due to temperature gradients. Finally, my *Leaves* example (Figure 3.1) involves 4000 leaf particles immersed in a large fluid grid. The leaves were subject to both the velocity field and a small gravitational force. I trained these examples by adding impulse forces and by simulating a car moving rapidly through the grid following one of four precomputed paths. In real-time, the user is able to automatically add wind forces, or move the cars with the mouse. Note that the car example was completely implemented on the CPU.

Table 3.2 presents timing comparisons against two semi-Lagrangian fluid solvers, one implemented on the CPU and the other on the GPU. For reference, the CPU comparison solver is faster than the Maya 6.0 fluid solver on comparable hardware. Texture size limits prevented GPU comparison against *Leaves*, although this reduced example did not use the GPU anyway.

There are some subtleties in the data. In th *2D* and *3D Boundaries* examples, I precomputed only the feedback matrices, calculating feedforward forces on the fly. Because volumetric density rendering is so slow ($\sim 75\%$ of each timestep in 3D), this saved memory at a negligible loss of frame rate, but increased boundary costs. For the *Leaves* example, I precomputed all coupling forces, yielding much lower boundary costs. Note that *Leaves* is mys fastest result despite the large number of voxels, and despite the completely CPU-based implementation. While much of the savings is due to using particles over densities, these timing results also underscore the voxel-independent time complexity of my fluid simulator.

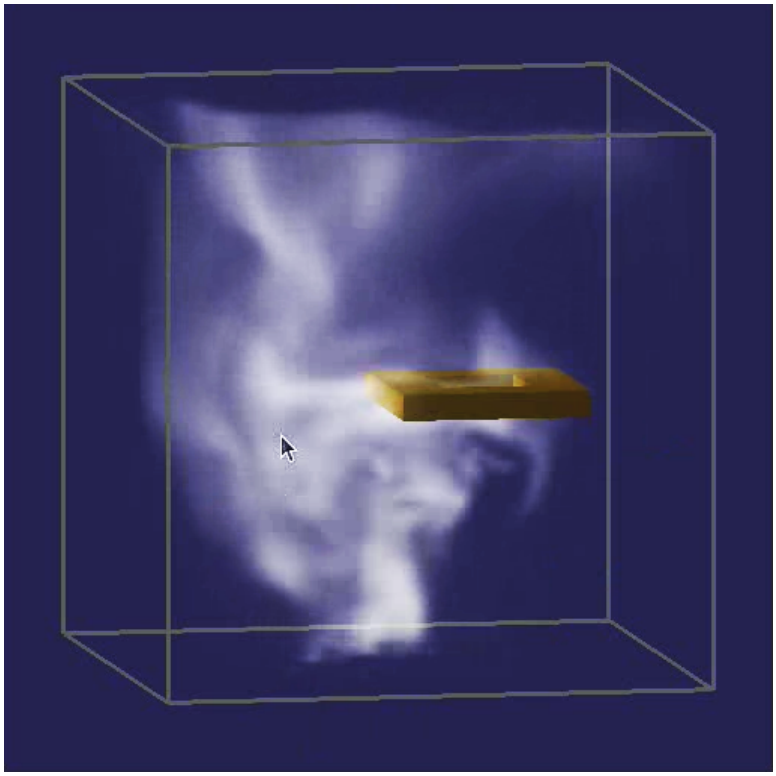


Figure 3.9: Real-time fluid-object interaction in a $64 \times 64 \times 64$ cube.

My results show that model reduction can preserve qualitatively important features such as turbulence (Figure 3.8(a)). However, there are also obvious inaccuracies relative to full simulation. There are several sources of error. First, the *choice and number of basis functions* plays an important role. Figure 3.10(a) shows how error varies with the basis size. For comparison, I included three lower resolution full simulations. These results show that spatial downsampling rapidly yields error at least as high as model reduction. For example, downsampling from 128×128 to 96×96 produces roughly the same error as projecting onto only 32 basis states. Because these trials were run without external input, the reduced simulations adhered closely to the full simulation. Another important source of error, however, is *divergence from observed data*. For example, the user may present runtime inputs on which the system has not been trained. While this is difficult to measure, Figure 3.10(b) shows one such example: I trained a 128×128 simulation with initial wind forces pointing at 0° and 90°. I then tested the reduced simulator with wind forces at other

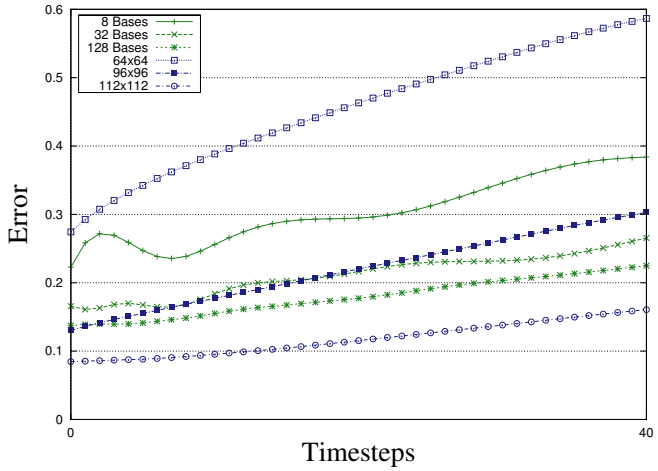
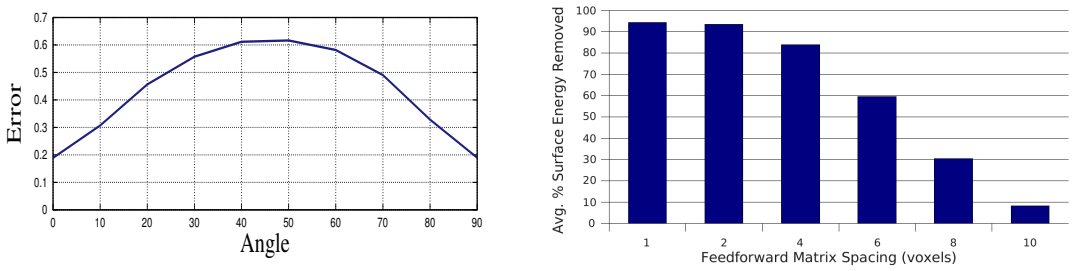


Figure 3.10: Simulation error for the model reduction of fluids. Error is measured against a full 128×128 semi-Lagrangian simulation of my *2D Boundaries* example. Green lines represent results of our reduced model at varying numbers of bases. Blue lines show errors under full simulation at varying resolutions.

angles. As the wind forces move toward 45° the simulation remains visually plausible but numerical accuracy diminishes, as expected. Both Figures 3.10(a) and 3.10(b) measure error using the relative l^2 norm $||\mathbf{u} - \mathbf{u}^*||_2 / ||\mathbf{u}^*||_2$ against the "ground truth" $\mathbf{u}^*$, a full 128×128 simulation. *Approximations due to the boundary basis* represent another source of error. I measure this as the percentage of surface energy removed per timestep by my technique. Perfect boundary conditions (100% energy removal) would only be possible with full set of boundary bases and exact feedforward computation. In practice, I use a truncated set of bases, and precomputed feedforward matrices drawn from a finite set $\mathcal{P}$ of boundary positions. Figure 3.11 shows the effect of using progressively smaller sets of translational boundary positions (as measured in the distance between samples). When feedforward matrix samples are one voxel apart, the simulation achieves $\sim 95\%$ surface energy removal per timestep for the c-shaped object in my *2D Boundaries* example. Finally, there are *global statistical measures of the performance*, such as energy preservation. As noted in section 3.3.1, my system actually can preserve energy when the diffusion coefficient $\nu = 0$. Otherwise, energy falloff can be precisely controlled (Figure 3.4).



(a) (b)

Figure 3.11: Further error metrics for the model reduction of fluids. (a) Observed force directions (0° and 90°) have lower error than unobserved force directions ($0° < \theta < 90°$). (b) Using fewer precomputed feedforward matrices (spacing the samples further apart) yields lower boundary fidelity.

3.7 Implementation Details

The most difficult section of code to write is the advection algorithm (section 3.3.1). On a MAC-style grid, velocities are defined as fluxes at the cell faces (Figure 3.12). For example, applying the advection equation (3.4) to the x-component of velocity and recentering at the cell face yields:

$$\begin{aligned}\dot{u}^x_{i,j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{\Delta h}\big(&+(u^x u^x)_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} &&-(u^x u^x)_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \\ &+(u^y u^x)_{i,j,k+\frac{1}{2}} &&-(u^y u^x)_{i,j+1,k+\frac{1}{2}} \\ &+(u^z u^x)_{i,j+\frac{1}{2},k} &&-(u^z u^x)_{i,j+\frac{1}{2},k+1}\ \big).\end{aligned} \tag{3.13}$$

Unfortunately, this equation queries velocities at non-MAC grid points. To resolve this, we must interpolate the velocities, as follows:

$$
\begin{aligned}
u^x_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} &= \frac{1}{2}\left(u^x_{i,j+\frac{1}{2},k+\frac{1}{2}} + u^x_{i+1,j+\frac{1}{2},k+\frac{1}{2}}\right). \\
u^x_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} &= \frac{1}{2}\left(u^x_{i,j+\frac{1}{2},k+\frac{1}{2}} + u^x_{i+1,j+\frac{1}{2},k+\frac{1}{2}}\right). \\
u^x_{i,j,k+\frac{1}{2}} &= \frac{1}{2}\left(u^x_{i,j-\frac{1}{2},k+\frac{1}{2}} + u^x_{i,j+\frac{1}{2},k+\frac{1}{2}}\right). \\
u^x_{i,j+1,k+\frac{1}{2}} &= \frac{1}{2}\left(u^x_{i,j+\frac{1}{2},k+\frac{1}{2}} + u^x_{i,j+1\frac{1}{2},k+\frac{1}{2}}\right). \\
u^x_{i,j+\frac{1}{2},k} &= \frac{1}{2}\left(u^x_{i,j+\frac{1}{2},k-\frac{1}{2}} + u^x_{i,j+\frac{1}{2},k+\frac{1}{2}}\right). \\
u^x_{i,j+\frac{1}{2},k+1} &= \frac{1}{2}\left(u^x_{i,j+\frac{1}{2},k+\frac{1}{2}} + u^x_{i,j+\frac{1}{2},k+1\frac{1}{2}}\right). \\
u^y_{i,j,k+\frac{1}{2}} &= \frac{1}{2}\left(u^y_{i-\frac{1}{2},j,k+\frac{1}{2}} + u^y_{i+\frac{1}{2},j,k+\frac{1}{2}}\right). \\
u^y_{i,j+1,k+\frac{1}{2}} &= \frac{1}{2}\left(u^y_{i-\frac{1}{2},j+1,k+\frac{1}{2}} + u^y_{i+\frac{1}{2},j+1,k+\frac{1}{2}}\right). \\
u^z_{i,j+\frac{1}{2},k} &= \frac{1}{2}\left(u^z_{i-\frac{1}{2},j+\frac{1}{2},k} + u^z_{i+\frac{1}{2},j+\frac{1}{2},k}\right). \\
u^z_{i,j+\frac{1}{2},k+1} &= \frac{1}{2}\left(u^z_{i-\frac{1}{2},j+\frac{1}{2},k+1} + u^z_{i+\frac{1}{2},j+\frac{1}{2},k+1}\right).
\end{aligned}
\tag{3.14}
$$

Plugging equations (3.14) into equation (3.13) yields the fully-discretized advection equation with all quantities properly defined for a MAC-style grid. Interpolation for the y- and z-components of the velocity is analogous.

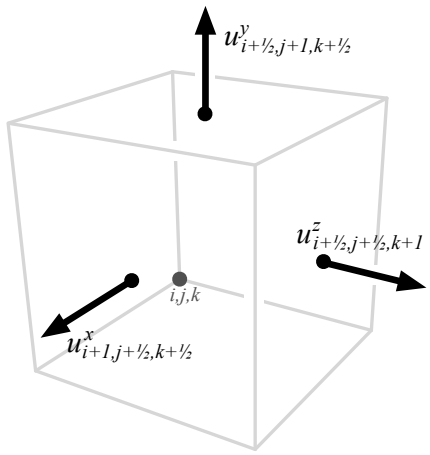


Figure 3.12: A MAC-style grid. Velocities are defined as fluxes at the cell faces.

Since the advection matrix A_u is highly sparse, I do not construct it explicitly. Instead, equations (3.14) and (3.13) are treated as a stencil to create the reduced advection matrix $\hat{A}_u$ which *is* constructed explicitly. A simple check that one's reduced advection matrix $\hat{A}_u$

is correct is that it be antisymmetric:

$$\hat{A}_u^T = -\hat{A}_u, \tag{3.15}$$

The antisymmetry of the reduced advection matrix also implies that it's eigen decomposition (equation (3.8)) has complex eigenvalues and eigenvectors. To compute this, I use the complex eigen decomposition function `dgeev` in LAPACK [1]. When the reduced advection algorithm is complete, one is technically left with a complex-valued reduced vector whose imaginary part should be zero to within machine precision. This vector can be converted back into the real-valued reduced velocity vector $\mathbf{r}$ by ignoring the imaginary part.

3.7.1 Parameter Settings

There are very few parameters which must be set in my technique. When creating a simulation, the only parameter is the number of reduced basis states. In my examples, I always used 64 basis states except for the simulation of a car going through leaves where I used 32. Generally, one should use as many basis states as possible subject to memory constraints. Theoretically, for very large number of basis states, the $O(m^3)$ reduced simulation time might become an issue, but this was not a problem in any of my simulations. As fewer basis states are used, the simulation looses detail and is less able to capture the nuances of human interaction. This degradation is further discussed in section 3.6 and illustrated in Figure 3.10.

When adding a moving object to the simulation, the implementor must decide how densely the object position and orientation is to be sampled when constructing the representative set of surface velocities (section 3.5). Again, the ideal is to use as many as possible, and the constraint is the time required to compute the reduced boundary basis. Generally, I sampled the object location at half the grid resolution and restricted this set to places in which I expected the object to be. For example, in the car simulations, the car was restricted to be on the road, and thus the surface velocity sampling was also restricted. A similar concern is the sampling of the coupling matrices, and again I generally sampled at half the grid resolution. As these samplings coarsen, the boundary conditions will be less

and less well enforced, meaning that fluid will traverse the boundary, a highly undesirable artifact visually. This effect is explored empirically in Figure 3.11(b).

The final parameter that can be set is the diffusion coefficient ν. This can be set to any nonnegative value without affecting the stability of the simulation (section 3.3.1). Setting $\nu = 0$ produced completely inviscid simulations which never loose energy. The result is that all user interaction will add energy to the system, which will rapidly yield very poor visual results. In my simulation, I used $\nu = 0.05$. The effects of different viscosity settings on energy dissipation are illustrated in Figure 3.4.

3.8 Discussion and Future Work

This chapter has presented a model reduction approach to real-time fluid simulation for interactive applications. I showed that moving objects can be integrated into the flow by endowing them with their own reduced models. Moreover, the coupling forces between the individual models can be computed in the reduced space as well. This treatment of moving objects overcomes one of the principle hurdles to adapting model reduction to interactive fluid simulation. In addition, the reduced advection term allows the user to precisely control the amount of energy dissipation in the simulation.

Within the gamut of fluid solvers, model reduction has a unique time complexity: the simulation runtime depends only on the basis size. Thus, complex phenomena can be observed on large grids and the number of variables is essentially proportional to the underlying degrees of freedom, such as the number of possible input forces. Grid resolution could be doubled, for example, without loss of speed. In practice, simulation and rendering are both dominated by the amount of immersed media. This makes model reduction particularly well suited to simulating immersed particles, where the velocity field need only be sparsely queried.

This speed, however, comes at the cost of accuracy, generality, and memory consumption. The simulation is unable to transfer into states not in the subspace, representing an inherent source of error particularly when the user presents inputs not in the training set. Moreover, because the basis states have global support, errors can have correspondingly

global effect on the field. This does not limit the user only to reproducing simulations in the training set: the system is robust to general user interaction, and to new objects. However, it would be surprising if completely new flows were generated. A related drawback is that the system must be trained before simulating. Changing the fixed boundary conditions, or creating new objects require lengthy precomputations. Once training is complete however, the system may be reused many times to produce simulation. As mentioned in section 3.5, the memory requirements of my algorithm are high. In fact, the grid size of my GPU-based 3D simulations was constrained by the GPU's 256 MB memory ceiling, and the cost of streaming grids to the GPU at each timestep prevents storing this data in CPU memory. Looking forward, however, I believe that the relative importance of model reduction will increase as memory costs decrease.

Another limitation of my technique is that the density simulation is computed in the full space. In principle, the density simulation could be reduced and the advection equation projected onto the joint reduced density/velocity space. However, issues related to density non-negativity and boundary conditions would have to be addressed, and it is questionable if a small number of modes could capture the required density configurations as well as they do the velocities. Moreover, as mentioned in section 3.6, the runtime bottleneck is actually the volumetric density rendering, not the simulation. If precomputed radiance transfer (essentially, model-reduced rendering) could be applied to the density field, then density reduction would certainly become an attractive possibility. In fact, this approach has recently been tried at low resolutions [50]. However, until this is perfected, immersed particles have the most promising cost relationship with model reduction.

I believe there are many other exciting branches to be explored. New model reduction techniques such as nonlinear Galerkin projection and balanced truncation might offer richer simulations for a similar number of modes. I am particularly interested in exploring the coupling between separate reduced models similar to my treatment of boundaries. Such coupling is crucial for generalizing model reduction to a wider set of phenomena, including compressible fluids, free-surface liquids, even particle systems such as hair and cloth.

Chapter 4

CROWD DYNAMICS

Figure 4.1: A variety of city scenarios simulated with our model.

Human crowds are ubiquitous in the real world, making their simulation a necessity for realistic interactive environments. Physically correct crowd models also have applications outside of computer graphics in psychology, transportation research, and architecture. This chapter focuses on real-time synthesis of crowd motion for thousands of individuals with intersecting paths. The goal is to simulate large groups with common goals, not for scenarios where each person's intention is distinctly different.

Real-time crowd simulation is difficult because large groups of people exhibit behavior of enormous complexity and subtlety. A crowd model must not only include individual human motion and environmental constraints such as boundaries, but also address a bewildering array of dynamic interactions between people. Further, the model must reflect intelligent path planning through this changing environment. Humans constantly adjust their paths to reflect congestion and other dynamic factors. Even dense crowds are characterized by surprisingly few collisions or sudden changes in individual motion. It has proven difficult to capture these effects in simulation, especially for large crowds in real-time.

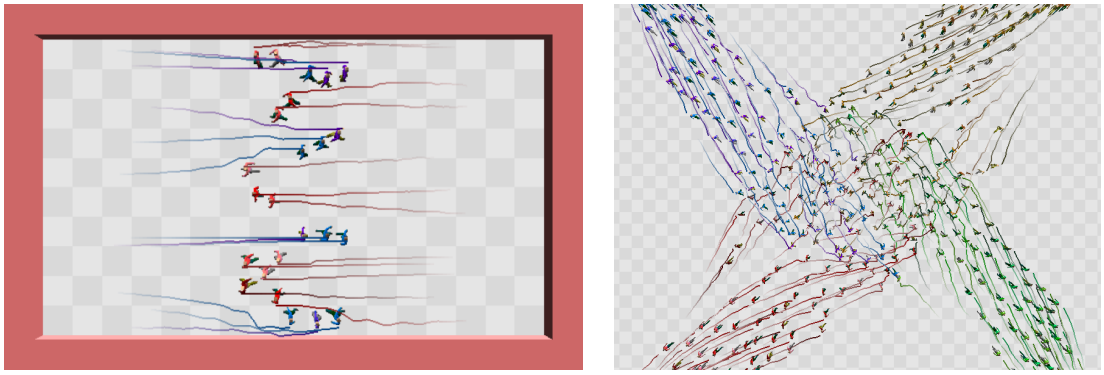
Virtually all previous work in the graphics and crowds literature has been *agent-based*, meaning that motion is computed separately for each individual. The agent-based ap-

[1]The work described in this chapter was originally presented as a paper [131] at SIGGRAPH 2006.

proach is attractive for several reasons. For one, real crowds clearly operate with each individual making independent decisions. Such models can capture each person's unique situation: visibility, proximity of other pedestrians, and other local factors. In addition, different simulation parameters may be defined for each crowd member, yielding complex heterogeneous motion. However, the agent-based approach also has drawbacks. It is difficult to develop behavioral rules that consistently produce realistic motion. Global path planning for each agent quickly becomes computationally expensive, particularly in real-time contexts. As a result, most agent models separate local collision avoidance from global path planning, and conflicts inevitably arise between these two competing goals. Moreover, local path planning often results in myopic, less realistic crowd behavior. These problems tend to be exacerbated in areas of high congestion or rapidly changing environments.

This chapter presents a real-time motion synthesis model for large crowds *without agent-based dynamics*. I view crowd motion as a per-particle energy minimization, and adopt a continuum perspective on the system. This formulation yields a set of dynamic potential and velocity fields over the domain that guide all individual motion simultaneously. My approach unifies global path planning and local collision avoidance into a single optimization framework. People in my model do not experience a discrete regime change in the presence of other people. Instead, they perform global planning to avoid both obstacles and other people. This dynamic potential field formulation also guarantees that paths are optimal for the current environment state, so people never get stuck in local minima.

Global path planning is frequently an unrealistic assumption. People often have limited vision and only partial knowledge of the terrain. A more accurate approximation would be long-term planning with limited visibility and knowledge. Still, my global planning assumption produces significantly smoother and more realistic crowd motion than the common and equally unrealistic assumption of strictly local knowledge. This is especially the case when people are threading through dense congestion. At the same time, my framework is not as general as agent-based methods: I trade-off individual variability for real-time planning of optimal crowd behavior with minimal computation per individual. To achieve the benefits of both models, agents can seamlessly be integrated with contin-



(a) (b)

Figure 4.2: (a) Two groups pass, naturally forming lanes. (b) A vortex forms as four groups cross.

uum crowds.

In short, this chapter presents a new *type* of crowd simulator driven by dynamic potential fields that unify global navigation and local collision avoidance into one framework. In particular two very simple ideas will be crucial to creating my results: one is a velocity-dependent term which induces lane formation, and the other is a distance-based term which stabilizes the flow. I also show how individuals can integrate knowledge of the future to produce more intelligent behavior. The computational cost of the algorithm depends on the number of grid cells used to compute the dynamic potential. At coarse discretizations, crowds of over ten thousand people can be simulated at several frames a second, while fine discretizations allow high-resolution dynamics at real-time frame rates even for several thousand people. I also show that my model exhibits important emergent phenomena observed in real crowds. In particular, people walking in opposite directions tend to form lanes, and group crossings can form vortices.

4.1 The Governing Equations

In this section I develop a mathematical model of crowd dynamics. I begin with a set of observations about crowd flow, with each observation culminating in a precise hypothesis. My model is directly derived from these hypotheses. I subsequently discuss the role of potential functions in my model, and present the exact form of the speed equation. I defer

the discretization and simulation of these equations to section 4.2.

The overarching force driving crowd flow is that people have a destination, or *goal*. Goals can be specific such as *go to 1549 35th street*, or general as in *go to the west side of town*. They can also be dynamic such as *chase this person*, *find a non-empty theater seat*, or *explore unseen parts of the environment*. Crowd scenarios without a definable goal such as browsing at the mall, or wandering aimlessly are not appropriate for the continuum crowd formulation. This technique considers goal selection to be an external parameter set by the animator.

HYPOTHESIS 1. Each person is trying to reach a geographic goal $G \subseteq \mathbf{R}^2$.

As people move toward their goal, the next most important consideration is their speed. In general, the method assumes that people move at the maximum speed possible given environmental conditions. For example, a city-dweller might walk at a brisk pace, while a medieval army might charge ahead at a fast run. The environment affects the speed in a number of ways: descending a slope causes people to increase their speed, while physical boundaries are impassible. Most importantly, the presence of other people affects speed. It is assumed that people have difficulty moving against the current which is proportional to the local crowd density. In the extreme case, two people cannot intersect.

HYPOTHESIS 2. People move at the maximum speed possible. This can be expressed as a *maximum speed field* f such that a person at location $\mathbf{x}$ moving in direction θ has velocity:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \theta)\mathbf{n}_\theta, \tag{4.1}$$

where throughout this chapter, $\mathbf{n}_\theta = [\cos\theta, \sin\theta]^T$ will denote the unit vector pointing in direction θ.

Even when people can move unobstructed, they may express preferences for certain paths. Pedestrians, for example, often do not cross a street until they reach a crosswalk. Also, people generally follow trodden paths when they exist, even if they yield longer

routes. I represent this idea as follows.

HYPOTHESIS 3. There exists a *discomfort field* g so that, all things being equal, people would prefer to be at point $\mathbf{x}$ rather than $\mathbf{x}'$ if $g(\mathbf{x}') > g(\mathbf{x})$.

As with goal selection, the discomfort field is exogenous to my model. However, in section 4.1.3, I discuss how dynamic discomfort can be used to enhance collision avoidance between people and other moving obstacles such as cars.

I now tie together the above hypotheses and describe how people choose paths. In general, people choose the minimum distance path to their destination. However, this preference is tempered by a desire to avoid congestion and other time-consuming situations. This can be seen as the classic trade-off between energy and time minimization. Additionally, people prefer to minimize their exposure to areas of high "discomfort." These ideas can be combined by assuming that people choose paths so as to minimize a linear combination of the following three terms:

- The length of the path.
- The amount of time to the destination.
- The discomfort felt, per unit time, along the path.

HYPOTHESIS 4. Let Π be the set of all paths from $\mathbf{x}$ to some point in the goal. Assuming that the speed field f, discomfort g, and goal G are fixed, a person at location $\mathbf{x}$ will pick the path $P \in \Pi$ minimizing

$$\underbrace{\alpha \int_P 1 ds}_{\text{Path Length}} + \underbrace{\beta \int_P 1 dt}_{\text{Time}} + \underbrace{\gamma \int_P g dt}_{\text{Discomfort}} . \tag{4.2}$$

Here α, β, and γ are weights for individual terms; ds means that the integral is taken with respect to path length while dt means the integral is taken with respect to time. These two variables are related by $ds = f dt$ where f is the speed. Using this equality, one may rewrite equation (4.2) as

$$\alpha \int_P 1ds + \beta \int_P \frac{1}{f} ds + \gamma \int_P \frac{g}{f} ds \tag{4.3}$$

which can be simplified to

$$\int_P C ds, \quad \text{where} \quad C \equiv \frac{\alpha f + \beta + \gamma g}{f} \tag{4.4}$$

is the *unit cost field*.

4.1.1 Optimal Path Computation

I now show how potential functions can be used to find optimal paths given the path cost described in equation (4.4). Suppose I have a function $\phi : \mathbf{R}^2 \rightarrow \mathbf{R}$ everywhere equal to the cost of the optimal path to the goal. Intuitively, it makes sense that for any person, the optimal strategy is to move opposite the gradient of this function, as this will decrease cost of the path most rapidly. Conversely, this function can be constructed by following the set of all optimal paths outwards from the goal, integrating cost along the way. Indeed, the *potential function* ϕ can be defined exactly this way: in the goal $\phi = 0$, and everywhere else ϕ satisfies the *eikonal equation*:

$$||\nabla \phi(\mathbf{x})|| = C, \tag{4.5}$$

where the unit cost C is evaluated in the direction of the gradient $\nabla \phi$. A theorem from the calculus of variations (see, for example [69]) guarantees that *all* optimal paths follow exactly the gradient of this function. In my system, this means that every person moves in the direction opposite the gradient, scaled by the speed at that point:

$$\dot{\mathbf{x}} = -f(\mathbf{x}, \theta) \, \frac{\nabla \phi(\mathbf{x})}{||\nabla \phi(\mathbf{x})||}, \tag{4.6}$$

where $\dot{\mathbf{x}}$ denotes the velocity, and $f(\mathbf{x}, \theta)$ is evaluated in the direction of motion.

Calculating a potential field may seem like a cumbersome way of finding an optimal

path. One can, however, make a simplifying assumption that justifies this method. Suppose a group of people all share an identical speed field, discomfort, and goal. This is often the case in crowds when a number of people are trying to get to the same location at approximately the same speed. In this case, one needs to calculate the potential function for the group only once, deriving optimal paths for all group members simultaneously. In reality, of course, people move at different speeds, have varied perceptions of discomfort, and seek different goals. Therefore, my technique divides the crowd into a set of groups, each with different characteristics. At each timestep the algorithm constructs a potential function ϕ for each group and then moves the people in that group according to equation (4.6). Since each group affects the speed f and discomfort g across all groups, the entire crowd motion is coupled.

Solving equation (4.5) is the slowest aspect of simulation. Thus, one should use as few groups as possible. However, as my results show, even one group exhibits interesting dynamics, and realistic crowd congestion phenomena can be attained with few groups.

4.1.2 Speed

The speed field f measures the maximum permissible speed of movement for every point and every direction in the domain. The speed model is ad-hoc, but very simple to compute, and it provides physically plausible behavior for the crowd. I begin with an intuitive description. Speed is a density-dependent variable. At low densities, speed is dominated by the terrain, remaining constant on flat surfaces, but changing with the slope. At higher crowd densities, the speed becomes dominated by the movement of nearby people: movement is inhibited when trying to move against the flow, but is unaffected when moving with the flow. Modeling this latter effect is one important difference between my model and that of Hughes [60]. Moreover, I found that this velocity-dependent term is crucial for modeling "lane formation" where people moving in opposite directions along a path naturally segregate into separate lanes, a phenomenon that has been widely noted in real crowds.

Since speed is density-dependent, I will first describe the *crowd density field* ρ. The algo-

rithm converts each person into an individual density field, denoted ρ_i for the ith person. This field should peak at the location of person i, and fall off radially. The specific form of this function is unimportant as long that it is no less than some threshold value $\bar{\rho}$ within a bounding disc of radius r, and no greater outside. The crowd density ρ is simply the sum of each individual density field. The algorithm simultaneously calculates the *average velocity field* $\bar{\mathbf{u}}$ which scales each person's density by his velocity, indicating the overall speed and direction of crowd flow:

$$\rho = \sum_i \rho_i, \quad \text{and} \quad \bar{\mathbf{u}} = \frac{\sum_i \rho_i \dot{\mathbf{x}}_i}{\rho}. \tag{4.7}$$

Here $\dot{\mathbf{x}}_i$ denotes the velocity of the ith person. Both sums are taken across all people in all groups.

I now describe specifically how density affects the speed. In areas of very low density ($\rho \leq \rho_{min}$ for some ρ_{min}), the speed f is equal to the *topographical speed* f_T. Assuming the terrain is bounded to lie within the minimum and maximum slopes s_{min} and s_{max}, the speed varies inversely with the slope:

$$f_T(\mathbf{x}, \theta) = f_{max} + \left(\frac{\nabla h(\mathbf{x}) \cdot \mathbf{n}_\theta - s_{min}}{s_{max} - s_{min}} \right) (f_{min} - f_{max}), \tag{4.8}$$

where $\nabla h(\mathbf{x}) \cdot \mathbf{n}_\theta$ is the slope of the height field h in direction θ.

In areas of high density ($\rho \geq \rho_{max}$ for some ρ_{max}), the speed f is equal to the *flow speed* $f_{\bar{\mathbf{u}}}$:

$$f_{\bar{\mathbf{u}}}(\mathbf{x}, \theta) = \bar{\mathbf{u}}(\mathbf{x} + r\mathbf{n}_\theta) \cdot \mathbf{n}_\theta. \tag{4.9}$$

The flow speed $f_{\bar{\mathbf{u}}}$ is essentially the average velocity $\bar{\mathbf{u}}$ evaluated at a distance r from the location $\mathbf{x}$. The offset causes people to evaluate the average velocity for the area *into* which they are trying to move. Indeed, if not for the offset, a person's speed would be dominated by their own previous speed, an undesirable effect. Also, the flow speed is clamped to be nonnegative, implying that the crowd can slow people down, but never carry them

backwards.

At medium densities ($\rho_{min} < \rho < \rho_{max}$), I linearly interpolate between the topographical and flow speeds:

$$f(\mathbf{x},\theta) = f_T(\mathbf{x},\theta) + \left(\frac{\rho(\mathbf{x} + r\mathbf{n}_\theta) - \rho_{min}}{\rho_{max} - \rho_{min}}\right)\left(f_{\vec{\mathbf{u}}}(\mathbf{x},\theta) - f_T(\mathbf{x},\theta)\right) . \tag{4.10}$$

Note that, like the flow speed, the density ρ is evaluated at an offset of $r\mathbf{n}_\theta$, again for the same reason: a person's own contribution to the density field should not self-obstruct their motion. Since it was assumed that a person's contribution to the density field ρ is no greater than $\bar{\rho}$ outside the disc of radius r, their speed in low congestion will always be equal to the topographical speed so long as $\rho_{min} \geq \bar{\rho}$.

4.1.3 Models of the Future

The model presented in previous sections can be further improved by accounting for two important aspects of crowd motion that involve predictive models of the future:

Predictive discomfort. Individuals in my crowd simulator account for the future by path planning through a constantly updated static view of the environment. That is, the model takes into account moving obstacles, but not that the obstacles are moving. Often this simplification is justified: real pedestrians avoid those walking in the opposite direction and intelligently plan around areas of high congestion. In this model however, when two people cross perpendicularly, they fail to anticipate one another. I address this by adding a small amount of discomfort in front of each person, causing others to avoid this region. Specifically, the algorithm advances each person's position by their velocity for several timesteps depositing to the discomfort field an amount proportional to the density that would be calculated were the person there. In simulations with many intersecting paths, predictive discomfort yields smoother flow, effectively incorporating the salient aspects of short-horizon dynamic planning, while staying within the framework of fast, per-timestep, 2D planning. In section 4.3, I extend this idea to collision avoidance with other moving objects such as cars.

Expected periodic field changes. A similar issue arises when the field deterministi-

cally changes over time. Consider an environment with two exit doors rapidly opening and closing. An exiting crowd would continuously switch direction back and forth toward the currently open door. Similarly, when a traffic light turns red the speed field must be temporarily set to zero to prevent crossing. Individuals in the basic model would naively conclude that crossing is impossible, and pick another path. Here I offer a different solution. When computing the potential, the technique replaces the actual inverse speed with the expected inverse speed computed over a longer time period. For traffic lights, the expected inverse speed is computed from the percentage of time the traffic light is green during a period. Similar computations are performed for other periodic changes (e.g. opening doors). By the linearity of expectations, the optimal path cost given in equation (4.2) can then be interpreted as the minimizing the *expected* time to the goal.

4.2 Implementation

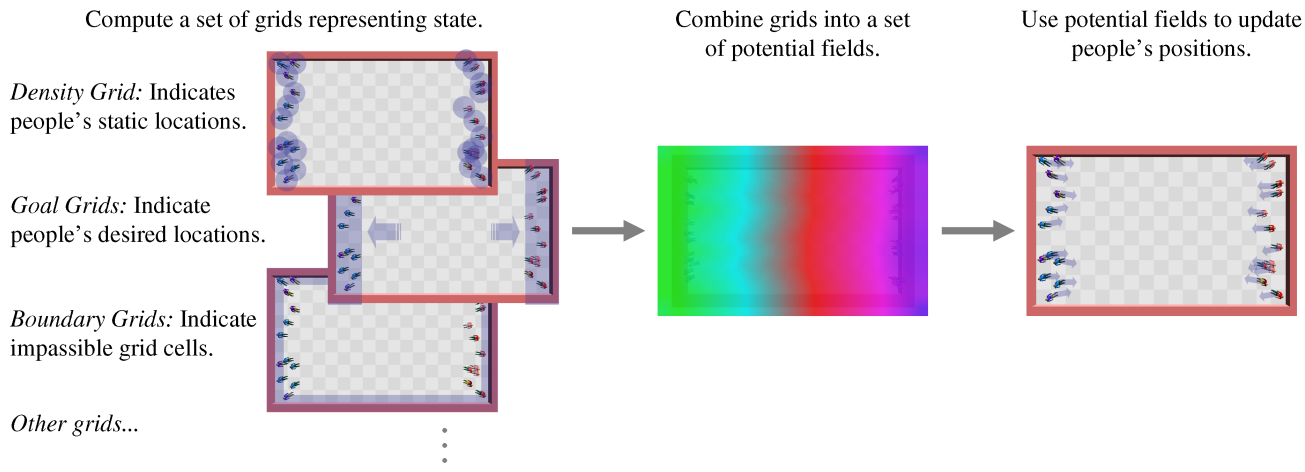


Figure 4.3: General algorithm overview.

To simulate this system, the model described in the last section must be discretized in time and space. The simulator advances through each timestep as follows:

For each timestep:

- Convert the crowd to a density field.

  For each group:

  - Construct the unit cost field C.
  - Construct the potential ϕ and it's gradient $\nabla\phi$.
  - Update the people's locations.

- Enforce the minimum distance between people.

To compute these fields, I discretize space into a regular grid, with physical variables defined at various locations within each grid cell. All physical fields are stored as 2D arrays of floating point numbers according to the schema shown in Figure 4.4(a). All scalar fields are defined at the center of each grid cell. This is also true of the average velocity $\bar{\mathbf{u}}$, which is stored as a pair of floats. All anisotropic fields–those depending on both position and direction–are stored with four floats per cell corresponding to $\theta = \{0°, 90°, 180°, 270°\}$, that is the east, north, west, and south faces of each cell. Finally, the velocity $\mathbf{u}$, the gradient of the height ∇h and of the potential $\nabla\phi$ are stored at the faces of each cell, in a MAC-style arrangement (see [35]). I now look more closely at how every step of the simulator is implemented.

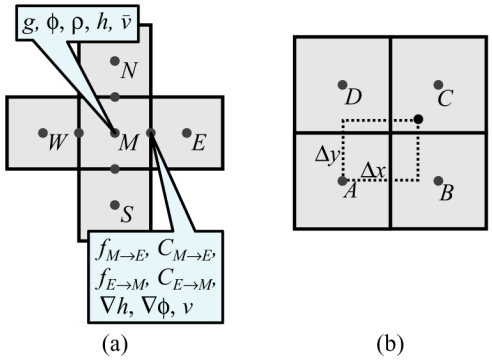


Figure 4.4: Discretized grid structure.

4.2.1 Density Conversion

My technique "splats" the crowd particles onto a density grid in order to compute the speed field, which is density dependent. The density conversion function must satisfy two requirements. First, the density field must be continuous with respect to the location of the people. Otherwise movement would cause sharp discontinuities in the density, and subsequently in the speed. Second, each person should contribute no less than $\bar{\rho}$ to their own grid cell, but no more than $\bar{\rho}$ to any neighboring grid cell. Intuitively, this requirement ensures that each individual is not affected by its own contribution to the density field. This requirement is the discrete analog to the disc of radius r surrounding each person described in section 4.1.2.

The first requirement is standard, and could be satisfied by any number of splatting techniques, including bilinear and Gaussian. The second requirement is unusual, and I have defined a new density conversion technique to satisfy it. For each person, I find the closest cell center whose coordinates are both less than that of the person. I then compute the relative coordinates $[\Delta x, \Delta y]$ of that person with respect to the cell center, as shown in Figure 4.4(b). The person's density is then added to the grid as

$$\begin{aligned} \rho_A &= \min(1-\Delta x, 1-\Delta y)^\lambda & \rho_B &= \min(\Delta x, 1-\Delta y)^\lambda \\ \rho_C &= \min(\Delta x, \Delta y)^\lambda & \rho_D &= \min(1-\Delta x, \Delta y)^\lambda, \end{aligned}$$

where the *density exponent* λ determines the speed of density falloff. This density conversion method is continuous with respect to the location of each person, and is defined so that each person contributes at least $\bar{\rho}$ to their grid cell, but no more than $\bar{\rho}$ to neighboring cells, with $\bar{\rho} = 1/2^\lambda$. Thus, the requirements are satisfied. The density field ρ and average velocity $\bar{\mathbf{u}}$ are computed simultaneously according to equation (4.7).

4.2.2 Unit Cost

There are two steps to computing the unit cost field C. First, compute the speed field f according to equation (4.10), then calculate the cost field C using equation (4.4). These fields are anisotropic so computing them involves not only iterating over each grid cell, but

also iterating over each of the four directions within each cell. For example, for cell M in Figure 4.4(a) one must compute $f_{M\to i}$ and $C_{M\to i}$ for $i \in \{E, N, W, S\}$. So that the pedestrians anticipate obstructions ahead, the algorithm evaluates the speed and discomfort at the cell *into which* the person would be moving if they chose that direction. For example, in the case of $f_{M\to E}$, one would plug the density ρ_E, discomfort g_E, and average velocity $\bar{\mathbf{u}}_E$ into equation (4.10).

4.2.3 Dynamic Potential Field Construction

Constructing the dynamic potential is the most complex and time consuming step of the algorithm. Equation (4.5) defines the potential as an implicit eikonal equation, and hence cannot be calculated directly. However, efficient methods have been developed to solve this type of equation, notably the fast marching method [136] and the fast sweeping method [135]. I have chosen the former, since the latter becomes inefficient when optimal paths must follow circuitous routes (because of obstacles, for example).

The fast marching algorithm is by now well known so I describe my implementation only briefly and refer the reader to [136] for more details. One begins by assigning the potential field ϕ the value of 0 inside the goal, and including these grid cells in the list of KNOWN cells; all other cells are UNKNOWN and set to ∞. Those UNKNOWN cells adjacent to KNOWN cells are included in the list of CANDIDATE cells and the algorithm approximates ϕ at these locations by solving a finite difference approximation to equation (4.5). The CANDIDATE cell with the lowest potential is then included in the KNOWN cells, and its neighbors are introduced into the CANDIDATE set by re-approximating the potential at these cells. This process is repeated, propagating the KNOWN cells outwards from the goal until all cells are defined. A heap data structure efficiently handles the list of candidate grid cells and gives the algorithm its $O(N \log N)$ running time, where N is the number of grid cells.

I now describe the finite difference approximation to equation (4.5). Suppose one were solving the equation for grid cell M in Figure 4.4(a). First, find the less costly adjacent grid cell along the both x- and y-axes:

$$m_x = \underset{i \in \{W,E\}}{\operatorname{argmin}} \{\phi_i + C_{M \to i}\} \qquad m_y = \underset{i \in \{N,S\}}{\operatorname{argmin}} \{\phi_i + C_{M \to i}\}.$$

Then use these upwind directions to calculate a finite difference approximation to equation (4.5) by solving for the larger solution to ϕ_M in the quadratic equation

$$\left(\frac{\phi_M - \phi_{m_x}}{C_{M \to m_x}}\right)^2 + \left(\frac{\phi_M - \phi_{m_y}}{C_{M \to m_y}}\right)^2 = 1. \tag{4.11}$$

If either m_x or m_y is undefined because both neighbors have infinite cost, then drop that dimension out of equation (4.11). Note that, up to the discretization of the angles, this formulation correctly handles the anisotropy of the unit cost field C. Once ϕ_M has been computed, take its difference with the neighboring grid cells in the upwind direction yielding $\nabla\phi$. Finally, renormalize the gradient, and multiply by the speed in the appropriate directions to compute the velocity field $\mathbf{u}$ at that point.

4.2.4 Crowd Advection

Having calculated the potential field ϕ, its gradient $\nabla\phi$, and determined the velocity field $\mathbf{u}$, one can then simply update each person's position by interpolating the velocity field. Each person's position is displaced by their velocity, effectively computing an Euler integration of equation (4.6). I have experimented with higher order Runge-Kutta integrators, but they do not appreciably affect the dynamics.

4.2.5 Minimum Distance Enforcement

In theory, my model ensures that no two people will intersect. Two people approaching one another will eventually experience density so high that the average velocity term dominates the speed equation and their speed toward each other will drop to zero. In practice, however, one can only resolve the dynamics up to the resolution of the grid, and two people in the same grid cell will sometimes intersect. This happens with relative infrequency as people naturally avoid one another in my system. However, when intersections do happen, they cause visually unpleasant artifacts.

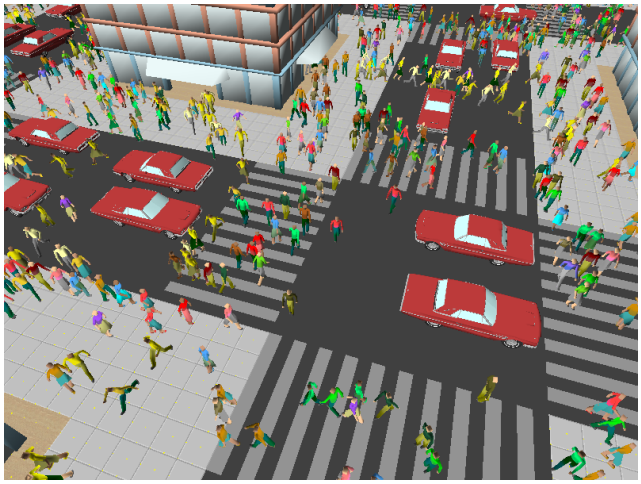


Figure 4.5: People in yellow shirts evacuate a building.

To address this, a pair-wise minimum distance is enforced between the people: simply iterate over all pairs within a threshold distance, symmetrically pushing them apart so that the minimum distance is enforced. This procedure does not strictly ensure that minimum distances are preserved, as modifying the position of one person changes that person's distance to all other people. Also, minimum distance enforcement occasionally introduces artifacts, such as when non-moving people are pushed by others. However, the vast majority of the time I found this method useful for eliminating intersection artifacts which could not be resolved by the grid. Note that minimum distance enforcement has linear time complexity if the crowd is first "binned" into a high-resolution neighbor grid.

4.3 Results

I have run a diverse set of crowd simulations with my system and found that the continuum method can produce smooth behavior for thousands of agents at interactive rates. I encourage the reader to view the animations in the companion video.[2] My implementation was in a mixture of Python and C++. All simulations ran on a 3.4GHz Pentium with a NVidia Quadro FX 3400 graphics card. Simulation updates took between 2 and 5 frames per second (fps). This running time includes a parallel rendering thread which calculated

[2]http://grail.cs.washington.edu/projects/crowd-flows/

"in-between" frames at 24 fps (12 fps for my 10,000 person example). Because my simulations usually exhibited smooth motion, the decoupling of the renderer from simulator was not apparent, yielding real-time, interactive simulations. The most noticeable motion artifacts in my simulations were due to the simplified human animation. I used basic locomotion cycles with precomputed in-between poses to avoid popping artifacts between cycles. A more sophisticated off-line motion model would produce higher quality results, but would preclude interactive animation.

Figure 4.2(a) shows a simulation of twenty-four people crossing in a hallway and forming lanes to avoid collision. This phenomenon is ubiquitous in real crowds, and it can be observed in all of my simulations. In Figure 4.2(b), I show a simulation of four groups crossing one another to reach opposite corners of the domain. A vortex forms as the groups cross paths. This emergent behavior is an exotic form of lane formation, and it also serves to prevent collisions. Such vortices have been observed in real crowds [54]. In these simulations, the checkered pattern on the floor indicates the grid size.

In comparisons with local, agent-based models one can see people not planning early for congestion, and thus not avoiding it until they are near. With continuum crowds, people plan to avoid congestion in advance and the overall movement assumes a smoother, more optimal flow. Note that it is certainly possible for local models to display vortex-like crowd flow as reported by Heïgeas, but such emergent features occur only for specific parameters of the model and only after the simulation has reached a steady state. It is also possible to mix continuum crowds with agents, as shown in Figure 4.7 where a continuum crowd interacts with Reynolds agents.

One of the features of my system is that very large scale simulations are possible on coarse grids. To demonstrate this, I simulated a 2000 person army retreating from an 8001 person army on a 60×60 grid (Figure 4.8). Despite the coarse resolution, interesting effects are apparent in which people change paths as congestion conditions evolve across the mountainous terrain. This simulation ran at 5 fps before rendering "in-between" frames.

I also simulated a school quad as classes get out. People exit from eight doorways and are randomly assigned a different destination door. This simulation was challenging because people crossed paths at many different angles. For this reason, I added ten timesteps

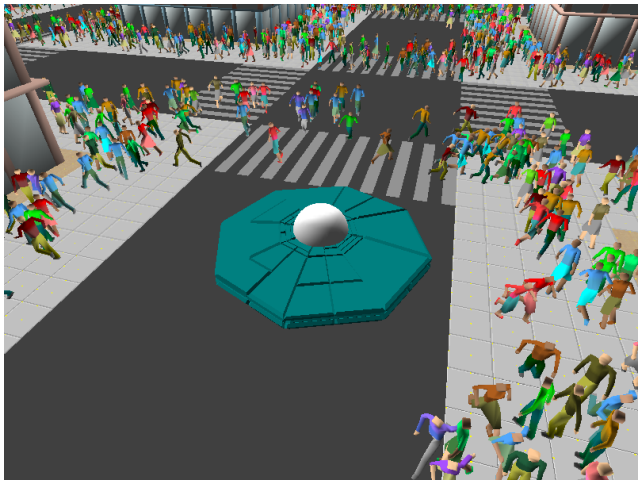


Figure 4.6: A continuum crowd reacts to a user-driven flying saucer agent in real-time.

of predictive discomfort, and the result is a smooth flow, even under the rapidly changing conditions.

Finally, I created a 16 square block city environment with crosswalks, streets, and sidewalks (Figures 4.1, 4.5, and 4.6). My simulations contained between one and two thousand people on this 120×120 grid. By adding discomfort to the streets, people preferred to stay on the crosswalks, crossing the streets only when congestion was too great. I was also able to simulate the 9 traffic lights by setting the speed onto the street to zero when the light is red. Since the expected value of the traffic light turning green is included in the

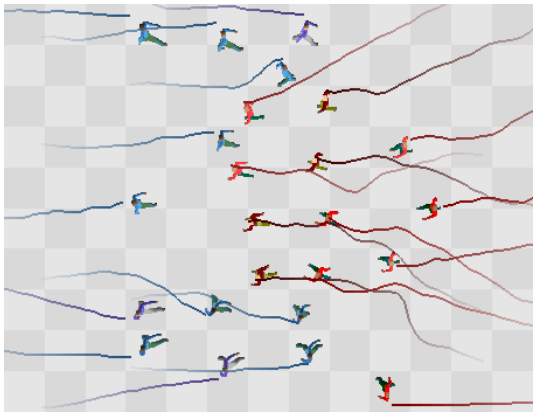


Figure 4.7: A continuum crowds group (blue) interacts with a group of Reynolds agents (red).

Figure 4.8: A large army chases a smaller one through mountainous terrain in real-time.

planning, pedestrians do not opt to go elsewhere once the light turns red. I also simulated a building evacuation (Figure 4.5), a scenario which is of interest for urban planning, and also serves to highlight the interesting congestion phenomena that occur when groups travel at different speeds. Finally, I created two simulations in which people predictively reacted to moving obstacles. I projected discomfort and increased speed in front of the obstacles so that people would move quickly out of the way. In my first example, cars are driving slowly through crowded streets. I consequently added only small amounts of discomfort and speed. In my second example (Figure 4.6), a user interactively flew a flying saucer agent through the simulation at much higher speeds. I correspondingly increased the amount and size of the region of increased speed and discomfort. Despite the 5 fps simulations rate, the pedestrians correctly avoided the flying saucer, and the 24 fps renderer provided smooth motion. The grid resolution is equal to the size of the sidewalk tiles shown in Figure 4.6.

4.4 Parameter Settings

To create a crowd simulation using this model, the animator must decide a number of issues, including the simulation terrain, how to inject and remove crowd members, and how to create "special effects" such as traffic lights and moving cars. Most importantly,

Table 4.1: Parameter settings for the crowd results. The left-most column lists simulation names, while the rest of the columns give parameter values. "*Lanes*" and "*Vortex*" are the linear and vortical lane formation examples. "*Armies*" is the result where one army chases another. "*Quad*" is the school quad animation, and "*City*" is the city animation. Space is measured in units of grid cell length, while time is measured in timesteps. Section 4.4 gives advice on choosing parameter values.

Info	**Speed**	**Density / Compression**				**Cost Coefficients**		
Name	f_{max}	ρ_{min}	ρ_{max}	δ_{min}	λ	α	β	γ
Lanes	0.120	0.001	0.056	0.500	10.000	0.000	1.000	0.000
Vortex	0.113	0.067	0.326	0.500	3.900	0.200	0.600	0.200
Armies	0.150	14.755	34.491	0.150	1.225	0.023	0.068	0.909
Quad	0.375	0.063	0.316	0.850	4.000	0.000	0.500	0.500
City	0.150	0.250	0.563	0.300	2.000	0.164	0.016	0.820

at least eight simulation parameters must be specified which determine effects such as how fast people move and how tightly packed the crowd will be. These parameters are summarized for the simulations we present in Table 4.1. Although eight may seem like a large number, in practice the animator can usually iterate toward good parameter settings fairly quickly. This section describes how to do so.

The parameters may be divided into three groups which control the speed, density, and types of paths which crowd members will choose. The animator should start by selecting the maximum speed f_{max} at which people move when unobstructed (equation 4.10). The units of speed are in grid cells per timestep, so faster speeds will make people move more quickly, but f_{max} must also increase when grid cells are small relative to human size.

After determining the speed, the animator should set the crowd compression parameters based on the desired crowd density. There are several aspects to crowd density. The average velocity exponent λ determines the shape of the density splat around each person (section 4.2.1). Higher values of λ create tightly concentrated densities so that people are more likely to brush past one another as they pass. For very high values of λ, the simulation can start to oscillate because people's movements will cause near-discrete changes in the density field, drastically effecting the value function. Generally, the animator should begin with $\lambda = 3$. When people become very tightly packed without moving, such as when waiting for traffic lights, the absolute minimum distance δ_{min} dominates the com-

putation (section 4.2.5). This number should be set as low as possible, while preventing two crowd members from actually intersecting, a highly undesirable visual artifact. The minimum and maximum densities, ρ_{min} and ρ_{max}, represent the density levels at which the flow speed $f_{\vec{u}}$ starts to dominate and people must move with the crowd (equation 4.9). Lowering these values will cause fewer, larger lanes to form, while raising them will cause more tight lanes to form. I have found $\rho_{min} = \left(\frac{1}{2}\right)^{\lambda}$ and $\rho_{max} = \left(\frac{3}{4}\right)^{\lambda}$ to be good initial values. After setting ρ_{min}, ρ_{max}, δ_{min}, and λ, the high-level flow characteristics of the crowds should be set.

The parameters α, β, γ control weights for the path length, time, and discomfort, respectively (equation 4.2). I have found that the best strategy is to start with $\alpha = \gamma = 0$ and $\beta = 1$. Then, gradually increase the discomfort coefficient γ until discomfort fields such as sidewalks are properly handled. Finally increase the path length coefficient α just enough so that any instabilities in the flow are removed. A simulation with $\alpha = 0$ will almost certainly have instabilities, while setting α too high means that people will move towards their destination along the shortest path regardless of the presence of other people.

Depending on the simulation, there may be other settings as well. For example, in the "*Armies*" simulation, the minimum and maximum slope had to be specified: $s_{min} = -0.3$ and $s_{max} = 0.3$. For the school "*Quad*" example, 15 timesteps of predictive discomfort were added in front of every crowd member with a density exponent of 1 and a scale factor of 0.1 times the density. In general, iterating over the above parameters will allow the animator to converge on crowd dynamics with appropriate properties relatively quickly.

4.5 Discussion and Future Work

This chapter presented a novel crowd simulation framework based on a continuum perspective rather than per-agent dynamics. My approach further develops and the abstract model proposed by Hughes [60]. Several of my enhancements to Hughes's model produce more realistic crowd behavior. Most notably, I changed the continuous density field into a particle description of the crowd and developed a full set of continuous dynamics for this representation. I added a velocity-dependent term to the speed computation

which is crucial for reproducing emergent phenomena that have been widely noted in real crowds. I also added a distance term to the optimal path computation which is psychologically plausible, and reduced the oscillations in Hughes's model. In addition, I described an algorithm for the efficient simulation of my continuum model. Finally, I showed how my model could be adapted to several interesting situations including traffic lights and predictive avoidance of moving objects.

My model has numerous advantages over previous systems in graphics. First, it unifies global planning and collision avoidance. This means that individuals in my simulations do not face conflicting requirements between local collision avoidance and global planning, and they exhibit smoother motion than has been previously reported. This brings my results closer to real crowds, which are characterized by surprisingly few sudden changes in direction. This has also allowed my system to capture a number of emergent phenomena that have been reported in the crowd literature, including lane formation and short lived vortices during turbulent congestion. Decoupling the simulator from the renderer allowed these simulations to run in real-time.

It is possible to integrate my model with agent models. In the results presented, the moving cars (driven by a simple rule) and the flying saucer (driven interactively) are all agents. The demo video[3] and Figure 4.7 show my model interacting with traditional Reynolds agents [109] exhibiting "unaligned collision avoidance" behavior.

Like all models, my continuum approach makes simplifying assumptions. Most notably, I do not take into account visual occlusions or any other form of uncertainty, effectively assuming that people really know the dynamic properties of the environment. While this assumption may be reasonable for daily urban commuting, limited vision certainly influences real crowds in unknown environments. In those cases, global knowledge planning is an approximation, in a similar way that local proximity knowledge is an approximation. I believe that global knowledge approximation is justifiable in the cases where my model most directly applies: large crowds navigating open and familiar areas. More importantly, compared to alternative approaches and other approximations, the use

[3]The demo video can be seen on the project website: http://grail.cs.washington.edu/projects/crowd-flows/

of global knowledge provides smoother crowd behavior and allows everyone to plan at once, achieving high simulation speeds.

I also assume that people can change direction without respect to inertia. While this is justified for walking crowds, it becomes unrealistic when people run. This could be addressed by giving each person a state consisting of position and velocity, and solving a 4D eikonal equation through this state space; however this method would not be real-time. Another useful extension would be to solve for the potential function across nonuniform grids. This would allow sparse data points in areas with few people, and finer discretizations in areas of congestion, which might yield a substantial speedup.

My system does not have the flexibility or individual variability of the full agent-based approach. Continuum crowds are designed specifically to model multiple large homogeneous groups of people who are moving in order to reach a specific goals. Since very large crowds are observed at a distance where it is harder to notice details of individual behavior, I feel that smooth movement, lane formation and planned motion are more important features than individual variability. At the same time, occasional variability can be achieved by inserting a number of agents within a continuum crowd. The continuum crowd model treats agents as moving obstacles and naturally handles the unpredictability of their behavior. I suspect that this hybrid model of continuum crowds with embedded agents may be the most natural use of continuum crows in larger interactive frameworks including games.

The continuum crowd model is not appropriate for all crowd behavior. For example, it does not take into account the regime where people are so tightly packed that contact forces between them dominate the physics. It is also limited by the requirement that people move with a common goal. At the same time, the continuum framework can be applied to wide range of scenarios that at first do not seem to meet the common goal hypothesis. These scenarios are achieved by dynamically changing goals and discomfort fields. For example, environment exploration can be modeled by setting goals to collectively unseen regions. I have similarly produced examples of “chasing” behavior, where goals are set to the location of target people, and avoidance behavior, where goals are set to a large-radius circle around people that should be avoided. The unassigned-seat theater-filling example

can also be modeled by removing each “seat goal” as they get occupied.

Chapter 5

HUMAN ANIMATION

Despite decades of advances, interactive character animation still lacks the fluidity and variability of real human motion. Capturing the fine nuances of human motion requires a high-dimensional motion repertoire and a multivariate space of input parameters which can change continuously, often unpredictably. For example, a human might fluidly move to avoid unexpected moving obstacles while maintaining a direction and walking next to another person. Unfortunately, algorithms for solving such problems remain highly sensitive to the size of the state space and number of concurrent goals.

This chapter looks at these issues through the lens of a difficult and important subproblem in character animation: synthesis of a kinematic controller that blends precaptured motion clips to achieve complex multi-objectives in realtime. In particular, I assume continuously changing environmental conditions and several dimensions of user control over the character's motion. For example, a virtual sports application might require that the character's speed, direction, and torso orientation all be independently controllable. Tasks like obstacle avoidance introduce additional control variables. For simple navigational control, interactive applications have typically used precomputed motion cycles and carefully constructed graphs that blend between these cycles. However building appropriate blend spaces and transitions requires a great amount of skill and manual adjustment. For complex objectives, researchers have also studied learning algorithms that automatically determine optimal sequences of actions. The primary challenges for both manual and learning approaches are high-dimensionality and the need for both rapid and natural changes in motion.

My main result is that *low-dimensional reinforcement learning* presents an attractive unifying approach to these problems. This approach outperforms greedy methods for navi-

[1]The work described in this chapter was originally presented as a paper [132] at SIGGRAPH 2007.

gational problems while retaining the low memory overhead. At the same time, despite the compact representation, my method can still solve difficult problems such as obstacle avoidance.

My technique has two components. First, an underlying motion model stitches together captured motion clips in realtime. Second, a controller selects sequences of clips to achieve some goal. The controller is based on a compact, parametric *value function* over all possible states and goals. This approach has the following advantages: given a corpus of motion capture data and a set of controller objectives, the system automatically produces a highly compact realtime controller that blends motion near-optimally to achieve user goals. There is no need to explicitly construct walk cycles or otherwise edit the underlying data. My technique is also expandable, in that the user can construct a set of controllers separately for different tasks, then transition between these controllers near-optimally, even if they were learned on different underlying sets of motion. This greatly reduces the cost of constructing complex controllers, as each individual task can be optimized separately before connecting the controllers together. A similar idea parallelizes precomputation along dimensions that do not change over time. Results are shown for a number tasks including navigational control of a character and moving obstacle avoidance. The latter can create crowd simulations in which each character fits a local linear model to the movement of near characters, then avoids those characters in realtime without an explicit collision avoidance algorithm.

5.1 Motion Model

My approach has two main components: a motion engine blends through captured motion clips to produce realtime human animation, while a control policy determines the best sequence of clips to achieve some multivariate control objective. I begin by describing the motion engine.

My model generates poses in realtime by blending sequences of prerecorded motion clips. Unlike standard motion graphs [72, 4, 78], my technique allows transitions between

any two clips, and automatically prevents foot-skate for a large class of motion. My model assumes that a set $\mathcal{C}$ of motion clips has been captured, where each clip $C \in \mathcal{C}$ consists of a sequence of poses: $C = (\mathbf{p}_1, \ldots, \mathbf{p}_m)$. Each pose $\mathbf{p} \in \mathbb{R}^n$ is a vector specifying all joint positions in a kinematic skeleton. I further assume that each clip C covers a single walk cycle and is divided into two subsequences C_{in} and C_{out}. The subsequences start and end during flight or mid-stance, and each covers one foot plant. Each subsequence must have a *constraint frame* that occurs during the middle of the ground contact phase (Figure 5.1).

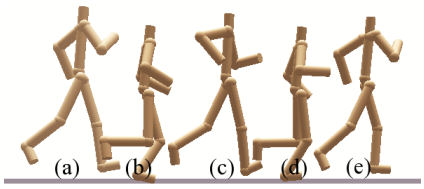


Figure 5.1: A clip. The first subsequence C_{in} runs from (a) to (c), while the second C_{out} comprises (c) through (e). The constraint frames are (b) and (d).

To create longer motions, I partially overlap successive clips and blend between them. The constraint frames prevent foot-skate during the foot plant without inverse kinematics. Two realizations make this possible. First, when blending from C to C', one may mirror or arbitrarily reorient the root of C' while preserving continuity. Second, kinematic blending is linear in the root of the skeleton (although nonlinear in all other skeletal nodes). Therefore, if the technique properly orients the foot and "re-roots" the skeleton at its foot, then it will satisfy a foot position constraint. Blending between clips C and C' is a four step process. First, mirror clip C' (if necessary) to blend along the same foot. Second, overlap the constraint frames from C_{out} and C'_{in} (Figure 5.2). Third, reorient clip C' is so that its ground-contact foot coincides with that of C at the constraint frame. Finally, blend from C_{out} to C'_{in}, *with the ground contact foot treated as the root of the kinematic skeleton.* By the linearity of blending at the root, if the foot is fixed during interval $[t_a, t_b]$ of C_{out} and during $[t'_a, t'_b]$ of C'_{in}, then there cannot be foot-skate on the interval $[t_a, t_b] \cap [t'_a, t'_b]$ of the blended animation.

In summary, in my model, any subsequence of clips produces a valid animation. Because my control algorithm avoids nonsmooth blends, ensuring good transitions does not

require an explicit graph structure, yet the model's high branching factor allows for quick motion changes. Another advantage of this model is that maximum blending occurs during ground contact when ground forces can account for changes in direction; minimal blending occurs during the flight phase, better preserving the linear and angular momentum of correct motion. The major drawback is that the system can enforce only one constraint per blend, thus preventing foot-skate for only one foot at a time. When two feet are on the ground, one foot might slide unless all double-stance clips have the feet spaced at a fixed distance. Nonetheless, I believe this model is highly useful for realtime foot-skate prevention for the wide class of motions which do not have double stance, such as walking and running.

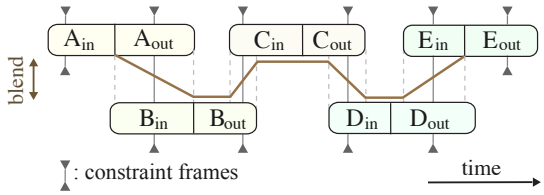


Figure 5.2: Clip blending. Blending the sequence $A, B, C, D, E \subseteq \mathcal{C}$. Constraint frames (shown as solid vertical bars) are aligned across successive clips. Blending occurs where two clips overlap.

5.2 Control

The motion model described above creates a valid motion for any sequence of clips. The controller must decide *which* sequence best achieves it goals quickly and naturally. Before explaining the control algorithm itself, I describe the state space $\mathcal{S}$ on which the controller is defined.

Since my model blends through sequences of clips, it would seem natural to define the state $X \in \mathcal{S}$ as the current clip. Unfortunately, this representation is insufficient for many tasks. For example, if the character is to walk along a line, the state must keep track of the character's orientation relative to this line. Other tasks may require other state variables. Therefore, although my control formulation is general, the specific state representation depends on the user's objectives for the controller. To create my results, the system learned four tasks:

- **Navigation.** The clips were divided into several gaits, including standing, walking, and running. The user controls the gait, linear motion path, and torso orientation.
- **Spinning Navigation.** The user controls the motion direction as the character spins in circles (figure 1, left).
- **Fixed Obstacle Avoidance.** The character follows a line, avoiding a fixed planar obstacle (figure 1, right).
- **Moving Obstacle Avoidance.** The character follows a direction, avoiding a planar obstacle with linear motion.

Although these objectives are learned independently, I consider them jointly in the state:

$$X = \left(C, x, z, \theta, u, v, \dot{u}, \dot{v}, \hat{G}, \hat{\tau}, \hat{\omega}\right). \tag{5.1}$$

Here, $C \in \mathcal{C}$ is the current clip, (x, z, θ) is the character's position and orientation on the x-z plane, (u, v) is the relative position of the obstacle, and $(\dot{u}, \dot{v})$ is the obstacle's speed (Figure 5.3). Finally $\hat{G}$, $\hat{\tau}$, and $\hat{\omega}$ are the character's *desired* gait, torso orientation, and spin, respectively. The latter three variables represent intentions, and only change if the user directly edits them (for example, by pushing the button to change the desired gait speed). Nonetheless, it will be useful to consider these part of the state.

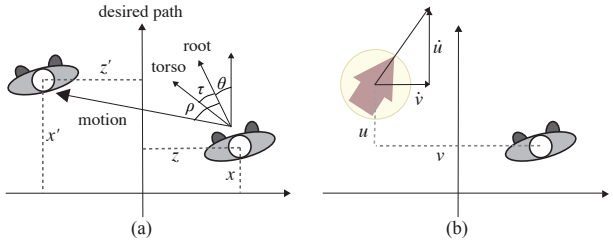


Figure 5.3: State variables for a digital character. Root orientation θ is measured relative to the desired path (along the positive x-axis). The torso orientation τ and actual motion direction ρ are measured relative to the root.

For every pair of clips, the blending algorithm (section 5.1) determines the length of the blend Δt, as well as the linear and angular change $(\Delta x, \Delta y, \Delta\theta)$ in the character's position.

Suppose the character transitions from clip C to C'. Then the state changes from $X = (C, \cdots)$ to $X' = (C', \cdots)$ according to the *transition function* $f : \mathcal{S} \times \mathcal{C} \rightarrow \mathcal{S}$ as follows:

$$f(X, C') = X' = \begin{pmatrix} C' \\ x' \\ z' \\ \theta' \\ u' \\ v' \\ \dot{u}' \\ \dot{v}' \\ \hat{G}' \\ \hat{\tau}' \\ \hat{\omega}' \end{pmatrix} = \begin{pmatrix} C' \\ x + \cos(\theta)\Delta x - \sin(\theta)\Delta z \\ z + \sin(\theta)\Delta x + \cos(\theta)\Delta z \\ \theta + \Delta\theta \\ u + \dot{u}\Delta t - \cos(\theta)\Delta x + \sin(\theta)\Delta z \\ v + \dot{v}\Delta t - \sin(\theta)\Delta x - \cos(\theta)\Delta z \\ \dot{u} \\ \dot{v} \\ \hat{G} \\ \hat{\tau} \\ \hat{\omega} \end{pmatrix}. \tag{5.2}$$

5.2.1 Cost

The user expresses goals by assigning a cost to each state and transition

$$\begin{aligned} c_s &: \mathcal{S} \rightarrow \mathbb{R} \quad : \text{ State cost} \\ c_t &: \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R} \quad : \text{ Transition cost} \end{aligned} \tag{5.3}$$

which are inversely proportional to how well these fulfill the goal. This framework is very broad and can handle a great number of objectives. I now describe the costs used specifically for my four tasks. Again, although these tasks were learned independently (Table 5.1), let us consider them jointly for now.

The gaits partition the clip set $\mathcal{C}$ into different kinds of motion. For the navigation task, I treat the desired gait $\hat{G}$ as a hard constraint so that if $X = (C, \cdots)$ but $C \notin \hat{G}$, then $c_s(X) = \infty$. Otherwise, the system ignores the gait and orients the coordinate system so that the desired path coincides with x-axis. The technique penalizes deviation from this path and proximity to the obstacle:

$$c_s(X) = \underbrace{\gamma_x|x|}_{\textit{deviation}} + \underbrace{\gamma_O \exp\Big(-(u^2+v^2)\sigma_O^{-2}\Big)}_{\textit{obstacle}}. \tag{5.4}$$

Here, γ_x and γ_O are scaling parameters, and σ_O controls the width of the obstacle. The transition cost c_t ensures smooth character motion in the right direction and with the correct torso orientation and spin:

$$c_t(X,X') = \underbrace{\gamma_\Psi \Psi(C,C')}_{\text{Physics.}} + \underbrace{\gamma_\rho|\theta+\rho|}_{\text{Direction.}} + \underbrace{\gamma_\tau|\tau-\hat{\tau}|}_{\text{Torso.}} + \underbrace{\gamma_\omega|\omega-\hat{\omega}|}_{\text{Spinning.}}. \tag{5.5}$$

Again, γ_Ψ, γ_ρ, γ_τ, and γ_ω are scaling constants. The physics cost $\Psi : \mathcal{C} \times \mathcal{C} \to \mathbf{R}$ measures the "physical error" in blending from C to C' as a weighted sum of squared differences of the position and velocity across all joints. The motion direction ρ is measured relative to the root (Figure 5.3). Finally, the average of torso direction τ and torso spin ω are compared to the *desired* torso angle and spin, $\hat{\tau}$ and $\hat{\omega}$, respectively.

5.3 Policies

Having cast the goals in terms of numerical costs, I now look at strategies for achieving these goals. I formalize this idea as a *policy* $\Pi : \mathcal{S} \to \mathcal{C}$, a function that picks the next clip based on the current state. The most obvious policy is greedy:

$$\Pi_{\text{greedy}}(X) = \underset{C' \in \mathcal{C}}{\text{argmin}} \left\{ c_t(X,X') + c_s(X') \right\}, \tag{5.6}$$

where X' is given by the transition function $X' = f(X,C')$ as in equation (5.2). This policy simply chooses the clip that directly minimizes the state and transition costs. In most cases this policy does not produce good results because motion requires *planning*: sacrificing short-term objectives to achieve more efficient results in the long run. For example, when told to turn around, a digital character with planning will step so as to produce a sharper, more realistic turn than under greedy control (Figure 5.4).

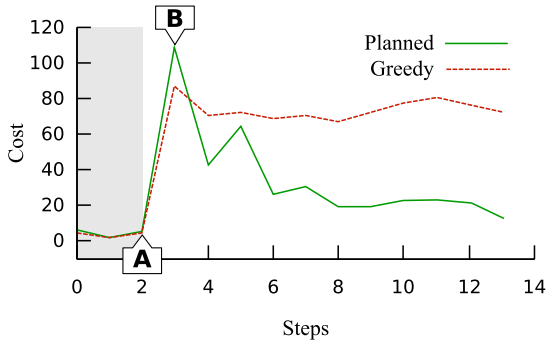


Figure 5.4: Planning vs. greedy controllers. This graph compares the cost $c_s + c_t$ of a greedy navigational controller (equation (5.6)) with my planning controller. The characters act almost identically until the user reverses the intended motion direction (A). The planning controller then chooses a high cost step (B) to enable a sharp and realistic turn one step later.

5.3.1 *The Optimal Policy*

Instead of minimizing the immediate cost, I will minimize the entire policy cost, taking into account the future. Suppose policy Π produces the sequence $(X_0, X_1, X_2, \ldots)$ with $X_t = f(X_{t-1}, \Pi(X_{t-1}))$. I measure the cost $c_\Pi(X_0)$ of this sequence as:

$$c_\Pi(X_0) = \sum_{t=0}^{\infty} \alpha^t \left[c_s(X_t) + c_t(X_t, X_{t+1}) \right]. \tag{5.7}$$

The discount factor $\alpha \in [0,1)$ accounts for future uncertainty and conveniently also ensures that the series converges. However, the main reason for this functional form is that it can be rewritten recursively:

$$c_\Pi(X_t) = c_s(X_t) + c_t(X_t, X_{t+1}) + \alpha c_\Pi(X_{t+1}). \tag{5.8}$$

Under general conditions, minimizing equation (5.7) for each initial state results in the optimal policy $\Pi_\star$ [12]. This allows us to define a *value function* $V : \mathcal{S} \to \mathbb{R}$ that measures the long-term state cost under the optimal policy: $V(X) = c_{\Pi_\star}(X)$. Thus for all X one can rewrite (5.8) as:

$$V(X) = c_s(X) + c_t(X, X') + \alpha V(X'). \tag{5.9}$$

where $X' = f(X, C')$ and $C' = \Pi_\star(X)$ is the optimal next clip. An ingenious and well known reversal of logic turns this equation into a formula for the optimal policy. If equation (5.9) truly reflects the optimal long-term cost, then the best policy must be to minimize the right hand side of this expression:

$$\Pi_\star(X) = \underset{C \in \mathcal{C}}{\operatorname{argmin}} \left\{ c_t(X, X') + \alpha V(X') \right\}. \tag{5.10}$$

Thus, the value $V(X)$ completely specifies the optimal controller.

5.3.2 *Near-optimal Policy*

One generally cannot compute the exact value for continuous state spaces, and therefore one must calculate near-optimal policies based on some approximation of the value function. Sample-based approaches work well, but are efficient neither in learning time nor in controller memory costs. I instead adapt a linear programming approach used primarily in operations research [29]. The idea is to define *basis* $\Phi = [\phi_1, \cdots, \phi_n]$ for the value function where each basis function $\phi_i : \mathcal{S} \to \mathbb{R}$ can be evaluated in closed form, such as polynomials or Gaussians. I then approximate the value function by a linear combination of these basis functions:

$$V \approx r_1\phi_1 + \cdots r_n\phi_n = \Phi\mathbf{r}. \tag{5.11}$$

Thus, I have reduced the problem of solving for the complete value function to the much simpler (and lower dimensional) problem of solving for the n-dimensional vector $\mathbf{r}$ to approximate it.

To derive this approximation, draw a set of state samples $\bar{\mathcal{S}} \subset \mathcal{S}$ at which the value function will be evaluated. Also, consider a set $\mathcal{L} \subseteq \bar{\mathcal{S}} \times \mathcal{S}$ of state transition pairs, each starting at a sample point. The algorithm starts with $\mathcal{L} \leftarrow \{\}$ and $\mathbf{r} \leftarrow \mathbf{0}$, and iterates toward a better value of $\mathbf{r}$ by alternating between the following two steps until convergence.

1. Compile a set of optimal actions according to the current V from every sample in $\bar{\mathcal{S}}$

and add them as constraints:

$$\mathcal{L} \leftarrow \mathcal{L} \cup \{(X, X') \mid X \in \bar{\mathcal{S}}\}$$

where X' is the optimal next state from X given the value function approximation $V \approx \Phi\mathbf{r}$.

2. Find V by solving the linear program

$$\max_{\mathbf{r}} \sum_{X \in \mathcal{S}} V(X)$$
$$\text{s.t.} \quad V(X) \leq c_s(X) + c_t(X, X') + \alpha V(X') \;\; \forall (X, X') \in \mathcal{L}.$$

Step 2 essentially inflates the value function as much as possible subject to the bounds. These inequality constraints generalize (5.9) to the case where the transition $(X, X') \in \mathcal{L}$ may not be optimal. This technique has proved very successful. The convex optimization step of the algorithm yields *globally* optimal approximations of the value function, up to the representative power of the basis and constraint sampling. Moreover, one can discard all samples after optimization, retaining only the highly compact basis coefficient vector $\mathbf{r}$.

5.3.3 Runtime Control

The runtime control algorithm is extremely fast and simple: every time a clip finishes, the controller scans through all clips, picking the minimum value transition as in equation (5.10). User inputs such as changing the desired gait, torso orientation, or motion path alter the underlying state variables in equation (5.1), and thus affect the next state transition.

5.3.4 Dimensionality

Despite the advantages of approximate dynamic programming, high dimensional control remains challenging. Basis approximations use far fewer variables than sample-based approaches, but the number of bases must generally still grow exponentially with the size of

the state. I mitigate this problem by employing two related but subtly different concepts: *switchability* and *separability*.

Both techniques exploit the property that some quantities remain constant through clip transitions. For example, equation (5.2) tells us that the desired gait $\hat{G}$ does not change. Somewhat more subtly, the expression $x + u$ (the distance between the object and the desired path along the x-axis) also remains unchanged, but *only* in the case of a stationary obstacle. Since the controller cannot affect these quantities, the system does not need to couple them during optimization. Instead, my technique tabulates the value function along these dimensions and learns them separately. The difference between switchability and separability is in how the value function is recombined afterwards.

• **Switchability** simply means switching between the tabulated value functions. This is well suited to discrete dimensions such as the desired gait $\hat{G}$. One can even produce near-optimal transitions across value functions that were optimized with different parameters or used different clips. This is because equation (5.10) holds even if state X is drawn from a different value function than X'. For example, when transitioning from a walking gait to a running gait, the optimal policy is still to pick the running clip with lowest value. This feature enables the user to optimize the parameters of several controllers independently, and automatically combine them with near-optimal transitions.

• **Separability** means blending the value function, producing a continuous value along the separable dimension. For example, one can learn separate value functions for various fixed obstacle locations and blend between these for intermediate obstacle positions. Separability can be interpreted as learning a single value function with linear interpolation (or "hat") bases along the separable dimension. The space of value functions is particularly well suited for blending for the same reason that the value functions can be represented with a small number of bases: the value functions for character animation tend to be smooth.

Switchability and separability do not fundamentally solve the dimensionality problem inherent in reinforcement learning. Nonetheless, they allow for some degree of sub-exponential memory consumption (in serial implementations) or sub-exponential time usage (in parallel implementations). Intrinsically dependent dimensions still require expo-

Table 5.1: Motion tasks. Columns 2-4: Active state variables. Columns 5-10: Nonzero cost terms are indicated by bullets.

Task	**Active State Variables**			**Cost Terms**					
	Switchable	Separable	Dependent	γ_x	γ_O	γ_Ψ	γ_ρ	γ_τ	γ_ω
N	$\hat{G}$	$\hat{\tau}$	C, x, θ	•	-	•	•	•	-
SN	-	-	C, θ	-	-	•	•	-	•
FOA	-	$x + u$	C, u, v, θ	•	•	•	•	-	-
MOA	-	$\dot{u}, \dot{v}$	C, u, v, θ	-	•	•	•	-	-

nential resources, however.

5.4 Results

To produce my results, I learned four controllers (section 5.2) to which I give the following symbolic names: navigation (N), spinning navigation (SN), fixed obstacle avoidance (FOA), and moving obstacle avoidance (MOA).

Table 5.1 summarizes the controllers. Columns 2-4 indicate the active variables for each task. Variables that are neither switchable nor separable are *dependent* and must be coupled during value function optimization. Note that for FOA, the distance between the obstacle and the desired motion line $x + u$ is fixed. Therefore, I treat this expression as separable. For moving obstacles, this expression is not fixed and cannot be treated as separable. In fact, to save memory in MOA, I ignore deviation from the desired path and measure only the character's angular direction.

The current clip C is both discrete and dependent. Therefore, I tabulate the value function along this dimension, but optimize for all clips at once. For these results I used 384 clips, manually partitioned into gaits by speed: 18 standing clips, 87 slow walks, 124 normal walks, 65 fast walks, and 90 runs. In practice, I allowed adjacent gaits to share some clips, smoothing gait transitions. The navigation (N) task used all of these clips; SN used a subset of only 60 walking clips, while FOA and MOA used a set of 18 walking clips.

The rest of the dependent dimensions are continuous and my technique learned continuous functions along these dimensions drawn from one of two families. The first is the polynomials:

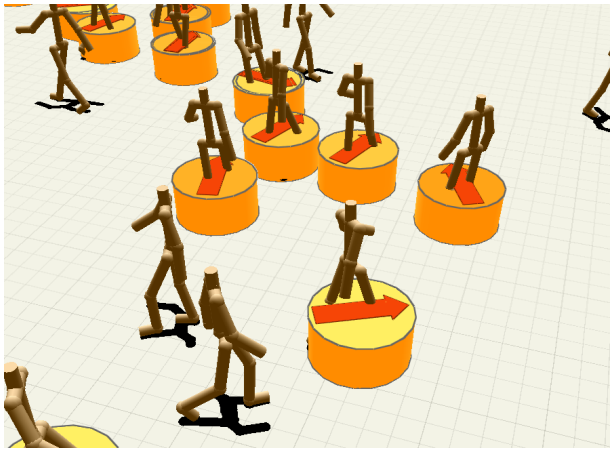


Figure 5.5: Moving obstacle avoidance. Moving obstacle avoidance models short-horizon crowd dynamics: each character who is viewed as an obstacle by another is marked by a cylinder. Thus, characters automatically avoid one another.

$$\mathcal{P}_N(x) \equiv \{f(x) = x^k \mid k = 0, \ldots, N\}.$$

The second is a family of N Gaussian bumps evenly spaced along the interval $[a, b]$:

$$\mathcal{G}_{a,b,N}(x) \equiv \left\{f(x) = e^{\frac{-(x-\bar{x})^2}{\sigma^2}} \;\middle|\; \bar{x} = a + i\frac{b-a}{N-1}, i = 0, \ldots, N-1\right\},$$

where $\sigma^2 = 2$ throughout this chapter. Multivariate families of functions are defined through outer products. For example, denote:

$$\mathcal{Q}(x)\mathcal{R}(y) \equiv \{f(x,y) = q(x)r(y) \mid q \in \mathcal{Q},\, r \in \mathcal{R}\}.$$

The bases for each task are shown in the last column of Table 5.1. In FOA for example, I used the basis $\mathcal{P}_4(u)\mathcal{P}_2(v)$ of polynomials (Figure 5.6, (1)-(15)) for general control, and added a set of Gaussian bases $\mathcal{G}_{0,\text{-}\frac{3}{2},4}(u)\mathcal{G}_{x+u,x+u,1}(v)$ (Figure 5.6, (16)-(18)) to improve the resolution of the value function in the crucial region around the obstacle.

All optimizations were performed with samples uniformly covering each dimension, with between 5 and 8 samples per dimension. For cyclic dimensions, samples divided up the interval [$\text{-}\pi, \pi$), while along linear dimensions samples covered a 4 meter interval

about the origin. My sampling is therefore much higher resolution than the value function bases. I found I could delete up to 50% of the samples at random without appreciable detriment to the controller. Because of the low-dimensional value function representation, learning the controllers was relatively quick: my 2D navigational controllers with several hundred clips never took longer than 45 minutes nor more than 2GB of memory to learn. The more complicated obstacle avoidance controllers never took longer than one hour to learn, and also required about 2GB of memory. Precomputation time is measured as the longest single process runtime when parallelizing along separable and switchable dimensions.

My runtime controller requires 0.101 ms to choose state transitions and can blend poses at 0.04 ms per frame (not including rendering) on a Dual Core 3.5 GHz Intel Xeon processor. Because of the compact basis representation, the runtime memory costs of my method are low. Navigational controllers require less than 13KB of memory while my avoidance controllers require less than 18KB with double precision floating point numbers. In fact, these numbers were dwarfed by the cost of storing the motion clips, which is 23MB for my navigational controller and 1.1MB for my avoidance controller. Note that these costs are amortized across all characters. The per-character memory footprint can be measured in bytes.

Examples of these controllers can be seen in the accompanying video.[2] Compared to greedy methods, my planning algorithm produces better motion with sharper turns, an effect shown in the video and demonstrated numerically in Figure 5.4. Surprisingly few bases are required to capture these results. I varied the polynomial degree along the θ axis for a simple navigational controller and compared the value at sample points to the 10th-degree approximation. While quadratic polynomials are about 11% different from the 10th-degree value, they nonetheless produce motion visually indistinguishable to quality to higher degrees. My results also demonstrate successful avoidance of fixed and moving obstacles. I created simple crowd simulations with the latter. Specifically, each character fits a moving obstacle model to all neighbors within a threshold distance. The highest

[2]http://grail.cs.washington.edu/projects/graph-optimal-control/

value neighbor is then treated as a moving obstacle. This avoidance approach is heuristic and can fail in situations with too many neighbors. However, I found it worked well at medium densities.

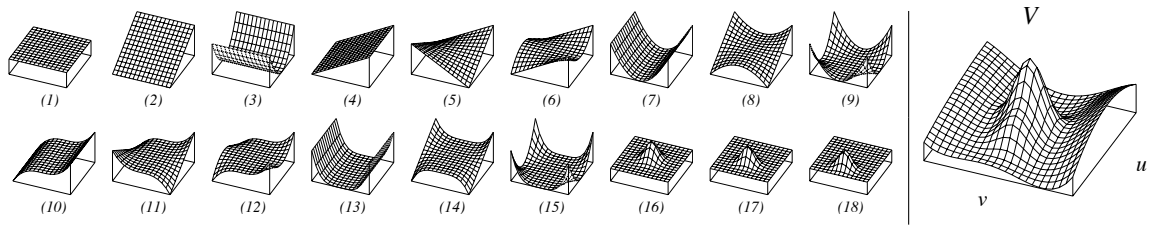


Figure 5.6: Basis functions for fixed obstacle avoidance. (1)-(15) are the polynomial bases for fixed obstacle avoidance (FOA). (16)-(18) are three of the four Gaussian bases. All bases are shown as slices through the u-v plane, holding θ, C, and $x + u$ constant. My method solves for the linear superposition of basis functions shown on the right.

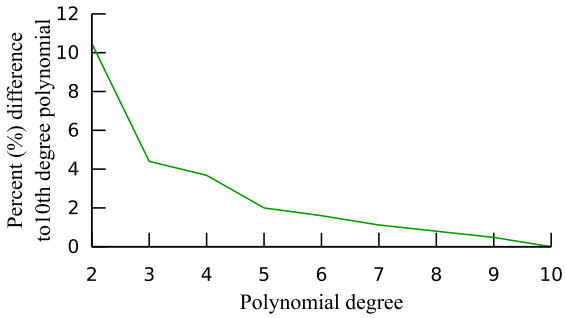


Figure 5.7: Value function convergence. This graph varies the degree d of the root orientation polynomial $\mathcal{P}_d(\theta)$ for a simple navigational controller. Errors are measured relative to the 10th-degree approximation as the sum of squared differences over a set of samples along the θ axis.

Parameter selection is considerably more difficult for this algorithm than it was for crowd simulation (section 4.4) because of the training time required to learn the basis. I suggest that the animator first set all coefficients to zero and then find the correct balance between the physics cost γ_Ψ and the direction γ_ρ. Weighting the cost too heavily towards the former makes the simulation smooth, but sluggish, especially during turns. Weighting too much towards the latter can yield unrealistic motion in which momentum is not preserved. With sufficient motion clips, the character tends to deviate only very slowly from his intended motion direction, and deviation γ_x can be set to a very low number, or even

zero (which should reduce the number of state variables). Finally, each of our examples had at most one additional nonzero cost coefficient, either obstacle cost γ_O, torso orientation γ_τ, or spin cost γ_ω. This variable can usually be discovered by "bracketing" the correct value between a pair of guesses which are too low and too high, and then binary searching towards the correct value. In our case, this search was considerably aided by using a parallel computing cluster which allowed us to search through several parameter values at once.

5.5 Conclusion and Future Work

This chapter presented a new control model for realtime character animation subject to high dimensional, continuously changing user control. The underlying motion model enables rapid transitions while enforcing foot-skate constraints for a wide class of motion. The control mechanism chooses sequences of clips using a compact, continuous value function representation that admits fast, near optimal control, even in high dimensional spaces.

While the solution presented does not solve the dimensionality problem in control, it indicates that combining a number of different strategies can mitigate this problem. The most important strategy is using basis functions to represent the value space. This has several advantages: the basis functions can correlate information between dimensions, reducing the pure dimensional explosion of sampling, while the wide support of each basis function enables significant reduction in required samples. Also, parallelizing along constant dimensions made higher dimensional controllers possible. This has allowed me to create complex high dimensional controllers in less than one hour of optimization.

My framework automatically produces near-optimal controllers given a set of objectives. There is no need to explicitly construct walk cycles or otherwise edit the underlying data. The system is able to blend through ground contact points, preventing foot-skate for single-stance motions in realtime without post-processing such as inverse kinematics. The technique is also expandable in that controllers can be constructed separately for different tasks, then combined with near-optimal transitions even when the constructors were

learned on different underlying motion sets. This greatly reduces the cost of constructing complex controllers, as each individual task can be optimized separately before connecting the results together. Finally this chapter demonstrated the technique for a number of tasks including navigation with independent control over gait and torso orientation as well as 2D obstacle avoidance. The latter can create crowd simulations in which each character fits a local linear model to the movement of near characters and then avoids those characters in realtime, all without explicitly defining collision avoidance strategies.

The major limitation of my technique is that it is too closely tied to blending precaptured motion data, thus requiring large amounts of such data to produce highly varied controllers. For example, the navigational controllers require the motion capture subject to perform a large number of actions: locomoting and turning at a variety of speeds and torso orientations. Ideally, one would like to determine automatically the minimal set of clips that would guarantee good controllers, perhaps computing additional clips via a synthesis process. A related problem is that while my system has a very high branching factor (transitions are possible between any two clips), the system is still limited to a discrete number of transitions. This makes it harder to achieve very fine control such as stepping exactly on certain points or walking on a narrow bar. In fact, my system always achieves better motion when given more clips. This is a situation where a more general notion of blending, such as "fat graphs" [119] or multi-way blending [114] might prove useful. I believe that my foot-skate constraint could be generalized to handle multi-way blends. Also, I expect that specialized blending techniques which always produce physically correct motion [113] would enable more low-cost transitions and therefore better motion with fewer clips.

Another important advance would be to treat the clips as samples from a continuous manifold of human motion. This would enable continuous bases along this manifold, further mitigating the curse of dimensionality.

Looking further, the next major advance would be to allow full kinematic motion based on the degrees of freedom in the skeleton. The principle obstacle to this research direction is the vast dimensionality of the human skeleton. In addition, human motion is based on a set of highly interconnected biomechanical systems such as bones, muscles, and tendons, and has much to do with robustness and styles that are learned over years of practice.

These issues add to the difficulty of the problem, and the field is still far from producing motion this way. With this goal in mind, however, it is especially important to explore strategies that address the dimensional explosion inherent in motion controllers.

Chapter 6

CONCLUSION

This thesis presents new techniques for real-time interactive simulation across three domains: fluid dynamics, crowd dynamics, and human animation. While the approaches presented are substantially different, they all illuminate a unifying theme: interactive dynamics requires new algorithms and new evaluation metrics. Even with exponential increases in processing speed, traditional simulation algorithms applied to very high dimensional phenomena will not achieve real-time performance in the near future. At the same time, human interaction and the unbounded length of simulation create new requirements for real-time simulation such as long-term statistical correctness which are not usually present in traditional algorithms.

I achieve these goals by developing *new representations* (or model reductions) of the underlying dynamics. These representations lead to algorithms that have better asymptotic time complexity than traditional algorithms. For fluid dynamics, I represent the velocity as a linear combination of basis states. This gives my system a unique runtime cost which is proportional to the number of basis states, but *independent* of the number of simulation voxels. To support interactivity, the system must handle moving objects immersed in the flow. I do so by endowing each object with its own reduced model. Importantly, the coupling forces between the individual models can be computed in the reduced space as well.

For crowd simulation, I adopt a continuum perspective rather than using per-agent dynamics. All computation is performed on a 2D grid that stores crowd velocity, density, and potential. The individual crowd members are advected through these fields without any intelligence. Besides having completely different (and very fast) runtime properties, this algorithm unifies global planning and collision avoidance. This means that individuals in my simulations do not face conflicting requirements between local collision avoidance and global planning, leading to smoother simulations over longer periods of time.

Finally, for my character animation model, the reduction is not on the simulation state, but in the planning space. I use a linear basis representation for the value function. The basis functions can correlate information between dimensions, reducing the pure dimensional explosion of sampling. Further, the wide support of each basis function also significantly reduces the variables we need to store. Finally, various forms of parallelization enable further increases in the dimensionality of my controllers. These results indicate that low-dimensional control models are capable of creating rich and responsive human motion based on motion capture data sets.

6.1 Future Horizons

There are many exciting avenues of future research using model reduction. One grand challenge is to "model-reduce everything." I want to extend model reduction across a vast range of natural phenomena and to the interlocking relationships between multiple systems. This requires extending the theory to more general classes of functions as I simultaneously develop methods to couple such systems together. For example, one might envision a model-reduced biomechanical simulation of a human standing in a model-reduced stream of water. The human's movement must create correct fluid disturbances, but both systems may only "talk to each other" in the reduced space. As a first step along this path, my recent work [133] explored interactions between model-reduced fluid simulations and arbitrarily moving rigid bodies, demonstrating such interactions for the first time in any field. I am convinced that this work on model reduction will profoundly effect real-time simulation in graphics. I think it is quite possible that soon all dynamics in video games will be model-reduced.

Another fundamental challenge is finding "optimal" low-dimensional representations. Very little is known about this when the goal is to faithfully reproduce a dynamical system. My own work used standard Principal Component Analysis for dimension reduction, but this ignores the underlying physics; much better solutions surely exist. Optimal dimension reduction also leads to vital questions about how to preserve invariants. For example I have shown that both incompressibility and energy can be preserved through fluid model

reduction. However, complex systems teem with important invariants, from volume to momentum, that must be preserved.

The dimensionality problem in control is parallel to that in simulation, and I believe the same avenue will be fruitful: model reduction. This will allow us control using only partial state information given by the reduced model. For example, a digital character might successfully move around without complete knowledge of its own state. Instead it would use a lower-dimensional representation of its state which would subsequently make learning much easier. Dimension reduction may even allow us to extract the *minimum* information required to perform certain control tasks.

An important and related challenge is how to build "easily controllable" systems. For example, it has been suggested [71] that animals move fluidly in part because tendons and other passive elements stabilize noisy neural control signals. How can such easily controllable elements be identified? Can these concepts be generalized to other control problems? I believe these questions closely relate to model reduction because easily controllable mechanisms are likely to be those that present a simple (reduced) representation to the control system.

Looking further into the future, if we could derive animal motion from first principles, we could begin to investigate the role that motion plays in a larger biological evolution. This opens up a suite of fascinating problems. How did animal motion and shape coevolve and influence one another? Can we synthesize the motion of now-extinct animals, and derive what selective pressures led to their morphology? Understanding animal motion also has practical implications, from designing prosthetic limbs and mechanical exoskeletons to robotics. I hope that progress on high-dimensional systems, catalyzed in part by my research, will help realize these applications while advancing our understanding of complex physical and biological phenomena.

BIBLIOGRAPHY

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] John David Anderson. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill, 6 edition, 1995.

[3] I. Aoki. A simulation study on the schooling mechanism in fish. *Bulletin of the Japanese Society of Scientific Fisheries*, 48:1081–1088, 1982.

[4] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, July 2002.

[5] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Transactions on Graphics*, 22(3):402–408, July 2003.

[6] R. C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, pages 92–112, 1989.

[7] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locallyweighted learning for control. *Artificial Intelligence Review*, 11(1-5):75–113, February 1997.

[8] J. Ausseur, J. Pinier, M. Glauser, and H. Higuchi. Predicting the Dynamics of the Flow over a NACA 4412 using POD. *APS Meeting Abstracts*, page D8, November 2004.

[9] Jernej Barbič and Doug James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics*, 24(3):982–990, August 2005.

[10] O. Burchan Bayazit, Jyh-Ming Lien, and Nancy M. Amato. Better group behaviors in complex environments with global roadmaps. In *Int. Conf. on the Sim. and Syn. of Living Sys. (Alife)*, pages 362–370, Dec 2002.

[11] Maren Bennewitz and Wolfram Burgard. Finding solvable priority schemes for decoupled path planning techniquesfor teams of mobile robots. Proceedings of the 9th International Symposium on Intelligent Robotic Systems (SIRS), 2001.

[12] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 2001.

[13] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM Transactions on Graphics*, 22(3):917–924, July 2003.

[14] David C. Brogan and Jessica K. Hodgins. Group behaviors for systems with significant dynamics. In *Autonomous Robots*, pages 137–153, 1997.

[15] J. Buhl, D. J. T. Sumpter, I. D. Couzin, J. J. Hale, E. Despland, E. R. Miller, and S. J. Simpson. From disorder to order in marching locusts. *Science*, 312(5778):1402–1406, 2006.

[16] J. Chen and N. Lobo. Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graphical Models and Image Processing*, pages 107–116, 1995.

[17] J. X. Chen, N. da Vittoria Lobo, C. E. Hughes, and J. M. Moshell. Real-Time Fluid Simulation in a Dynamic Virtual Environment. *IEEE Computer Graphics and Applications*, pages 52–61, May-June 1997.

[18] Stephen Chenney. Flow tiles. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 233–242, July 2004.

[19] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2):182–203, April 2003.

[20] Paul W. Cleary, Soon Hyoung Pyo, Mahesh Prakash, and Bon Ki Koo. Bubbling and frothing liquids. *ACM Transactions on Graphics*, 26(3):97:1–97:6, July 2007.

[21] Richard R. Clements and Roger L. Hughes. Mathematical modelling of a mediaeval battle: the battle of Agincourt, 1415. *Math. Comput. Simul.*, 64(2):259–269, 2004.

[22] Rinaldo M. Colombo and Massimiliano D. Rosini. Pedestrian flows and nonclassical shocks. *Mathematical Methods in the Applied Sciences*, 28(13), 2005.

[23] Otávio C. Cordeiro, Adriana Braun, Cesar B. Silveira, Soraia R. Musse, and Gerson G. H. Cavalheiro. Concurrency on social forces simulation model. First International Workshop on Crowd Simulation, November 2005.

[24] M. Couplet, C. Basdevant, and P. Sagaut. Calibrated reduced-order POD-Galerkin system for fluid flow modelling. *J. Comput. Phys.*, 207(1):192–220, 2005.

[25] Iain D. Couzin and Nigel R. Franks. Self-organized lane formation and optimized traffic flow in army ants. *Proceedings: Biological Sciences*, 270(1511):139–146, 2003.

[26] Iain D. Couzin, Jens Kraus, Richard Jame, Graeme D. Ruxton, and Nigel R. Franks. Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, 218(1):1–11, 2002.

[27] Iain D. Couzin and Jens Krause. Self-organization and collective behavior in vertebrates. *Advances in the Study of Behavior*, 32:1–75, 2003.

[28] András Czirók and Tamás Vicsek. Collective motion. In *Fluctuations and Scaling in Biology*. Oxford University Press, 2001.

[29] Daniela Pucci de Farias. *The Linear Programming Appraoch to Approximate Dynamic Programming: Theory and Application*. PhD thesis, Stanford, June 2002.

[30] J. L. Deneubourg, S. Goss, N. Franks, and J. M. Pasteels. The blind leading the blind: Modeling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, 2(5):719–725, 1989.

[31] Leah Edelstein-Keshet, James Watmough, and Daniel Grünbaum. Do travelling band solutions describe cohesive swarms? An investigation for migratory locusts. *Journal of Mathematical Biology*, 36(6):515–609, 1998.

[32] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. In *Discrete Differential Geometry*, Chapter 9 of Course Notes. ACM SIGGRAPH, 2005.

[33] D. Enright, S. Marschner, and R. Fedkiw. Animation and Rendering of Complex Water Surfaces. In *Computer Graphics (SIGGRAPH 2002)*, pages 736–744. ACM, July 2002.

[34] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1):83–116, 2002.

[35] R. Fedkiw, Jos Stam, and H. Jensen. Visual simulation of smoke. In *Computer Graphics (SIGGRAPH 2001)*, pages 15–22. ACM, August 2001.

[36] Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. *ACM Transactions on Graphics*, 24(3):904–909, August 2005.

[37] Franck Feurtey. Dept. of ee. Master's thesis, Univ. of Tokyo, 2000.

[38] C.A.J. Fletcher. *Computational Techniques for Fluid Dynamics*. Springer-Verlag, 2 edition, 1991.

[39] G. Flierl, D. Grünbaum, S. Levin, and D. Olson. From individuals to aggregations: the interplay between behavior and physics. *Journal of Theoretical Biology*, 196(4):397–525, 1999.

[40] M. Fogleman, J. Lumley, D. Rempfer, and D. Haworth. Application of the proper orthogonal decomposition to datasets of internal combustion engine flows. *Journal of Turbulence*, 5(23), June 2004.

[41] N. Foster and R. Fedkiw. Practical Animation of Liquids. In *Computer Graphics (SIGGRAPH 2001)*, pages 23–30. ACM, August 2001.

[42] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[43] N. Foster and D. Metaxas. Modeling the Motion of a Hot, Turbulent Gas. In *Computer Graphics (SIGGRAPH 97)*, pages 181–188. ACM, August 1997.

[44] N. R. Franks, N. Gomez, S. Goss, and J. L. Deneubourg. The blind leading the blind in army ant raid patterns: Testing a model of self-organization (Hymenoptera: Formicidae). *Journal of Insect Behavior*, 4(5):583–607, 1991.

[45] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 29–38, August 1999.

[46] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. Snap-together motion: Assembling run-time animation. *ACM Transactions on Graphics*, 22(3):702–702, July 2003.

[47] Siome Goldenstein, Menelaos Karavelas, Dimitris Metaxas, Leonidas Guibas, Eric Aaron, and Ambarish Goswami. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics*, 25(6):983–998, December 2001.

[48] N Goodnight, C Woolley, D Luebke, and G. A. Humphreys. Multigrid solver for boundary value problems using programmable graphics hardware. In *Proceeding of Graphics Hardware*, page 102, 2003.

[49] Shay Gueron and Simon A. Levin. Self-organization of front patterns in large wildebeest herds. *Journal of Theoretical Biology*, 165(4):541–552, 1993.

[50] Mohit Gupta and Srinivasa G Narasimhan. Legendre fluids: A unified framework for analytic reduced space modeling and rendering of participating media. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2007)*, August 2007.

[51] Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189, 1965.

[52] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. Physically-based visual simulation on graphics hardware. In *Graphics Hardware 2002*, pages 109–118, September 2002.

[53] Laure Heïgeas, Annie Luciani, Joelle Thollot, and Nicolas Castagné. A physically-based particle model of emergent crowd behaviors. In *GraphiCon*, 2003.

[54] D. Helbing, L. Buzna, and T. Werner. Self-organized pedestrian crowd dynamics and design solutions. *Traffic Forum*, 12, 2003.

[55] Dirk Helbing, Péter Molnár, Illés J. Farkas, and Kai Bolay. Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design*, 28:361–383, 2001.

[56] Dirk Helbing, Péter Molnár, and Frank Schweitzer. Computer simulations of pedestrian dynamics and trail formation. In *Evolution of Natural Structures*, pages 229–234, 1994.

[57] P. Holmes, J. L. Lumley, and G. Berkooz. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, Cambridge, MA, 1996.

[58] L. Hongwan, F. K. Wai, and C. H. Chor. A study of pedestrian flow using fluid dynamics. Technical report, 2003.

[59] Roger L. Hughes. A continuum theory for the flow of pedestrians. *Transportation Research Part B*, 36(6):507535, july 2002.

[60] Roger L. Hughes. The flow of human crowds. *Annu. Rev. Fluid Mech.*, 35:169–182, 2003.

[61] Leslie Ikemoto, Okan Arikan, and David Forsyth. Learning to move autonomously in a hostile environment. Technical Report UCB/CSD-5-1395, June 2005.

[62] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transactions on Graphics*, 25(3):805–811, July 2006.

[63] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics*, 22(3):879–887, July 2003.

[64] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artifcial Intelligence Research*, 4:237–285, May 1996.

[65] J. T. Kajiya and B. P. von Herzen. Ray Tracing Volume Densities. *Computer Graphics (SIGGRAPH 84)*, 18(3):165–174, July 1984.

[66] A. Kamphuis and M. H. Overmars. Finding paths for coherent groups using clearance. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 19–28, July 2004.

[67] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, pages 49–57, August 1990.

[68] Neil Ketchell. A technical summary of the aea egress code. Technical Report 1, AEA Technology, 2002.

[69] Ron Kimmel and James A. Sethian. Optimal Algorithms for Shape from Shading and Path Planning. *Journal of Mathematical Imaging and Vision*, 14:237–244, 2001.

[70] Ansgar Kirchner and Andreas Schadschneider. Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. In *Physica A*, volume 312, pages 260–276, 2002.

[71] Daniel E. Koditscheka, Robert J. Full, and Martin Buehlerc. Mechanical aspects of legged locomotion control. *Arthropod Structure and Development*, 33:251–272, 2004.

[72] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, July 2002.

[73] Jens Krüger and Rüdiger Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics*, 22(3):908–916, July 2003.

[74] Taesoo Kwon and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 29–38, July 2005.

[75] Fabrice Lamarche and Stéphane Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3):509–518, September 2004.

[76] Manfred Lau and James J. Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 299–308, September 2006.

[77] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[78] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500, July 2002.

[79] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 79–87, July 2004.

[80] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. Motion patches: building blocks for virtual environments annotated with motion data. *ACM Transactions on Graphics*, 25(3):898–906, July 2006.

[81] Herbert Levine, Wouter-Jan Rappel, and Inon Cohen. Self-organization in systems of self-propelled particles. *Physical Review E*, 63, 2001.

[82] Tsai-Ten Li and Hsu-Chi Chou. Motion planning for a crowd of robots. In *IEEE International Conference on Robotics and Automation*, September 2003.

[83] Tsai-Yen Li, Ying-Jiun Leng, and Shih-I Chang. Simulating virtual crowds with a leader-follower model. In *Proceedings of 2001 Computer Animation Conference*, Nov 2001.

[84] Wei Li, Xiaoming Wei, and Arie Kaufman. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19(7-8):444–456, 2003.

[85] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics*, 21(3):408–416, July 2002.

[86] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, 23(3):457–462, August 2004.

[87] Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. Multiple interacting liquids. *ACM Transactions on Graphics*, 25(3):812–819, July 2006.

[88] Celine Loscos, David Marchal, and Alexandre Meyer. Intuitive crowd behaviour in dense urban environments using local laws. In *Theory and Practice of Computer Graphics (TPCG'03)*, June 2003.

[89] J. L. Lumley. *Stochastic Tools in Turbulence*, volume 12 of *Applied Mathematics and Mechanics*. Academic Press, New York, 1970.

[90] M. Marion and R. Temam. Nonlinear Galerkin methods. *SIAM J. Numer. Anal.*, 26(5):1139–1157, 1989.

[91] Massive Software. http://www.massivesoftware.com, 2006.

[92] S. Maurel, J. Boree, and J. Lumley. Extended proper orthogonal decomposition: Application to jet/vortex interaction. *Flow, Turbulence and Combustion*, 67(2):125–36, June 2001.

[93] James McCann and Nancy Pollard. Responsive characters from motion fragments. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), July 2007. *To appear.*

[94] Ronald A. Metoyer and Jessica K. Hodgins. Reactive pedestrian path following from examples. *The Visual Computer*, 20(10):635–649, 2004.

[95] Alexander Mogilner and Leah Edelstein-Keshet. A non-local model for a swarm. *Journal of Mathematical Biology*, 38(6):534–570, 1999.

[96] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995.

[97] S. R. Musse and D. Thalmann. A model of human crowd behavior: Group interrelationship and collision detection analysis. In *Computer Animation and Simulation '97*, pages 39–51, September 1997.

[98] Soraia R. Musse, Christian Babski, Tolga Capin, and Daniel Thalmann. Crowd modelling in collaborative virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 115–123, 1998.

[99] D. Nguyen, R. Fedkiw, and H. Jensen. Physically Based Modeling and Animation of Fire. In *Computer Graphics (SIGGRAPH 2002)*, pages 736–744. ACM, July 2002.

[100] Akira Okubo. *Diffusion and Ecological Problems*. Springer, 1980.

[101] OpenSteer. http://opensteer.sourceforge.net, 2006.

[102] Sang Il Park and Myoung Jun Kim. Vortex fluid for gaseous phenomena. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 261–270, July 2005.

[103] Lynne E. Parker. Designing control laws for cooperative agent teams. In *IEEE International Conference on Robotics and Automation*, pages 582–587, 1993.

[104] Julia K. Parrish, Steven V. Viscido, and Daniel Grünbaum. Self-organized fish schools: An examination of emergent properties. *Biological Bulletin*, 202(3):296–305, 2002.

[105] Nuria Pelechano, Kevin OBrien, Barry Silverman, and Norman Badler. Crowd simulation incorporating agent psychological models, roles and communication. First International Workshop on Crowd Simulation, November 2005.

[106] Julien Pettré, Jean-Paul Laumond, and Daniel Thalmann. A navigation graph for real-time crowd animation on multilayered and uneven terrain. First International Workshop on Crowd Simulation, November 2005.

[107] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics*, 21(3):501–508, July 2002.

[108] Craig Reynolds. Steering behaviors for autonomous characters, 1999.

[109] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 25–34, July 1987.

[110] C. W. Rowley, I. G. Kevrekidis, J. E. Marsden, and K. Lust. Reduction and reconstruction for self-similar dynamical systems. *Nonlinearity*, 16:1257–1275, July 2003.

[111] Clarence W. Rowley and Jerrold E. Marsden. Reconstruction equations and the Karhunen-Loéve expansion for systems with symmetry. *Phys. D*, 142(1-2):1–19, 2000.

[112] C.W. Rowley, D.R. Williams, T. Colonius, R.M. Murray, and D.G. MacMartin. Linear models for control of cavity flow oscillations. *J. Fluid Mech.*, January 2006.

[113] Alla Safonova and Jessica K Hodgins. Analyzing the physical correctness of interpolated human motion. In *In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 171–180, August 2005.

[114] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3), July 2007. *To appear.*

[115] R. Schmit and M. Glasuer. Low dimensional tools for flow-structure interaction problems: Application to micro air vehicles. *APS Meeting Abstracts*, pages D1+, November 2002.

[116] Arno Schödl and Irfan Essa. Machine learning for video-based rendering. *Advances in Neural Information Processing Systems*, 13:1002–1008, July 2001.

[117] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics*, 24(3):910–914, August 2005.

[118] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, New York, NY, USA, 2005. ACM Press.

[119] Hyun Joon Shin and Hyun Seok Oh. Fat graphs: Constructing an interactive character with continuous controls. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 291–298, September 2006.

[120] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[121] S. Sirisup and G. E. Karniadakis. A spectral viscosity method for correcting the long-term behavior of POD models. *J. Comput. Phys.*, 194(1):92–116, 2004.

[122] L. Sirovich. Turbulence and the dynamics of coherent structures. I - Coherent structures. II - Symmetries and transformations. III - Dynamics and scaling. *Quarterly of Applied Mathematics*, 45:561–571, October 1987.

[123] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. *ACM Transactions on Graphics*, 26(3):107:1–107:9, July 2007.

[124] Jos Stam. Stable Fluids. In *Computer Graphics (SIGGRAPH 99)*, pages 121–128. ACM, August 1999.

[125] Jos Stam. A simple fluid solver based on the FFT. *Journal of graphics tools*, 6(2):43–52, 2001.

[126] G. Still. *Crowd Dynamics*. PhD thesis, University of Warwik, UK, 2000.

[127] Mankyu Sung, Michael Gleicher, and Stephen Chenney. Scalable behaviors for crowd simulation. *Computer Graphics Forum*, 23(3):519–528, September 2004.

[128] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 291–300, New York, NY, USA, 2005. ACM Press.

[129] John Toner and Yuhai Tu. Flocks, herds, and schools: A quantitative theory of flocking. *Physical Review E*, 58(4):4828–4858, 1998.

[130] Chad M. Topaz and Andrea L. Bertozzi. Swarming patterns in a two-dimensional kinematic model for biological groups. *SIAM Journal on Applied Mathematics*, 65(1):152–174, 2004.

[131] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Transactions on Graphics*, 25(3):1160–1168, July 2006.

[132] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. *ACM Transactions on Graphics*, 26(3), July 2007.

[133] Adrien Treuille, Andrew Lewis, and Zoran Popović. Model reduction for real-time fluids. *ACM Transactions on Graphics*, 25(3):826–834, July 2006.

[134] Michael A. Trick and Stanley E. Zin. Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics 1*, page 255277, 1997.

[135] R. Tsai, H. Zhao, and S. Osher. Fast sweeping algorithms for a class of hamilton-jacobi equations. *SIAM Journal of Numerical Analysis*, 42(6), 2005.

[136] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[137] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50, July 1994.

[138] Hongcheng Wang, Qing Wu, Lin Shi, Yizhou Yu, and Narendra Ahuja. Out-of-core tensor approximation of multi-dimensional matrices of visual data. *ACM Transactions on Graphics*, 24(3):527–535, August 2005.

[139] Huamin Wang, Peter J. Mucha, and Greg Turk. Water drops on surfaces. *ACM Transactions on Graphics*, 24(3):921–929, August 2005.

[140] Andrew Witkin and Zoran Popović. Motion Warping. *Proceedings of SIGGRAPH 95*, pages 105–108, August 1995.

[141] P. Witting. Computational Fluid Dynamics in a Traditional Animation Environment. In *Computer Graphics (SIGGRAPH 99)*, pages 129–136. ACM, August 1999.

[142] Enhua Wu, Youquan Liu, and Xuehui Liu. An improved study of realtime fluid simulation on GPU. *Computer Animation and Virtual Worlds*, 15(3-4):139–146, 2005.

[143] G. Yngve, J. O'Brien, and J. Hodgins. Animating Explosions. In *SIGGRAPH 2000 Conference Proceedings, Annual Conference Series*, pages 29–36, July 2000.

[144] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Transactions on Graphics*, 26(3):99:1–99:8, July 2007.

VITA

Adrien Treuille received his B.S. in Computer Science from Georgetown University in 2001 and his Ph.D. in Computer Science from the University of Washington, Seattle in 2008.