

HYBRID TECHNIQUES FOR HIGH-FIDELITY
PHYSICAL SIMULATION OF SOLIDS AND FLUIDS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Andrew Paul Selle

June 2008

© Copyright by Andrew Paul Selle 2008
All Rights Reserved

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Ronald Fedkiw) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Pat Hanrahan)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Scott Klemmer)

Approved for the University Committee on Graduate Studies.

Abstract

This dissertation presents hybrid algorithms for the simulation of fluids and solids as well as methods for coupling between the two. Physical simulations have become prevalent in computer graphics due to their ability to predict phenomena with too many degrees of freedom for hand animation. Nevertheless, it has been challenging to obtain simulations with sufficiently high quality in a reasonable amount of time, leading to the development of a large variety of geometric representations and simulation algorithms, each suited to a particular class of problems. While this makes choosing and implementing simulation technology difficult, this diversity of techniques results from attempts to balance tradeoffs. This suggests that increased simulation fidelity and efficiency can be obtained by synthesizing hybrid techniques, and this thesis presents examples of such techniques. In addition to algorithmic improvements we also consider the use of distributed memory parallelism (MPI) to improve the tractability of highly detailed simulations.

In the first chapters, we concentrate on increasing the fidelity of fluids by reducing numerical dissipation. Truncation error in the solution of the hyperbolic advection equation is reduced by a new unconditionally stable advection method. Two semi-Lagrangian advection steps are combined to obtain second-order accuracy in space and time while also providing a constructive reinterpretation of MacCormack's advection method. To reduce dissipation in smoke, water and explosions, we also present a Lagrangian/Eulerian scheme that couples a Lagrangian vortex particle method with an Eulerian pressure-velocity method. Additionally, we consider coupling Lagrangian deformable and rigid thin shells to Eulerian fluid flows using robust ray casting, fully preventing fluid leaks across the thin boundary.

In the next chapters, we discuss simulation of solids using Lagrangian techniques. We introduce improved altitude springs for the simulation of volumetric tetrahedra, which we apply to bending in cloth and torsion in hair. We also introduce a simple mass-spring model for hair simulation that uses tetrahedral simulation techniques for simulating straight and curly strands of hair. For time integration we develop an unconditionally stable fully-implicit linear spring that can be included in a semi-implicit Newmark time integration scheme. For collisions and interactions, we develop a hybrid scheme that uses inexpensive history-based repulsions coupled with more accurate geometric collisions. We also present a method for obtaining accurate friction of deformable objects with collision bodies when a semi-implicit or fully-implicit time evolution approach is used.

Acknowledgments

I would like to thank my parents Thomas Selle and Stephanie Copoulos-Selle for all the support and guidance they have provided through the years. Obviously much of my interest in graphics stems from the visual aesthetics they helped instill in me. They and my sister, Rachel, have been understanding of all the Christmas' that I have missed for SIGGRAPH papers.

I would also like to thank my advisor, Ron Fedkiw, who has had a profound impact on my life inside and outside the lab. From when we first met and he asked me, "Are you a nerd?" I knew that he was an advisor that would challenge me to go outside my comfort zone so I could achieve more than I thought possible. Besides technical skills, he taught me that the relationships we form with others are as important to research as the algorithms we design.

The rest of my research group has also had a large impact on my education as I had never worked with such a large group of extremely bright individuals. I would first like to acknowledge my co-authors in the group, Eran Guendelman, Frank Losasso, Tamar Shinar, Eftychios Sifakis, Avi Robinson-Mosher, Jonathan Su, Geoffrey Irving, and Michael Lentine. Special thanks goes to Frank, Geoffrey and Eftychios who I worked closely with for four of my five years. I would also like to thank Jonathan and Michael who were my second authors that both dealt well with my sometimes demanding and impatient personality. Others in the group that I have worked with were, Rachel Weinstein, Craig Schroeder, Joseph Teran, Jerry Talton, Sergey Koltakov, Zhaosheng Bao, Kevin Der, Jon Gretarsson, Frederic Gibou, Jeong-Mo Hong, Nipun Kwatra, Neil Molino, Igor Neverov, Duc Nguyen, Robert Strzodka. Also, I would like to thank CURIS students who were extremely helpful in the production of papers. These

include, Jiyai Chong, Eilene Hao, Cynthia Lau, Joyce Pan, Justin Solomon, and Melody Wu.

I also had the privilege of collaborating on a scientific computing paper with several researchers from the Georgia Institute of Technology. So, I would also like to thank to ByungMoon Kim, Yingjie Liu and Jarek Rossignac.

During my time at Stanford, I also had the privilege of consulting at both Intel Corporation and Industrial Light + Magic. At Intel, I worked closely with Radek Grzeszczuk, Chris Hughes, Daehyun Kim, Sanjeev Kumar, Y.K. Chen, Pradeep Dubey, and Jim Hurley. I am grateful with them for the collaboration which deepened my understanding of architecture and its impact on my research. At Industrial Light + Magic I have had interactions with too many people to mention. Kim Libreri was instrumental in giving me the opportunity to work in production at Industrial Light + Magic. This brought me in contact with Willi Geiger, John Knoll, David Meny, Craig Hammock, Joakim Aresson, all of whom I enjoyed working with. In research and development, I would like to acknowledge Nick Rasmussen, Brice Criswell, Zoran Kacic-Alesic, Don Hatch, Stephen Bowline, Julian Hodgson, Chris Twigg, Jeff Smith, and Lin Shi.

I also thank my reading committee consisting of Pat Hanrahan, Scott Klemmer and Ron, as well as the other members of my defense committee, Joe Marks and Heinz Pitsch. I appreciate their time and effort in providing me feedback.

I would also like to acknowledge Michael Gleicher for introducing me to computer graphics, and working with me as an undergraduate research assistant. He provided significant encouragement and advice, leading me to pursue a graduate degree.

Finally, I would like to thank Christina Mester, who has given me her unconditional support and love. I am looking forward to many more years together.

Contents

Abstract	v
Acknowledgments	vii
1 Introduction	1
2 High Order Advection	4
2.1 Introduction to Advection Equations	4
2.2 First order upwinding	5
2.3 Semi-Lagrangian Schemes	6
2.4 BFECC and MacCormack Methods	7
2.5 Temporal Accuracy and Stability	10
2.6 The Wave Equation	12
2.7 New Extrema	16
2.8 Zalesask's Disc	17
2.9 Incompressible Flow	20
2.10 Conclusion	31
3 A Hybrid Incompressible Fluid Solver	32
3.1 Introduction	32
3.2 Pressure Velocity Methods	33
3.2.1 Vorticity Confinement	34
3.3 Vortex Particle Methods	36
3.4 Hybrid Solver	38

3.4.1	Solving the Particles	39
3.4.2	Vorticity Forcing	40
3.5	Examples	42
3.6	Conclusion	42
4	Coupling Thin Shells to Fluids	46
4.1	Introduction	46
4.2	Previous Work	48
4.3	Robust One-Sided Interpolation of Data	49
4.4	Fluid Simulation	51
4.4.1	Computing the Intermediate Velocity	51
4.4.2	Solving for the Pressure	52
4.4.3	Water	54
4.5	Cloth and Thin Shell Simulation	57
4.6	Rendering	58
4.7	Examples	59
4.8	Conclusions	60
5	Solid Simulation	64
5.1	Introduction	64
5.2	Meshes	64
5.3	Materials	65
5.3.1	Constitutive Models	65
5.3.2	Linear Springs	66
5.3.3	Bending	66
5.3.4	Tetrahedral Altitude Springs	67
5.4	Time Integration	72
5.5	Implicit Linear Springs	73
5.5.1	Implicit Linear Springs	73
5.6	Conclusion	74

6	High Fidelity Cloth Simulation	75
6.1	Previous Work	79
6.2	Algorithm Overview	79
6.3	The Outer Collision Loop	81
6.4	History-Based Self-Repulsions	83
6.5	Cloth-Object Collisions	86
6.6	Examples	91
6.7	Conclusion	94
7	Hair Simulation	99
7.1	Introduction	99
7.2	Related Work	100
7.2.1	Aggregate Hair Simulation	101
7.2.2	Strand dynamics	102
7.3	Constitutive Model	104
7.3.1	An Altitude Spring Hair Model	104
7.4	Time Integration	107
7.4.1	Strain Limiting	109
7.5	Interaction and Collisions	111
7.5.1	Body Collisions	111
7.5.2	Stiction	112
7.5.3	Self-Repulsions and Collisions	113
7.6	Examples	114
7.7	Limitations	115
7.8	Discussion	118
8	Parallel Simulation	121
8.1	Fluids Parallelism	122
8.1.1	Advection Equations	122
8.1.2	Incompressible Solve	123
8.1.3	Particle Level Set	123
8.2	Solids Parallelism	124

8.2.1	Time Integration	125
8.2.2	Repulsions and Collisions	126
8.2.3	Solids Analysis	128
8.3	Conclusion	128
9	Conclusions	130
	Bibliography	133

List of Tables

2.1	MacCormack ODE Accuracy	10
2.2	Wave equation via upwinding, CFL=0.75	15
2.3	Wave equation via semi-Lagrangian, CFL=1.75	15
6.1	Parameters of our model	92
6.2	Example resolutions, timing, and processor count	93
6.3	Timing breakdown of algorithm	94
7.1	Comparison between various hair methods	103
7.2	Hair simulation performance numbers	114

List of Figures

2.1	Semi-Lagrangian method	6
2.2	BF ECC and MacCormack method conceptual diagrams	8
2.3	ODE Stability Regions	11
2.4	Numeric and Analytic Truncation Error vs CFL	15
2.5	New Extrema in BF ECC and MacCormack Methods	17
2.6	Zalesask’s disc, upwind, CFL=0.75	18
2.7	Zalesask’s disc, semi-Lagrangian advection, CFL=0.75	19
2.8	Zalesask’s disc, semi-Lagrangian advection, CFL=1.75	19
2.9	Inviscid incompressible flow, CFL=0.75	21
2.10	Inviscid incompressible flow, CFL=1.75	22
2.11	Inviscid incompressible flow comparing clamping techniques	23
2.12	Inviscid free-surface falling drop, $t = 6/24$	25
2.13	Inviscid free-surface falling drop, $t = 10/24$	26
2.14	Inviscid free-surface falling drop, $t = 18/24$	27
2.15	Inviscid free-surface ball splashing into a pool, $t = 4/24$	28
2.16	Inviscid free-surface ball splashing into a pool, $t = 10/24$	29
2.17	Inviscid free-surface ball splashing into a pool, $t = 40/24$	30
3.1	Example vorticity confinement fields	35
3.2	Simulations with varying vorticity confinement	36
3.3	Hybrid solver result for free-surface pool of water	44
3.4	Hybrid solver result for smoke explosion	44
3.5	Hybrid solver result for turbulent river	45

4.1	Robust geometry and one-sided interpolation of data	50
4.2	Neumann boundary conditions for thin shells	54
4.3	Bridson’s collision-aware subdivision methods for cloth	58
4.4	Collision-aware subdivision for cloth in fluids	59
4.5	Rigid thin shell cup fills and pours	60
4.6	Rigid “Buddha” cup fills and pours	61
4.7	Smoke two-way coupled with a cloth	61
4.8	Rigid boat fills with water and sinks	62
4.9	Suspended cloth supports and sheds water	62
4.10	Water stream interacts with suspended cloth curtain	63
5.1	Simple cloth constitutive model	67
5.2	Point/face and edge/edge altitude springs	68
5.3	Tetrahedral altitude springs recovering a mesh	71
5.4	Axial bending springs used in tandem with Bending Springs	71
5.5	Large time step size with implicit integration	74
6.1	Real-world and art show highly intricate folds and wrinkles	76
6.2	Low versus high resolution cloth simulations	76
6.3	Simulation of cloth twisting tightly around a cylinder	78
6.4	Inversion criteria for point/triangle and edge/edge interaction pairs	83
6.5	Diagram of improved body-collision handling on an incline plane	87
6.6	A single particle on incline plane	88
6.7	A single tetrahedron on incline plane	89
6.8	Piece of cloth dropped down an incline plane	95
6.9	100 pieces of cloth dropped with 10,000 triangles each	96
6.10	A piece of cloth with 1.8 million triangles is draped over a ball	96
6.11	Performance versus fixed and adaptive total loops	97
6.12	Cloth curtain with 1.7 million triangles	98
6.13	Cloth cover with 2 million triangles falls off a wardrobe	98
7.1	Options for creating perturbed hair particles	105

7.2	Straight and curly hair models	105
7.3	Triangle path interpretation of tetrahedral hair model	106
7.4	Experimental hair tests reproduced by numerical simulations	107
7.5	Varying curly parameters of hair simulation	108
7.6	Hair tuft examples	110
7.7	Eulerian level set interpolation	111
7.8	Medium length straight hair undergoing wind (500,000 particles) . . .	116
7.9	Medium length straight hair on animated head (500,000 particles) . .	116
7.10	Long curly hair on animated head (250,000 particles)	117
7.11	Long straight hair on animated head (1,000,000 particles)	117
7.12	5,000 hairs versus 10,000 hairs	120
8.1	Cloth interdependencies changing over course of simulation	125
8.2	Parallel collision/repulsion pair Gauss-Seidel ordering	127
8.3	Parallel speedup for cloth simulation	129

Chapter 1

Introduction

Computer graphics as a field is concerned with reproducing the visual aspects of a virtual world. If the goal is a physical and photorealistic world this involves creating a model for the physical world. Typically this modeling process is divided into geometry, rendering, and animation. In this thesis we consider animation, which involves specifying a time varying description of the world. Specifically, we are concerned with animation of objects that are too complicated to animate by hand or with phenomenological procedural models. Examples includes smoke, water, cloth, and hair. The dominant approach to modeling these types of phenomena has been physical simulations, which use numerical methods to approximate physical equations that evolve discrete geometric models over time.

Physical simulation has been studied for many years in the graphics community, because physical simulation has the advantage that simulations are plausible (that is, observable in the real world). The disadvantages are that simulation is often considered difficult to control, because parameters do not have a direct effect, and that simulations are very slow. The former problem continues to be challenging and will certainly require significant research, but the latter problem has been helped by increases in personal computer performance. Additional commodity computer power continues to increase with graphics processing units (GPUs) and chip multiprocessors. Such advances in computer power have caused physical simulation to enter a new golden age in computer graphics.

Even though simulation has become common, the problem of developing techniques adept at modeling all phenomena remains challenging. Originally, simple simulation algorithms were developed within the graphics community, but these have given way to algorithms from computational physics. This echoes the progression of rendering research which went from the development of useful (and intuitive) phenomenological shading and reflectance algorithms to light transport methods that were heavily influenced by radiative transfer literature. The influx of computational methods has had an enormous effect on the quality and prevalence of simulation, but it also has brought into question the role of graphics researchers developing numerical techniques; again, this is analogous to rendering researchers.

In this thesis, we show examples that graphics applications, while having specific utility to graphics, can also yield generally useful techniques for other fields (including computational physics). In particular, while it is true that simulation in graphics considers many of the same problems that computational physics does, it is also true that it has a unique perspective. The fundamental evaluation of any graphics algorithm is whether an image is plausible. While plausibility can be defined in many ways, in this thesis we use the most common definition—photorealism. This metric immediately frees a researcher from absolutely considering physicality of the model because only plausibility of the visual projection of that model is important. This means that we can follow a more intuitive research path. Once a visually plausible technique is developed, a researcher can formalize an algorithm. This is analogous to the process a mathematician might take to proving a theorem.

Another important goal of graphics is producing visually complex worlds, which tend to require interactions of many objects of multiple types. This is in contrast to computational physics which is typically concerned with simple tests that are experimentally verifiable. The techniques that computational physicists use are centered around simulating one phenomena alone such as a fluid or a solid. While coupling between two materials is also considered, all-way coupling of many complex phenomena is typically not. Unfortunately, these techniques for individual phenomena must choose representations and algorithms ideally suited for only that phenomena. Thus, if another phenomena is simulated, completely different choices must be made. This

becomes a problem when, in coupling two phenomena, one chooses a single representation which is optimal for one phenomena but sub-standard for the other. Instead, our techniques will attempt to instead use multiple representations simultaneously so that each phenomena can be simulated in the most appropriate way.

Another important goal in graphics is to produce visually rich scenes with many details. For rendering, modeling and animation, this requires high resolution textures, geometries, and simulations. Unfortunately, adding resolution to a simulation is typically much more expensive than for rendering and modeling. Thus, we seek to provide algorithms that are more efficient at producing detail given a fixed resolution. Numerical methods with high orders of accuracy are usually advocated for this purpose, but order of accuracy specifies only a convergence rate, and outside the asymptotic regime, low order methods with better constants might also be possible. Thus, we consider other techniques for increasing the fidelity of a simulation, often with the use of hybrid techniques.

Along these lines, we present a series of physical simulation techniques for graphics. In Chapter 2, we present an unconditionally stable second order accurate advection scheme that recasts MacCormack advection as a scheme that can be built up with multiple individual semi-Lagrangian advection steps. In Chapter 3, we present a coupled Eulerian/Lagrangian fluid solver that combines Lagrangian vortex particles with a standard Eulerian pressure/velocity solver. In Chapter 4, we present a one-way coupling technique for simulating Eulerian fluid responding to Lagrangian deformable and rigid thin-shells using ray tracing. In Chapter 5, we present our basic approach to solid modeling and introduce a new altitude spring constitutive model. In Chapter 6, we present a cloth collision technique that uses history aware repulsions/attractions together with geometric self-collisions. In Chapter 7, we present a hair technique that uses a tetrahedral view to model hair torsion and bending. In addition, this dissertation also considers the parallelization of solids and fluids algorithms in Chapter 8.

Much of the contents of this thesis is drawn from several previous publications in which I was the primary or main author. In particular, these include [61, 134, 132, 135, 133].

Chapter 2

High Order Advection

2.1 Introduction to Advection Equations

The movement of material is one of the most important aspects of physical simulation. In this chapter we consider the process of bulk convection through a material in an Eulerian setting. That is, a density ρ is measured on a fixed grid and a velocity field causes the density to move between grid points. The model equation for this transport is the linear convection equation

$$\rho_t + \mathbf{V} \cdot \nabla \rho = 0 \tag{2.1}$$

which is a hyperbolic partial differential equation. The velocity field \mathbf{V} is assumed to be divergence free; that is, $\nabla \cdot \mathbf{V} = 0$. Here we are advecting density, but other quantities can be advected such as velocity, strain, or even a signed distance function that tracks an interface.

Since advection is such an important problem many techniques are available to solve it. In this chapter we will look at a few such schemes, and we will develop one that is well suited to graphics as it is stable, can be built from simple and easy to implement building blocks and provides higher accuracy than the most commonly used schemes. Before we do this, we motivate the discussion with some simple schemes.

2.2 First order upwinding

There are many ways the advection equation can be discretized in time and space. A very simple and effective first order scheme is to discretize in time using a forward Euler scheme and in space using an upwind finite difference scheme. We will consider this scheme with the 1D wave equation

$$u_t + au_x = 0 \quad (2.2)$$

where a is the velocity. This is discretized as

$$\begin{cases} \frac{u_i^{n+1} - u_i^n}{\Delta t} + a \frac{u_{i+1}^n - u_i^n}{\Delta x} = 0 & a > 0 \\ \frac{u_i^{n+1} - u_i^n}{\Delta t} + a \frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 & a < 0 \end{cases} \quad (2.3)$$

where Δt is the time step size, Δx is the grid cell size. One downside of this method is that it has a time step restriction of $\Delta t < \Delta x/|a|$. In addition, this method is only first order accurate in time and space. The second order errors that occur look very much like viscosity, so this error is typically called numerical viscosity. In particular, we can compute the truncation error of this discretization by considering the Taylor expansion of the method i.e.

$$u_i^{n+1} = u_i^n - a\Delta t u_x + a \frac{\Delta t \Delta x}{2} u_{xx} - O(\Delta t^3).$$

The truncation error (defining CFL number 1, i.e. $\lambda = \Delta t a / \Delta x$) is

$$\frac{1}{2}(\lambda - 1) \frac{\Delta t^2 a}{2} u_{xx} + O(\Delta t^3)$$

which fits the model diffusion term Du_{xx} where D is the diffusion constant.

The obvious way to get rid of this diffusion is to use higher order methods that match more terms of the Taylor expansion of u^{n+1} . However, there are pitfalls to higher order advection schemes that must be considered. Numerical advection has been well studied and there are too many schemes to mention here, but we refer the interested reader to [145] and the survey paper.

2.3 Semi-Lagrangian Schemes

For the remaining discussion we will focus on semi-Lagrangian schemes which have been of major interest in computer graphics due to their unconditional stability and simplicity. Courant et al. [29] proposed a simple method of characteristics scheme for discretizing advection equations. These semi-Lagrangian type schemes are popular in many areas (for example in the atmospheric sciences community [142]), because they can be made unconditionally stable. The simplest semi-Lagrangian scheme traces back a straight line characteristic and uses trilinear interpolation to estimate the data. In particular, this scheme traces a characteristic ray backward from the location \mathbf{X} of a grid point we wish to update in the opposite direction of the velocity $\mathbf{V}(\mathbf{X})$ and interpolates at its end point, see Figure 2.1. In other words,

$$\phi(\mathbf{X}, t_{n+1}) = I(\phi^n, \mathbf{X} - \Delta t \mathbf{V}(\mathbf{X}))$$

where $I(\phi, \mathbf{X}_i)$ is linear interpolation of ϕ from the grid points surrounding the interpolation point \mathbf{X}_i . Notice that this method is identical to upwinding in one spatial dimension if a CFL < 1 is used. Also note that like the upwinding scheme above, this approach is first order accurate in space and time.

The order of accuracy can be improved by tracing back curved characteristics and using higher order interpolation (e.g. [103]). However, this can significantly increase the complexity and computational cost of the method especially since high order polynomial interpolants require limiters to avoid oscillations, new extrema and possible instability. See for example the appendix of [45] which illustrates the use of a non-oscillatory cubic spline interpolant. Another way to improve the fidelity of

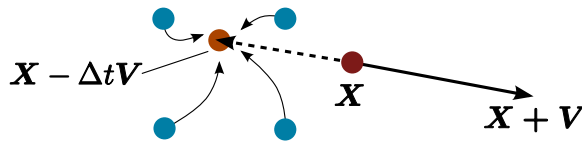


Figure 2.1: The semi-Lagrangian method traces a characteristic ray in the direction opposite to the velocity and interpolates the data from this “upstream” point.

semi-Lagrangian schemes is via auxiliary information. For example, after [141] popularized the semi-Lagrangian method in the field of computer graphics, [45] showed that vorticity confinement [143] could be used to alleviate the high amount of dissipation, allowing for visually intricate, albeit non-physical flows (see also [134] which used vortex particles). Similarly, [39] showed that the simple first order accurate semi-Lagrangian scheme can be used to obtain very accurate level set tracking as long as particles are tracked with higher order accuracy. The original particle level set method [38] used fifth order accurate Hamilton-Jacobi WENO [79] and third order accurate TVD Runge-Kutta [138] making it difficult to extend the method to more complicated data structures. In contrast, the simple semi-Lagrangian approach proposed in [39] allowed for straightforward extension to octree [96, 95] (see also [144]) and run length encoded [76] data structures.

2.4 BFECC and MacCormack Methods

Back and forth error compensation and correction (BFECC) was first proposed in [36] with the aim of reducing mass loss in level set methods (see [113, 112]). The key idea was to realize that a reversible differential equation could be evolved forward and then backward in time to obtain an error estimate. The difference between the final result and the original data is approximately twice the advection error. While inappropriate for non-reversible differential equations such as the heat equation or level set reinitialization [147], it is useful for problematic advection terms in hyperbolic differential equations. First, the forward advection operator A is applied to get $\hat{\phi}^{n+1} = A(\phi^n)$, and then the backward advection operator A^R is applied to get $\hat{\phi}^n = A^R(\hat{\phi}^{n+1})$. The result is used to estimate the error in an advection step as $\hat{\phi}^n - \phi^n = 2e$ or $e = (\hat{\phi}^n - \phi^n)/2$. Next, this error estimate is used to adjust the initial condition of the final advection via $\bar{\phi}^n = \phi^n - e$ and then $\phi^{n+1} = A(\bar{\phi}^n)$. If A was linear, this could be viewed as evolving both the equation and the error forward in time, i.e. $\phi^{n+1} = A(\phi^n - e) = A(\phi^n) - A(e)$. Intuitively, this treats e as if it were a time n quantity that needs to be advected forward in time to time $n + 1$. Since there is no strong evidence that e is a time n quantity, one could just as well add it directly to the

time $n + 1$ state to obtain $\phi^{n+1} = A(\phi^n) - e = \hat{\phi}^{n+1} - e$. And since $\hat{\phi}^{n+1}$ has already been computed in the error estimation step, this strategy alleviates the need to carry out a third advection step making the method significantly cheaper. Figure 2.2 shows a conceptual diagram of these two methods applied.

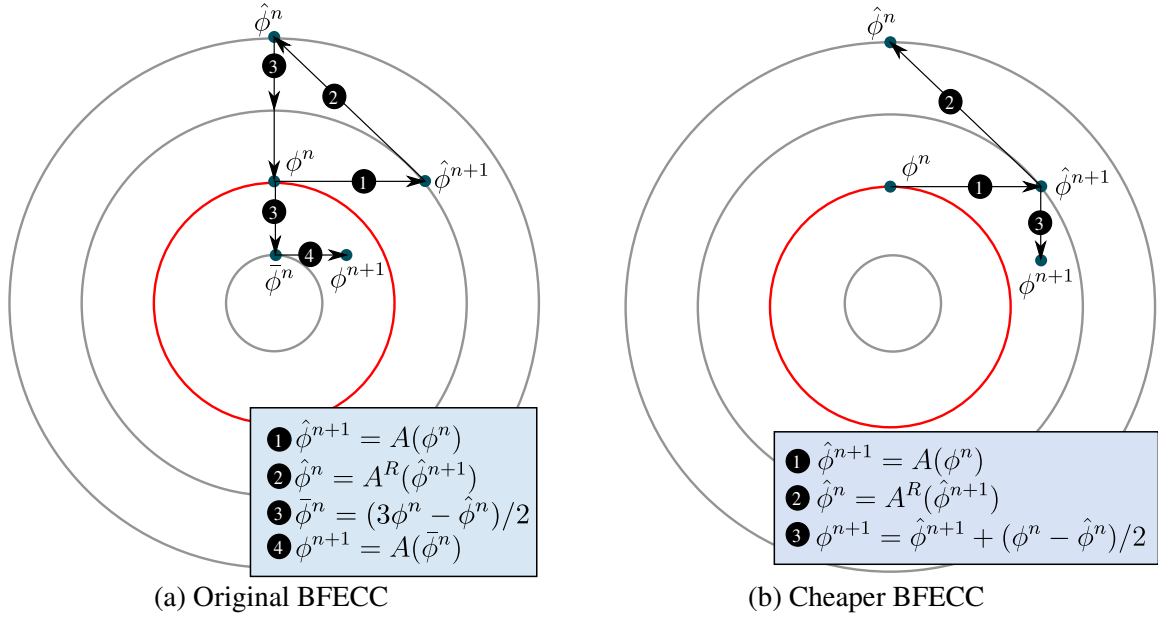


Figure 2.2: A conceptual diagram of the steps of the original BFECC method and the cheaper modified BFECC method (which is actually a modified MacCormack scheme).

Originally, [36] focused on the application of BFECC to the level set equation $\phi_t + V \cdot \nabla \phi = 0$ using forward Euler time integration combined with first order accurate upwinding and downwinding for the forward and backward time evolution, respectively. More recently, a series of papers [84, 85, 37] showed that the forward and backward advection operators could be replaced by a first order accurate unconditionally stable semi-Lagrangian scheme without changing the desirable properties of the method. This produces an unconditionally stable, fully second order accurate semi-Lagrangian method composed of simple first order accurate building blocks. This has the desirable simplicity of TVD Runge-Kutta methods [138], and interestingly contains backward time evolution similar to the higher order accurate TVD Runge-Kutta

methods. Thus, the order of accuracy of the simple semi-Lagrangian scheme can be raised from one to two by increasing the amount of work by a factor of three, since three semi-Lagrangian advections are required. Alternatively, as pointed out above, the error estimate could be added directly to $\hat{\phi}^{n+1}$ removing the last advection step, thus requiring only twice the effort of the first order accurate scheme.

The motivation for this cheaper version of the BFECC scheme came from the MacCormack method [97], which uses a combination of upwinding and downwinding to achieve second order accuracy in space and time. Consider the cheaper version of the BFECC scheme applied to the one dimensional wave equation $\phi_t + \phi_x = 0$, with $\lambda = \Delta t/\Delta x$, $\Delta^-\phi = \phi_i - \phi_{i-1}$ and $\Delta^+\phi = \phi_{i+1} - \phi_i$. The forward advection step is $\hat{\phi}_i^{n+1} = \phi_i^n - \lambda\Delta^-\phi^n$, the backward advection step is $\hat{\phi}_i^n = \hat{\phi}_i^{n+1} + \lambda\Delta^+\hat{\phi}^{n+1}$, and the error estimate is $e_i = (\hat{\phi}_i^n - \phi_i^n)/2 = (\hat{\phi}_i^{n+1} - \phi_i^n + \lambda\Delta^+\hat{\phi}^{n+1})/2$. Finally, $\phi_i^{n+1} = \hat{\phi}_i^{n+1} - e = (\phi_i^n + \hat{\phi}_i^{n+1} - \lambda\Delta^+\hat{\phi}^{n+1})/2$. This is the same as equation (2b) from the original [97] if one switches Δ^- with Δ^+ throughout. That is, [97] proposed unstable downwind differencing for the forward step, and unstable upwind differencing for the backward step, whereas we propose the stable versions for both steps. Note that this slight modification is also typically referred to as a MacCormack method or modified MacCormack method, see e.g. [175] and [1]¹. Thus while this particular modification of BFECC is not novel, it adds insight to the (modified) MacCormack method allowing us to extend it to be unconditionally stable via simple semi-Lagrangian building blocks. Moreover, when viewed in this fashion many other improvements can be made. For example, one can automatically revert to the first order accurate scheme when the upwind and downwind building blocks pull data from non-commensurate regions (e.g. if one gets information from the fluid and the other from a solid wall boundary). It also becomes straightforward to apply limiters to prevent new extrema, which is important since the error correction step of both BFECC and our newly proposed MacCormack scheme can produce new extrema leading to instability.

¹page 224

2.5 Temporal Accuracy and Stability

Consider the model ordinary differential equation $y' = \lambda y$ and the Taylor expansion $y^{n+1} = y^n + \lambda \Delta t y^n + \Delta t^2 \lambda^2 y^n / 2 + \Delta t^3 \lambda^3 y^n / 6 + O(\Delta t^4)$. Forward Euler time integration is $y_{fe}^{n+1} = (1 + \Delta t \lambda) y^n$ with a leading order truncation error of $\Delta t^2 \lambda^2 y^n / 2$. The subsequent step backwards in time from y_{fe}^{n+1} to \hat{y}^n is $\hat{y}^n = (1 - \Delta t \lambda) y_{fe}^{n+1} = (1 - \Delta t^2 \lambda^2) y^n$, so the error is $e = (\hat{y}^n - y^n) / 2 = -\Delta t^2 \lambda^2 y^n / 2$. BFECC computes a new initial value $\bar{y}^n = y^n - e$ which is advanced forward in time via $y_{fb}^{n+1} = (1 + \Delta t \lambda) \bar{y}^n$, i.e.

$$y_{fb}^{n+1} = \left(1 + \Delta t \lambda + \frac{1}{2} \Delta t^2 \lambda^2 + \frac{1}{2} \Delta t^3 \lambda^3 \right) y^n$$

with a leading order truncation error of $-\Delta t^3 \lambda^3 y^n / 3$. The version of the MacCormack scheme we consider in this chapter is

$$y_m^{n+1} = y_{fe}^{n+1} - e = \left(1 + \Delta t \lambda + \frac{1}{2} \Delta t^2 \lambda^2 \right) y^n$$

with a leading order truncation error of $\Delta t^3 \lambda^3 y^n / 6$. That is, the MacCormack method is identical to second order accurate Runge-Kutta for ordinary differential equations. All this can be illustrated by solving $y' = -y/2$ with $y_0 = 1.3$ as shown in Table 2.1.

	Forward Euler		BFECC		MacCormack (RK2)	
Δt	Error	Order	Error	Order	Error	Order
1.000	-6.61e-02	–	-3.35e-02	–	1.73e-02	–
.500	-3.35e-02	1.0	-7.13e-03	2.2	3.40e-03	2.3
.250	-1.67e-02	1.0	-1.59e-03	2.2	7.66e-04	2.2
.125	-8.36e-03	1.0	-3.72e-04	2.1	1.82e-04	2.1

Table 2.1: $y' = -y/2$ with $y_0 = 1.3$. As predicted the error of the MacCormack scheme is about half in magnitude and positive compared the to the back and forth scheme.

Next, we consider the regions of stability. As usual, forward Euler requires $|1 + \Delta t\lambda| \leq 1$ so $\Delta t\lambda = x + yi$ must satisfy $(x + 1)^2 + y^2 \leq 1$. Similarly, BFECC requires $(1 + x + \frac{1}{2}x^2 - \frac{1}{2}y^2 + \frac{1}{2}x^3 - \frac{3}{2}xy^2)^2 + (xy + y + \frac{3}{2}x^2y - \frac{1}{2}y^3)^2 \leq 1$. The graphs of the stability regions are shown in Figure 2.3. Note that the BFECC method includes a significant portion of the imaginary axis similar to third order TVD Runge-Kutta.

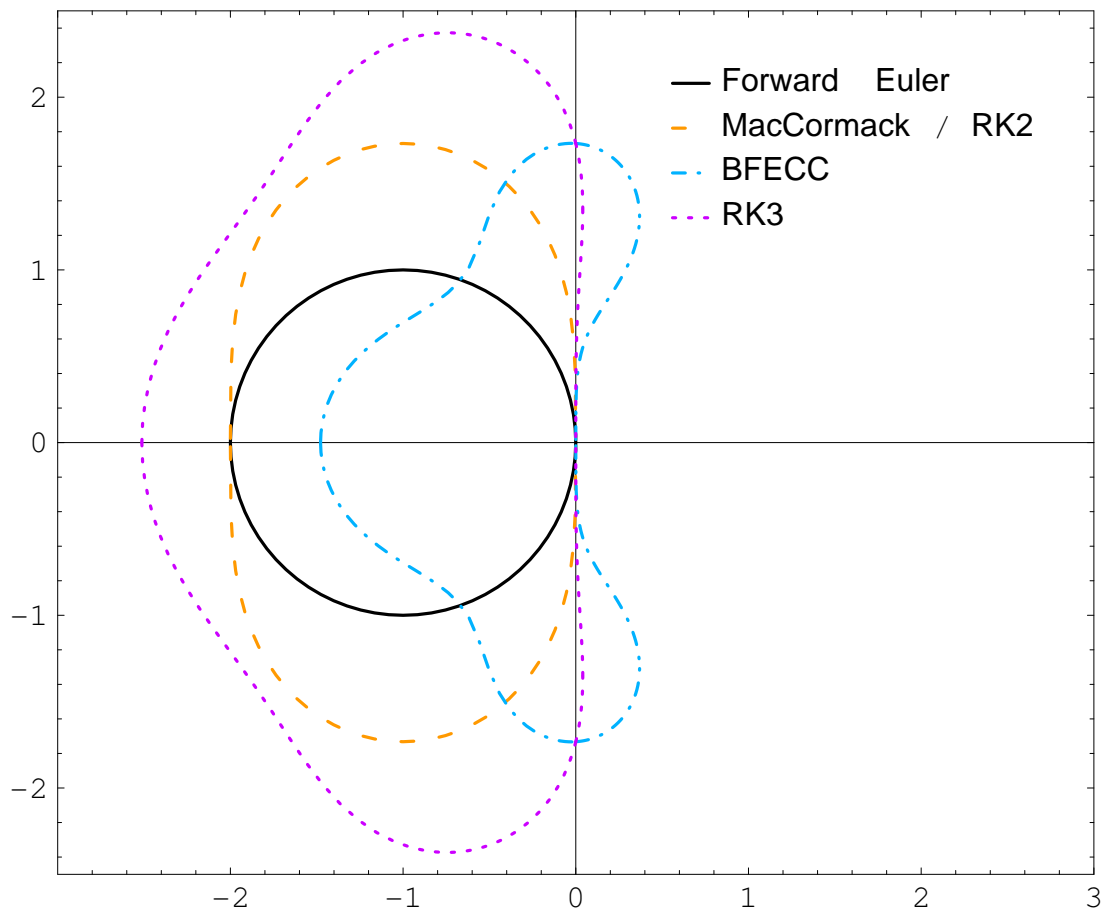


Figure 2.3: Stability regions for ordinary differential equations.

2.6 The Wave Equation

Next we analyze the spatial and temporal properties of the BFEC and MacCormack schemes by revisiting the wave equation. Without loss of generality we will assume that $a = 1$ in (2.2) so that

$$u_t + u_x = 0,$$

alleviating the need to consider the two upwinding cases explicitly. Rearranging the discretization from (2.3) and defining the CFL number $\lambda = \Delta t / \Delta x$,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n) = (1 - \lambda)u_i^n + \lambda u_{i-1}^n.$$

This is used to write BFEC's discretization as a step forward in time,

$$\hat{u}_i^{n+1} = (1 - \lambda)u_i^n + \lambda u_{i-1}^n,$$

followed by a step backward in time,

$$\begin{aligned} \hat{u}_i^n &= (1 - \lambda)\hat{u}_i^{n+1} + \lambda \hat{u}_{i+1}^{n+1} \\ &= ((1 - \lambda)^2 + \lambda^2)u_i^n + \lambda(1 - \lambda)(u_{i-1}^n + u_{i+1}^n) \end{aligned}$$

followed by a correction of the original data using the estimated error,

$$\begin{aligned} \tilde{u}_i^n &= u_i^n - (\hat{u}_i^n - u_i^n)/2 \\ &= u_i^n - \lambda(1 - \lambda)(u_{i+1}^n - 2u_i^n + u_{i-1}^n)/2, \end{aligned}$$

followed by a step forward in time using this error corrected data,

$$\begin{aligned} u_i^{n+1} &= (1 - \lambda)\tilde{u}_i^n + \lambda \tilde{u}_{i-1}^n \\ &= \left(-\frac{1}{2}\lambda^2 + \frac{1}{2}\lambda^3\right)u_{i-2}^n + \left(\frac{1}{2}\lambda + 2\lambda^2 - \frac{3}{2}\lambda^3\right)u_{i-1}^n \\ &\quad + \left(1 - \frac{5}{2}\lambda^2 + \frac{3}{2}\lambda^3\right)u_i^n + \left(-\frac{1}{2}\lambda + \lambda^2 - \frac{1}{2}\lambda^3\right)u_{i+1}^n. \end{aligned}$$

Using the Taylor expansions of u_{i-2}^n , u_{i-1}^n and u_{i+1}^n , and using $u_t + u_x = 0$ leads to

$$\begin{aligned} u_i^{n+1} &= u_i^n - \Delta x (u_i^n)_x \lambda + \frac{\Delta x^2}{2} (u_i^n)_{xx} \lambda^2 - \frac{\Delta x^3}{6} (u_i^n)_{xxx} (-3\lambda^2 + \lambda + 3\lambda^3) \\ &\quad + \sum_{i=4}^{\infty} \xi_{\text{bf}}(i, \Delta x, \Delta t) \\ &= u_i^n + \Delta t (u_i^n)_t + \frac{\Delta t^2}{2} (u_i^n)_{tt} + \frac{\Delta t^3}{6} (u_i^n)_{ttt} \left(-\frac{3}{\lambda} + \frac{1}{\lambda^2} + 3 \right) \\ &\quad + \sum_{i=4}^{\infty} \xi_{\text{bf}}(i, \Delta x, \Delta t) \end{aligned}$$

where $\xi_{\text{bf}}(i, \Delta x, \Delta t) = O(\Delta x^{i-2} \Delta t^2 + \Delta x^{i-3} \Delta t^3 + (i \bmod 2) \Delta x^{i-1} \Delta t)$. Thus, the final terms can be simplified to $\sum_{i=4}^{\infty} \xi_{\text{bf}}(i, \Delta x, \Delta t) = O(\Delta x^2 \Delta t^2 + \Delta x \Delta t^3 + \Delta x^4 \Delta t)$.

The MacCormack method instead uses the error estimate to correct the already advected data resulting in

$$\begin{aligned} u_i^{n+1} &= \hat{u}_i^{n+1} - (\hat{u}_i^n - u_i^n)/2 \\ &= \left(\frac{1}{2} \lambda + \frac{1}{2} \lambda^2 \right) u_{i-1}^n + (1 - \lambda^2) u_i^n + \left(-\frac{1}{2} \lambda + \frac{1}{2} \lambda^2 \right) u_{i+1}^n. \end{aligned}$$

Taylor expansions and $u_t + u_x = 0$ can be used to rewrite this as

$$\begin{aligned} u_i^{n+1} &= u_i^n - \Delta x (u_i^n)_x \lambda + \frac{\Delta x^2}{2} (u_i^n)_{xx} \lambda^2 - \frac{\Delta x^3}{6} (u_i^n)_{xxx} \lambda + \sum_{i=4}^{\infty} \xi_m(i, \Delta x, \Delta t) \\ &= u_i^n + \Delta t (u_i^n)_t + \frac{\Delta t^2}{2} (u_i^n)_{tt} + \frac{\Delta t^3}{6} (u_i^n)_{ttt} \frac{1}{\lambda^2} + O(\Delta x^2 \Delta t^2 + \Delta x^4 \Delta t) \end{aligned}$$

since $\xi_m(i, \Delta x, \Delta t) = O(\Delta x^{i-2} \Delta t^2)$ when i even and $\xi_m(i, \Delta x, \Delta t) = O(\Delta x^{i-1} \Delta t)$ when i is odd.

For a fixed CFL number (i.e. fixed λ) with $\Delta t = \lambda \Delta x$, the final summations in both methods become $O(\Delta x^4)$ or identically $O(\Delta t^4)$. Note that the first three terms of these expressions agree with the exact solution, so the fourth term in each expression is the leading local truncation error. We write the truncation error as

$E(\lambda)\Delta t^3 u_{ttt}/6 + O(\Delta t^4)$ where $E_{bf}(\lambda) = 1/\lambda^2 - 3/\lambda + 2$ for BFECC and $E_m(\lambda) = 1/\lambda^2 - 1$ for the MacCormack method. Note that these results were derived for the wave equation under the assumption that $\lambda \leq 1$. Since an unconditionally stable approach allows for larger λ 's, a similar piecewise analysis can be carried out for $\lambda \in (1, 2]$, $\lambda \in (2, 3]$, etc. to obtain

$$E_{bf}(\lambda) = \begin{cases} 2 + \frac{1}{\lambda^2} - \frac{3}{\lambda} & \lambda \leq 1 \\ 2 - \frac{6}{\lambda^3} + \frac{13}{\lambda^2} - \frac{9}{\lambda} & 1 < \lambda \leq 2 \\ 2 - \frac{30}{\lambda^3} + \frac{37}{\lambda^2} - \frac{15}{\lambda} & 2 < \lambda \leq 3 \end{cases} \quad (2.4)$$

and

$$E_{mac}(\lambda) = \begin{cases} -1 + \frac{1}{\lambda^2} & \lambda \leq 1 \\ -1 - \frac{6}{\lambda^3} + \frac{7}{\lambda^2} & 1 < \lambda \leq 2 \\ -1 - \frac{30}{\lambda^3} + \frac{19}{\lambda^2} & 2 < \lambda \leq 3 \end{cases} \quad (2.5)$$

and so on. Using these expressions, we can compute the leading truncation error at each grid point x_i as $|\Delta t^3 u_{ttt}(x_i, t)E(\lambda)/6|$ and the average error over n grid points as $(1/n) \sum_{i=1}^n |\Delta t^3 u_{ttt}(x_i, t)E(\lambda)/6|$. Computing this for a single time step of $u_t + u_x = 0$ with $u(0, x) = \sin 4\pi x$ and $n = 400$ while varying the CFL number λ produces the graph depicted in Figure 2.4 (left). Solving the same problem numerically for one time step and graphing the computed average error magnitude similarly yields the graph depicted in Figure 2.4 (right).

Next, consider solving the wave equation with $u(0, x) = \sin 4\pi x$ and periodic boundary conditions to a final time of $t = .5$ (one period). Table 2.2 compares the average errors obtained using first order accurate upwinding, BFECC and the MacCormack method with a CFL of .75. Since the CFL is less than one, upwind building blocks are used for both BFECC and the MacCormack method. Spatial refinement shows the expected results for all three methods. Table 2.3 shows similar results for a CFL of 1.75 where semi-Lagrangian building blocks are used for BFECC and the MacCormack method. Note that the the first order accurate upwinding was replaced with the first order accurate semi-Lagrangian method as well.

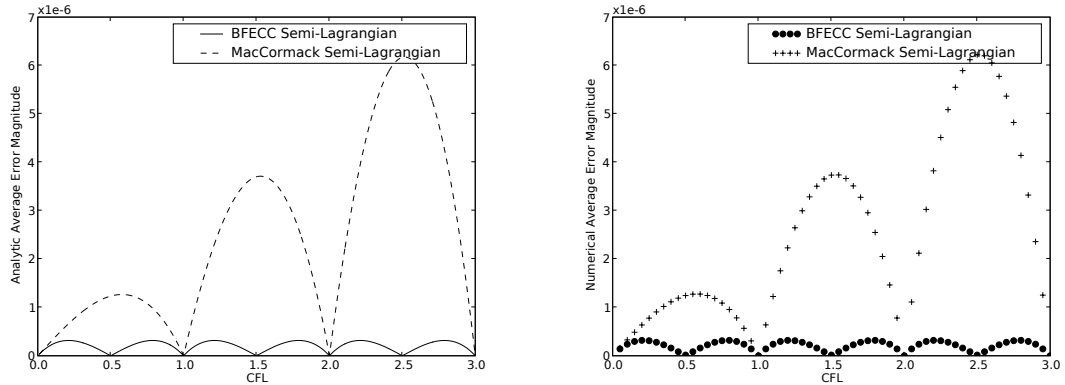


Figure 2.4: Error versus CFL shown for analytic (left) and numerical simulation (right)

n	Upwinding		BFECC		MacCormack	
	Error	Order	Error	Order	Error	Order
101	6.0e-2	–	1.3e-3	–	4.7e-3	–
201	3.1e-2	1.02	3.3e-4	2.09	1.2e-3	1.96
401	1.6e-2	1.01	8.2e-5	1.97	2.9e-4	2.01
801	7.8e-3	1.00	2.1e-5	1.99	7.2e-5	2.00

Table 2.2: Wave equation solved with upwind building blocks and CFL=.75.

n	Semi-Lagrangian		BFECC		MacCormack	
	Error	Order	Error	Order	Error	Order
101	2.6e-2	–	5.6e-4	–	5.3e-3	–
201	1.3e-2	.98	1.4e-4	2.09	1.3e-3	2.21
401	6.7e-3	1.02	3.5e-5	1.96	3.4e-4	2.05
801	3.3e-3	1.02	8.8e-6	1.99	8.4e-5	2.05

Table 2.3: Wave equation solved with semi-Lagrangian building blocks and CFL=1.75.

2.7 New Extrema

Whether first order accurate upwind building blocks (and a CFL less than one) or simple semi-Lagrangian building blocks are used, the first step forward in time and the subsequent step backward in time produce monotone data. That is, \hat{u}^n is bounded by \hat{u}^{n+1} which is in turn bounded by u^n . However, nothing special can be stated about the manner in which the error is computed, or the result obtained when the error estimate is applied to either u^n in BFEC or \hat{u}^{n+1} in the MacCormack scheme. In particular, this error correction step can lead to new extrema and possible instability. Figure 2.5 shows the oscillations obtained when using BFEC and the MacCormack scheme to advect a square wave for one period. A simple choice of limiter consists of limiting the final solution at each grid point to be bound by the values used in computing the first advection step from u^n to \hat{u}^{n+1} . For example, if the base of the semi-Lagrangian ray used in this first advection step interpolated data from the interval $[x_j, x_{j+1}]$, then we would postprocess the final result (from either BFEC or MacCormack) to be clamped between $\min(u_j^n, u_{j+1}^n)$ and $\max(u_j^n, u_{j+1}^n)$. This readily generalizes to multiple spatial dimensions, adaptive and non-Cartesian grids. Another commonly used limiter reverts to a first order accurate method when the higher order accurate method would overshoot (see e.g. [70]). For our method this requires no further computation as an out of bounds value is simply replaced with the value from the first semi-Lagrangian advection step, that is, no error correction is used. Figure 2.5 shows the results obtained by applying the clamping limiter to advection of the square wave initial data, and though it is not depicted we also ran the example using the reversion to first order accurate approach which yielded nearly identical results to clamping. We also compared clamping to reversion using a 2D flow example in Figure 2.11 in which we noticed no significant difference between the two methods. Despite this, we prefer the reversion approach as it yields better results when the semi-Lagrangian MacCormack scheme is applied to free surface flows where the reversible PDE assumption of our modified MacCormack scheme (and also BFEC) breaks down due to the linear extrapolation of velocities to outside the fluid region.

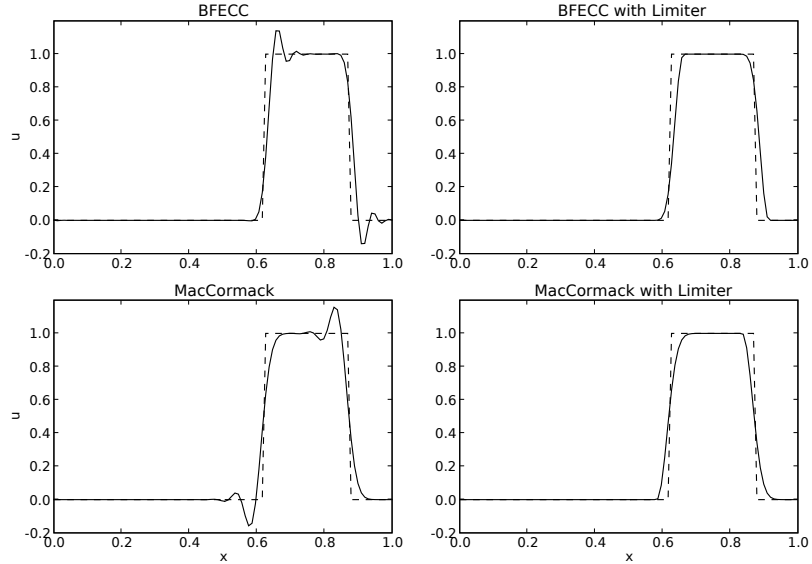


Figure 2.5: Both BFECC and the MacCormack method can create new extrema, as shown here where a square wave is advected for one period. Of course, these oscillations can be avoided with a simple limiting technique.

2.8 Zalesask's Disc

We use the the standard Zalesask's Disc test to illustrate the behavior of these methods for multidimensional advection. The initial data is a slotted circle centered at $(50, 75)$ with radius 15 and a 5 by 25 size slot. We show the results after advecting one rotation (to $t = 628$) in a rotational velocity field $V = \frac{\pi}{314}(50 - y, x - 50)$. Although BFECC was originally proposed for level set methods, this particular example is not meant to test these methods ability to treat interfaces. Instead, we are focusing on how these methods behave for multidimensional advection. Thus, we do not consider reinitialization [147] or the addition of particles [38]. In fact, [39] showed that the addition of accurately advected particles allows one to reduce high order accurate level set advection to first order accuracy without adverse results. Still, accurate multidimensional advection is quite useful for various other problems such as for incompressible flows. In the figures, we plot the zero isocontours for visualization

purposes but stress that the errors are computed across the entire domain (i.e. all isocontours). In particular, we compute the order of accuracy by comparing the error e on the coarse grid to the error e' on the fine grid at coincident node locations as $\log_2 \frac{e^{(i,j)}}{e'^{(2i-1,2j-1)}}$ and average all of these values to obtain the values shown in the tables below.

Figure 2.6 compares first order accurate upwinding with the versions of BFECC and the MacCormack scheme obtained using first order accurate upwind building blocks. Figure 2.7 compares the simple first order accurate semi-Lagrangian method with the versions of BFECC and the MacCormack scheme obtained using simple first order accurate semi-Lagrangian building blocks. Note that the semi-Lagrangian method is fully multidimensional, so even though the CFL is .75 in both Figure 2.6 and Figure 2.7 the resulting numerical methods are all different (in contrast to one spatial dimension where semi-Lagrangian and upwind building blocks are identical for CFL's less than one). Finally, Figure 2.8 repeats the calculations from Figure 2.7 using a higher CFL of 1.75.

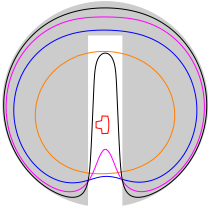
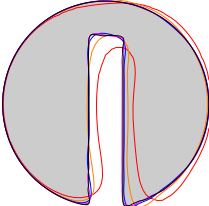
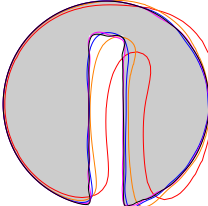
		Upwinding		BFECC		MacCormack	
							
Grid Size		Error	Order	Error	Order	Error	Order
■	101^2	1.8	–	8.9e-2	–	1.6e-1	–
■	201^2	9.6e-1	1.05	3.9e-2	1.79	6.3e-2	1.93
■	401^2	5.2e-1	1.04	1.8e-2	1.77	3.2e-2	1.59
■	801^2	2.8e-1	1.02	7.6e-3	1.98	1.3e-2	2.08
■	1601^2	1.5e-1	1.01	3.1e-3	2.04	5.5e-3	2.08

Figure 2.6: Zalesask’s disc rotation comparing simple upwinding with the versions of BFECC and the MacCormack scheme obtained using first order accurate upwind building blocks (CFL=.75).

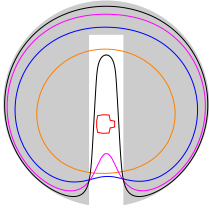
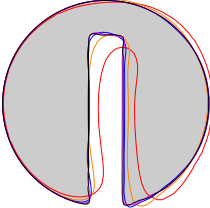
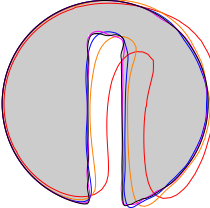
	Semi-Lagrangian		BFECC		MacCormack	
						
Grid Size	Error	Order	Error	Order	Error	Order
■ 101^2	1.8	–	9.9e-2	–	1.7e-1	–
■ 201^2	1.0	1.04	4.1e-2	1.73	6.7e-2	1.95
■ 401^2	5.4e-1	1.03	2.0e-2	1.67	3.3e-2	1.60
■ 801^2	2.9e-1	1.02	8.4e-3	1.93	1.4e-2	2.06
■ 1601^2	1.5e-1	1.01	3.5e-3	2.00	5.9e-3	2.06

Figure 2.7: Zalesask’s disc rotation comparing the simple first order accurate semi-Lagrangian method with the versions of BFECC and the MacCormack scheme obtained using simple first order accurate semi-Lagrangian blocks (CFL=.75).

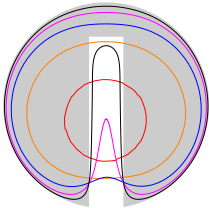
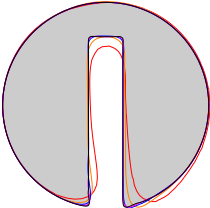
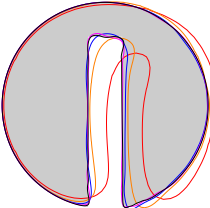
	Semi-Lagrangian		BFECC		MacCormack	
						
Grid Size	Error	Order	Error	Order	Error	Order
■ 101^2	2.0	–	5.4e-2	–	1.5e-1	–
■ 201^2	1.1	1.02	2.4e-2	1.81	6.0e-2	1.82
■ 401^2	5.6e-1	1.02	1.0e-2	1.84	3.0e-2	1.62
■ 801^2	3.0e-1	1.01	4.0e-3	1.98	1.2e-2	1.99
■ 1601^2	1.5e-1	1.02	1.5e-3	2.07	5.2e-3	2.01

Figure 2.8: Zalesask’s disc rotation comparing the simple first order accurate semi-Lagrangian method with the versions of BFECC and the MacCormack scheme obtained using simple first order accurate semi-Lagrangian blocks (CFL=1.75).

2.9 Incompressible Flow

Consider inviscid incompressible flow on a standard MAC grid. First the advection terms are updated to obtain an intermediate velocity, and then a Poisson equation is solved for the pressure which is subsequently used to make the intermediate velocity divergence free.

Consider a $[0, 3] \times [0, 1]$ domain divided into 300×100 MAC grid cells with a radius .125 circle centered at (.5, .5). The horizontal velocity is set to one at the left and right boundaries, and the vertical velocity is set to zero at the top and bottom boundaries. Zero derivative Neumann boundary conditions are used for the pressure on all four boundaries. Figure 2.9 compares the first order accurate semi-Lagrangian method to BFECC and the MacCormack method using first order accurate semi-Lagrangian building blocks. The MacCormack method is shown both with and without the extrema clamping procedure discussed earlier. Note that each component of the velocity field is separately advected and limited. Note that BFECC and the MacCormack method both achieve similarly higher Reynolds number flows than the first order accurate semi-Lagrangian method. Whereas Figure 2.9 was obtained with a CFL of .75, comparable results are shown in Figure 2.10 with a higher CFL of 1.75. In Figure 2.11 we depict the results of applying the extrema clamping strategies discussed in Section 2.7.

Both BFECC and the MacCormack method do not perform well when information is mixed from disparate regions of the flow. This happens frequently near domain boundaries leading to poor error estimates in these regions. Thus, for both BFECC and the MacCormack method, we revert to the first order accurate semi-Lagrangian scheme at any grid point whose semi-Lagrangian characteristic could source from a solid wall or solid circle boundary (i.e. $\text{CFL} \times \max(\Delta x, \Delta y)$).

Now consider free surface flows which combine incompressible flow with level set advection for interface tracking. We use the basic free surface model described in [40] with the improved second order accurate free surface pressure boundary condition proposed in [41] and the improved fast marching method from [95]. In addition, we use first order accurate semi-Lagrangian level set advection together with second

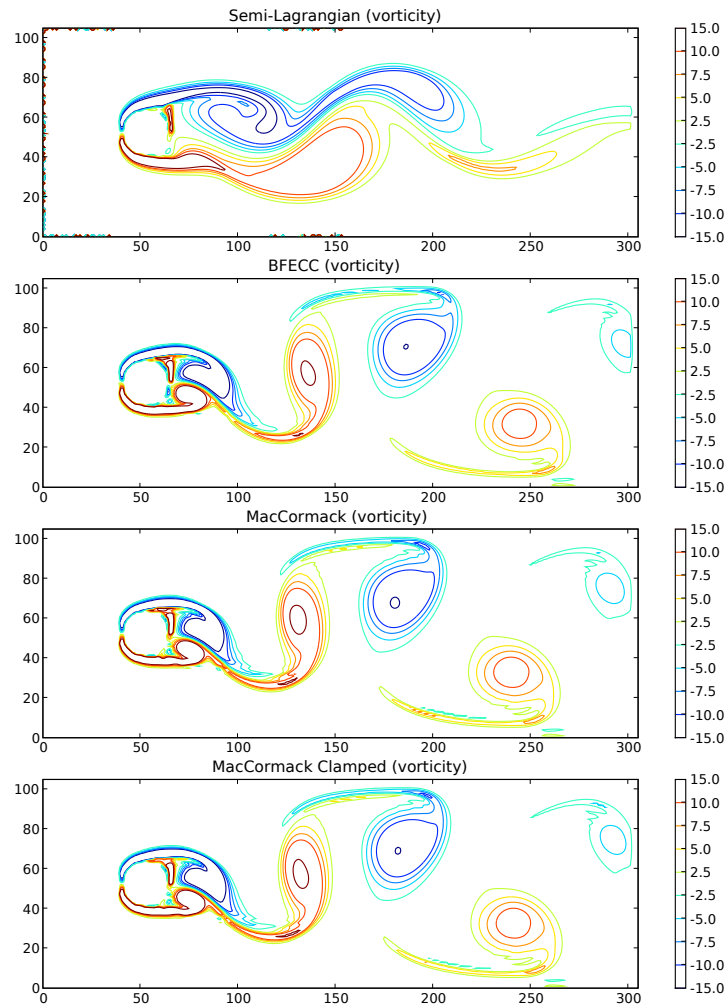


Figure 2.9: Inviscid incompressible flow comparing the first order accurate semi-Lagrangian method to BFECC and the MacCormack method using first order accurate semi-Lagrangian building blocks (CFL=.75).

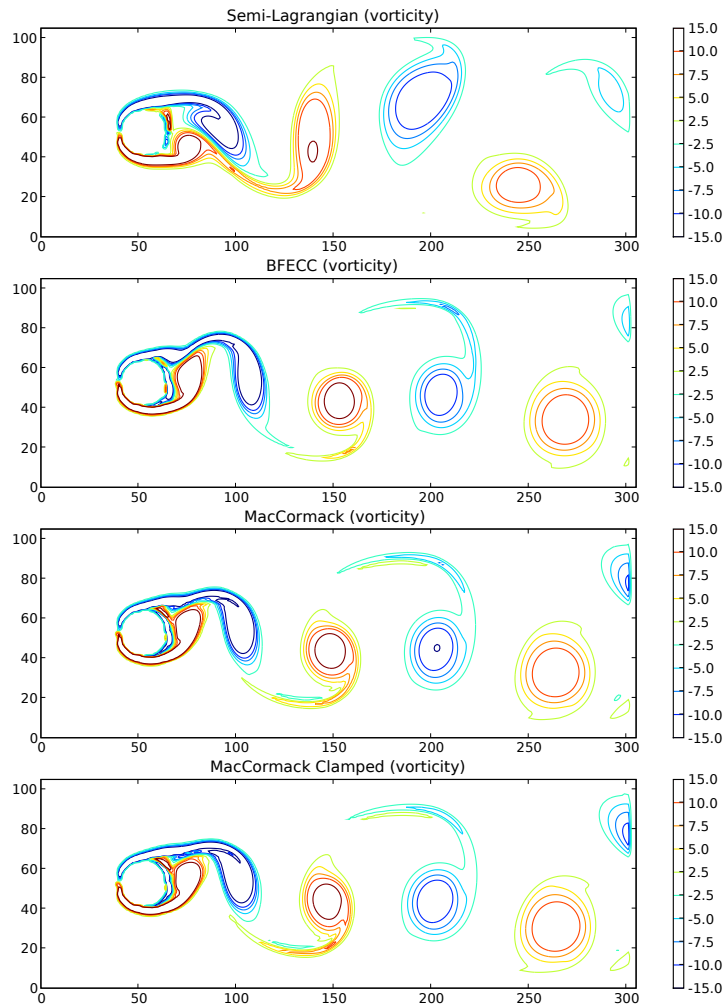


Figure 2.10: Inviscid incompressible flow comparing the first order accurate semi-Lagrangian method to BFECC and the MacCormack method using first order accurate semi-Lagrangian building blocks (CFL=1.75).

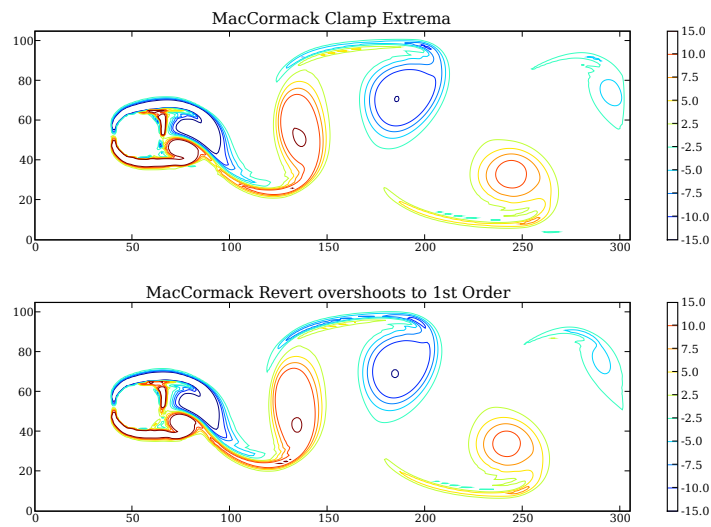


Figure 2.11: Inviscid incompressible flow comparing the semi-Lagrangian MacCormack method with clamping to the semi-Lagrangian MacCormack method with reversion to first order when new extrema are detected (CFL=.75).

order accurate Runge-Kutta advection for the particles because [39] showed that the Particle Level Set Method with a second order accurate particle advection scheme obviates the need for high order level set advection.

As [40] does, we extrapolate a divergence free velocity field across the interface for the level set and incompressible advection, resulting in lower order values in those regions. The BFECC or semi-Lagrangian MacCormack error computation then depends on these less accurate values which creates new extrema which we could clamp. Unfortunately, clamping tends to pollute the solution with the incorrect error correction, thus when new extrema are detected we advocate reverting to first order accurate semi-Lagrangian advection instead.

Consider a spherical radius .5 drop located at (.5, .75, .5) falling (with gravity vector $(0, -9.8, 0)$) into a pool at height .412134 contained in a walled domain of $[0, 1] \times [0, 1.5] \times [0, 1]$ as shown in Figure 2.12, 2.13 and 2.14. As with the flow past circle example, advection was reverted to the first order accurate semi-Lagrangian method in cells close to the domain boundaries. Without any limiter the simulation becomes unstable so we apply the following limiter variants: (1) clamping of new extrema, (2) clamping of new extrema but always using the first order accurate semi-Lagrangian method near the interface and (3) reverting to the first order accurate semi-Lagrangian method at cells where new extrema are found. (1) and (3) do well at producing higher Reynolds number flows while (2) mitigates most of the benefit of the second order accurate approach.

Finally, consider a radius .2 solid ball splashing into a pool of water in a $[0, 1.5] \times [0, 1] \times [0, 1]$ domain. The ball is kinematically moved from $(1.25, .55, .5)$ to $(.8, .1, .5)$ between time 0 and .075 using linear interpolation, and the fluid has the gravity vector $(0, -9.8, 0)$. Figure 2.15, 2.16 and 2.17 show a comparison between the first order accurate semi-Lagrangian method and the three limiter strategies described above. Again we note that without a limiter, the simulation is unstable. Also, as with the flow past circle example and the falling drop example, advection near the domain boundaries and the object boundaries is reverted to the first order accurate semi-Lagrangian method.

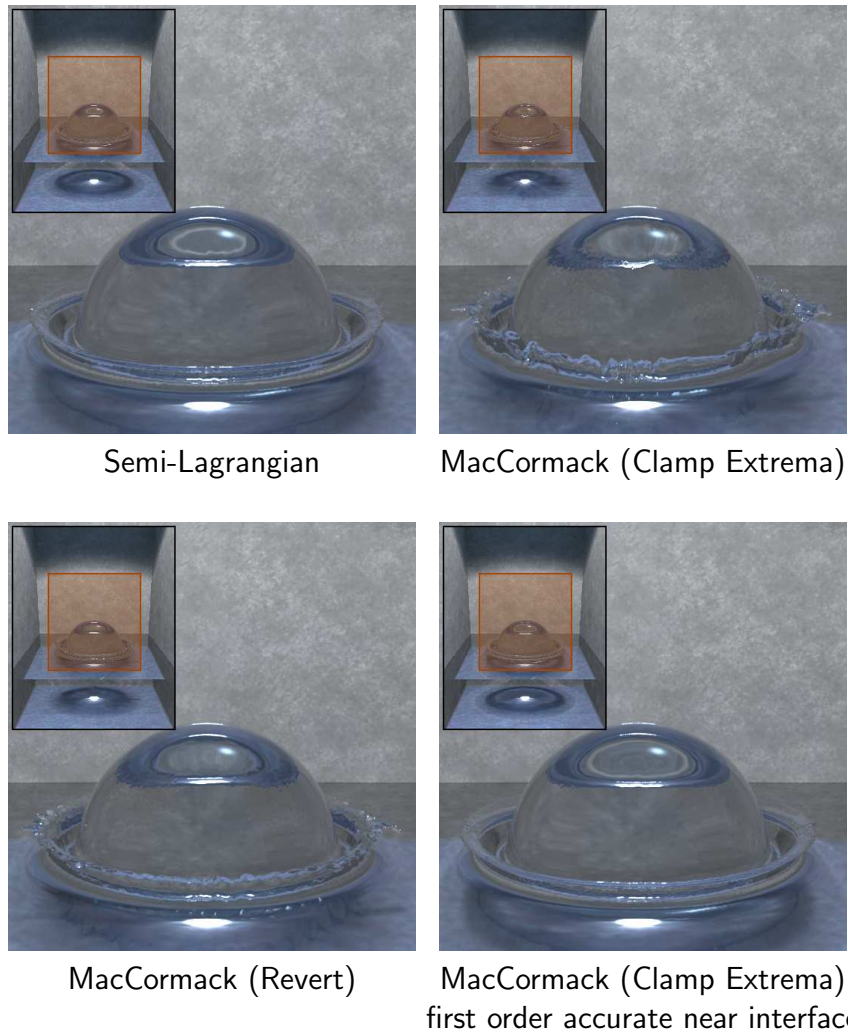


Figure 2.12: Inviscid incompressible free surface flow of a drop falling into a pool. Notice how the MacCormack methods (top right, bottom left) produce additional detail. Using the first order accurate semi-Lagrangian advection near the interface (bottom right) removes much of the advantage of the second order accurate method (bottom right). ($200 \times 300 \times 200$ grid, $t = 6/24$, $CFL=1.75$).

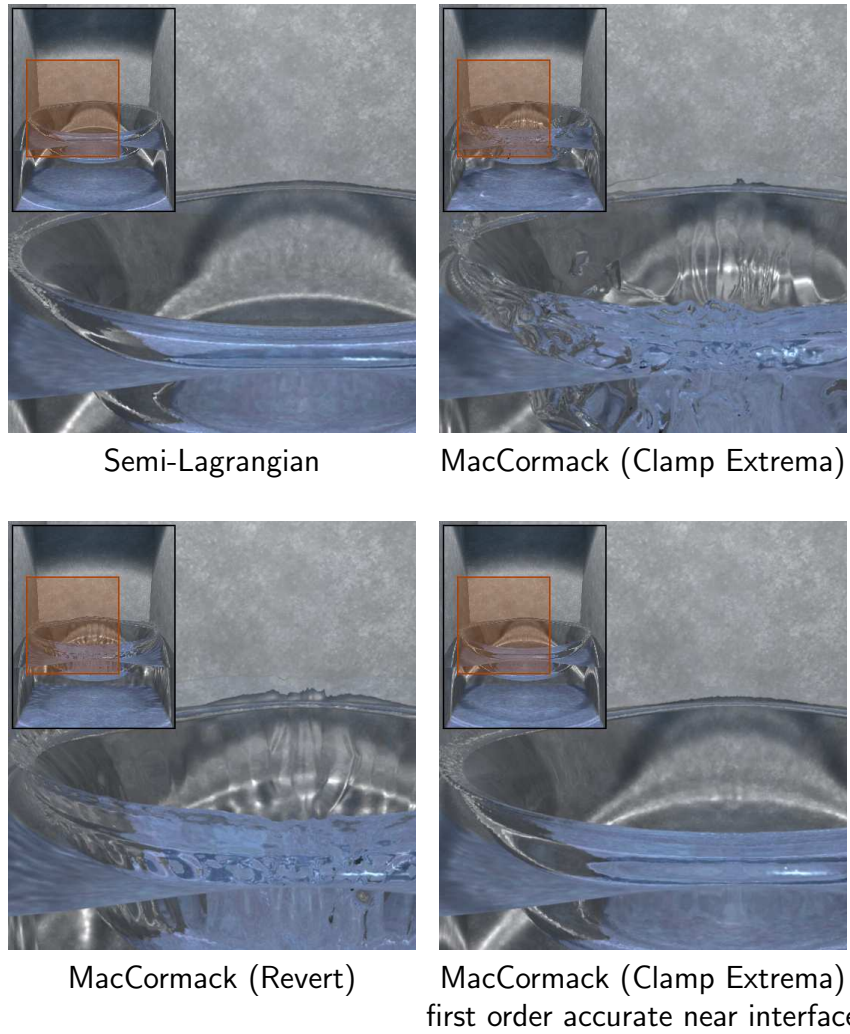


Figure 2.13: Inviscid incompressible free surface flow of a drop falling into a pool ($200 \times 300 \times 200$ grid, $t = 10/24$, CFL=1.75).

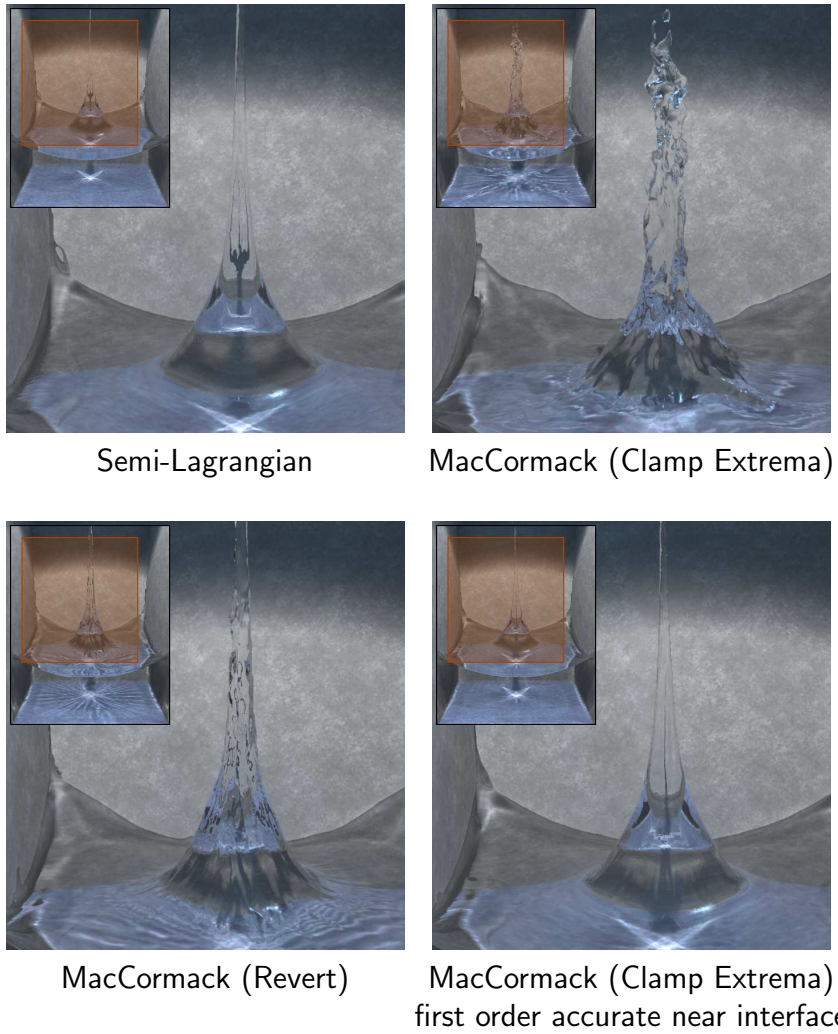


Figure 2.14: Inviscid incompressible free surface flow of a drop falling into a pool ($200 \times 300 \times 200$ grid, $t = 18/24$, CFL=1.75).

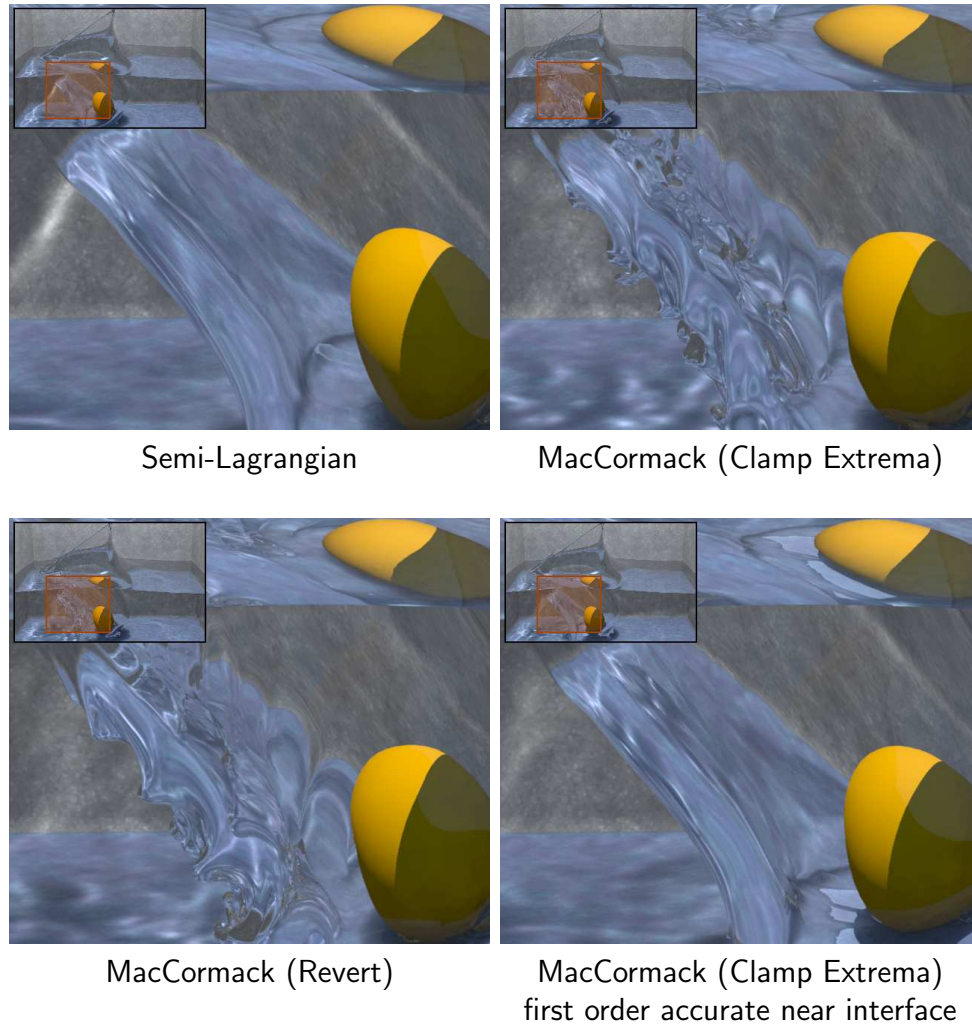


Figure 2.15: Inviscid incompressible free surface flow of a solid ball thrown into a pool. Note that the clamp and reversion MacCormack variants (upper right, lower left) capture the vortex sheeting due to the ball's penetration that is missed when using the semi-Lagrangian method or the clamped MacCormack method that uses the first order accurate semi-Lagrangian method near the interface (upper left, lower right). ($450 \times 300 \times 300$ grid, $t=4/24$)

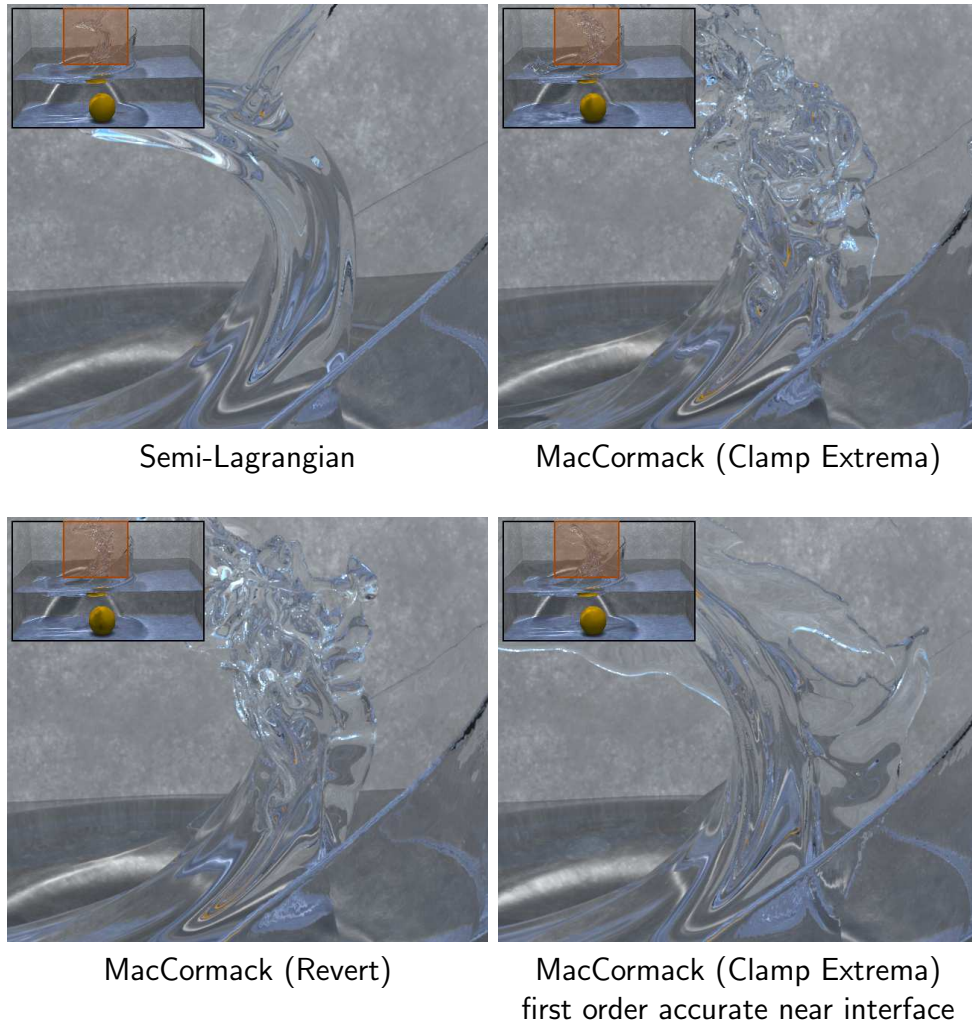


Figure 2.16: Inviscid incompressible free surface flow of a solid ball thrown into a pool. ($450 \times 300 \times 300$ grid, $CFL=1.75$, $t=10/24$)

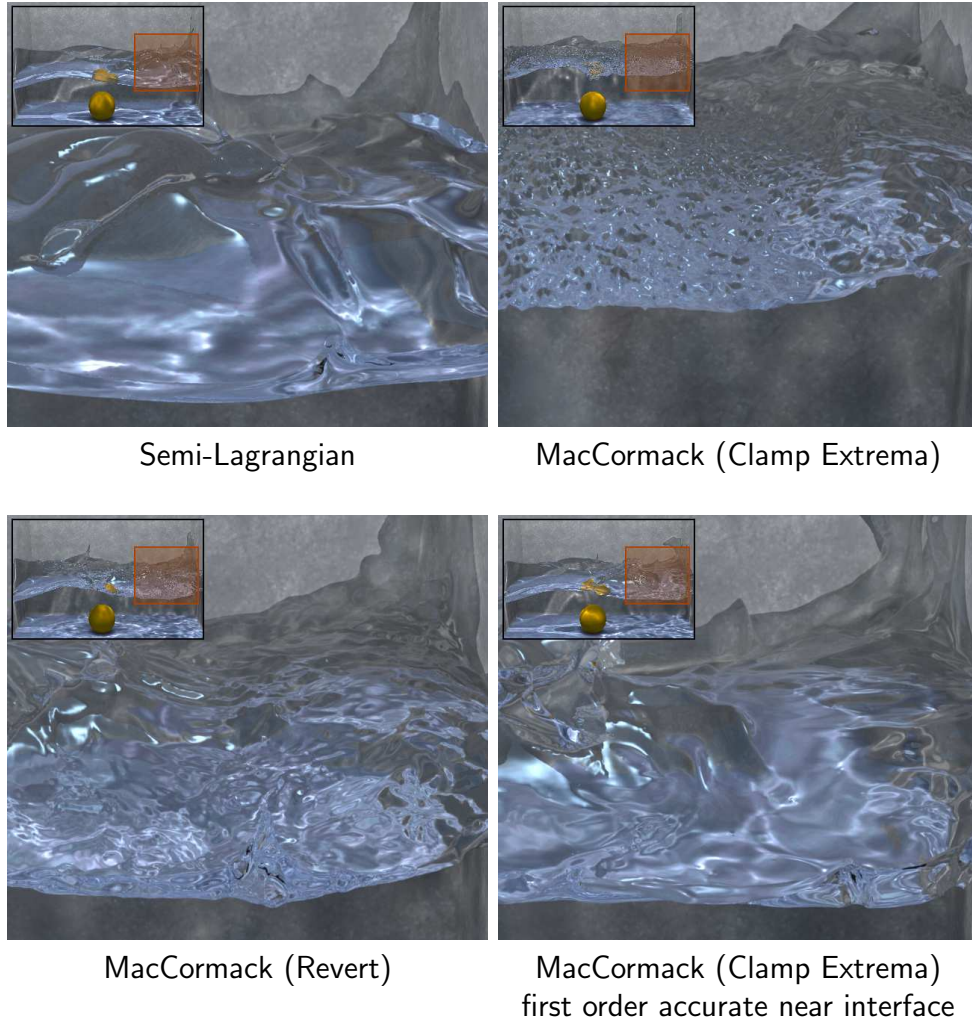


Figure 2.17: Inviscid incompressible free surface flow of a solid ball thrown into a pool. ($450 \times 300 \times 300$ grid, $CFL=1.75$, $t=40/24$)

2.10 Conclusion

Motivated by the BFECC method, we rewrote a version of the MacCormack method into a similar form facilitating its extension to unconditional stability while still preserving second order accuracy in space in time. The MacCormack method only requires two first order accurate advection steps whereas BFECC is 50% more expensive requiring three advection steps. Although the semi-Lagrangian versions of BFECC and the MacCormack scheme are not fully monotone, we proposed a simple limiter that preserves monotonicity. We illustrated the expected behavior of the method on both simple and more complex problems, noting that difficulties could arise if the semi-Lagrangian rays mix information from disparate regions such as from both solid wall boundaries and fluid regions. In those instances, it is simple to revert to the first order accurate semi-Lagrangian method. We also saw that using an increasing number of simple semi-Lagrangian rays can improve accuracy. In particular one ray yields the first order accurate semi-Lagrangian method, two rays yield the second order accurate MacCormack method and three rays yield the second order accurate BFECC method, which has a lower truncation error magnitude than the MacCormack method in some problems. Thus an obvious question is whether one can do better than the BFECC method with three rays, perhaps even obtaining third order accuracy.

Chapter 3

A Hybrid Incompressible Fluid Solver

3.1 Introduction

While the numerical simulation of fluids is now common in the special effects industry, highly turbulent phenomena such as explosions remain challenging. It is difficult to resolve these effects even on the highest resolution grids using state of the art techniques. Regardless, directors frequently desire these exciting and compelling effects, and filming them practically is not always possible especially when complex camera motions (such as flying through an explosion) are required.

In the last chapter we discussed how to evolve quantities given a velocity field, while in this chapter we introduce a technique for obtaining such a field. In particular, we consider the solution of the incompressible Navier-Stokes equation to simulate water, smoke, etc. First, we review the standard velocity/pressure solver, the technique that is commonly used in graphics. We will also review vorticity confinement, a method for increasing the detail of a simulation. In addition, we consider vortex methods, a commonly used alternative formulation for incompressible fluid simulation. These techniques are then combined into a hybrid technique that better handles turbulent flows.

Central to this discussion is the balance between Eulerian (grid-based) and Lagrangian (particle-based) approaches. Fluids typically are simulated using Eulerian approaches, even though this makes them subject to numerical dissipation of velocities. Vorticity confinement, grid subdivision, and higher order methods are used to combat these problems. Some researchers also turn to particle based approaches, which have the ability to conserve quantities exactly. This chapter introduces a method that combines particle methods and grid methods to balance tradeoffs. Moreover, we demonstrate the ability to generate very turbulent effects that cannot be achieved with grid based methods (even with vorticity confinement) or particle based methods alone.

3.2 Pressure Velocity Methods

The incompressible Navier-Stokes equations can be written as

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla) \mathbf{u} + \frac{1}{\rho} \nabla p = \mu \nabla^2 \mathbf{u} + \mathbf{f} \quad (3.1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (3.2)$$

with velocity $\mathbf{u} = (u, v, w)$, pressure p , density ρ , and \mathbf{f} representing the external force. These equations are typically solved on a grid, but particle methods such as SPH (e.g. [33, 65, 122, 107]) avoid the memory requirements of a three dimensional grid, but exhibit other difficulties such as the cost of finding the nearest neighbors, complications involved with enforcing incompressibility, particle redistribution, etc. We can solve Equation 3.1) and Equation 3.2 using a forward Euler time discretization and the Chorin projection method [25] on a MAC grid [68] as follows (ignoring the viscosity term):

1. Advect velocity $\mathbf{u}^* = \mathbf{u}^n - \Delta t (\mathbf{u} \cdot \nabla) \mathbf{u}$
2. Add external velocities $\mathbf{u}^* + = \Delta t \mathbf{f}$
3. Compute pressures by solving $\nabla^2 p = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*$

4. Compute divergence free velocity field $\mathbf{u}^{n+1} = \mathbf{u} - \Delta t \nabla p$.

In fact, both [51] and [141] used this method, leading to the recent popularity of computer graphic smoke simulation using the three dimensional Navier-Stokes equations. However, A major trouble with this method is numerical dissipation in the advection term shown in Chapter 2. Figure 3.2(a) shows a smoke simulation suffering from numerical dissipation. Under refinement, dissipation goes to zero, but refinement into this asymptotic regime is prohibitively expensive. So, some authors have considered adaptive data structures like octrees [96], RLE structures [76] or AMR [9] to allow spatially varying resolution. These schemes are often expensive as the overhead of adaptivity is high, even if theoretically these methods are more efficient. It can also be difficult to choose refinement criteria that ensure adequate grid resolution everywhere interesting flow might develop, and poor refinement criteria result in small scale detail never being formed. Sometimes, a computation can be reduced to a two-dimensional spatial problem. For example, [126] introduced a method for simulating large scale explosions that avoids the high memory requirements of three dimensional grids by simulating a series of two dimensional slices that are placed in three dimensional space and used to define a wind field to advect particles. The technique produced impressive nuclear explosions, but is not as applicable to problems that have less inherent symmetry. Moreover, interesting phenomena such as fuel pocket combustion, etc. cannot be modeled in the free space between slices where interpolation is relied on to generate the velocity field. The other obvious method for reducing numerical diffusion is higher order advection methods like the ones presented in Chapter 2. While these methods reduce diffusion, they cannot eliminate it, and they are more complicated to implement.

3.2.1 Vorticity Confinement

Vorticity confinement in [45] (see also [143]) has been one of the most important enabling technologies for graphics fluid simulation. The technique amplifies existing grid vorticity, allowing much more turbulent flows than the underlying grid would typically support. In fact, with this technique, simple and stable semi-Lagrangian

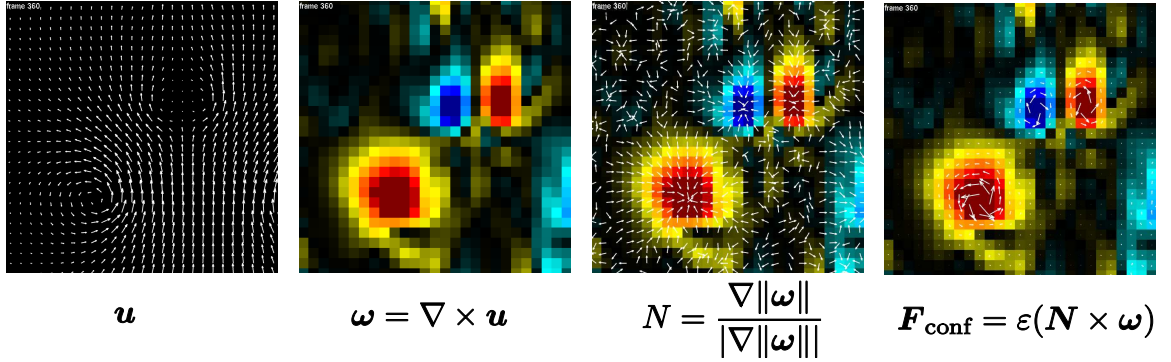


Figure 3.1: Graphical depiction of computation of vorticity confinement fields.

advection approaches have typically been used in lieu of higher order schemes. In particular, the vorticity confinement force is computed by taking the curl of the velocity field to obtain the vorticity

$$\boldsymbol{\omega} = \nabla \times \mathbf{u}. \quad (3.3)$$

Then, a vector field

$$\mathbf{N} = \frac{\nabla \|\boldsymbol{\omega}\|}{\|\nabla \|\boldsymbol{\omega}\|\|}$$

is constructed that points towards the concentrations of vorticity, and this is used to construct an orthogonal force

$$\mathbf{f} = \varepsilon \Delta x (\mathbf{N} \times \boldsymbol{\omega})$$

that amplifies the vorticity concentration, where the scaling by grid size Δx ensures convergence under refinement and the strength is controlled with the scalar ε . See Figure 3.1 for these quantities computed on an example velocity field. This technique works well, producing the examples in Figure 3.2(b,c).

Despite the usefulness of this approach, some major drawbacks remain. For example, a three dimensional computational grid requires a lot of memory, so it can be difficult to simulate large scale phenomena. Also, vorticity confinement can only amplify *existing* grid vorticity, so if the resolution of the grid is not fine enough to

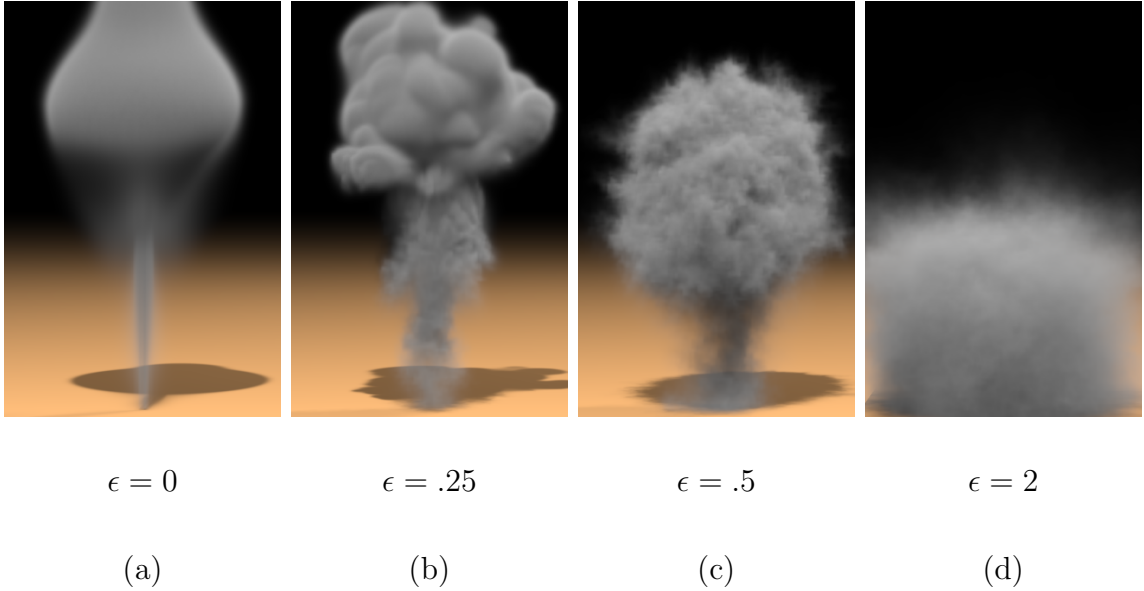


Figure 3.2: Simulations with varying vorticity confinement illustrate that too much confinement causes artifacts and instabilities. In fact, a large value of ϵ actually prevents the smoke from properly rising.

capture object interaction, combusting fuel pockets, upwelling, etc., vorticity confinement cannot recover them. Lastly, if ϵ is set too high, it causes unstable results, like those shown in Figure 3.2(d).

3.3 Vortex Particle Methods

Vortex methods are another class of fluid simulation techniques that are based on vorticity Equation 3.3 instead of velocity and pressure. The vortex evolution equations are derived by taking the curl of equation Equation 3.1, obtaining

$$\omega_t + (\mathbf{u} \cdot \nabla) \omega - (\omega \cdot \nabla) \mathbf{u} = \mu \nabla^2 \omega + \nabla \times \mathbf{f} \quad (3.4)$$

where the velocity advection term has been split into a vorticity advection term $(\mathbf{u} \cdot \nabla) \omega$ and a vortex stretching/tilting term $(\omega \cdot \nabla) \mathbf{u}$. Equation 3.2 vanishes as any vorticially defined field is divergence free.

Although these equations can be solved on a grid (e.g. for graphics [179]), particle based methods (for graphics [52]) have the distinct advantage of avoiding numerical dissipation that smears out the flow making it appear more viscous. A vortex particle stores a vorticity value ω which includes both a magnitude and direction. A kernel (one typically uses a clamped Gaussian with compact support or a tent function) is used to define the vorticity in a region of space nearby the particle. Given a collection of particles, the vorticity at a point is defined by summing the contributions from all nearby particles. The flow evolves as the particles move around and their vorticity values change. For example, viscous flow strongly dissipates large velocity gradients according to the $\mu \nabla^2 \omega$ term. This is typically implemented with some sort of particle exchange method or with the aid of a background grid.

The major disadvantages are in finding boundary conditions for the *vector* valued Poisson equation especially for moving and deforming solid objects, dealing with particle redistribution techniques to adequately represent and resolve the flow, and difficulties associated with the vortex stretching term (that happens to be identically zero in two spatial dimensions as in [52]). In addition, purely particle based vortex methods suffer from many of the same issues that apply to SPH methods. In fact, Neither of the graphic papers mentioned above [179, 52] handled obstacles, and both operated only in two spatial dimensions. For some examples of such methods in computational physics, see [92, 121].

Nevertheless, the advantage of particle-based vortex methods is that carrying vorticity on the particles causes vorticity to be conserved, allowing inviscid, high Reynolds number turbulent flows, i.e. one avoids the grid based damping artifacts (that one uses vorticity confinement to reduce). In addition, particle methods are optimal from a memory storage standpoint for adaptively resolving a flow field.

To combat some of the problems with Lagrangian vortex particle methods, while still maintaining their advantages, authors such as [166, 28]) frequently use a background grid for some parts of the computation. In particular, the solution of Equation 3.4 requires a velocity field, which can be determined from the vorticity values stored on the individual particles. This is still typically a rather complex process, because one has to solve a *vector* valued Poisson equation and deal with complicated

boundary conditions. Besides computing the velocity field, viscosity (vorticity diffusion) and vortex stretching/tilting is also handled on the grid. A step of a vortex particle solver that uses a background grid proceeds as follows:

1. Evolve the particles $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{u}(\mathbf{x}^n)$
2. Map vorticity onto the grid
3. Stretch and tilt vorticity
4. Reconstruct velocity field \mathbf{u}

3.4 Hybrid Solver

Taking stock of the advantages of pressure/velocity (PV) solvers compared to vortex particle solvers (VP), we see that their advantages and disadvantages complement each other perfectly. PV solvers have numerical diffusion where VP solvers have none. VP solvers have difficulty generating a velocity fields where this is easy for PV solvers. VP solvers have difficulty sampling all locations uniformly, whereas PV solvers use a uniform grid. VP solvers also have difficulty with boundary conditions whereas they are intuitive in PV solvers. Thus, our goal is to create a solver that conserves vorticity by using the Lagrangian representation but retains the simplicity and effectiveness of the grid based PV solver. We note that [35] and [47] also had this goal, and they proposed coupling the two methods together in two and three dimensions, respectively, with what they referred to as “ad hoc” techniques. Some of the problems with their coupling procedures were discussed in [48, 49], and we also discuss downsides of their method in Section 3.4.2.

Thus, we will store and evolve particles that hold vorticity. This typically would require us to solve a vector valued Poisson equation to reconstruct a velocity field. However, one of the major benefits of our approach is that this step can be avoided entirely, as we instead use the velocity field determined by solving Equations 3.1 and 3.2 which only requires the solution of a simple *scalar* Poisson equation with straightforward boundary conditions. Moreover, a standard vortex method needs to

carefully place particles to resolve the flow. However, our technique does not require perfect distribution (and redistribution) of particles, because the grid based method adequately resolves the flow at least as well as in [45]. Our vortex particles just provide increased details where they happen to exist. Thus we did not need to redistribute or reseed particles for any of our examples. This is a major contribution of using the grid based solver to determine the velocity field. Additionally, since our goal is to eliminate dissipation, we ignore the diffusion/viscosity term of the vortex equations, similar to how we ignore the viscosity term in Equation 3.1. In the next sub-sections we discuss the precise way the individual solvers are coupled in both directions.

3.4.1 Solving the Particles

Given the velocity field, \mathbf{u} , determined via the grid based method, trilinear interpolation is used to define a velocity for advecting each particle. This accounts for the $(\mathbf{u} \cdot \nabla)\boldsymbol{\omega}$ term in Equation 3.4. We typically inject particles with random initial vorticity at a uniform rate at a source, and let them passively advect through the flow. However, particles could also be created on the fly either near objects or near concentrations of high vorticity, and given the initial vorticity of the surrounding flow. Another nice feature of our approach is that the grid based solver creates a velocity field with proper boundary conditions. And since the particles are advected with that velocity field, they tend to avoid interpenetration with obstacles. However, if particles do enter solid geometry, we could delete them or project them back out of the object using an object level set. Since we use a high density of particles (typically thousands), either option suffices.

Besides advecting the particles, we need to consider the effects of the vortex stretching term in Equation 3.4. This is done by computing the derivatives of the velocity field on the grid with central differences, trilinearly interpolating them to the particle location, and then augmenting the vorticity on the particle with $\boldsymbol{\omega} += \Delta t(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$. In isolation, this term can be thought of as an ordinary differential equation (ODE) that changes both the magnitude and direction of the particle's vorticity. Unfortunately, the vorticity magnitude can exponentially increase when

the ODE has a positive eigenvalue based on the fluid velocity gradient. To ensure stability one could clamp the magnitude, only allow it to decrease, etc. However, since the goal of our particle based method is to preserve vorticity concentration, we rescale the final vorticity to preserve its magnitude in all of our simulations. In that case, the effect of this term is to *spin* the particle’s vorticity vector without affecting its magnitude. This limits the numerical accuracy of the vortex particle method, but is consistent with our reliance on the the grid based method to provide most of the bulk flow features with the vortex particles providing an extra level of detail via vorticity concentration preservation. Along the same lines, we completely ignore the $\nabla \times \mathbf{f}$ term noting that forces (such as buoyancy) still have influence as they affect the velocity field via Equation 3.1.

3.4.2 Vorticity Forcing

Equation 3.4 can be rewritten in conservation form

$$\omega_t^T + \nabla \cdot (\mathbf{u}\omega^T - \omega\mathbf{u}^T - \mu(\nabla\omega)^T - \mathbf{f}^*) = 0 \quad (3.5)$$

where we have written the equations in row instead of column form, and \mathbf{f}^* is the skew symmetric cross product matrix based on \mathbf{f} . This equation demonstrates that vorticity should be conserved (neither created nor destroyed), highlighting one of the major problems with the work of [47]. They used an “ad hoc” method to transmit the vorticity from the particles to the grid based velocity field that does not conserve the total vorticity of that velocity field, i.e. they change the values of the grid based velocity without regard for vorticity conservation. We believe that vorticity conservation is what leads to better quality, especially *visual* quality. Without this, fluid swirling, etc., seems to appear magically. Our key innovation is to use the force \mathbf{f} in Equation 3.1 to drive the grid based velocity field towards the desired vorticity. Although Equation 3.5 dictates that *all* body forces conserve vorticity, the vorticity confinement force is the only one we know of that can introduce vorticity in the fashion required.

The simplest approach is to use the particles' vorticity magnitude *only* (ignoring direction) to define a spatially varying confinement strength ϵ , transferring the particles values of this parameter to the grid with the distribution kernel mentioned above. This allows vorticity confinement to be activated independent of the existing grid based vorticity, but ignores the directional component of the particle's vorticity.

A promising technique is to form an analytic confinement force independently for each particle. The distribution kernel, $\xi_p(\mathbf{x} - \mathbf{x}_p)$, for a particle together with the particle vorticity, $\boldsymbol{\omega}_p$, defines an analytic vorticity $\tilde{\boldsymbol{\omega}}_p(\mathbf{x}) = \xi_p(\mathbf{x} - \mathbf{x}_p)\boldsymbol{\omega}_p$. Choosing a kernel that is rotationally symmetric and strictly decreasing with distance from the particle center implies that $\mathbf{N}_p(\mathbf{x}) = (\mathbf{x}_p - \mathbf{x})/\|\mathbf{x}_p - \mathbf{x}\|$, and the confinement force is then $\mathbf{F}_p(\mathbf{x}) = \epsilon_p(\mathbf{N}_p \times \tilde{\boldsymbol{\omega}}_p)$. We can sum the contributions from all the particles to obtain a grid based force field for use in Equation 3.1. This technique was used to generate Figures 3.3, 3.4 and 3.5. In addition, one can interpolate the grid based vorticity to the particle location and reduce the strength of the particle based force as the grid based vorticity approaches the particle's vorticity. Of course, in practice the grid is typically too coarse for the grid vorticity to match the vorticity of all the particles. Alternatively, one could transfer the magnitude *and* direction of the particle's vorticity to the grid, and compare this to the existing grid based vorticity. The difference between these can be used to calculate a vorticity confinement force (replacing vorticity with this difference in the formulas). However, we have not found these last two options to be necessary.

Finally, we note that vorticity confinement is rather robust for reasonably well chosen parameter values, but can destroy a simulation or cause instabilities if ϵ is set too high as shown in Figure 3.2. Since we use a vorticity confinement style force to drive the grid based vorticity towards the particle based vorticity, similar issues arise in our method. However, as in standard vorticity confinement, a large range of parameter values seem to perform quite nicely. Although one could limit our vorticity confinement forces as the grid based vorticity approaches the particle based vorticity (as mentioned above), we have not found this necessary.

3.5 Examples

We implemented our method on both uniform and octree grids and generated a variety of examples demonstrating its versatility. The extra computational cost incurred by using vortex particles was negligible (less than 5%). Most of our examples used a clamped Gaussian kernel

$$\xi_p(\mathbf{x} - \mathbf{x}_p) = e^{-\|\mathbf{x} - \mathbf{x}_p\|^2/2r^2} / (r^3(2\pi)^{3/2})$$

when $\|\mathbf{x} - \mathbf{x}_p\| \leq r$, and 0 otherwise. In Figure 3.4, we seeded about 6000 particles during an initial divergence driven expansion lasting .5 seconds. Particles are seeded with random position while directions are placed tangent to the cylinder centered at the source region’s midpoint oriented upward. We use a radius extending about 4 grid cells (for octrees we compute the radius using the smallest cells) and a particle vorticity of 2×10^{-3} . Figure 3.3 demonstrates that our technique also works well for liquids. Particles are seeded randomly at the inflow with vorticity pointing up or down to create toroidal eddies characteristic of rivers. To create larger vortices the kernel radius is increased to cover 40 grid cells and the particle vorticity magnitude is 1×10^{-2} for the top figure and 5×10^{-2} for the bottom figure. Figure 3.5 depicts a stream illustrating that we can handle complex geometries. The parameters are similar, except that particles that enter geometry are deleted. Also, we used a 4 grid cell particle radius in order to model a larger scale scene. The technique was also used at Industrial Light + Magic for several feature films including “Star Wars Episode III: Revenge of the Sith”, “Poseidon”, and many others. About 200 particles were used with a radius of about 3 grid cells in a $100 \times 100 \times 100$ simulation, and we used a tent kernel.

3.6 Conclusion

In summary, our method could be viewed as a traditional grid based Navier-Stokes solver with special forces added to obtain interesting fluid flows. These forces are

obtained via a particle based approach to the vorticity formulation of the Navier-Stokes equations. Specifically, the requirements of our method are to (1) use vorticity carrying particles to preserve vorticity concentrations, and (2) target the grid based vorticity towards the particle based vorticity using a *vorticity conserving* body force, based on the successful vorticity confinement approach.

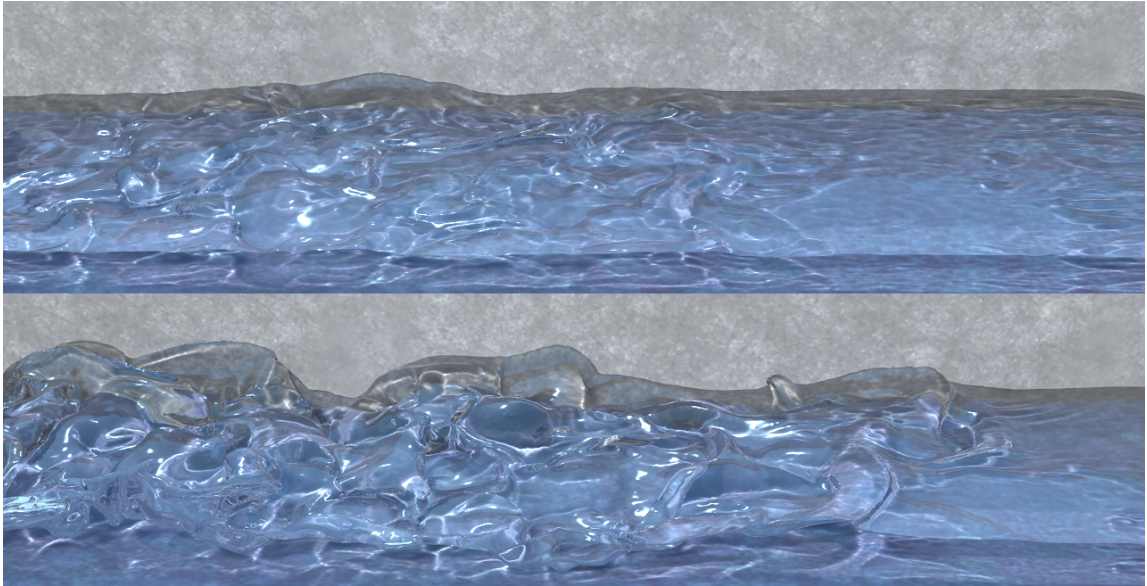


Figure 3.3: Vortex particles seeded at the inflow (left) create turbulence in water flowing from left to right. The top and bottom show lower and higher amounts of particle induced vorticity. ($320 \times 128 \times 320$ effective resolution octree grid, approximately 600 vortex particles)

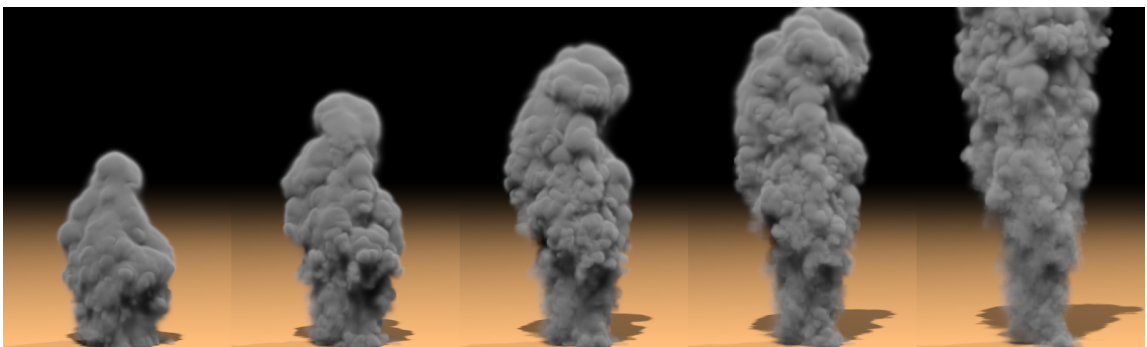


Figure 3.4: Time evolution of a smoke explosion enhanced with vortex particles seeded as the smoke undergoes expansion at the source. ($180 \times 260 \times 180$ uniform grid, approximately 6000 vortex particles)

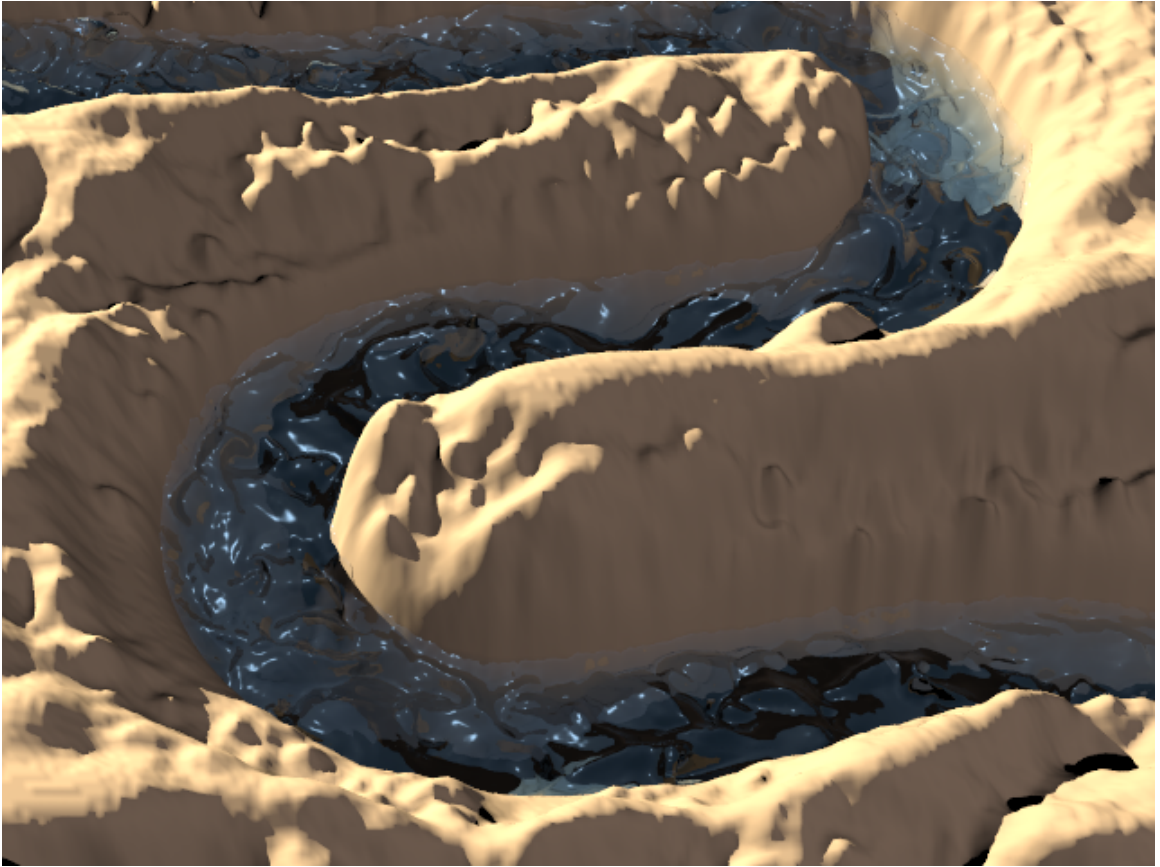


Figure 3.5: Vortex particles interact with complex geometry creating a turbulent water stream. ($272 \times 112 \times 272$ effective resolution octree grid, approximately 800 vortex particles)

Chapter 4

Coupling Thin Shells to Fluids

4.1 Introduction

Visual effects are typically used in entertainment for storytelling, so these effects need to interact with a particular environment and objects. This interaction is responsible for much of the compelling behavior of simulations. An artist typically creates these models and animations by hand, and they must be integrated as boundary conditions for the fluid simulation. This is made difficult by the fact that solids and fluids use different representations. Fluids such as water (e.g. [50, 40]) are simulated using Eulerian techniques (those described in previous chapters) while solid objects such as cloth, rigid bodies, or animations are specified using Lagrangian methods (discussed in later future chapters). If the Lagrangian object is thick, it is typically easy to rasterize it on a grid and enforce boundary conditions in an Eulerian sense, but for thin objects like rigid or deformable shells, direct rasterization is more difficult.

Very little research has been carried out on algorithms that couple infinitesimally thin Lagrangian-based solids to Eulerian-based fluids, and few computational strategies exist. Moreover, they are mostly focused on single phase fluids, whereas our main interest is fluids with interfaces such as between water and air. Probably the most common strategy for *single phase* fluids is based on the immersed boundary method of [116, 117], and [183] used this method to calculate the motion of a thin flexible filament (a curve) in two spatial dimensions. A thin solid object feels and

reacts to fluid forces as molecules collide against it, and the net force on the thin solid comes directly from the pressure differential across it. The immersed boundary method cannot handle this pressure jump and instead forces the pressure to be continuous across the thin solid, and thus (nonexistent) pressure jumps cannot be used to apply forces to the solid. Instead, they simply set the solid velocity to be equal to the velocity of the surrounding fluid, and use ad hoc methods to provide resistance to the fluid motion. For example, [183] smeared out the filament over a number of grid cells converting it into a higher density fluid, and added artificial forces to the right hand side of the Navier-Stokes equations. Similar to penalty methods for rigid body contact constraints, these forces can only coerce a desired fluid reaction and often require small time steps for stability and accuracy. [82] pointed out that smeared out pressures profiles (such as those used in the immersed boundary method) can cause parasitic currents when used to make the velocity divergence free (see also [53]). A key to our method is the replacement of penalty forces with analytic constraints on the fluid velocity forcing it to flow as dictated by the velocity of the solid. Heuristically similar to the analytic methods of [4] for solving contact phenomena in rigid bodies, we replace the stiff inaccurate penalty forces of the immersed boundary method with a robust constraint that requires the solution of a linear system of equations greatly reducing the errors. Conveniently, we are already solving a linear system for the pressure, and it is readily modified to include the no flow constraint exactly as opposed to the only approximate enforcement via penalty forces. This is essentially a sharp interface approach similar in spirit to the immersed interface method (see e.g. [91]). However, we note that neither the immersed interface method nor the immersed boundary method has been used to solve solid/fluid coupling problems in the presence of liquid interfaces or thin films as we do here.

In this chapter we consider the problem of one-way coupling of a time varying thin object interacting with a fluid. We use robust ray tracing to define boundary conditions on the fly that respect the arbitrarily thin Lagrangian object geometry. One way coupling implies that the fluid will feel the effects of the solid, but the solid will not be affected in any way by the pressures from the fluid. This simplifying

assumption is common, especially in visual effects where an artist usually hand animates an object like a boat for storytelling purposes, and the water is simulated around that animation. Though we do not discuss two way coupling, the techniques presented in this chapter are used as the basis for two-way coupling approaches such as [128]. Additionally, in [61], we introduced these one-way coupling approaches and also presented a split two-way coupling approach.

4.2 Previous Work

We use the simple pressure/velocity fluid simulation techniques discussed in the previous chapters together with semi-Lagrangian advection. Though these examples do not use the hybrid vortex particle solver or the MacCormack advection scheme, there is no reason they could not. For simulating water, we use the hybrid particle and implicit surface approach of [50], which led to the particle level set method of [40]. Surface tension [41, 71, 26, 96], viscoelastic fluids [54], control [100, 102], explosions [46], fire [89, 109], etc. could be used with this method, though their use of data quantities would have to be made robust as described in Section 4.3.

Various authors have used simplified fluid dynamics to blow around solid objects, e.g. [178, 176], and many have used simplified wind models to simulate flags flapping in the wind, e.g. [93]. [65] used gridless SPH techniques to couple air flows to hair simulation, but since hair is one-dimensional it does not restrict or contain the fluid as cloth does. SPH models for water were considered in [122, 107], and methods of this type were coupled to deformable solids in [108] using virtual boundary particles. In fact, [88] coupled an SPH model for water to thin deformable cloth pointing out that particle based fluid methods can be coupled without leaking using robust point face collisions, although their method will leak if the time step is not chosen sufficiently (sometimes severely) small. Of course, this can be alleviated with a more robust point face collision method as in [15]. The drawback of using SPH methods is that it is difficult to obtain the smooth liquid surfaces characteristic of level set methods, and recently [19] proposed a method for the two way coupling of rigid bodies to level set based fluid simulations. They first rasterize the rigid body velocity onto the

grid, and then solve the fluid equations everywhere treating the rasterized rigid body velocities as if they were fluid (this was also done in [50] for modeling slip boundary conditions). They then modify the velocities in the rigid body region to account for collisions and buoyancy before averaging them to a valid rigid body velocity with a constant translational and rotational component. The authors point out that their method leaks if the objects are too thin (whereas we consider arbitrarily thin objects), and deformable materials were not considered.

We provide examples of water and smoke interacting with thin rigid bodies and cloth. The novelty of the method is in the treatment of the fluid and the interaction between the fluid and the solid, not in the simulation of the solids themselves. Thus, we use the basic cloth model from [16] including their bending formulation (see also [58]), and the self-collision algorithm of [15]. We note that there are many other interesting strategies for cloth including the dynamics model proposed in [6], the bending model proposed in [23], and the self-interference untangling strategy of [7]. We use a basic method for rigid body simulation, which doesn't need to be any more sophisticated than that in [67, 106] for our purposes. The interested reader is also referred to [5, 60] and the references therein for contact and collision handling techniques.

4.3 Robust One-Sided Interpolation of Data

Our goal is to completely prevent the leaking of smoke and water across thin rigid and deformable solids represented by moving triangles. To do this, we use visibility and occlusion to determine which point combinations can be used to produce interpolations, derivatives, etc. of variables such as ϕ , ρ and \mathbf{u} . This is accomplished via robust ray casting against thickened triangle wedges as in [15], see Figure 4.1 (left). From the perspective of any reference point in space, the world is broken up into three regions: *visible* points, *occluded* points, and points *inside* some triangle wedge of the object. This partitioning is accomplished by casting a ray from the reference point to the point in question as shown in Figure 4.1 (right). The triangle wedges guarantee that only visible points are labeled *visible*, but may incorrectly mislabel some

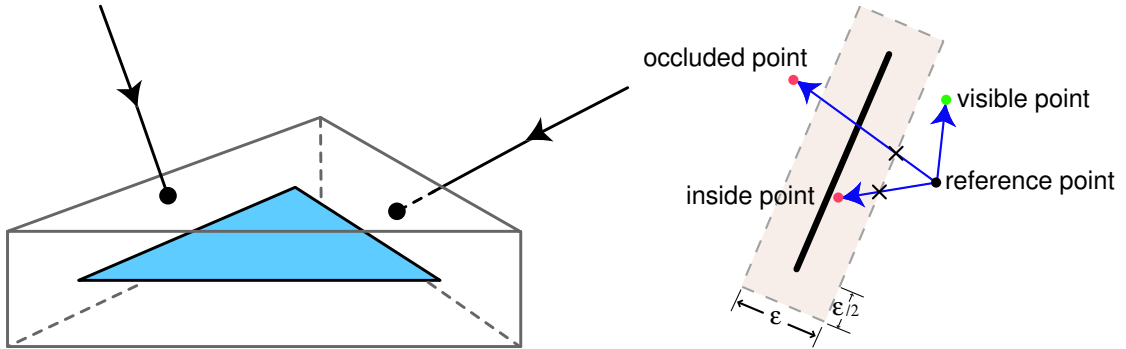


Figure 4.1: We intersect with a triangle wedge that is formed by extending the edges and face in normal directions by $\epsilon/2$ (left). Given a reference point, another point can be classified as *visible*, *inside* an object, or *occluded* by the result of a ray cast (right).

visible points as being *inside* the object or even *occluded* near boundaries leading to a fuzzy interpretation of the object that is robust against roundoff errors. Given a reference point, if one rules out all *occluded* and *inside* points when constructing stencils for interpolation, differentiation, etc. at this reference point, then valid one sided approximations are guaranteed.

When a thin solid moves, a point originally on one side of the object surface may be swept over by the surface and end up on the other side of it. In this case, the values contained by that point are invalidated for all subsequent interpolation, since it represents information from the other side of the object. Detecting such points is crucial to preventing leaks and is accomplished on a per-triangle basis. Each time step, we move the triangle nodes with linear trajectories, and consider a point *invalid* if it intersects the triangle itself (at the center of the wedge) during the time step. Checking this amounts to solving a cubic equation as in [15]. For robustness, we additionally consider a point *invalid* if it is either inside the triangle wedge at the beginning or at the end of a time step. Any point that does not start or stop *inside* a triangle wedge will robustly register a collision with an interior triangle if it crosses from one side of the object to the other. In the case of octrees, refinement leads to new point values that are also marked as *invalid*. Coarsening only involves the removal of nodes, and thus nothing special need be done.

We provide valid values for *invalid* nodes using a Gauss-Jacobi iterative scheme to propagate information. Each iteration, every *invalid* node is assigned the average of its valid *visible* one ring neighbor values and marked *valid*. This technique of averaging uncovered points is similar to the blending methods used by others, see e.g. [8]. Complicated object geometry or folding may produce nodes that are still *invalid* after all iterations are complete. These nodes have no *valid visible* neighbors, and thus we again iterate in a Gauss-Jacobi fashion except this time using specially chosen values when visibility rays intersect an object. For example, we use the object velocity, a zero density, an ambient or object temperature, the positive distance to the object, etc.

A standard axis-aligned box hierarchy is used for the triangulated surface accelerating intersection tests, etc. Moreover, for each triangle, a slightly enlarged bounding box is used to label all the voxels from the fluid simulation that are in close proximity to the surface thus possibly requiring special treatment.

4.4 Fluid Simulation

We use the pressure/velocity fluid solver as outlined in Section 3.2, again ignoring the viscosity term. While we have implemented the algorithm on both uniform and octree grids [96], we gear the exposition toward uniform grids.

4.4.1 Computing the Intermediate Velocity

To update the velocity field we use semi-Lagrangian advection as described in Section 2.3. To do an update of a MAC grid face, we must first obtain a full velocity vector \mathbf{u} by averaging the other two components to the face. However, the averaging operator must consider that the path to a value from which we wish to average may be blocked by a solid. Thus, we ray trace from the face location to the locations we wish to average from and if we hit an object, we average with the object velocity at the intersection point instead. Following this, we have a full velocity vector, which we can use with the semi-Lagrangian method. We trace a semi-Lagrangian ray from \mathbf{x}

to $\mathbf{x} - \Delta t \mathbf{u}$, for each velocity face, intersecting it with a triangle wedge that is double the usual size (i.e. using $\epsilon' = 2\epsilon$ in Figure 4.1 (left)). This guarantees that the base interpolation point is *visible* to the point we are updating, and that the subsequent 8 rays we send out from this base interpolation point can accurately predict visibility (through a 2 ray path) for the interpolation stencil. If any of these 8 secondary rays intersect the object, we use the local object velocity for that term in the trilinear interpolation formula. After computing the intermediate velocity field, the object is moved to its new location and we label all nodes colliding with the moving object as *invalid*. Finally, these nodes are given *valid* values as described in Section 4.3.

For water, gravity is simply added to each velocity field in the usual manner. For smoke, we add source terms that depend on the smoke's density ρ and the fluid's temperature T , i.e. $\mathbf{f} = -\alpha\rho\mathbf{z} + \beta(T - T_a)\mathbf{z}$ where \mathbf{z} is the upward direction, α and β are tunable parameters, and T_a is the ambient temperature. The smoke's density and the fluid's temperature are treated (advection, visibility, cross over, invalid, etc.) together with the velocity as explained above, using a zero density and T_a for visibility rays that intersect the object.

To compute the vorticity confinement force, we calculate the curl of the velocity field $\omega = \nabla \times \mathbf{u}$ at the cell centers, replacing any velocity vectors that are not *visible*. Then the confinement force is computed in the usual way (see Section 3.2.1) at the cell centers and averaged back to the faces (again using robust averaging).

4.4.2 Solving for the Pressure

The intermediate face velocity is made divergence free via

$$\mathbf{u} = \mathbf{u}^* - \Delta t \nabla p. \quad (4.1)$$

where the cell centered pressure values are calculated by solving a Poisson equation of the form

$$\nabla^2 p = \nabla \cdot \mathbf{u}^* / \Delta t. \quad (4.2)$$

This equation is solved by assembling a symmetric system of linear equations, one for each MAC grid cell (that contains fluid) with the pressure defined at the cell center. In the case of water, we set Dirichlet $p = 1$ boundary conditions in air cells. A Neumann boundary condition implies that the pressure derivative at a cell face is zero, and thus the intermediate velocity is not modified as can be seen in Equation 4.1.

We found that thin films of water can quickly compress and lose mass against thin solid objects if one is not careful in how the boundary conditions are handled. In fact, correctly handling the boundary conditions is of utmost importance for mass conservation in general, as discrepancies between the fluid and object velocity cause fluid to flow into or out of an object losing or gaining mass respectively. Our method for handling this is one of the key observations and contributions of this chapter. First, we note that the velocity we compute for the fluid during the divergence free projection will be used in the *next* time step, and thus we need to make this commensurate with what the solid will do in the *next* time step. In order to do this, we calculate the size of the next fluid time step, evolve the solid object forward in time by the size of this time step allowing the solid to take as many substeps as it needs to remain accurate and stable, calculate an *effective velocity* for each node in the solid by dividing its positional change by the size of the next fluid time step, and finally rewind the solid to its current position at the end of the current fluid time step. Now the *effective velocity* represents exactly what the solid will do in the next time step, and we use Neumann boundary conditions to force the fluid to move in exactly this manner allowing for excellent resolution of thin films of water colliding against cloth and thin shells. We cast rays from a cell center to the six neighboring cell centers to see if an object cuts through the line segment connecting the pressures as shown in Figure 4.2. And if so, we set a Neumann boundary condition at the cell face and set the constrained velocity there equal to the appropriate component of the *effective velocity* of the object. Then the divergence is computed in the standard fashion, Equation 4.2 is solved, and the results are used in Equation 4.1.

A rather common difficulty with simulating highly deformable thin objects such as cloth in a fluid flow is that the cloth folds over on itself and pockets of fluid get separated from the flow. These are simple to identify by performing a flood

fill algorithm over the fluid cell centers using the Neumann and Dirichlet conditions as the fill boundaries. If any region is surrounded entirely by Neumann boundary conditions, then the coefficient matrix assembled using Equation 4.2 has a null space corresponding to the vector of all 1's and is not invertible. However, there is a version of the conjugate gradient algorithm that can be applied to this matrix, if we first enforce the compatibility condition [119]. This is enforced independently in each region that has a null space using the area and velocity of the faces on the boundary to calculate the net flow per unit area into or out of the region. Then for each boundary face, we use this and the face area to obtain new temporary velocities that enforce no net flow across the region boundary. Finally, we solve for the pressure and make this region divergence free.

4.4.3 Water

We simulate water using the particle level set method of [40] with $\phi \leq 0$ denoting water and $\phi > 0$ denoting air. Since we only solve for velocity values in the water, each time step we extrapolate the nodal velocities across the interface into a 3-5 grid cell band to obtain velocity boundary conditions. To do this, we first order all the grid cells in the band based on their values of ϕ , noting that this ordering is only valid after reinitializing ϕ to be a signed distance function (see Section 4.4.3). Then we solve the vector equation $\nabla\phi \cdot \nabla\mathbf{u} = 0$ for the nodes in ϕ increasing order. To prevent velocities from leaking across objects, we rule out neighbor values that are not *visible*. It is possible that some points have no *visible* neighbors, and we temporarily label

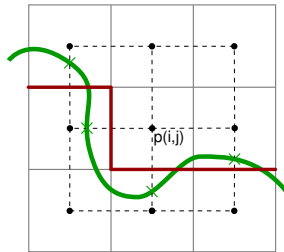


Figure 4.2: Neumann boundary conditions (denoted in bold red) are enforced at a cell face if the ray between two adjacent cell centers (where pressures are defined) intersects an object.

these points *invalid*. After extrapolation is complete, all *invalid* points are given *valid* values as explained in Section 4.3.

Level Set Method

[39] showed that the particle level set method relies primarily on the particles for accuracy whereas the role of the level set is to provide connectivity and smoothness. Thus, they showed that high order accurate level set advection could be replaced with a semi-Lagrangian characteristic based scheme without adversely affecting the accuracy. The level set is defined at the grid nodes, and thus we trace the same semi-Lagrangian rays as for velocity advection. When gathering the 8 values for trilinear interpolation, we replace nodes that are not *visible* with values averaged from a subset of the other 7 grid nodes of the cell whenever possible. Each point that needs to be redefined first looks to see if any of its one ring connected neighbors (in that 8 grid node cell) are *visible* to the base interpolation point. If so, they are averaged to obtain a new value for the node in question. Otherwise, we check and average the three 2 ring neighbors, or if that fails we consider the single 3 ring neighbor. If the process fails, there are no *visible* nodes in the cell and the point in question cannot be updated. We mark this node's ϕ value *invalid* and fix it in a postprocess (see Section 4.3).

The level set is maintained to be a signed distance function using the fast marching method (see e.g. [112]). Typically, the nodes adjacent to the water interface are found by checking for sign changes between neighbors in the Cartesian grid directions (or along edges in the octree grid). We add to this list any node with $\phi \leq 0$ that has a neighbor that is not *visible*, and subtract from this list any node with $\phi > 0$ that does not contain a *visible* neighbor with $\phi \leq 0$. These last two adjustments ensure that an interface exists up against the solid object, and that water does not have influence across the thin triangulated surface. Typically, each node in this list is given an initial ϕ value by considering how far it is from the interface in each of the Cartesian grid directions that have a sign change. However, if the current node has $\phi > 0$, we ignore directions where the neighbor is not *visible*. And if $\phi \leq 0$, we use the minimum between the distance to the solid and $|\phi|$ in directions that are not *visible*. This last

adjustment prevents water from incorrectly sticking to objects. After initialization, we employ the fast marching method in the usual fashion ruling out neighbors that are not *visible* when updating a given point (similar to extrapolation of velocity values).

Particle Level Set Method

Negative particles need to collide with solid objects to prevent water from leaking through those objects, and we collide them using a collision distance that is preassigned to each particle by drawing a random number between $.1\Delta x$ to Δx . To collide a particle with an object, we find the closest point on the object and compute the object normal at that location. We would like the particle to be at least its collision distance away from the object, and if it is not we move it in the normal direction by the required amount. If the particle intersects any object during this move we either delete it or just don't move it. We found that properly colliding negative particles against objects significantly improves the ability to properly resolve thin films of fluid against an object.

The particle velocity is determined by casting rays to the 8 neighboring nodal velocities in the same fashion as discussed above for the base of a semi-Lagrangian ray (see Section 4.4.1). For negative particles that are closer than their collision distance to the object, we clamp the normal component of their velocity to be at least that of the object so that they do not get any closer to it. [39] showed that second order accurate particle evolution was quite important, especially when using the semi-Lagrangian method for level set advection. To achieve this, we first evolve the particles forward in time robustly colliding against the (stationary) object. Then we interpolate a new velocity at this location and average it with the original velocity to get a second order accurate velocity, before moving the particle back to its original location. For negative particles, we clamp the normal component of this averaged velocity if they are either within their collision distance to the object or they collided with the object when they were originally evolved.

Before moving each particle with its second order accurate velocity, we check for intersections between the particle center and the *moving* object. We delete positive particles that intersect the object, but attempt to adjust negative particles using the

triangle they intersect. With the particle and triangle in their initial position, we record which side of the triangle the particle is on using the triangle normal. Then with the particle and triangle in their final position, we move the particle normal to the triangle so that it is on the same side as before and offset by its collision distance in the normal direction. Finally, we check this new particle path against the moving object and delete the particle if it still intersects the object. After advection, all negative particles are adjusted to be at least their collision distance away from the object as discussed above.

After updating both the level set and the particles, we modify the level set values using the particles. This is done in a collision aware fashion using only *visible* particles. The final step in the particle level set method is to adjust the radius of the particles based on the values of ϕ , and possibly delete particles that have escaped too far from the interface. This is accomplished by evaluating ϕ at the center of the particle in the same fashion as is done at the base of a semi-Lagrangian ray. Periodically, every 10-20 frames, particles are reseeded to get a better representation of the interface. Initially, this is performed disregarding the object altogether (for efficiency). Then as a postprocess, we evaluate ϕ at the center of the particle and delete particles with the wrong sign.

4.5 Cloth and Thin Shell Simulation

The novelty of our method lies both in the treatment of the fluid and in the interaction between the fluid and the solid, not in the simulation of the solids themselves. All that is required is positions of the nodes of the triangulated surface at discrete time intervals, and from this we can calculate a velocity for each point assuming that it is piecewise constant between fluid time steps. When velocities within a triangle are required, we interpolate using the barycentric coordinates. For rigid body simulation, we use the method of [60], although our examples require no technology beyond that in [67, 106]. For cloth, we use the basic cloth model from [16] including their bending formulation, and the self-collision algorithm of [15].

4.6 Rendering

To make sure renderings do not exhibit visual leaking artifacts, robust one-sided interpolation of data is also needed. Water is typically rendered using implicit surface ray tracing techniques, and if non-robust interpolation is used, fluid will appear to leak through the objects. Fortunately, the techniques outlined in Section 4.3, work well for rendering as well. We simply use the typical root solving approach for ray tracing implicits, but we use the robust interpolation scheme to evaluate the ϕ implicit function. Since we use the same acceleration structures as we do for the simulation, the additional rays that used for robust interpolation need not be sent if the evaluation point is far from a solid. Moreover, this effectively removes visual artifacts caused by smoke and water showing through to the other side of objects, and air pockets showing through to the smoke and water side.

Subdivision surfaces are typically used to render cloth or even coarse rigid bodies, however, these subdivision schemes may cause visual artifacts if the subdivided points of the solid cross fluid grid points, even with robust interpolation. A similar problem occurs if a cloth simulation is rendered using smooth (loop) subdivision, where the smooth subdivision is self interpenetrating even though the coarse simulation was robustly collision-free. [15] presented a collision-aware subdivision scheme that solves this problem by viewing subdivision as a pseudo-dynamics problem, where a linear (non-smooth planar subdivision) is time evolved to the desired smooth subdivision, while processing collisions (see Figure 4.3). This is possible because both loop and linear subdivision share the same topology and mesh, making particular

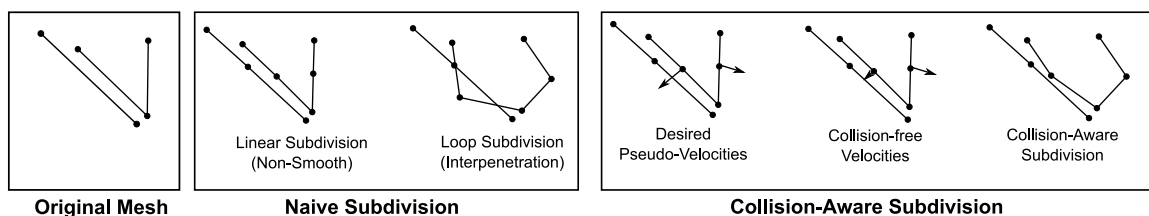


Figure 4.3: Depiction of various subdivision approaches. Notice how the Bridson approach uses a pseudo-velocity modification procedure to create a collision-free subdivision (bottom).

particle positions the only difference between the schemes. Thus, we obtain an effectively smoothness/collision-free subdivision trade off by modulating between the two possible positions.

For fluids, we can perform a similar collision-aware subdivision scheme using Bridson’s technique. We could use his algorithm directly and treat all of the grid points as points that the cloth is not allowed to cross during the evolution from the linear subdivision state to the loop subdivision state. This would work, but it could unnaturally constrict the cloth during subdivision, especially on a very fine grid. Instead, we can allow the cloth to cross fluid grid points, but detect when this happens and invalidate the data that was there, just like the crossover that can occur during a simulation. Revalidation is performed after subdivision using averaging just as in the simulation. See Figure 4.4.

4.7 Examples

We were able to simulate computational grids with effective resolutions as large as $256 \times 256 \times 192$ for the fluid and as many as 90k triangles for the rigid and deforming bodies using a 3 GHz Pentium 4. The computational cost ranged from 5 to 20 minutes per frame, and thus the longest examples took a couple of days.

First, consider examples showing the one-way coupling of this method in isolation. Figure 4.5 depicts a kinematically controlled cup dipped, raised, and poured, demonstrating that our technique can model liquid behavior on both sides of a triangulated surface independently. Figure 4.6 shows similar behavior for more complex geometry.

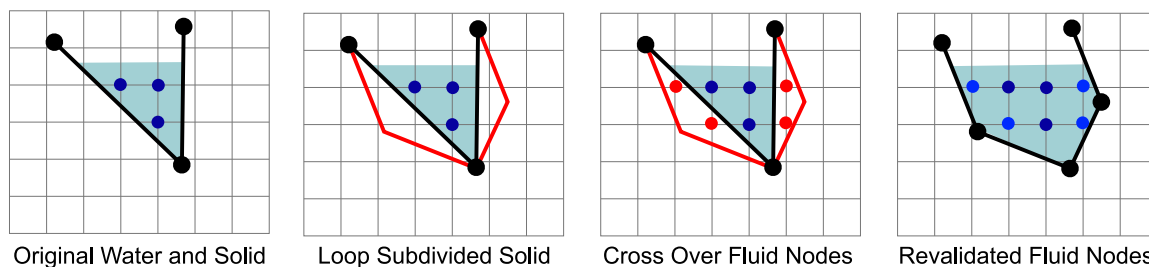


Figure 4.4: Depiction of our fluid-aware subdivision approach.

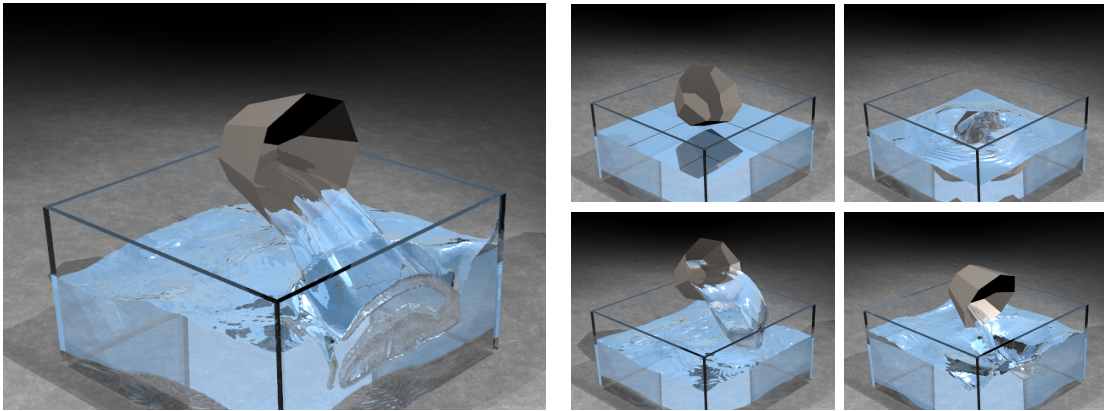


Figure 4.5: A thin rigid kinematically controlled cup is filled with water, and then poured out ($160 \times 192 \times 160$ effective resolution octree).

As stated before, [61] also showed two-way coupling examples using our technique. For example, Figure 4.7 depicts a smoke stream flowing toward a suspended cloth curtain, and the two-way coupling generates interesting wrinkles and folds as well as smoke motion. Two-way coupling is possible for rigid shells as well, and Figure 4.8 shows a fully *dynamic simulation* of a boat floating until a stream of water sinks it. Figure 4.9 shows a stream of water flowing over a piece of cloth demonstrating full two-way coupling of cloth and water. Note specifically that the cloth supports the water (without leaks) and produces highly detailed thin water sheets flowing off the sides. Figure 4.10 depicts a stream of water flowing onto a cloth curtain causing it to deform.

4.8 Conclusions

We have presented a new computational algorithm for the coupling of incompressible flows to thin objects represented by Lagrangian triangulated surfaces. Examples were presented to demonstrate that this algorithm works well for one phase fluids such as smoke and for fluids with interfaces such as water. These examples demonstrated that the method robustly prevents leaking of material across the thin boundaries.

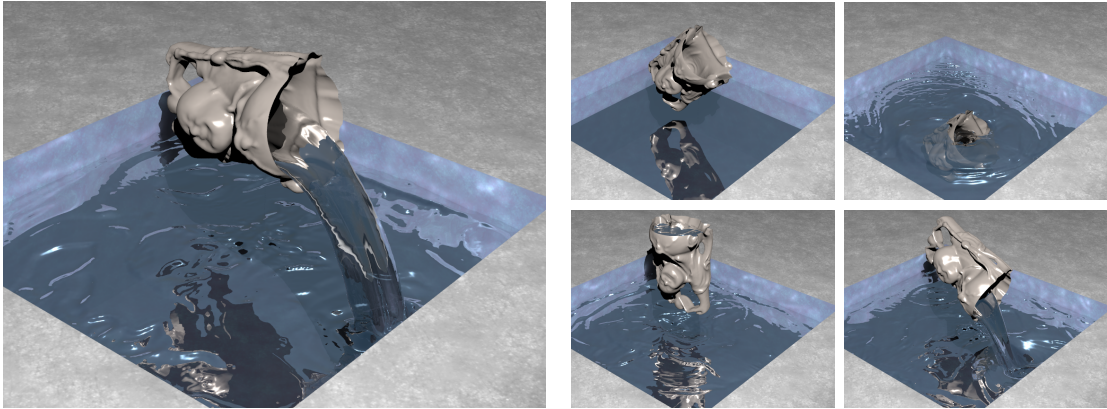


Figure 4.6: A rigid kinematically controlled “Buddha cup” dipped, filled and poured out (192^3 effective resolution octree, 60K triangles in the rigid cup).

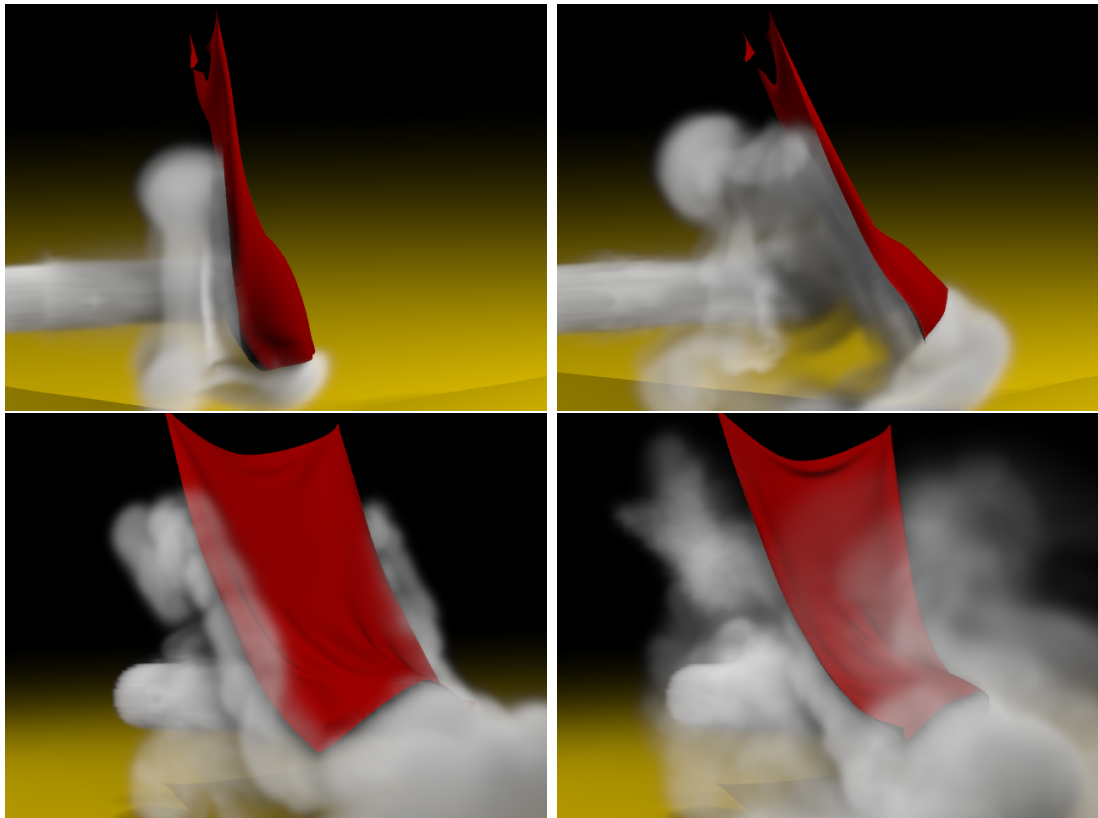


Figure 4.7: Two way coupled cloth and smoke ($210 \times 140 \times 140$ uniform grid, 30K triangles in the cloth).

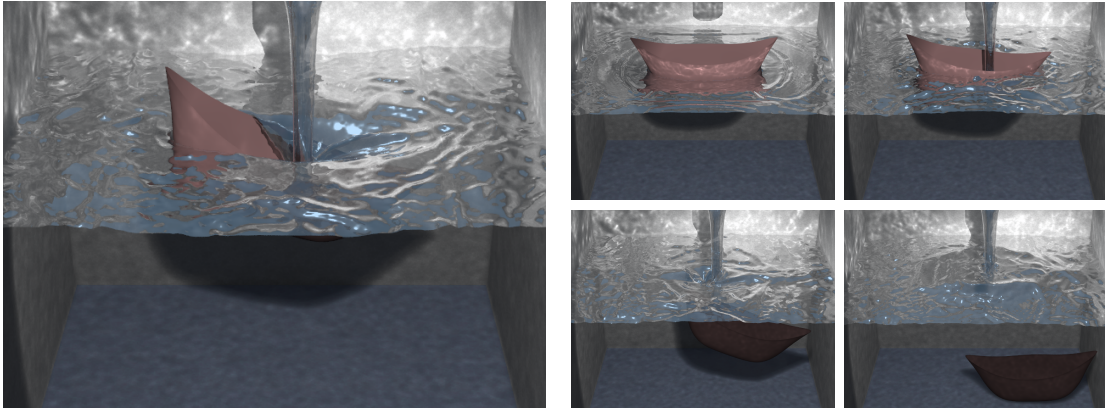


Figure 4.8: A full dynamic simulation of a rigid body shell two way coupled with water. The boat is heavier than the water, but retains buoyancy due to Archimedes' principle (effectively replacing displaced water with the air in its hull). Filling its hull with water causes it to sink, until it dynamically collides with the ground. ($148 \times 148 \times 111$ uniform grid, 2.5K triangles in the dynamically simulated rigid boat)

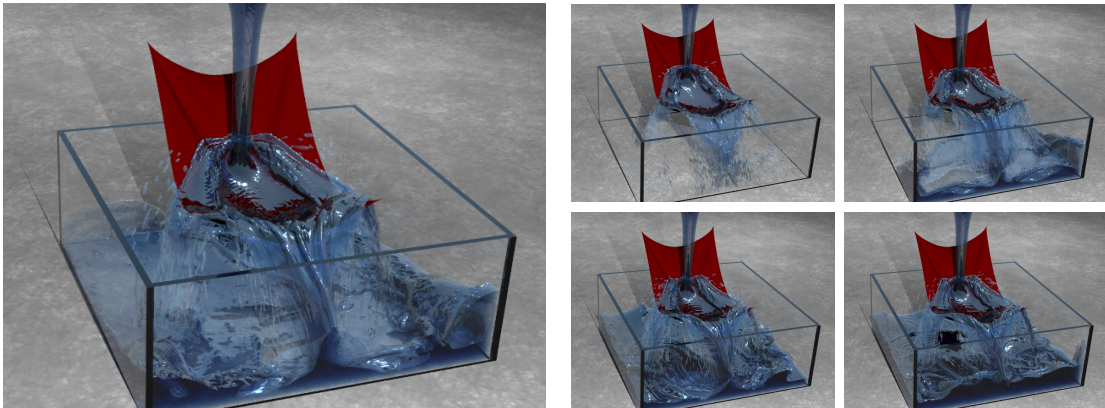


Figure 4.9: Two way coupled water flows over cloth (suspended at four corners), demonstrating that thin objects can support a sheet of water without leaks (200^3 effective resolution octree grid, 30K triangles in the cloth).

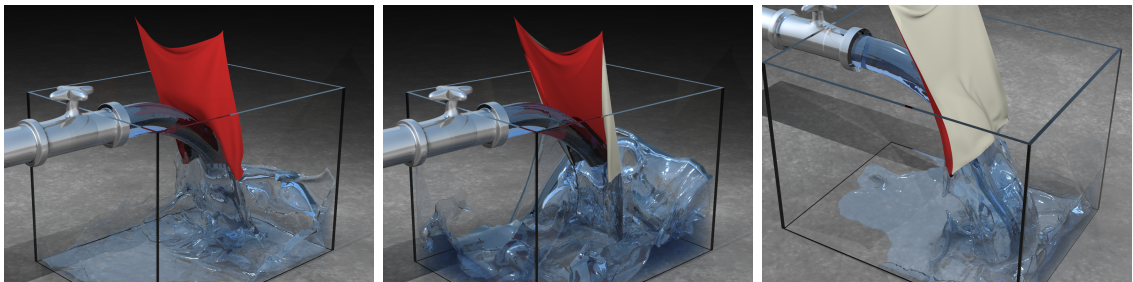


Figure 4.10: Water and cloth interacting with full two way coupling ($256 \times 256 \times 192$ effective resolution octree, 30K triangles in the cloth).

Chapter 5

Solid Simulation

5.1 Introduction

In this chapter we move from fluid simulation to solid simulation, which can be used to simulate hair, flesh, and cloth for synthetic characters in visual effects and games. Compared to fluids, solid simulation typically seeks to resolve a particular geometry. This is because, solids are elastic so that they target some rest shape. Thus, Lagrangian techniques are popular with tetrahedral meshes for flesh, a triangle meshes for cloth, and a segment meshes for hair.

5.2 Meshes

The spatial discretization of a geometry into a particular mesh is very important as it affects what degrees of freedom are possible. For sheets of cloth, we typically use a Herring-Bone mesh to triangulate a quad grid so that we do not bias the bending directions. This is important, as the edges of the triangles are the only places where the mesh can bend. The quality of the mesh has a direct impact on simulation as we will see. Extreme angles, tiny elements, etc. all slow down the simulation by creating bad conditioning and stringent time step restrictions.

To avoid these problems we prefer uniform meshes if possible. If a particular surface or volume geometry needs to be meshed, we use [104], which starts with

a BCC lattice (uniform or adaptive) and moves boundary particles to conform to the input surface via either dynamics or optimization. Irregular Delaunay meshes are another popular choice, but they have more difficulty obtaining well-conditioned meshes. For hair simulation, which we consider in Chapter 7 we show how to construct special meshes for hair.

5.3 Materials

5.3.1 Constitutive Models

Given a mesh and a desired material we wish to simulate, we can create a *constitutive model*, a mapping from the strain (deformation) of a material to its stress. For simplicity, we will consider this intuitively in one spatial dimension. Suppose a piece of material with initial length l_0 is stretched to l . Then the strain is

$$\varepsilon = \frac{l - l_0}{l_0}.$$

The quantity is negative if the length is compressed, positive if stretched, and zero if there is no deformation. We then have a Young's modulus E which can be used to map to a restoring force

$$F = E\varepsilon = E \left(\frac{l}{l_0} - 1 \right).$$

E as a constant is a linear stress-strain relationship—a constitutive model. Constitutive models, stress, strain can be generalized to three dimensions, giving the quantities used for finite element methods. Separating the constitutive model from the computation of a per particle force is quite useful. For example, see [148] for an example of an anisotropic constitutive model that was used for muscle simulation. Nevertheless, in this dissertation we do not use finite elements, electing instead to use springs, because springs tend to be several times faster to compute and tend to be more robust.

5.3.2 Linear Springs

The notion of a one-dimensional spring is easily extended to higher spatial dimensions, by mapping its action to a vector direction. In particular, the force becomes

$$\mathbf{F}_{\text{elastic}} = k \left(\frac{\|\mathbf{x}_2 - \mathbf{x}_1\|}{l_0} - 1 \right) \hat{\mathbf{d}} \quad (5.1)$$

where $\mathbf{x}_{\{1,2\}}$ are the spatial positions of the spring end points at time n , l_0 is the rest length and $\hat{\mathbf{d}} = (\mathbf{x}_2 - \mathbf{x}_1) / \|\mathbf{x}_2 - \mathbf{x}_1\|$. This defines an elastic force that returns a spring to its rest length, however, one also typically requires damping

$$\mathbf{F}_{\text{damp}} = d(\mathbf{v}_2 - \mathbf{v}_1)^T \hat{\mathbf{d}} \hat{\mathbf{d}}.$$

Notice here that we only apply damping in the direction of the spring which avoids much of the floating artifacts caused by a non-directional damping force

$$\mathbf{F}_{\text{ether}} = d(\mathbf{v}_2 - \mathbf{v}_1).$$

We can place these simple linear stress/strain springs on every edge of a triangle mesh for cloth, every edge of a tetrahedron for flesh, or every edge of a segment for hair. Obviously, this does not model any particular material, but nevertheless, it is simple to implement and widely used. One can also extend this definition to provide a piecewise linear stress/strain relationship. This is useful for modeling materials that stretch easily to a point and then become very stiff. One can also differentiate between the stiffness for compression and stretching. The special case of two linear pieces is called a biphasic spring.

5.3.3 Bending

The biggest drawback to the spring based approach is that forces can only be applied in a few discrete directions. Thus, users typically augment their models with additional springs that target other directions. Obtaining the correct bending behavior on cloth is a significant research area, and has been addressed by [23, 16, 58, 10, 155, 165].

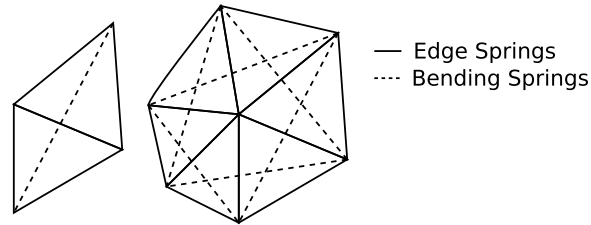


Figure 5.1: Constitutive model of cloth consisting of both springs on triangle edges and springs across shared edge of triangle pairs.

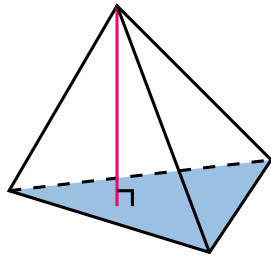
The simplest bending approach is to add another linear spring between the two unshared vertices of a triangle pair that shares an edge. (see Figure 5.1). This works surprisingly well, but has some limitations. In particular, the edge springs are artificially strengthened if the two triangles are planar. Also, if the rest configuration of the two triangles is not coplanar, and the triangle deforms to to be flat, then the bending spring (which is in-plane) cannot restore the appropriate rest configuration bending angle. One approach to correcting this problem is creating a force that preserves a dihedral angle between the two faces [16, 58].

5.3.4 Tetrahedral Altitude Springs

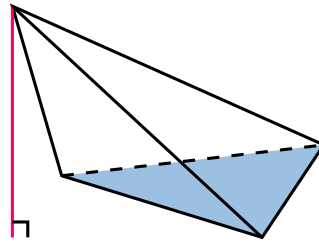
A major difficulty in volumetric simulations (both mass-spring and finite volume) is that a tetrahedron may collapse to zero volume or even invert to negative volume. In traditional finite elements, this inversion causes the simulation to fail (which can be remedied by [77]). In the case of a mass spring model with springs on each of the six tetrahedron edges, the tetrahedron can collapse to a plane and never recover because all spring force directions are in that plane. If the tetrahedron is inverted, the springs have an artificial rest state where the tetrahedron has the same absolute value of volume as the rest shape.

To prevent these problems, [104] places an *altitude spring* between each particle of the tetrahedron and a virtual node projected onto the plane of the opposite face (see Figure 5.2 (a) and (b)). Equal and opposite forces are applied to both the particle and the virtual node where the virtual node distributes its force barycentrically to

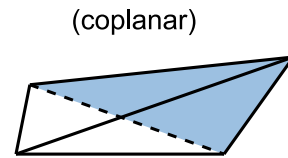
Point/Face Altitude Springs



(a) Spring has all non-negative barycentric weights

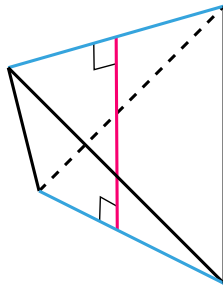


(b) Spring has negative barycentric weights

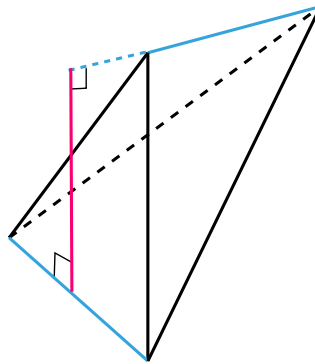


(c) Degenerate: all point/face springs have negative barycentric weights

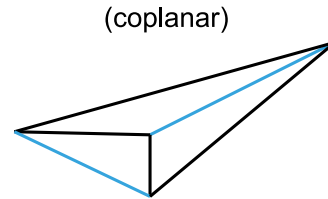
Edge/Edge Altitude Springs



(d) Spring has all non-negative barycentric weights



(e) Spring has negative barycentric weights



(f) Degenerate: all edge/edge springs have negative barycentric weights

Figure 5.2: Varying examples of point/face and edge/edge altitude springs are shown. Examples (a) and (d) represent ideal cases for the two types of altitude springs, whereas (b) and (e) show altitude springs of each type with negative barycentric weights. (c) shows all point/face altitude springs having negative weights, but there is still one edge/edge altitude spring that has non-negative weights. (f) shows all edge/edge altitude springs having negative weights, but one point/face altitude spring has non-negative weights.

the particles of the face. Unfortunately, altitude springs have problems in the case of highly stretched or degenerate tetrahedra as the virtual node can be outside the triangle. In this case, barycentric weights can be negative resulting in arbitrarily large forces on particles of the triangle (i.e. large forces on non-negative weights balance out large forces on negative weights). One might try to correct this by using only point/face pairs with non-negative barycentric weights, but in many configurations, no such altitude springs exist (Figure 5.2(c)).

The key to solving this problem is noting that in the point/face degenerate case two edges are *crossing*, which yields a non-degenerate direction for restoring the positive volume of the tetrahedron. In particular, the edge/edge altitude spring direction is mutually orthogonal to the two lines containing the edges (Figure 5.2(d)). Each spring endpoint is embedded on one of the lines using barycentric weights. Here, the barycentric weights can also be negative if the virtual nodes are not on the segment (Figure 5.2(e)) in which case the edge/edge altitude spring can also produce arbitrarily large forces. Fortunately, we have the following result about tetrahedra.

Theorem 1 *For any non-zero volume tetrahedron, the edge/edge or point/face spring that currently has the least length is guaranteed to have all non-negative barycentric weights, preventing unbounded forces.*

We first show that for a given tetrahedron with at least one positive area triangle, there always exists a point/face or edge/edge pair that has non-negative weights. Consider every non-zero area triangle $\triangle ABC$ and project the fourth point D into its plane as $P(D)$. Then there are three cases: (1) $P(D)$ is inside $\triangle ABC$, meaning the point/face altitude spring has non-negative weights. (2) $P(D)$ is outside $\triangle ABC$ and the convex hull consists of $\triangle P(D)AB$ (relabelling without loss of generality). Then the projection of $\triangle P(D)AB$ is inside $\triangle DAB$ so $P(C)$ is inside $\triangle DAB$ making that a point/face altitude with non-negative weights. (3) $P(D)$ is outside $\triangle ABC$ and the convex hull is a quadrilateral so that segments intersect yielding an edge/edge pair with non-negative weights.

Next, the minimum altitude has non-negative weights because any invalid edge-edge or point/face pair is not the minimum altitude. Suppose an invalid point/face pair, then if the points projection on the face yields a quadrilateral, then a smaller

distance must exist between the resulting edge/edge pair because the projected point is the highest distance from the plane by definition and is incident on one of the crossing edges. A similar argument occurs if the point face is invalid with the convex hull being a triangle. In the case where the edge/edge pair is invalid, then if it is coplanar, it is tied in altitude with everything. Otherwise, the length can be reduced by choosing a different pair. ■

Thus, when performing a simulation of a tetrahedron, we can choose to use the single edge/edge or point/face spring that currently has the least length. Point/face altitude springs compute length as $l = 6V/\|\mathbf{u} \times \mathbf{v}\|$ where \mathbf{u} and \mathbf{v} are the vectors of the base triangle and V is the signed volume of the tetrahedron, while edge/edge altitude springs use the same formula but with \mathbf{u} and \mathbf{v} replaced by the edge vectors. There are seven potential springs, 4 point/face and 3 edge/edge. A valid altitude spring exists unless all particles become colinear, i.e. every tetrahedron face is degenerate and every opposing edge is parallel. Thus to robustly select which spring to use, we find the largest $\|\mathbf{u} \times \mathbf{v}\|$ of any possible edge/edge or point/face spring. If $\|\mathbf{u}\|$ or $\|\mathbf{v}\|$ is too small, then an edge has collapsed, and it will be restored by normal edge springs so no altitude spring is used. If $\sin^2 \theta = \|\mathbf{u} \times \mathbf{v}\|^2 / (\|\mathbf{u}\|^2 \|\mathbf{v}\|^2)$ is too small, then particles in the tetrahedron are colinear, and we use an edge/edge spring with a direction orthogonal to the line. Otherwise, the altitude spring is valid and guaranteed to have non-negative barycentric weights. We illustrate the efficacy of our altitude springs in Figure 5.3(a) where a tetrahedral torus model collapses to zero height and then the stiffness is increased on both the normal edge springs and our altitude springs causing the model to recover (similar to the test in [77]).

We also briefly reconsider the simple bending model for cloth or shells, where linear springs are placed on the triangle edges while bending springs connect opposite vertices across an edge shared by two triangles. If two triangles become coplanar this spring is in-plane and cannot recover the rest curvature, so an axial bending spring can be added that connects a virtual node on the shared edge to a virtual node on the bending spring (Figure 5.4). However, this is exactly an edge/edge altitude spring with the two triangles and the bending spring representing a tetrahedron. Thus instead of an axial bending spring we use our generalized altitude spring model

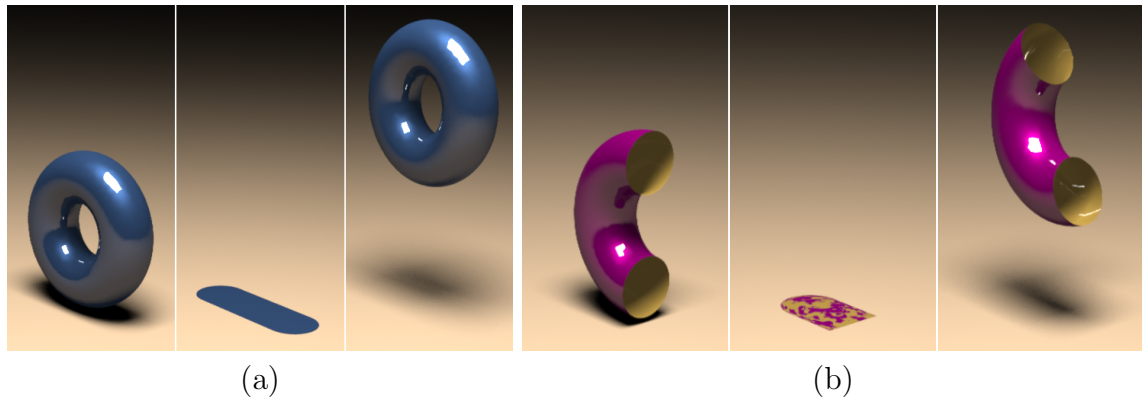


Figure 5.3: A demonstration of our new altitude springs on both volumetric tetrahedral models (left) and thin shell models (right). (a) A volumetric torus with no forces collapses to a puddle and subsequently recovers after the spring strengths are increased. (b) A similar test performed using a thin shell.

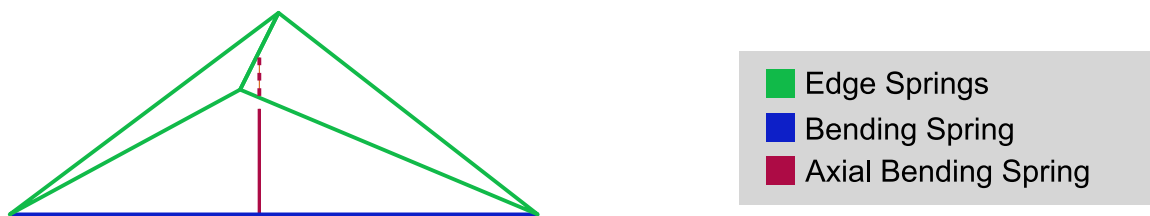


Figure 5.4: A pair of triangles sharing an edge can have its bending modeled by two springs, a bending spring between the unshared vertices and an axial bending spring between a virtual node on the shared edge and a virtual node on the bending spring. The bending spring edge and the shared edge form a tetrahedron so this axial bending spring is really an edge/edge altitude spring.

described above. A simulation using this model is shown in Figure 5.3(b) where a shell model collapses to zero height, and then the stiffness of all springs is increased allowing it to recover.

5.4 Time Integration

Once a constitutive model is chosen, simulation requires integrating the equations of motion through time. Many schemes have been developed for time evolution of solids. Elastic terms typically incur linear time step restrictions whereas damping terms incur quadratic restrictions. Thus, many practitioners have turned to fully implicit methods (see e.g. [6]), which unfortunately tend to damp high frequency elastic behavior, producing simulations that appear to be embedded in a high-viscosity fluid. One can combat this by using semi-implicit approaches such as [151]. More recently, [16] proposed using the Newmark scheme which computes damping forces implicitly and elastic forces explicitly so that there is only a linear time step restriction and high frequency elastic behavior is preserved. For better accuracy, we use the variant of [139], where the Trapezoidal rule application is replaced by a backward Euler step followed by an extrapolation step. This method can be broken up into the following steps:

1. $\mathbf{v}^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2} \mathbf{a}(t^{n+1/2}, \mathbf{x}^n, \mathbf{v}^{n+1/2})$
2. $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1/2}$
3. $\mathbf{v}^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2} \mathbf{a}(t^{n+1/2}, \mathbf{x}^{n+1/2}, \mathbf{v}^{n+1/2})$
4. Extrapolate $\mathbf{v}^{n+1} = 2\mathbf{v}^{n+1/2} - \mathbf{v}^n$

where Δt is the time step, \mathbf{v} is the velocity, \mathbf{a} is the acceleration, and $\mathbf{x}^{n+1/2} = (\mathbf{x}^n + \mathbf{x}^{n+1})/2$.

5.5 Implicit Linear Springs

5.5.1 Implicit Linear Springs

Integrating stiff springs using explicit or even semi-implicit (implicitly damped) time integration can be intractable due to stringent time step restrictions. To increase the efficiency of our simulations we could use a fully implicit spring model, but this usually requires Newton-Raphson iteration ([6] used one iteration) as well as techniques to symmetrize the forces. Instead, we propose a new discretization of linear springs with elastic forces that are truly linear in position.

Consider the spring Equation 5.1) at time t^{n+1} between two points \mathbf{x}_1^{n+1} , \mathbf{x}_2^{n+1} is

$$\mathbf{F}^{n+1} = k (\|\mathbf{x}_2^{n+1} - \mathbf{x}_1^{n+1}\|/l_0 - 1) \hat{\mathbf{d}}^{n+1}$$

where $\hat{\mathbf{d}}^{n+1} = (\mathbf{x}_2^{n+1} - \mathbf{x}_1^{n+1})/\|\mathbf{x}_2^{n+1} - \mathbf{x}_1^{n+1}\|$ is the spring direction. If we rewrite the force as

$$\mathbf{F}^{n+1} = \frac{k}{l_0} \left((\mathbf{x}_2^{n+1} - \mathbf{x}_1^{n+1})^\top \hat{\mathbf{d}}^{n+1} - l_0 \right) \hat{\mathbf{d}}^{n+1}$$

it is clear that fixing the spring direction at time t^n makes the force linear in \mathbf{x}^{n+1} , i.e.

$$\mathbf{F}^{n+1} = \frac{k}{l_0} \left((\mathbf{x}_2^{n+1} - \mathbf{x}_1^{n+1})^\top \hat{\mathbf{d}}^n - l_0 \right) \hat{\mathbf{d}}^n.$$

Substituting $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \mathbf{v}_i^{n+1}$ yields

$$\mathbf{F}^{n+1} = \frac{k}{l_0} \left((\mathbf{x}_2^n - \mathbf{x}_1^n)^\top \hat{\mathbf{d}}^n - l_0 \right) \hat{\mathbf{d}}^n + \Delta t \frac{k}{l_0} (\mathbf{v}_2^{n+1} - \mathbf{v}_1^{n+1})^\top \hat{\mathbf{d}}^n \hat{\mathbf{d}}^n.$$

The first term is the typical explicitly integrated elastic force, and the second term is the typical damping force from a semi-implicit method except that the damping coefficient is fixed at $\Delta t k/l_0$. Notably, this indicates that if one adds the exact correct amount of damping to the semi-implicit discretization, then the spring remains stable. In summary, springs with fixed directions have elastic forces that are linear in the position and only require a single implicit linear solve for the damping forces to be unconditionally stable. Of course the usual damping can still be added making the damping coefficient $\Delta t k/l_0 + b$, where b is the usual damping coefficient. We note

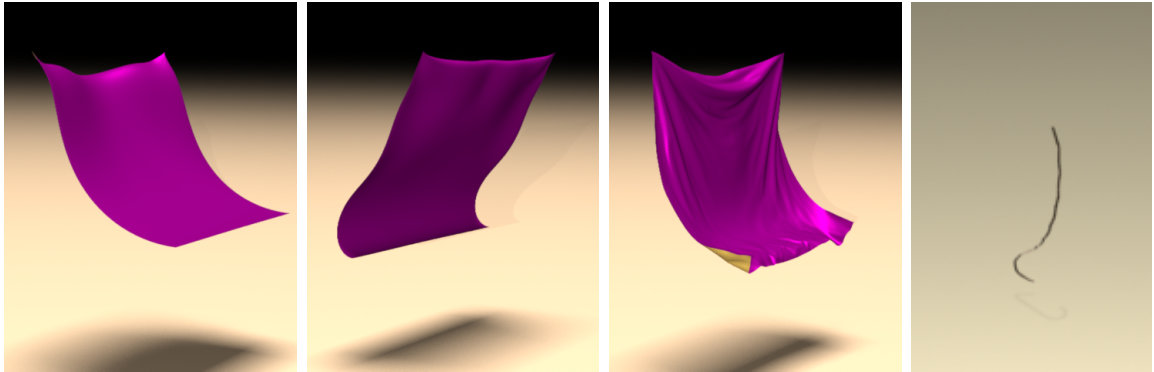


Figure 5.5: Both cloth (left two figures) and hair (far right) integrated using one time step per frame. However, excessively large time steps in implicit methods lead to larger damping. The third figure from the left depicts the exact same cloth simulation with a smaller time step showing that the larger damping stems from the time step size, not the method.

that while there are other approaches to deriving linear springs (see e.g. [34, 42]), our particular one has the advantage that the linear term projects only in the spring direction, giving less damped results. Finally, note that these springs are readily included in the Newmark semi-implicit time integration just as any other force which is linear in the positions (similar to the zero restlength binding springs in [139]). Figure 5.5 shows cloth and hair integrated at the frame rate using these springs.

5.6 Conclusion

We have presented the general outline of techniques required for solid simulation: a mesh, a constitutive model, and a time integration scheme. We also have presented an improved volumetric altitude spring constitutive model that prevents inversion using only simple linear springs. To allow fully implicit integration of linear springs in the context of a Newmark method, we have also provided a discretization that is truly linear in position for linear stress/strain springs. However, in this chapter we did not discuss the remaining important parts of solids simulation—collision and interactions. In the next two chapters we consider self and body interactions, and we use the techniques presented in this chapter for the simulation of cloth and hair.

Chapter 6

High Fidelity Cloth Simulation

Cloth simulation is pervasive in film, e.g. the untangling strategies for *Monsters Inc.* [7], the collision and stiction methods for *Terminator 3* and *Harry Potter* [16], and the wrinkle system for *Shrek 2* [31]. Cloth simulation also promises to have significant future impact on the clothing industry [27] (see also [32]). While, some researchers have focused on real-time simulation for computer games or non-hero characters [99] that have lower quality requirements, our focus is on hero characters, online shopping, and related applications that need photorealistic cloth and clothing. Achieving such realism requires higher resolution because the number of bends and folds is limited by the underlying cloth discretization.

Thus the focus of this chapter is allowing simulation of extremely high resolution meshes and producing interactions commensurate with this level of detail. Cloth papers typically consider the simulation of relatively few triangles: [56] used 10-40 thousand elements, [146] used 5-38 thousand elements, and [164] mostly considered a few thousand elements but their highest resolution simulation was a very thin ribbon with 80 thousand elements that exhibits bending but no folds or wrinkles. These resolutions cannot resolve or simulate folds and wrinkles at the granularity of Figure 6.1. Most simulation techniques would fail if resolution were increased because of two problems: robustness and tractability. These problems typically manifest themselves in time integration and self-collisions/contact (see Figure 6.3). To solve these problems we use a combination three techniques described below.



Figure 6.1: Two examples of cloth showing the types of intricate folds and wrinkles we wish to simulate; both would require high resolution simulation. A satin bed sheet (left) shows that larger scale cloth typically has more folds and wrinkles. Intricate detail is also visible on a Greek sculpture (right) sculpted to resemble a light fabric. Both images are public domain from the Creative Commons of Wikipedia.

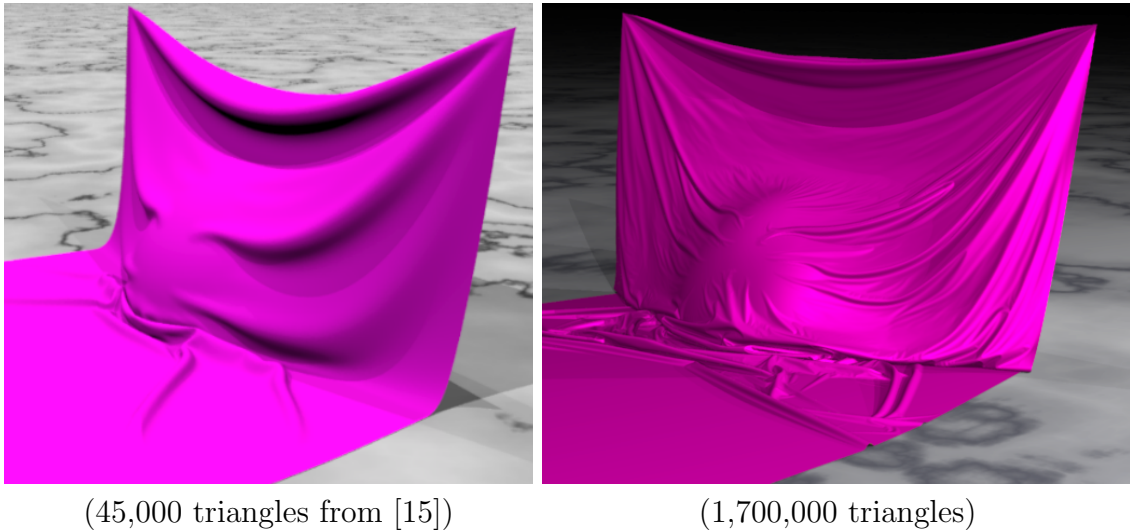


Figure 6.2: High resolutions are necessary to represent and simulate highly detailed, so our goal is to maintain the same robustness and quality while having better scalability.

First, we must improve time integration as at high resolutions, it begins to dominate. Semi-implicit methods have a worse time step restriction as the mesh is refined while fully implicit methods could take the same time step size but would then be less accurate and require more iterations to solve their linear systems. For these reasons, parallelism becomes an attractive approach, and in Section 8.2 we describe how this is done for cloth.

Second, we improve self-collisions so that tractability of collisions is improved while maintaining robustness guarantees. Self collisions and interactions are important not only because they prevent interpenetration but because they also force the cloth to form folds and wrinkles. Recent works have demonstrated that it is possible to stop all collisions even in complex scenarios [124, 75, 15, 74], while other works have shown that untangling is useful as well [159, 162, 164] especially in situations such as pinching [7]. Geometric collisions (using swept primitives) can resolve complex interactions accurately, but they become intractable as the resolution increases. Repulsion based approaches (even with untangling) are not robust enough as the approximate geometric information they use degrades as simplex density increases. Even robust hybrid approaches such as [15] still require the use of a geometric collision algorithm for every time integration step that contains a collision, which becomes intractable as the mesh resolution increases and the time steps become smaller. We propose an extension to the hybrid repulsion/collision technique, defining a history-based repulsion/attraction scheme that allows us to rely less on geometric self collisions while still remaining fully robust, allowing tractable scaling to higher resolutions. Notably, the knowledge of a collision free state enables the application of smarter repulsion/attraction forces without heuristics to estimate the untangled configuration (e.g. voting algorithms and methods that preclude the use of edge/edge collisions).

Third, we introduce more accurate friction handling between cloth and collision objects to ensure that the extra simulated resolution is used effectively. Cloth object friction and interaction is especially important as the object a cloth is interacting with defines much of its behavior. We propose a novel technique for cloth-object collision and friction that is significantly more accurate than previous methods applied to a semi-implicit or implicit time stepping scheme. See Figure 6.8.

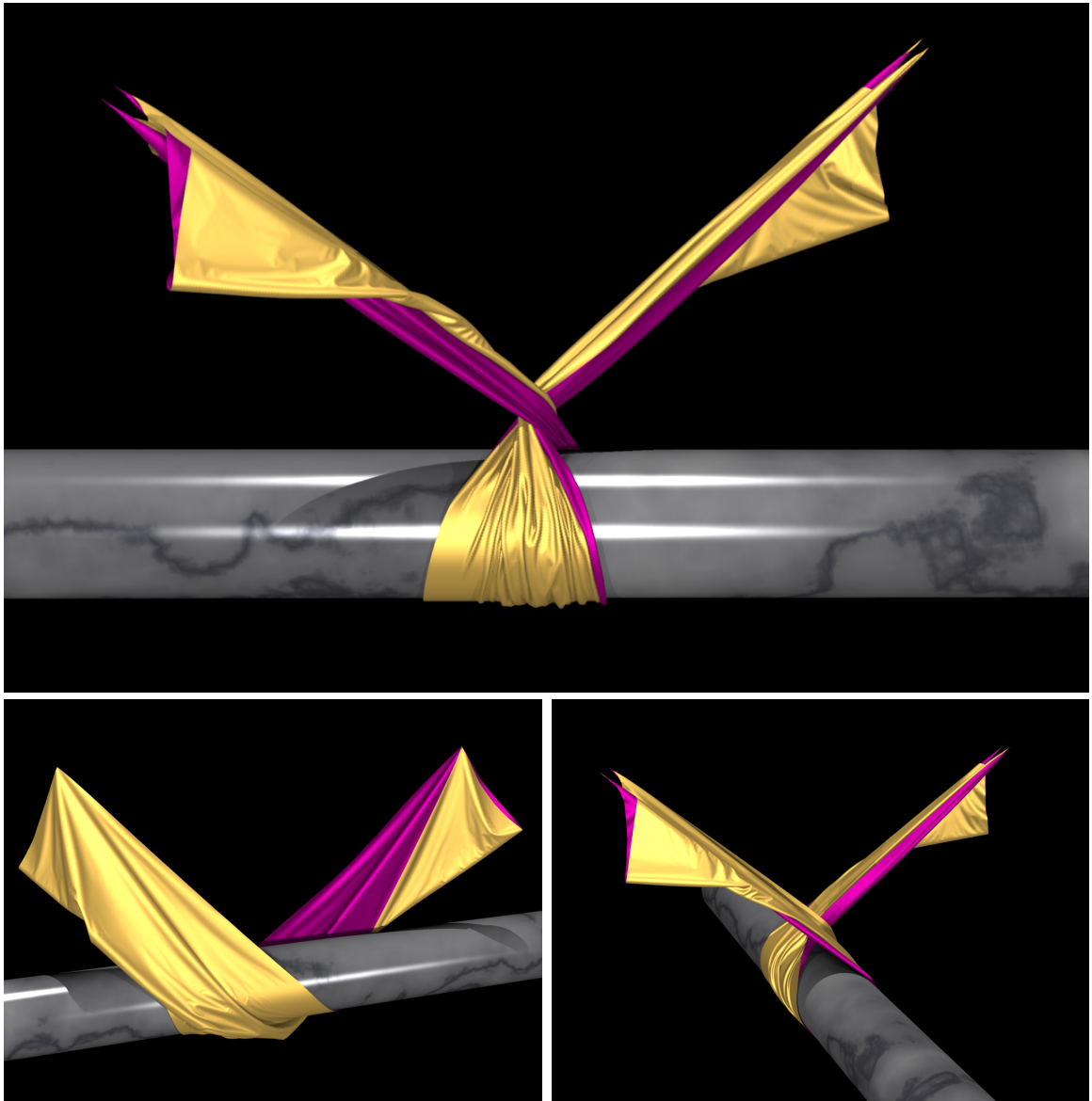


Figure 6.3: A piece of cloth with a half-million triangles is forced to twist by a rotating cylinder. Even under such high tension, the cloth remains self-intersection free showing the robustness of our algorithm at high resolution.

6.1 Previous Work

Computer graphics cloth simulation extends back at least 20 years, and early examples include [177, 151, 150, 152, 18, 111, 14]. A good background on cloth modeling is provided in [72]. We use a variant of the semi-implicit method introduced by [16], but other examples of time integration for cloth include [6, 101, 115, 13, 163, 110]. Our goal is to obtain folds and wrinkles in a physically based fashion from the interplay of in-plane constitutive forces and bending forces, as opposed to adding the wrinkles via a separate modeling system, e.g. [64, 24, 31]. Although we do not address constitutive models for in-plane forces, our preliminary tests show that finite elements and mass-springs models behave similarly, but we refer the interested reader to [43, 77]. Bending models have been addressed by [23, 16, 58, 10, 155, 165]. Although we currently use a static resolution grid, an adaptive approach would allow even more resolution, see e.g. [59]. For collision detection, we use straightforward algorithms and extensions based on well known work but refer the interested reader to [56, 146] and the references therein. We also note the work on improving efficiency in low curvature regions [160, 159].

6.2 Algorithm Overview

We use a simple mass-spring constitutive model (as was described in Section 5.3.1) with edge springs as well as bending springs that connect unshared vertices of adjacent triangles which allows simulating arbitrary triangle meshes. More accurate models such as finite-element constitutive models are possible, but we are interested in accurately modeling collisions and interactions so we found this simple model to be sufficient. We use the semi-implicit time integration scheme described in

Since our focus is collisions and interactions we will augment the basic time semi-implicit integration scheme of Section 5.4 with the missing interaction steps. Our approach to integrating time integration and collision detection/response is similar to [15], but with some key differences. As they did, we have an outer collision loop that puts the mesh into a collision free state. Within this outer collision loop a nested

time integration scheme is used to produce a candidate final position and velocity for each particle in the mesh. The collision loop then calculates an effective velocity by subtracting the candidate position from the last collision free state and modifies this effective velocity until it obtains a displacement that removes all collisions. If collisions were found, [15] rewound to the last collision free state and repeated the simulation with half the number of time steps per collision step, eventually resulting in 1 time step per collision step. In Section 6.3 and Section 6.4, we describe how our history-based attraction/repulsion framework allows us to circumvent this difficulty and thus achieves better performance while maintaining robustness (see Figure 6.11). The i th iteration of our *outer collision loop* step proceeds as:

A. Compute repulsion pairs and their orientation history

B. Perform k_i time integrations (*inner loop*)

1. $\tilde{v}^{n+1/2} = v^n + \frac{\Delta t}{2}a(t^{n+1/2}, x^n, \tilde{v}^{n+1/2})$
2. Modify $\tilde{v}^{n+1/2}$ with elastic and inelastic self-repulsion
3. $\tilde{x}^{n+1} = x^n + \Delta t\tilde{v}^{n+1/2}$
4. Collide with body objects to obtain x^{n+1} and v_\star^n
5. $v^{n+1/2} = v_\star^n + \frac{\Delta t}{2}a(t^{n+1/2}, x^{n+1/2}, v^{n+1/2})$
6. Extrapolate $\tilde{v}^{n+1} = 2v^{n+1/2} - v_\star^n$
7. Modify \tilde{v}^{n+1} to v^{n+1} for friction with objects
8. Modify v^{n+1} with friction and inelastic self-repulsion

C. Detect and resolve moving collisions.

Here Δt is the time step, $a(t, x, v)$ is the acceleration and $x^{n+1/2} = (x^n + x^{n+1})/2$. In Step A we search hierarchies for repulsion pairs as described in Section 6.4. In Step B we perform k_i time integrations which consists of several steps: Step B.1 is a backward Euler solve to obtain a temporary velocity, which is subsequently modified with self-repulsions in step B.2 (Section 6.4), before being used to advance the cloth positions forward in time in step B.3. Then in step B.4 our novel cloth-object collision algorithm (Section 6.5) is applied, obtaining the final collision corrected position

x^{n+1} and intermediate velocity v_\star^n . Next, we apply a backward Euler solve in step B.5 followed by an extrapolation in step B.6, which are equivalent to applying the trapezoidal rule to velocity but are significantly better conditioned than the standard formulation (see [139]). Finally, in step B.7 we modify this final velocity to obtain the appropriate cloth-object friction as dictated by our new cloth-object collision algorithm (Section 6.5) and subsequently apply self-repulsions in step B.8 (Section 6.4). Finally step C ensures that no collisions remain by detecting and removing collisions described further in Section 6.3.

We also employ distributed memory parallelism using message passing so that our algorithm can be used on more than one machine rather than being constrained to a single shared memory machine. Work is distributed across m processors by partitioning the particles into m disjoint sets. Our parallelization strategy and contributions are discussed in Section 8.2.

6.3 The Outer Collision Loop

During the course of a simulation, an outer loop of collision steps is performed where the i th collision step evolves time using k_i time integrations and then resolves collisions. As in [15] this loop maintains the invariant that positions are collision free at the beginning and end of each of these collision iterations. If $k_i > 1$ and a geometric collision was detected, [15] rewound time and reran the collision iteration with $k_i/2$ and this process would continue until $k_i = 1$. Only then were robust geometric self collisions used to zero the relative velocity of interacting collision pairs. We instead never rewind, and run with a fixed k_i between 8 and 16 ensuring tractability in complicated high resolution interactions. [15], by contrast, applied repulsions only at the end of the collision step, using the stored self collision free positions. They could not apply repulsions per time step because as positions moved they had no way of tracking side coherence so repulsions could exacerbate tangling that produced more rigid groups and thus visual artifacts. The key to allowing per time step repulsions without creating artifacts is to store history information together with repulsions discovered

in step A so that the repulsions applied per time in step B.2 and B.8 do the right thing (which we describe in Section 6.4).

In step C, geometric collisions test whether an interaction pair (point/triangle or edge/edge) intersects by checking if the linear trajectories from the last known collision free positions to the current time integrated positions (after k_i time steps) interfere. The linear trajectory implicitly defines the notion of an effective velocity which equals the net change in position divided by the total time elapsed since the last collision free state. Potentially colliding pairs are found by box hierarchy searches with bounding boxes containing the swept trajectory primitives. Pairs of leaf boxes are further pruned by checking for intersections in a coordinate frame that moves with the average effective velocity of the involved particles.

Potentially colliding pairs are processed in Gauss-Seidel fashion using the algorithm in [15] which requires the solution of a cubic equation to determine if the four points involved become coplanar. The inelastically applied collisions are handled by zeroing the relative effective velocity and using this to compute the new positions. If the final positions of the collision pair are in too close proximity, an elastic self repulsion is applied (exactly as in self-repulsion) to push them further apart. We stress that, unlike in [15], this elastic repulsion impulse uses the average time step size taken during the collision loop as opposed to the total time elapsed during the collision loop because we can rely on future per time step repulsions. For any affected particle, we use its new position to calculate a new effective velocity.

After processing all potentially colliding pairs and obtaining new effective velocities, new collisions could be generated. Thus, we iterate the entire algorithm until no collisions are found. The second and later iterations are significantly less expensive than the first since we can reduce the cost of a hierarchy search by considering only box pairs whose expansion contains nodes modified in the previous iteration. As in [15], we rely on rigid “impact zones” when collisions cannot be resolved after a number of iterations. Not noted by other authors we found that rigid groups sometimes form between coplanar, colinear, or colocated points causing the inversion of the inertia tensor to fail. These situations occurred as we scaled to higher resolutions which explains why previous authors did not observe them. Thus we robustly compute the

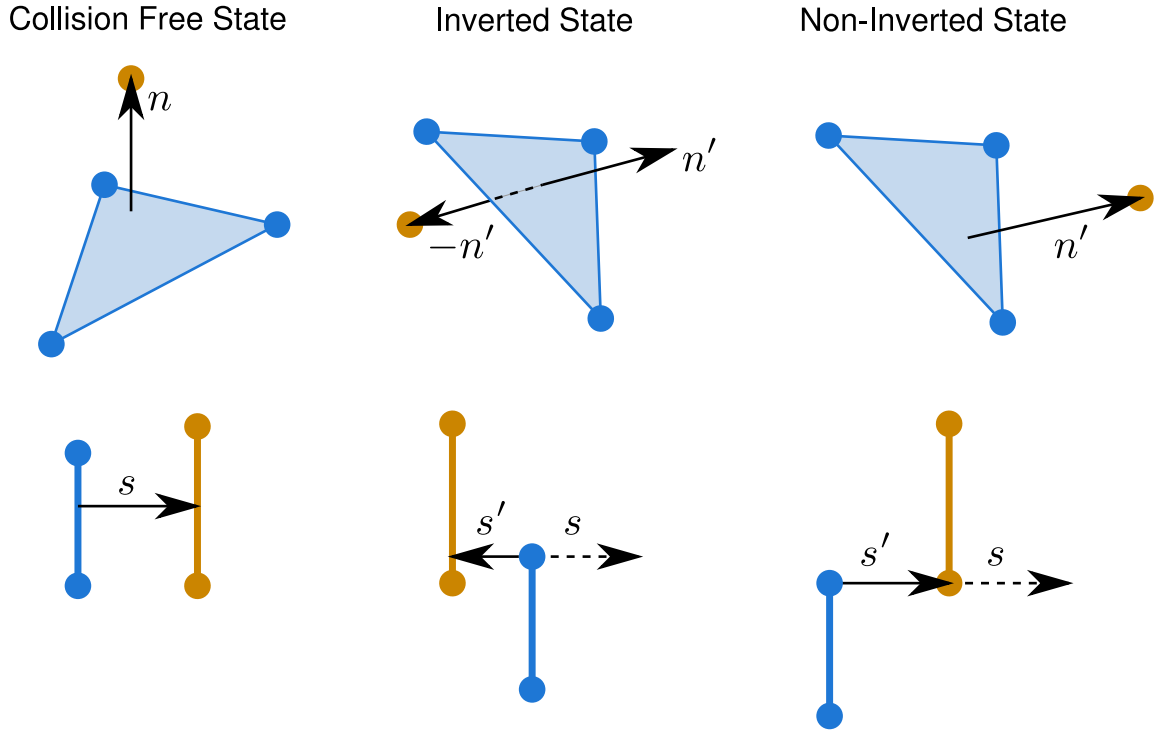


Figure 6.4: Inversion criteria for point/triangle and edge/edge interaction pairs. (Top) Point/triangle pairs are considered inverted if the point is in a different triangle half-plane. (Bottom) Edge/edge pairs are considered inverted if the shortest direction vector rotates too far from its collision free orientation.

pseudoinverse of the symmetric 3×3 inertia tensor using a singular value decomposition. Once the algorithm finds no further collisions, step C is complete and we satisfy the collision-free invariant with the same robustness and visual quality as [15].

6.4 History-Based Self-Repulsions

Since geometric self collisions are expensive to compute and we would like to perform them less frequently, we rely on self-repulsions to help prevent and simplify resolution of collisions. Similar to collisions we consider both point/triangle and edge/edge interactions. Pairs are obtained in step A using a bounding box hierarchy which is

discussed more extensively in Section 8.2. For any type of interaction, we first apply it for all point/triangle pairs followed by all edge/edge pairs, since there are fewer point/triangle pairs and their behavior is typically more robust. In step B.2 of the time integration loop, first an *inelastic collision* impulse is used to stop approaching interaction pairs, and then if necessary a spring-based *elastic repulsion* is used to push interaction pairs further apart. Note that the velocity used in step B.3 is discarded and that steps B.5 and B.6 fully integrate the velocity from time n to time $n+1$. Frictional effects are only calculated in step B.8 in order to implement self repulsions. The amount of friction is determined based on the normal forces caused by the inelastic repulsions used to stop cloth from encroaching on itself. Since elastic collisions can produce artifacts, we use the elastic repulsions only to modify the velocity that will be used to update the positions to reduce the chance of collisions while only using inelastic repulsions for the actual update to the velocity in step B.8.

[15] also made use of repulsions to ease the requirements on the geometric collision stage, but he applied them immediately before the collision stage (i.e. in our step C) using the linearized effective velocities. We instead use actual simulation velocity state and apply repulsions at per time step granularity resulting in increased stability and robustness. We follow [15]’s formulation of repulsions. Here we consider point-face, but edge-edge is similar. For an inelastic repulsion the impulse is $I_c = m\mathbf{v}_N/2$ where \mathbf{v}_N is the normal velocity and m is the mass of all the pair’s particles. For an elastic repulsion of spring stiffness k we use

$$I_r = -\min\left(\Delta t k d, m\left(\frac{.1d}{\Delta t} - \mathbf{v}_N\right)\right) \quad (6.1)$$

where

$$d = h - (x_4 - w_1\mathbf{x}_1 - w_2\mathbf{x}_2 - w_3\mathbf{x}_3) \cdot \hat{n} \quad (6.2)$$

and w_i are the barycentric weights of the free point x_4 projected to the triangle, $x_{\{1,2,3\}}$ are the triangle’s point locations, h is the repulsion thickness, and \hat{n} is the triangle normal. Note the elastic repulsion is limited to 0.1 of the interpenetration, removing the need for any damping parameter and the inelastic repulsion does not require damping as it provides infinite damping in the normal direction. The modification

of the friction is also accomplished with an impulse that uses the change in normal velocity applied by the inelastic or elastic collision Δv_N .

Unfortunately, repulsions only work if the interaction pairs contain the correct notion of sidedness, as they otherwise work against an interference free state if the pairs have crossed (e.g. if a point spuriously crosses a face). Several authors have suggested switching to attractions (e.g. [7]) after cloth has non-physically interpenetrated itself, however it can be difficult to ascertain whether attractions or repulsions should be applied and thus sometimes the interactions are turned off altogether. Since crossing may occur during the time step, checking for interpenetration at discrete times is insufficient. [15] avoids this problem by only applying repulsions in the collision free state, which has the downside of only allowing repulsions at the granularity of the outer collision loop. Since mesh elements can move considerably during the time integration, many potential repulsions are missed, reducing the benefit of repulsions in avoiding collisions as well as reducing the amount of small scale bending and folding that could be produced.

Our key idea is to compute and store interaction data from the collision free state in step A and to subsequently use this data to apply history-based repulsions and attractions during all time integration steps during step B. This increases efficiency because we do not apply a per time step collision detection scheme but instead detect all potentially interacting repulsion pairs during the outer collision loop. Although this tends to produce more pairs than is necessary, and in fact we purposefully use non-aggressive pruning to capture more pairs because the elements can move significantly during a sequence of time steps, it is quite efficient to perform a simple prune at each time step to see if the potentially interacting pairs are indeed interacting. The total cost of all operations in our per time step repulsion method (including detection and application) is typically 7% of the code run time. This approach contrasts with others' who have used voting schemes to construct sidedness (e.g. [161]) as our use of the collision free state to determine orientation alleviates the need for majority (voting) approaches.

During the outer collision loop in step A, after finding all potentially interacting

pairs, we compute and store the relative orientation for later use in our history-based repulsion scheme. For the point/triangle case, we apply a repulsion whenever the point remains on the correct side of the triangle as determined by its normal (Figure 6.4 top), and otherwise switch to an attraction. This is implemented by ensuring the normal \hat{n} always points toward the free point at the collision free time. If not then the points are reordered. i.e. if we have pair (x_1, x_2, x_3, x_4) then if $(x_2 - x_1 \times x_3 - x_1) \cdot (x_4 - x_1) < 0$ then consider the pair as (x_1, x_2, x_4, x_3) . Then as a point x_4 passes from the correct side through the triangle, d will increase in Equation 6.2, leading to a larger impulse in a correcting direction in Equation 6.1. For the edge/edge case, we store the shortest direction vector between the two interacting edges in the collision free state \mathbf{s}_0 , and later compare this with the shortest direction vector \mathbf{s} in the current state (Figure 6.4 bottom). This formulation does not penalize rotation of segments in parallel planes, but it does penalize rigid body rotations. This makes the edge/edge history-based heuristic less reliable than the point-face one so we only flip the current shortest vector if $\mathbf{s}_0 \cdot \mathbf{s} < \epsilon$ (where $\epsilon = 0$ is aggressive and $\epsilon = .9$ is conservative).

6.5 Cloth-Object Collisions

Bridson [16] suggested a level set based collision algorithm that processes each position \tilde{x}^{n+1} and velocity v^n as follows. Suppose $\phi(x)$ measures the signed distance from a point in space to all objects in the scene $\phi(x) < 0$ is inside and $\phi(x) > 0$ outside. Then if $\phi(\tilde{x}^{n+1}) < 0$, a collision is detected with depth $d = |\phi(\tilde{x}^{n+1})|$ and normal $N = \nabla\phi(\tilde{x}^{n+1})$, assuming that $\nabla\phi$ has already been normalized. The position is then projected in the normal direction $x^{n+1} = \tilde{x}^{n+1} + dN$ as shown in Figure 6.5 (left). For the sake of exposition the following steps define a function $v_\star = \Gamma(v)$ that adjusts the velocity to remove any inward normal component and include the effects of friction. The normal and tangential velocity components are computed as $v_N = v^T N$ and $v_T = v - v_N N$ for the particle and $v_{BN} = v_B^T N$ and $v_{BT} = v_B - v_{BN} N$ for the body. Here v_B is the velocity of the body at time $n + 1$. The modified normal velocity $v_{\star N} = \max(v_N, v_{BN})$ ensures that the relative velocity does not point into the body.

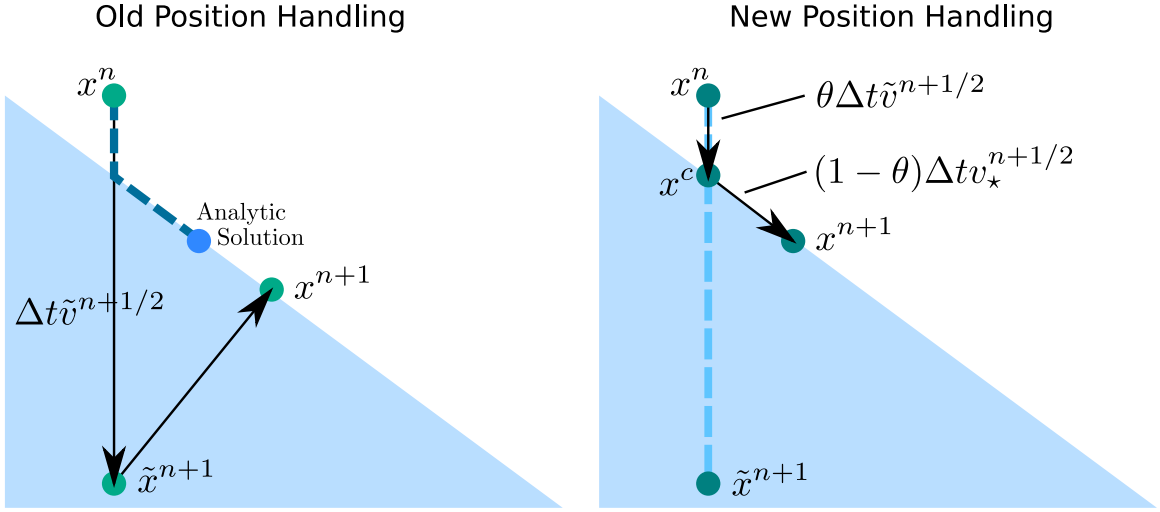


Figure 6.5: Particles are depicted falling towards an incline plane. (Left) A simple level set adjustment to position incorrectly projects in the normal direction. (Right) Our improved method first finds the collision point and then steps to the final position.

The tangential velocity is modified to $v_{\star T} = v_{BT} + \max\left(0, 1 - \mu \frac{v_{\star N} - v_N}{|v_{T,rel}|}\right) v_{T,rel}$ where $v_{T,rel} = v_T - v_{BT}$ and μ is the coefficient of friction. The final result is

$$v_\star = v_{\star N} N + v_{\star T}.$$

The two sources of inaccuracy in this previous method emanate from errors in the position update obtained by projecting in the normal direction and the subsequent changes in velocity incurred during the conjugate gradient solve. Although [16] constrained the normal velocity during their second conjugate gradient solve, the friction and motion in the tangential direction were still adversely affected. While infinite friction could be obtained by also constraining the tangential velocity, one cannot accurately obtain finite non-zero friction, and moreover one cannot constrain the velocity in the first conjugate gradient solve as it would cause cloth to stick to objects.

To compute a more accurate collision adjusted position, we find the point where the level set changes sign $x^c = x^n + \theta \Delta t \tilde{v}^{n+1/2}$ where $\theta = \phi(x^n) / (\phi(x^n) - \phi(\tilde{x}^{n+1}))$

by linear interpolation. If the object is moving, ϕ depends on time as well, and we replace $\phi(x^n)$ with $\phi(x^n) + \Delta t v_{BN}$ in the definition of θ noting that all evaluations of the level set function ϕ occur with the time $n + 1$ collision body. Next we compute $v_\star^{n+1/2} = \Gamma(\tilde{v}^{n+1/2})$ which is used to move from x^c to the final modified position $x^{n+1} = x^c + (1 - \theta)\Delta t v_\star^{n+1/2}$. Note that $(1 - \theta)\Delta t$ is the remaining fraction of our

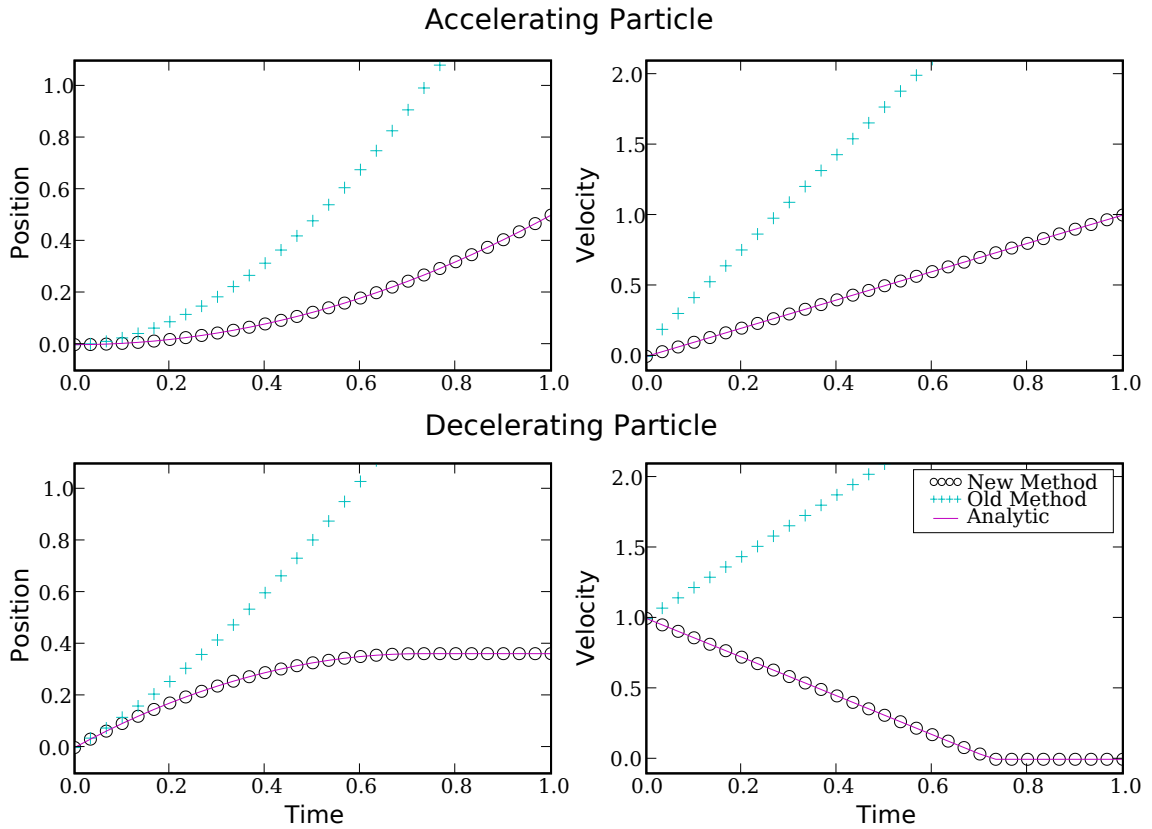


Figure 6.6: Plot of displacement and tangential velocity over time for a single particle moving down an incline plane. Our result is coincident with the analytic result for both accelerating and decelerating particles, while the previous method accelerates too fast in both cases. In particular the analytic velocity is $v(t) = v_0 + (\sin \theta - \mu \cos \theta)gt$ when not stopped. If $\mu > \tan \theta$ stopping happens when $v(v_0 / (g(\sin \theta - \mu \cos \theta))) = 0$. The analytic position is $x(t) = v_0 t + (\sin \theta - \mu \cos \theta)gt^2 / 2$. We use $\theta = \pi/5$, $g = 9.8$. For accelerating we use $v_0 = 0$ and $\mu = .6$ and for decelerating we use $v_0 = 1$ and $\mu = .9$

time step after the collision as shown in Figure 6.5 (right). Although $v_{\star}^{n+1/2}$ yields the correct post collision velocity for use in the position update, we also compute $v_{\star}^n = \Gamma(v^n)$ for subsequent use in the trapezoidal rule (steps B.5 and B.6) which is more correct at time n . While steps B.1 to B.3 of the time integration scheme are used to obtain the correct position, steps B.5 and B.6 are used to update the velocity which also must be corrected for contact. Specifically, any forces applied during the trapezoidal rule velocity update will contain both tangential and normal components so that the conjugate gradient algorithm must project the normal components of the forces to zero to keep points in contact constrained to $v_{\star N}^n$.

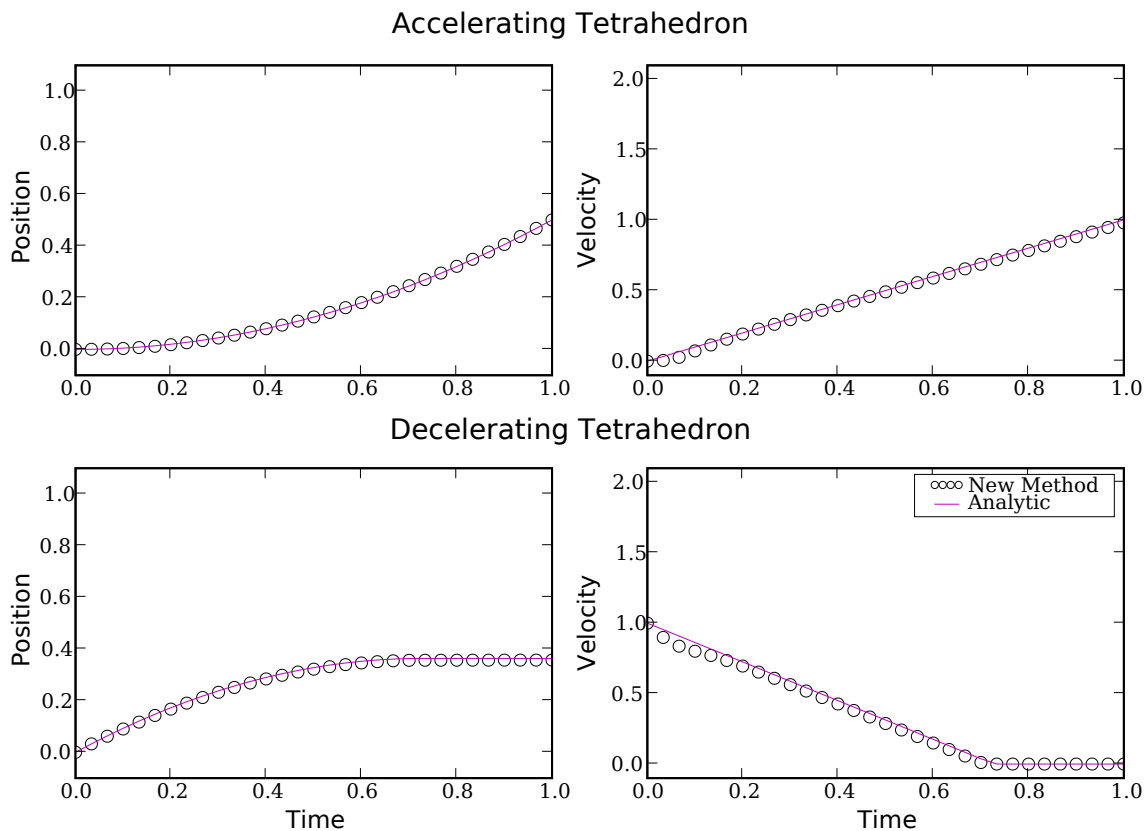


Figure 6.7: A single tetrahedron slides down an incline plane, matching the analytic solution for both accelerating and decelerating test cases. The analytic formulas and parameters are the same as the particle's.

Similar to [6] we can determine the net normal force applied during the trapezoidal rule (or for backward Euler) and use it to correct our friction algorithm. Although [6] used a simple model, we use the more complex Γ function. For the sake of exposition, assume that the final velocity step was a backward Euler step to t^{n+1} instead of the trapezoidal rule, then the velocity update equation is

$$\tilde{v}_P^{n+1} = v_\star^n + P\left(\frac{\Delta t}{m}F_i(t^{n+1}, x^{n+1}) + \frac{\Delta t}{m}F_d(t^{n+1}, x^{n+1})\tilde{v}_P^{n+1}\right)$$

where F_i is the velocity independent force, $F_d\tilde{v}^{n+1}$ is the linear velocity dependent damping force, P projects away the normal component of each colliding point with $I - NN^T$, and \tilde{v}_P^{n+1} are the solved velocities (which have normal components constrained by projections). Using the same forces without applying the projection gives a different result

$$\tilde{v}_{NP}^{n+1} = v_\star^n + \frac{\Delta t}{m}F_i(t^{n+1}, x^{n+1}) + \frac{\Delta t}{m}F_d(t^{n+1}, x^{n+1})\tilde{v}_P^{n+1}$$

which includes the global effects of the projection in F_d but applies all forces to the velocities without projection. Although \tilde{v}_{NP}^{n+1} does not have a zero normal relative velocity for nodes in contact, its damping forces have been properly globally conditioned for contact, i.e. they use the \tilde{v}_P^{n+1} velocity. To include contact in step B.7, we take \tilde{v}_{NP}^{n+1} and apply Γ to obtain $v^{n+1} = \Gamma(\tilde{v}_{NP}^{n+1})$. Note that Γ will compute the same normal force that P applied during the conjugate gradient solve. In addition, Γ will apply the appropriate friction that P did not apply.

To test our new algorithm we consider a single particle sliding down an incline plane. We consider two cases: one with a particle slowing down and coming to rest and another with a particle starting from rest and accelerating. (Note in these simple cases, the incline plane does not move, the normal does not change and the particle starts and stays in contact, i.e. $\theta = 0$) Figure 6.6 compares the previous algorithm, our improved version, and the analytic solution. Since the single particle test case does not require conjugate gradient, as gravity is the only force, we show in Figure 6.7 the same test repeated with a tetrahedron that uses non-trivial damping forces. Figure 6.8 shows an example where cloth rolls with static friction.

If many collision objects are used during a simulation, the cost of evaluating ϕ can be prohibitive, especially for memory-intensive collision objects which we desire to process only once. For efficiency we use a uniform spatial partition for collision body occupancy and iterate over cloth points, creating a list of potential interactions. Subsequently, each collision body is accessed only once and all potentially interacting points are processed with it.

One limitation of this algorithm is that the linearizations used for the position correction can be problematic on high curvature objects. Another limitation of this algorithm is that it queries for point penetration within the collision body, so if a collision body is excessively thin or velocities are high, a collision might be missed. In practice this is rarely an issue and in fact this type of collision approach is frequently used for a character's body in order to simulate clothing. Alternatively, body collisions can be handled within our self-collision framework instead, although the much more efficient cloth-body algorithm should be favored if it is applicable.

6.6 Examples

We ran our simulations with meshes ranging from a half-million to 2 million triangles on between 2 to 4 quad-processor Opteron 2.8 GHz machines connected by gigabit ethernet, but obtained similar performance profiles on commodity dual-core dual-processor machines. We used 16 time steps between each collision processing step though we reduced this number to 4 or 8 during a few collision intensive sections. Automatically choosing the number of time steps between collision processing steps is important future work but we emphasize that approaches that scale to 1 time step per collision loop whenever collisions occur [15] or use 1 time step per collision loop [74] become intractable on large meshes. Even on lower resolution meshes where there are fewer collisions, we still get a benefit by running with 16 total loops (e.g. Figure 6.11). None of our examples used viscous air (ether) drag to damp the velocities, and only the wardrobe and leaves examples used wind drag. We rendered cloth using a standard ray tracer. Although the results of cloth simulations can have their resolution artificially increased in a subdivision postprocess (even avoiding collisions

Parameter	Description	Value
E_e	Edge Youngs Modulus (Stiffness)	4.14
ξ_e	Edge Spring Overdamping Fraction	15
E_b	Bending Youngs Modulus (Stiffness)	0.41
ξ_b	Bending Spring Overdamping Fraction	8
CG Tolerance	Convergence for Backward Euler	0.001
Mass	Mass of node in mesh	1e-6
g	Gravitational constant	9.8
h	Repulsion Thickness	0.01
k	Repulsion Spring Constant	30
k_i	Collision Total Loops	4-16

Table 6.1: Parameters of our model.

as in [15]), our simulated examples were of sufficient resolution to require no subdivision except for the example that used 100 pieces of lower resolution 10 thousand triangle cloth. This is an important step as subdivision is only a stop-gap measure that makes renderings visibly smooth, and it cannot introduce detail missing from the low-resolution simulation. We have summarized some of the key parameters that we used to generate our examples in Table 6.1.

We begin with our lowest resolution example, a half-million triangle version of a twisting cloth torture test shown in Figure 6.3, demonstrating the robustness of our algorithm even in difficult collision situations. This example averaged 30 minutes per frame, but as the two ends become intertwined the strain on the knotted self-collision area becomes very high and frames take longer to complete. In Figure 6.9 we demonstrate that we can handle many pieces of cloth, simulating 100 separate falling cloths with 10 thousand triangles each, totaling 1 million triangles. We employ a wind-drag model that produces interesting deformation and flutter, facilitating inter and intra-cloth interactions. For each vertex, we apply a linear drag in the normal direction as a simple approximation to the pressure force, noting that a better approximation would be quadratic in velocity. We maintain a high wind speed near the ground in order to push the lightweight pieces of cloth around, causing further interactions and increasing the number of dynamic collisions that must be resolved even after the pieces of cloth hit the ground. At 5 minutes per frame across two machines, this was our lowest cost simulation.

Example	Resolution	Avg. frame time	Processors
Twisting Cloth	0.5 million	30 mins	16
Cloth Leaves	1 million	5 mins	8
Spinning Ball	1.8 million	20 mins	16
Curtain & Ball	1.7 million	45 mins	16
Wardrobe	2 million	6 mins	16

Table 6.2: List of our examples with their resolutions, average time per frame, and number of processors they were run on.

The next examples demonstrate a single high resolution mesh with simulation parameters chosen to accentuate folds and wrinkles and their subsequent collisions and interactions. Figure 6.10 shows the spinning ball example from [15, 146] with an increased resolution of 1.8 million triangles. To promote a very high level of detail we avoid using overly stiff cloth and air drag, use high sphere but relatively low ground friction, and raise the vertical position of the sphere slightly so that the initial draping of the cloth results in even more self collisions at the base of the sphere. This example averaged 20 minutes per frame. Figure 6.12 shows the curtain and ball example from [15] at a resolution of 1.7 million triangles. The multiple layers of contact that form when the cloth folds over itself make this example particularly difficult, especially when the sphere pushes through the clump of layers. The average frame time for the simulation was 45 minutes. Figure 6.13 shows a 2 million triangle cloth draped over a wardrobe. We initially animated several vertices of the cloth to obtain a more interesting draping effect. After an initial settling period, we change the coefficient of friction on the wardrobe allowing the cloth to fall and form many folds and wrinkles under the effect of wind drag. The cloth in this simulation resembles the photograph of draped cloth in Figure 6.13 and the satin depicted in Figure 6.1. This simulation averaged just over 6 minutes per frame. A summary of the resolutions and timings of our examples is shown in Table 6.2, and we have provided a breakdown of our algorithm timing in Table 6.3.

We found parallelism essential as it allowed us to scale to very high resolutions although it alone was not responsible for our results. In fact we first parallelized the [15] algorithm but found it insufficient, motivating our other improvements. For

Simulation Step	% of total sim	% of subpart
Compute repulsion pairs/history (Step A)	3-7%	
Inner time integration loop (Step B)	45-60%	
Applying per time step repulsions		2-4%
Detect & resolve collisions (Step C)	37-48%	
Applying collisions		20-45%
Computing swept bounding boxes		5-20%
Traversing hierarchies		5-15%
Inter-processor communication		40-70%

Table 6.3: A breakdown of the time spent in each part of our algorithm.

descriptions of the parallel algorithm and a graph of speedup on our curtain and ball example see Section 8.2.3.

6.7 Conclusion

We have addressed several issues in cloth simulation to allow simulating high resolution cloth, enabling us to represent and simulate intricate folds and wrinkles. Using a multi-processor approach on commodity hardware we demonstrated simulations of cloth with up to 2 million triangles. To make collisions more efficient on these large meshes, we employ a history-based attraction/repulsion scheme that takes advantage of the last known collision free state. Applying the repulsions/attractions at every time step reduces the frequency at which the more expensive geometric collision algorithm is used. In addition, to ensure resolution is used effectively, we correctly handle cloth-object friction which facilitates formation and preservation of folds and wrinkles at both low and high resolutions.

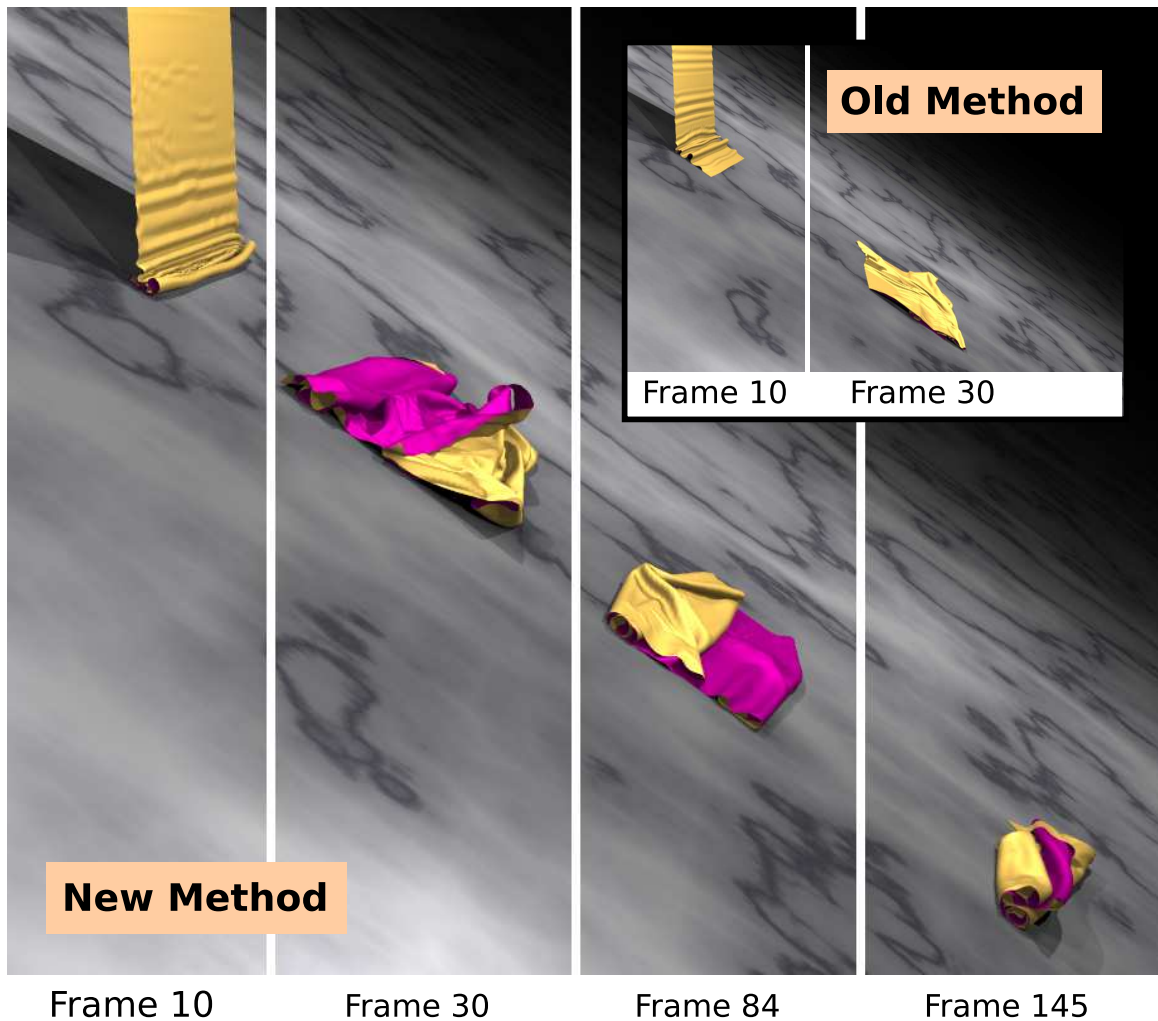


Figure 6.8: A piece of cloth with initial tangential velocity falls on an incline plane with high coefficient of friction, and undergoes static friction causing it to roll. The cloth simulated by the previous method slips immediately on contact with the ground and moves faster (incorrectly) down the incline. Note the resolution here is about 80,000 triangles, showing that even on lower resolutions we achieve better results.

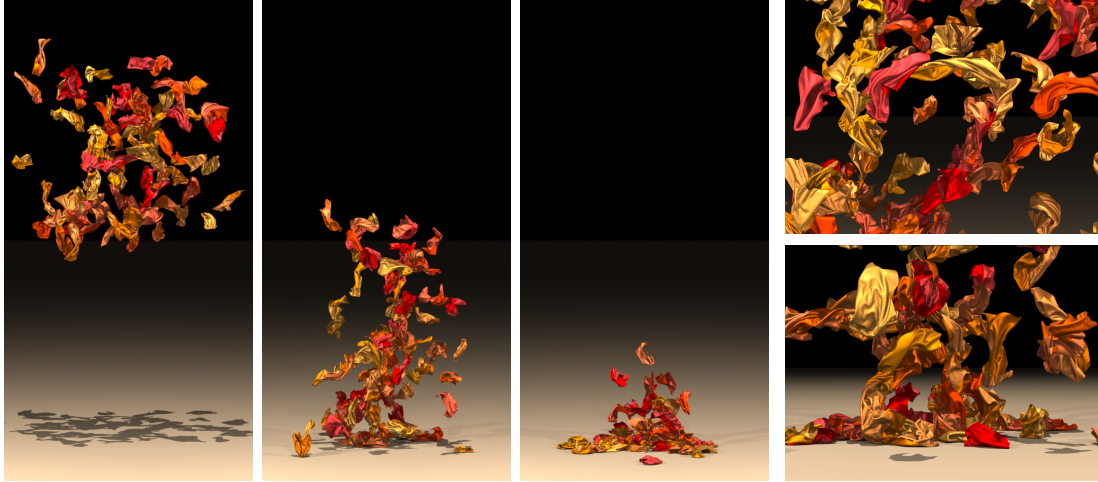


Figure 6.9: We drop 100 pieces of cloth with 10,000 triangles each (1 million triangles total). A linear wind drag model causes them to flutter and interact. The simulation time was under 5 minutes per frame.

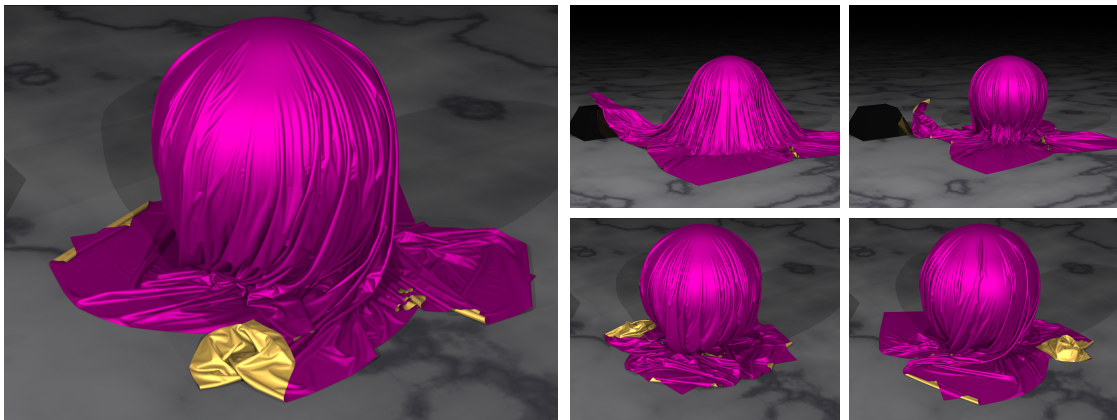


Figure 6.10: A piece of cloth with 1.8 million triangles is draped over a ball. Low ground friction causes the cloth to tightly pack beneath the sphere. As the ball begins to spin, the high frictional coefficient causes the detailed folds of the cloth to twist.

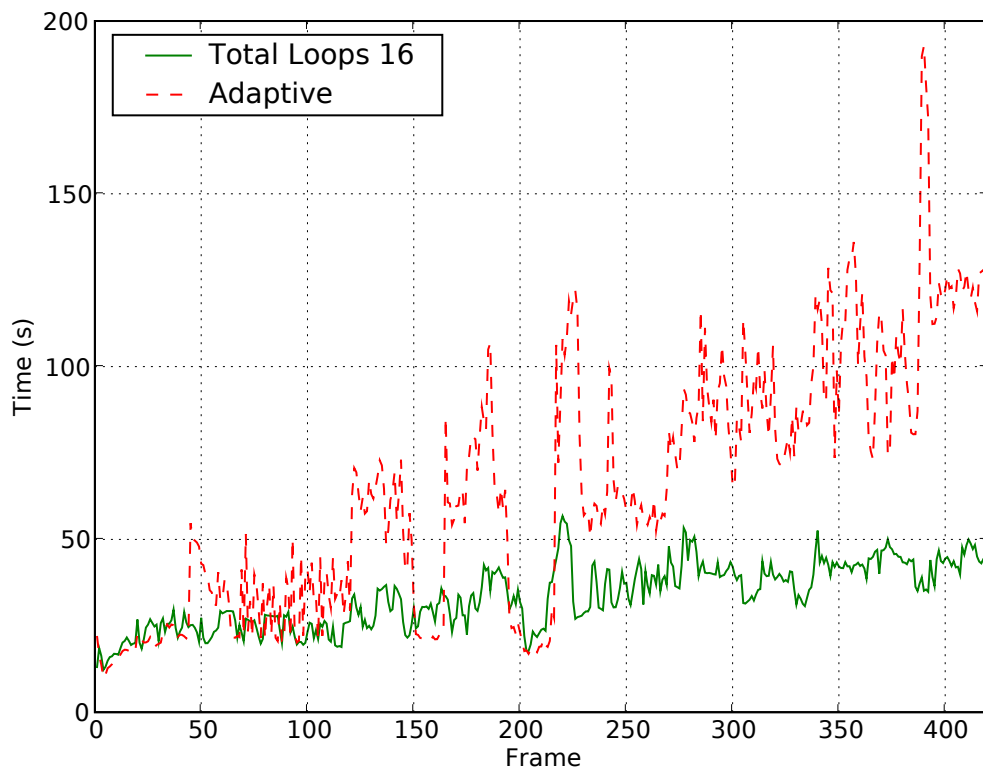


Figure 6.11: Timing comparison of using adaptive collision resolution with rewind (Bridson) and the fixed total loop strategy that we can apply due to our history based repulsion scheme.

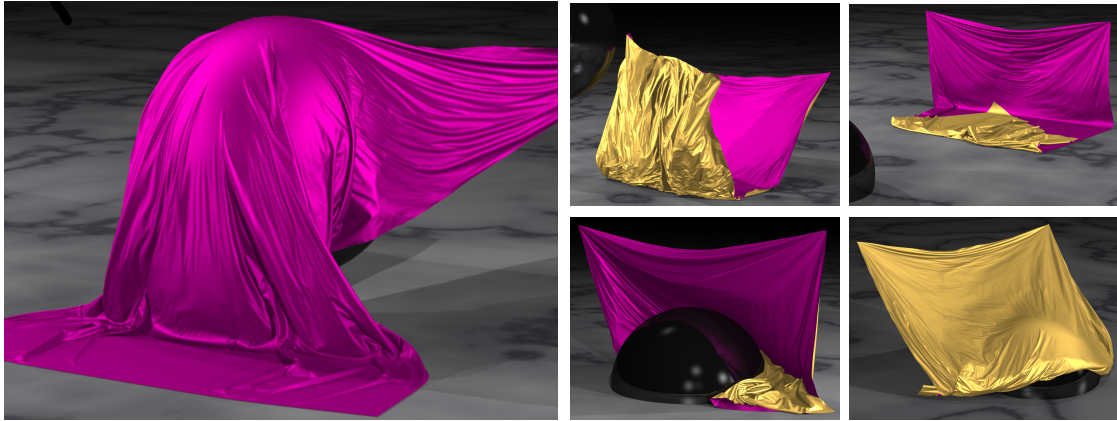


Figure 6.12: A piece of cloth with 1.7 million triangles is flipped by a sphere causing it to fold over itself. The sphere then pushes through the complicated clump of folds.

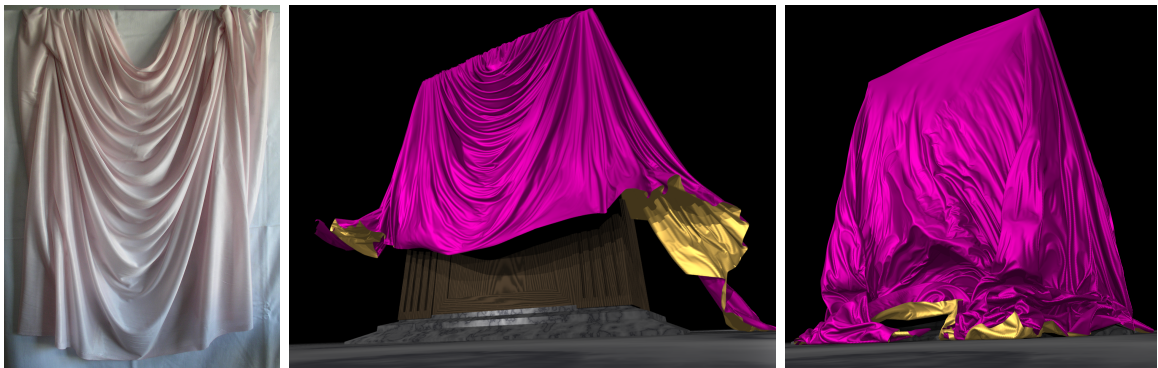


Figure 6.13: (Left) A photograph of a real piece of cloth draped showing intricate folds and wrinkles. The middle and right depicts a synthetic piece of cloth with 2 million triangles draped over a wardrobe. The coefficient of friction between the cloth and the wardrobe is reduced, causing the cloth to fall. Small scale wrinkles and fluttering are introduced by our wind drag model as shown in the right image. Notice how the synthetic example has more folds and wrinkles which is due to it representing a larger size piece of cloth.

Chapter 7

Hair Simulation

7.1 Introduction

Hair dynamics is one of the most challenging phenomena to simulate because of the sheer number of hairs on the head and the complexity of the motion. Even so, hair simulation has important applications for visual effects, animated features, virtual hair styling and online stores. Consequently, researchers have developed methods for its simulation, such as the seminal work of [130, 2] (see also the survey [170]). Most research has managed the complexity of hair simulation by focusing on the bulk behavior of hair, but we seek to capture the subtle and intricate details of hair, requiring us to consider the non-collective behavior.

Some authors suggest that simulating every hair on the head is unnecessary as hair behaves as a collective (see e.g. [118]); they instead use aggregate hair simulation techniques. While the efficiency of these approaches is attractive, simulating aggregate geometry also has drawbacks. In particular, using fewer degrees of freedom (DOFs) forces hair to behave collectively even in situations where it should not (e.g. trendy, messy or tangled hairstyles). Increasing the DOFs by simulating each individual hair, as we propose, can capture these effects, but it also requires handling more interaction constraints explicitly. Another drawback of clumped models is that even though hair volume is easily maintained by using fluid incompressibility or repulsions with large radii, intricate collisions and interactions with small or high curvature objects (e.g.

the ear, a comb, etc.) cannot be resolved. Additionally, the geometric inconsistency between simulated and rendered geometry can produce artifacts such as self or body interpenetrations. Lastly, subtle details such as stray hairs, clump separation, etc. are not handled by aggregate models. To address the above drawbacks, we propose using a non-clumped simulation technique. Though this requires more computation, this allows us to achieve non-collective behavior.

In this chapter, we attempt to simulate as many individual hairs as possible by using a constitutive model that is simple and fast but also can model twist and curly hair. To do this, we use a mass/spring model which is computationally inexpensive and adept at modeling both object collisions and self-collisions. This also has the advantage of being easily adaptable to an existing cloth or flesh simulator, thus leveraging existing time integration and interaction/collision handling. Unfortunately, mass-spring models have difficulty modeling twist, so we introduce a new altitude-spring based hair constitutive model that can model torsion. Fortuitously, the same model we use for hair torsion also provides an improved bending model for cloth and shape preservation for tetrahedra. We also improve time integration by introducing an unconditionally stable implicit linear spring model and a biased strain limiting approach for hair. Body collisions are more efficiently/accurately handled by introducing a better temporal interpolation scheme for level sets. Hair/hair interactions are modeled by using geometric collisions and a stiction model. Our method is demonstrated in several examples in which we only render simulated hairs.

7.2 Related Work

Though our focus is simulation, modeling hair geometry and styles is also an important research topic see [87, 22, 169]. While these methods have modeled geometry with hair counts approaching the number in the head (e.g. [180] uses fifty thousand hairs), techniques for hair simulation have typically simulated fewer.

7.2.1 Aggregate Hair Simulation

Researchers use different degrees of hair aggregation to manage computational complexity. From most aggregate to least, these can be categorized as continuum, clumped strand, adaptive clumped strand and non-clumped strand models. Early work focused on single hair non-aggregate simulation using either mass/spring models [130] or projective dynamics [2] (a length-preserving generalized coordinate model), but both methods neglect torsion and self-collision/interactions. Thus, these methods parallelize trivially as each strand has no data dependencies on any other strand, meaning they could be easily scaled to 100,000 hairs. Nevertheless, the lack of twist and interaction modeling of these methods would not yield high fidelity simulations.

Hair modeled as a continuum implicitly treats interaction, which improves efficiency while maintaining hair volume. [66] uses a fluid continuum that models contact using viscosity together with rigid body chains embedded in the volume. [3] models hair using unconnected particles embedded in a fluid continuum, but their renderings show that sometimes stray particles appear to float in space. [118] simulates mass/spring guide hairs (neglecting twist), rasterizing them onto a level set grid to model interaction and volume, producing stylized and uniform hair. [62] uses a simplified mass/spring mechanical model on lattice points to define a deformation field for embedded rendering hairs. While continuum approaches are efficient at modeling bulk behavior, they have difficulty modeling discontinuous effects like hair separation, and the highly intricate interacting geometry like curly hair, or individual hair twist.

Other authors have used a small set of discrete aggregate hair wisps or clumps that are simulated separately. [120] uses a mass-spring envelope lattice per guide hair in which rendering hairs are embedded. Unlike our mass-spring model, theirs is designed to be used with large envelope volumes (especially for interactions). [21] simulates guide hairs consisting of rigid body links attached with springs, synthesizing rendered hairs with a statistical model; collisions are modeled using penalty forces. [63] uses a sparse set of articulated rigid body (ARB) chains to produce hairs, foliage, and character ears. Collisions require the solution of a quadratic program making it less desirable for dense hairs. [12] also simulates hair using a finite-element constitutive model (discussed below) using penalty methods for collisions.

Even though clumped models do obtain some discontinuous behaviors not handled by continuum models, clumped models still tradeoff some accuracy for efficiency. To capture more detail in only some areas of a model, authors have recently considered adaptive clumped models. [11] uses a tree of varying size rigid body wisps and provided heuristics for forming and breaking. [174, 173] use projective dynamics and three discrete levels of detail (LODs) based on hair motion or camera distance to allow real-time simulation. While targeted at real-time, this method could be used with different settings to achieve higher quality simulations offline. However, the fundamental assumption of all adaptive schemes is that some areas do not require high fidelity, meaning computational savings can be achieved by varying simulation resolution. Unfortunately, this is not always the case; for example, in highly dynamic flipping or wind motions, the same detail is needed everywhere. Here, uniform approaches are more desirable as they do not incur the overhead of managing adaptivity. Nevertheless, if adaptivity is appropriate, these schemes provide a useful framework within which our mass-spring model could be integrated (see future work).

7.2.2 Strand dynamics

Researchers also have considered many mechanical models for strands which can often be applied outside of their original aggregate context. In fact, many papers only demonstrate a single strand, even though their mechanical models could be applied to hair. [114] introduced the high order Cosserat (finite-element) model for flexible rods, although he did not consider dynamics. [57] modeled flexible tubes while also using a Cosserat-based scheme. [140] used a spatially adaptive dynamic Cosserat model for interactively tying knots in ropes in real-time. [17] also simulated knot tying, instead using an articulated rigid body system; gravity was neglected to help user interaction. The preceding three methods use interpenetration to detect collisions, so they would miss collisions if strand thickness was reduced to our hair strand size or if velocity was increased. [167] uses a specialized solver that includes torsion as a state variable to simulate thread.

Researchers have used mass-spring systems, projective dynamics, articulated rigid bodies, and other generalized or finite elements models as discussed above. Mass-spring models are advantageous as they are simple, commonly implemented, and handle collisions and constraints easily, but typically cannot model torsion, curly hair, or exact length preservation. Projective dynamics solves the length preservation problem at the expense of using generalized coordinates, making collisions and constraints more difficult. Generalized coordinate ARB schemes conserve length and allow twist to be modeled, but also make collisions and constraints more difficult. ARBs are also relatively expensive per body, especially if linear complementarity problem (LCP) collision methods are used. Higher order finite element models like Cosserat schemes (e.g. [12]) can create intricate helical guide curves using very few DOFs. However, if one wanted to simulate thousands of interacting hairs, more DOFs per hair would be required to resolve collisions and contact, making higher-order methods less attractive. In particular, [12] scales quadratically as the number of elements per hair are increased, and other high order methods are also more expensive per element. Moreover, the use of generalized coordinates in [12], [140], and [167] makes collision/contact response more challenging. A summary of the tradeoffs of various strand models (including ours) is shown in Table 7.1.

Phenomena	Mass/ Spring	Proj. Dyn.	Rigid (chain)	Super- helices	Our Method
Bending	Yes	Yes	Yes	Yes	Yes
Torsion	No	No	Yes	Yes	Yes (alt. spring)
Non-stretching	No	Yes	Yes	Yes	Yes (strain limit)
Curliness	No	No	No	Yes	Yes (alt. spring)
Constraints	Easy	Tricky	Tricky	Tricky	Easy

Table 7.1: Adapted table from [Ward et al 2007] that shows a comparison between various hair methods. The last column shows that our method relieves several problems with mass/spring models.

7.3 Constitutive Model

Since our goal is to model as many realistically interacting hairs as possible, we prefer a mass-spring model. For efficiency, we use tetrahedra instead of representing coordinate systems explicitly. Furthermore, we avoid generally slower finite element based models, instead using simple linear stress-strain springs. In addition, to preserve effective coordinate systems, we use the new and improved altitude spring model introduced in Section 5.3.4.

7.3.1 An Altitude Spring Hair Model

If we model a hair as a series of connected line segments, stretching can be modeled with edge springs between every consecutive particle, and bending can be modeled with bending springs between every other particle. The edge springs and bending springs together form triangles that implicitly represent the orientation of the hair. Twist can be modeled by attaching torsion springs that connect each particle to a particle three particles away from it (see Figure 7.2(a)). These springs imply tetrahedra allowing the use of our generalized altitude springs to prevent collapse. This scheme works well on curly hair where no set of three consecutive particles is colinear so every tetrahedron has non-zero volume. Unfortunately, for straight hair all the particles are colinear meaning that the triangles formed by edge and bending springs have zero area so orientation (and thus twist) cannot be represented. We address this by introducing additional particles perturbed from the main hair axis to obtain non-zero area triangles.

If we duplicate and perturb the middle particle of two colinear segments, we could handle the original middle particle in three ways. If we did not constrain it, a quadrilateral is formed that can deform arbitrarily (Figure 7.1(a)). If we constrained it, then all bending degrees of freedom are lost (Figure 7.1(b)). If we left it unconstrained, but also attached a spring between it and the perturbed particle, bending is incorrectly constrained in one direction (Figure 7.1(c)). Thus, instead of creating a single new particle, we create two new particles at the midpoints of the two segments and perturb them to form two triangles (Figure 7.1(d)). These triangles should be

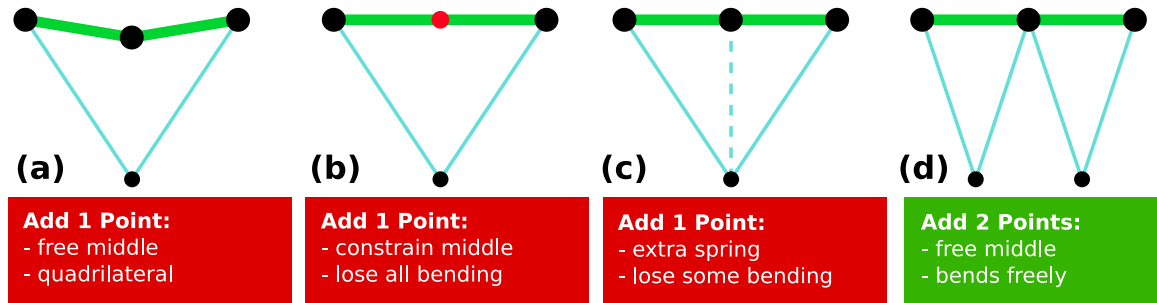


Figure 7.1: Various options for creating perturbed points to form triangles in straight hair. (d) yields both non-degenerate triangles and free bending.

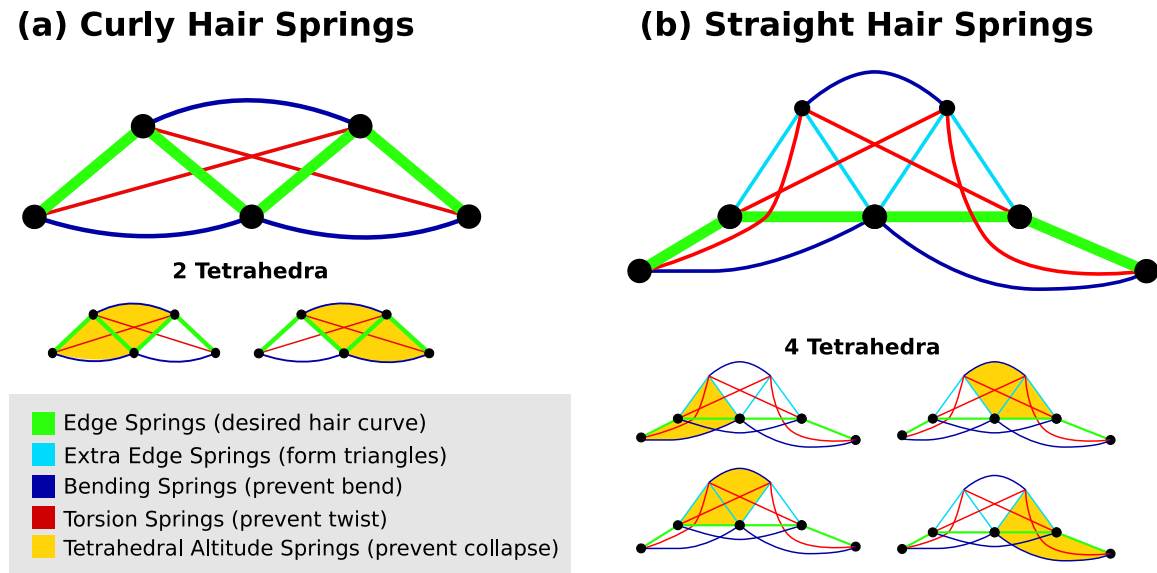


Figure 7.2: Straight and curly hair models using edge, bending, torsion, and altitude springs preserving the implied tetrahedra.

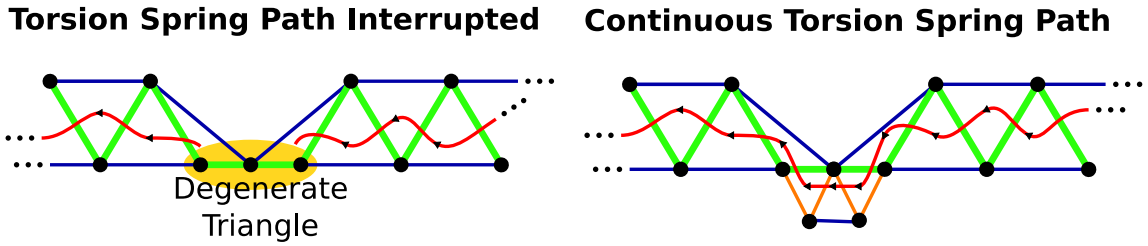


Figure 7.3: Triangles define orientations for penalizing twist, and torsion springs “trace” a continuous path through the non-degenerate triangles—but they are blocked at straight hair segments (left). The subdivision and perturbation of our method removes degeneracies so the path becomes continuous (right).

fairly rigid so they are given springs on their edges that are as strong as the hair edge springs. From these triangles we can add bending and torsion springs to form a full hair model as shown in Figure 7.2(b) where a tetrahedron is formed whenever a torsion spring connects two opposite particles of a triangle pair. The process of ensuring triangles are non-degenerate can be seen as a way of ensuring a path of torsion springs through the hair (Figure 7.3). Thus instead of using an explicit coordinate system we model an implicit one by using offset particles together with extra springs, causing a marginally higher simulation cost, but still fitting into a simple mass-spring framework.

Thus our algorithm prepares a curve for simulation by first sampling discrete points x_1, \dots, x_m equally in arclength. Then a segment x_i, x_{i+1} is subdivided if it is colinear with x_{i-1} or x_{i+2} . We perturb new particles off of the original curve such that the edge lengths of the newly created triangles are equal to the length of the parent segment. In addition, if there are many consecutive segments with perturbed particles, we rotate the perturbed particles about the hair axis to ensure good direction sampling. Afterwards, we consider every segment in order and build the appropriate spring connections. Figure 7.4 demonstrates the plausibility of our approach on single hair examples by comparing videos of real hair to our simulations. Figure 7.5 shows that we can simulate different curly stiffnesses (top) and degrees of curliness (bottom). We also note that one could always subdivide and perturb instead of detecting colinearity, albeit at the cost of unnecessary particles. One downside of

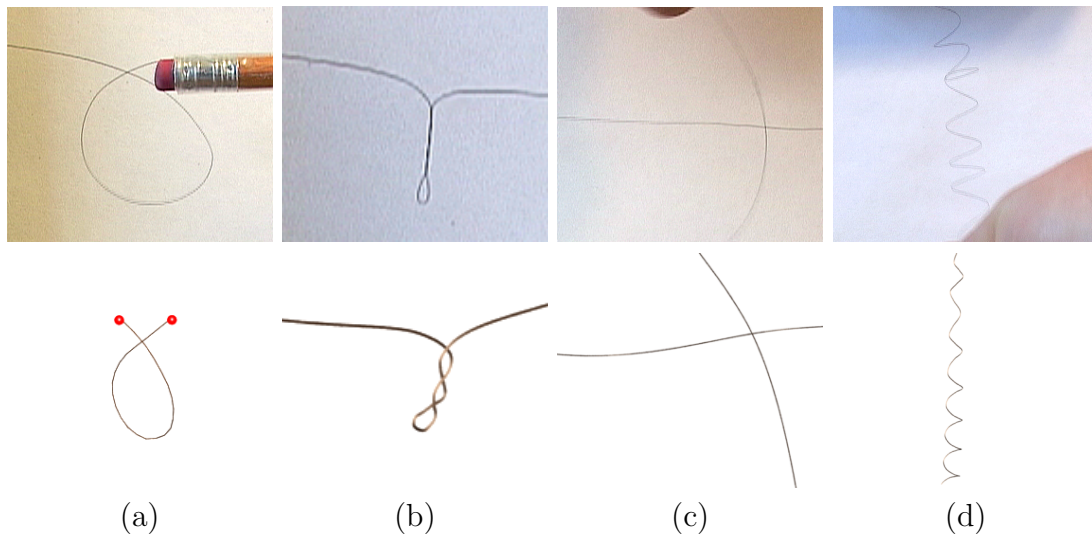


Figure 7.4: Four single hair tests showing various behaviors observed in real hair (top) and reproduced by our model (bottom). (a) a hair is forced on either endpoint to buckle into a loop. (b) a hair is twisted at one end point causes tangling. (c) a hair is pulled across another to exhibit stickiness. (d) a curly hair is stretched and released.

this model is that there is some mixing of stretch, bend, and torsion forces which is analogous to using bending springs in cloth.

7.4 Time Integration

We augment the time integration scheme of Section 5.4 with repulsion steps and strain limiting steps integrate fromn t^n to time t^{n+1} as follows:

1. $\mathbf{v}^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2} \mathbf{a}(t^{n+1/2}, \mathbf{x}^n, \mathbf{v}^{n+1/2})$
2. Modify $\mathbf{v}^{n+1/2}$ with strain limiting
3. Modify $\mathbf{v}^{n+1/2}$ with self-repulsions
4. $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1/2}$
5. Body collisions modify \mathbf{x}^{n+1} and \mathbf{v}^n

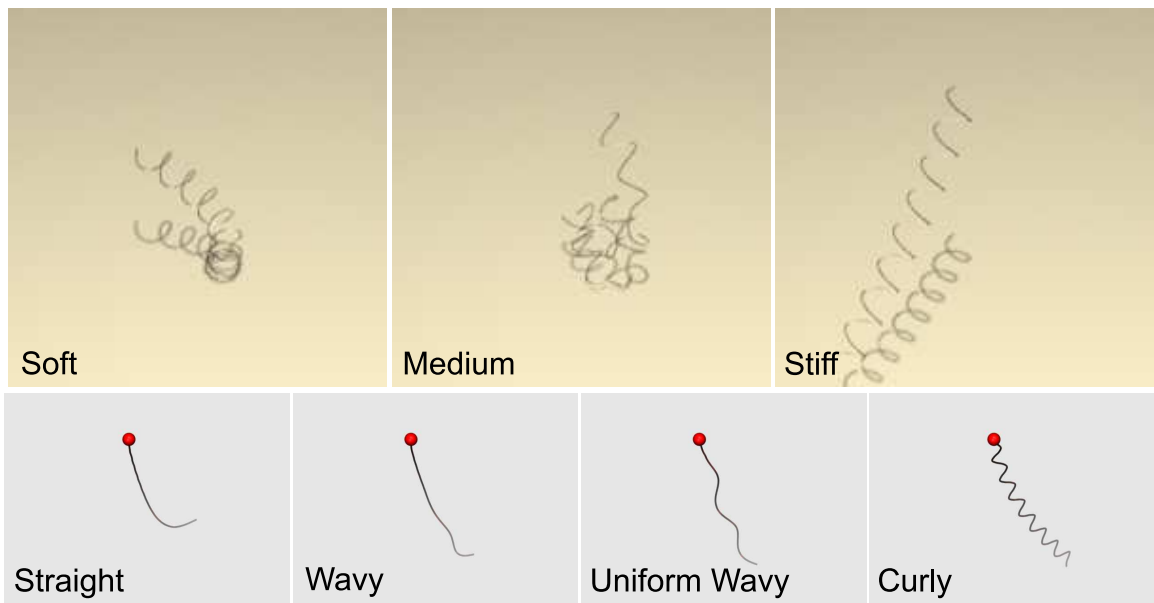


Figure 7.5: (Top) A single curly hair dropped with varying stiffnesses showing that we can represent different types of curly hair. (Bottom) Varying degrees of waviness showing that our model can handle the full spectrum of hair curliness.

6. Self-collisions modify \mathbf{x}^{n+1} and \mathbf{v}^n
7. $\mathbf{v}^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2} \mathbf{a}(t^{n+1/2}, \mathbf{x}^{n+1/2}, \mathbf{v}^{n+1/2})$
8. Extrapolate $\mathbf{v}^{n+1} = 2\mathbf{v}^{n+1/2} - \mathbf{v}^n$
9. Modify \mathbf{v}^{n+1} with self-repulsions

where $\mathbf{x}^{n+1/2} = (\mathbf{x}^n + \mathbf{x}^{n+1})/2$. Notice in particular that here we are ignoring details of the outer loop from Chapter 6, as these details are the same. The velocity computed in step 1 is processed with our new strain limiting approach (Section 7.4.1) as well as self-repulsions before being used to update the positions in step 4. After the position update, body collisions are applied first in step 5 and then self-collisions in step 6 (both of which are discussed in Section 7.5). The velocity from steps 1-4 is discarded, and step 7 and 8 evolve the velocity forward in time using the trapezoidal rule, though increased stability can be obtained with backward Euler.

7.4.1 Strain Limiting

Complex head motions can cause severe stretching especially in springs which have one of their two endpoints constrained to a character's head as seen in Figure 7.6(a). To alleviate high strain in cloth, [123, 15] used strain limiting approaches that apply momentum conserving velocity impulses to particles attached by springs that exceed 10% deformation. Since correcting one spring potentially damages another, iteration is typically used. Since hair is relatively light compared to the head, we employ a biased strain limiting approach that marches from the root of the hair and projects the length by moving only the particle further from the root in the direction formed by the two particles. For the extra particles from the subdivision and perturbation in Section 7.3.1, we also project one of their two edge springs (the one closer to the root). Iteratively, once we find a new candidate position, we apply a velocity impulse to the $\mathbf{v}^{n+1/2}$ velocity to achieve that position. Note that the strain limiting in step 2 above only affects the velocity used to update the positions and has no direct effect on the velocity used for evolution in steps 5-9. One might alternatively suggest using an extremely stiff spring with a fully implicit integrator, however, implicitly integrated



Figure 7.6: A tuft of hair simulated with (a) a basic mass spring model with no torsion springs or self-interaction, (b) our mass-spring model including altitude springs, implicit springs and strain limiting, and (c) our mass-spring model with self-adhesion and self-collisions. Note in (a) the long straight hairs that extend from the scalp to the neck actually represent one spring which is severely stretched. Also note how much more realistic (c) looks with the addition of self-collisions and stiction.

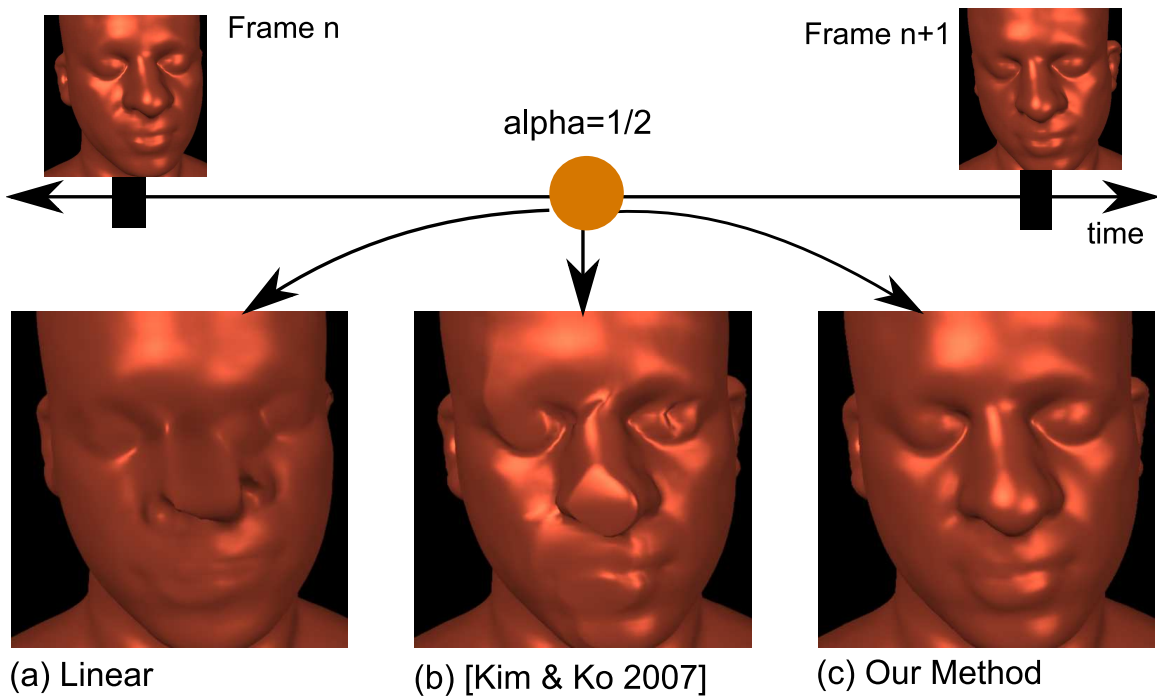


Figure 7.7: Interpolation midway between two key frames is shown using three methods: (a) naïve linear interpolation, (b) Kim and Ko’s one sided Eulerian evolution and (c) our new two-sided evolution interpolation.

stiff springs cause an implicit solve to converge slower. One could also use a method similar to [55], which would also likely be more expensive than strain limiting.

7.5 Interaction and Collisions

7.5.1 Body Collisions

We represent object collision geometry as a level set signed distance function which is negative inside and positive outside. Body collisions (and friction) proceed as described in Section 6.5 with any intersecting point having its relative normal velocities constrained to prevent collision during the second backward Euler solve in step 7. As collisions can flatten the structure of hair, flattening curls, a fold preservation

approach similar to theirs might be useful. As the head and torso are undergoing complex motion we create level sets at 60 frames per second (the frequency of our motion capture data). To represent level sets at an arbitrary time t , one could naively linearly interpolate from the closest two frames. Instead we employ a semi-Lagrangian advection based interpolation scheme similar to [86]. Using their scheme directly produces a discontinuous interpolation (not a problem for motion blur) as they only evolve *forward* from a time t^n . Instead we use the nearest two frames, interpolating forward and backward evolution via $\phi(\mathbf{x}, \alpha) = (1 - \alpha)\phi^n(\mathbf{x} - \Delta t\alpha\bar{\mathbf{v}}) + \alpha\phi^{n+1}(\mathbf{x} + \Delta t(1 - \alpha)\bar{\mathbf{v}})$ where $\bar{\mathbf{v}} = (\mathbf{v}^n(\mathbf{x}) + \mathbf{v}^{n+1}(\mathbf{x}))/2$ with the collision body velocities \mathbf{v}^n and \mathbf{v}^{n+1} , and $\alpha \in [0, 1]$ is the interpolation fraction. See Figure 7.7.

7.5.2 Stiction

Hair/hair interaction effects include friction and static charge adhesion which cause hair to stick together. In fact, the tendency of hair to merge together is one of the motivations for wisp and clump models. Springs can be useful for making and breaking dynamic constraints as in [80]. Similar techniques were then used in cloth [16] and articulated rigid bodies [30]. These techniques also were applied to hair in [20] (though they do not dynamically create new connections) and [171, 172] for wet and styled hair (though connections were created only in response to interactive styling product application).

We created a stiction model in the same spirit as the approaches listed above. First, we create segment pairs when the shortest vector between two edges is less than a distance threshold ϵ_s . Then we store the interpolation fraction along each segment (thus defining two embedded nodes) and create a spring between them that has restlength ϵ_s (making a segment pair resist not only being pulled apart but also being compressed together). These springs are maintained with the same embedded node locations until the length of the spring exceeds a separation threshold. For efficiency, we search for pairs of segments using a bounding box hierarchy containing each segment, and we only allow n connections to be formed to a segment by using a max heap to keep the n closest segments to any given segment.

Connections may be broken as hairs separate beyond the distance threshold. [171] broke connections after a force threshold was exceeded, but we avoid this, as it could improperly break a connection when two hairs are forced together. Other authors typically reduce the spring stiffness with distance to model different hairs separating at various times. We do not do this because we observed that real hair tends to exhibit discontinuous separation behavior (see Figure 7.4(c)) which is explained by the concentric scales that comprise a hair (see e.g. [127]). This is an example of one of the departures we make in modeling individual hairs as opposed to clumped or continuum models.

Adjusting stiction parameters allows a user to model various hair behaviors. For example, extra spring connections for stickiness along with a heavier mass could be used to model wet hair (see references above). Hair can also be made more or less clumped by varying the number of connections and the stiffness. Additionally, hair volume can be increased by using larger restlengths and stiffnesses. Doing so would retain a similar time step, as stiffer springs reduce the time step while longer restlengths increase it. Moreover, increasing the volume would cause hairs to be further apart, reducing the number of collision pairs—making the simulation easier. This is akin to the effect that repulsions have in the [15] algorithm, where they reduce the need for more expensive geometric collisions. However, one potential hazard of increasing the restlength is that springs become more directionally biased which could affect rotational stability, but this might be remedied by using a tetrahedron based stiction force akin to our constitutive model.

7.5.3 Self-Repulsions and Collisions

For self-repulsions and self-collisions we use the same techniques as those presented in Chapter 6. However, we note that while cloth interpenetration produces an obviously invalid state, any hair state is collision-free, and self-collisions are only apparent during motion. Thus, added computational efficiency can be achieved by ignoring some collisions; in fact we do not use rigid impact groups [123]. In that vein, we found that the stiction approach described above greatly reduces the number of edge/edge

collisions but unfortunately also has the effect of bringing edges closer together causing many false positives to be detected. Thus for efficiency and robustness, we ignore collision pairs that have a stiction connection. A comparison between Figures 7.6(b) and 7.6(c) illustrates the improved realism due to stiction, self-repulsions and self-collisions.

7.6 Examples

We have produced a number of examples ranging from a single hair, a tuft of hair, and a full head of hair. We employ the standard [81] anisotropic reflectance model and use deep shadow maps [94]. One could also use the more accurate reflectance and scattering models of [98, 105]. We stress that we do not create any additional hair geometry at render-time, only rendering what we simulated.

We built a head and torso geometry by combining a laser scan of a head with the torso from the Visible Human Data Set [157]. We captured realistic and highly dynamic motion using an optical motion capture system. The resulting motion capture skeleton was used as a boundary condition for a quasistatic flesh simulation [149], and the triangulated mesh was deformed by barycentrically embedding each vertex. Each frame of the surface was then rasterized to create a level set for head and torso collisions. To grow hairs, we painted hair densities and seeded points using [156]. The first few points at the hair root were embedded in the head model barycentrically to obtain a full coordinate constraint.

Hair Type	Particle Count	Avg m/frame	% Newmark	% Collide	% Stiction
Curly	250k	4 m	44 %	10 %	20 %
Straight short	500k	10 m	70 %	3 %	13 %
Straight short wind	500k	12 m	70 %	9 %	13 %
Straight long (5k)	500k	17 m	64 %	16 %	14 %
Straight long (10k)	1,000k	38 m	64 %	3 %	21 %

Table 7.2: Performance of simulating various hair types. We provide time/frame as well as a percent breakdown between time integration, collisions and stiction.

We simulate full heads of hair in three different styles: long straight (Figure 7.11), medium straight (Figure 7.9) and long curly (Figure 7.10). In addition to body and head motion we also show hair affected by external forces such as wind in Figure 7.8. These examples illustrate clumping behavior as well as stray individual hairs giving a multiresolution feel that is difficult to capture with clumped or continuum models. The typical time steps were 6×10^{-4} s, although we chose our time step adaptively based on the strain rate and Courant condition.

We ran our examples with one to four quad processor Opteron machines in as little as a few minutes per frame to as much as an hour per frame, when highly dynamic motion capture data strained the system. Large examples (shown in Table 7.2) used four machines and smaller examples used one machine. Parallelization was carried out in a straightforward manner discussed in Chapter 8. Simulation time (shown in Table 7.2) varies based on the velocities of the underlying head animations and wind forces even though the number of segments and time step is the same (required CG iterations may change). Note the “straight long” example where we ran 5k and 10k versions shows that doubling the number of elements approximately doubled the time which we found encouraging as conjugate gradient has larger than linear complexity. Comparing these numbers to other published results is not especially meaningful because of different hardware and levels of optimization. Also, since our examples have many more elements than others’ examples, we typically do not fit into either L1/L2 cache so a linear speedup of other methods should not be expected. Furthermore, we used high velocity motions which will make convergence slower than lower velocity examples.

7.7 Limitations

We attempted to simulate as many hairs on the head as possible and in doing so we generated many interesting, highly-detailed examples with up to a million particles. Unfortunately, we did not manage to simulate 100k hairs. Architectural improvements will naturally remedy this problem to some extent, but developing more efficient time evolution and collision approaches is also important. Thus an obvious limitation is



Figure 7.8: A simulation of 10,000 medium length straight hairs with 25 segments each (500,000 total particles) on a stationary head while interacting with wind.



Figure 7.9: A simulation of 10,000 medium length straight hairs with 25 segments each (500,000 total particles) on a character moving his head in a circular motion.



Figure 7.10: A simulation of 5,000 long curly hairs with 50 segments each (250,000 total particles) on a character spinning around from back to front.



Figure 7.11: A simulation of 10,000 long straight hairs with 50 segments each (1,000,000 total particles) on a character shaking his head from side to side.

that simulating more elements is more expensive than sparse clumped models making our method less suited for simulating hair in real time and less cost effective when a high level of detail is not required or necessary.

Using the Bridson approach for self-repulsions and collisions (in particular our variant from Chapter 6) is a limitation as it is primarily designed for interactions of edges embedded in a surface. Although it can be used for hair collision, it is not an efficient way of doing so as it has trouble with very complex stacking configurations that sometimes occur in hair. In our model, we used stiction instead of collisions for some pairs to help with this. This is analogous to Bridson’s use of rigid groups when the correct collision responses could not be determined. Improving collision response techniques in the presence of extreme stacking would allow us to apply collision restitution to all pairs. One promising approach would be to use a tetrahedral mesh together with altitude spring based techniques. This approach would also help maintain hair shape (and volume) by preventing pointwise rotations during collision response. Another issue is that wrinkles in hair tend to be flattened by body collisions which could possibly be remedied by adapting the cloth fold preservation technique of [16]. Our spring-based torsion model also contains some parasitic bending influence which is analogous to some of the problems with linear bending spring models. Research had led to the development of better non-linear bending models, but recently bending approaches have come full circle by focusing on more efficient linear bending models. Nevertheless, improved mass-spring torsion models would be an interesting avenue of research.

7.8 Discussion

Our method departs from recent hair simulation methods in several ways. First, we directly simulate every hair we render so we can capture all the DOFs needed for high fidelity hair. We use a mass-spring model that also can capture torsion which allows us to use standard time integration and interaction schemes. In order to capture electrostatic attraction, we employ a per hair stiction and friction model

whereas clumped models assume (and fix) the intra-clump interaction of hairs. Inter-clump interaction is modeled, but only in the aggregate sense, limiting discontinuous interaction behavior within clumps. Lastly, hair dynamics on aggregate models use clumped mass instead of more physical small mass hairs (which we use).

Our method represents one extreme of the performance/quality tradeoff existing between simulating individual hairs and simulating clumps or continuums. A practitioner that desires to simulate hair chooses a technique in this spectrum to satisfy his needs be them performance or fidelity. For example, in a film, a hero shot that needs very detailed single hair behavior and interactions would use our technique, whereas this is overkill for a character in the background. One might instead wish to use less computationally expensive methods such as super helices because it can represent interesting curly shapes with very few DOFs. In a film that requires stylized hair, where uniformity is desired, a volumetric/continuum method such as [118] would be ideal. In extremely time-limited scenarios such as a video game or simulator, an approach such as [3] might be applied.

An interesting avenue of research is considering combinations of techniques so that different levels of detail can be achieved in the same simulation. First, the number of hairs could be varied in our method which among other things would allow our method to be used in a clumped fashion to reduce computational cost. Thus, all the techniques for interpolating rendering hairs would be useful. One could also augment our interaction models with continuum approaches such as smooth particle hydrodynamics [66], etc. Another promising approach would be to use faster simulation techniques to iterate a basic look for a character. Then, our method could be influenced by the guide hairs to produce a simulation with the art direction and control of the coarser simulation and the intricate details of a large scale simulation.

In order to simulate 100,000 hairs, the bottlenecks at that particular resolution must be considered. The stiffness matrix and unknowns for time integration increase linearly in the number of hairs. Collision detection can theoretically scale poorly, but in our long straight example going from 5,000 to 10,000 hairs, we found that the relative percent of time on collisions (detection and response) went down, and in all of the examples in Table 7.2, they were below 16% of total simulation time.

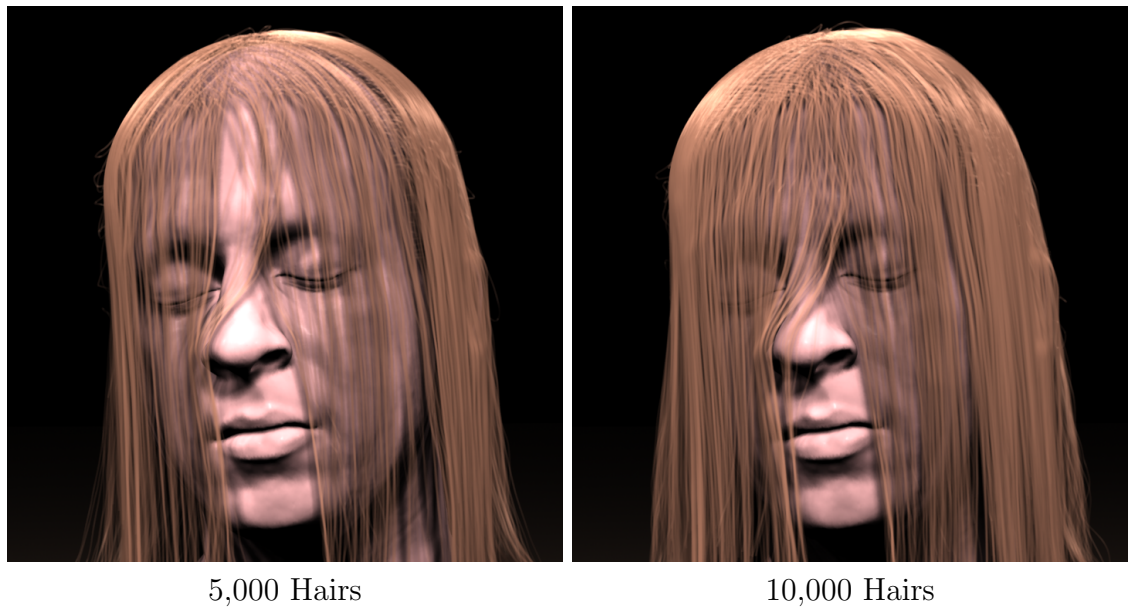


Figure 7.12: A comparison of a simulation with 5,000 hairs versus one with 10,000 hairs. Notice the significant improvement when doubling the number of hairs, and that in this pose we would benefit from even more hairs.

Even though we skipped collision *response* for stiction pairs, we did *detect all* pairs (including stiction pairs) using an axis-aligned bounding box hierarchy. As we scale to 100,000 hairs it is likely that although detection will not be a problem, more complex stacking configurations will necessitate better collision response. However, we note that detection might be sped by employing the acceleration techniques used in [168].

In conclusion, We have presented a novel mass-spring hair model with the goal of simulating every individual hair on the head. Even though mass-spring models have recently been unpopular, we have shown that they can in fact model torsion to simulate hair effectively. However, the most immediate challenge is scaling the model to more hairs. Figure 7.12 shows a simulation with 5,000 hairs compared to one with 10,000 hairs showing that double the number of hairs has a significant impact on the simulation quality justifying our goal of simulating 100,000 hairs. We believe we have taken important steps towards obtaining higher fidelity hair simulation, and we look forward to future work.

Chapter 8

Parallel Simulation

In the past years, the rate of serial performance increase with new hardware has slowed. Instead, computer architecture is increasingly relying on chip multiprocessors to obtain effective speedup. This has placed the burden of obtaining higher performance to algorithm designers and software engineers. Consequently, parallel algorithms for physical simulation need additional attention. While, most super computing applications typically consider parallelism, most practitioners outside the high performance computing community typically do not. Algorithm designers need to start evaluating their techniques through the lens of future parallelism. In this chapter we describe how some of the algorithms previously described in this dissertation are parallelized.

Parallelizing physical simulation applications typically requires spatial decomposition. This is because methods typically step forward from an initial time, and each future time is dependent on the result of each previous method. This presents a challenge, because the overhead of parallelism usually means that speedups cannot be achieved without increasing the problem size. Fortunately, for most visual effects applications, this is acceptable as practitioners usually are simulating with too little resolution to start with. However, making applications more real-time is much more challenging. In this discussion we focus on distributed memory parallelism which is typically higher overhead, but allows the simulation of very large problem sizes. The decomposition techniques outlined here are of course still applicable to shared memory

systems. We point the interested reader to [73], an analysis of physical applications on massively multi-core shared memory systems.

As an outline, we mostly follow the approach of transmitting data read dependencies of a computation, so no write synchronization hazard exists. This model usually wastes some computation and data bandwidth in exchange for less complicated synchronization.

8.1 Fluids Parallelism

Distributed memory computation for fluids has been very common in the high performance computing arena. In particular, compressible flow which is typically solved in a fully explicit manner is easy to parallelize with simple domain decomposition. In this discussion, however, we describe the parallelization of incompressible flow and consider the parallelization of the particle level set method.

8.1.1 Advection Equations

Consider the parallelization of advection (Equation 2.1) using the semi-Lagrangian method. We divide a uniform Eulerian grid into several non-overlapping pieces. Each piece is assigned to a single processor and can border up to 8 other domains (if you include corners). Each piece is responsible for updating only the grid points within it. The semi-Lagrangian backtrace ray may lead to a point that requires interpolation from grid points not owned by the processor, requiring the use of external data. However, this is similar to the problem of an interpolation point lying outside the domain. For that problem, we typically use an array with extra ghost cells on each edge of the domain. We can do the same here, but instead of filling the boundaries with domain boundary conditions, we use MPI to transmit data from another processor. Then, since each grid point is updated on the processor owning the sub-domain it belongs to, there is no write contention or synchronization issues.

8.1.2 Incompressible Solve

To make an incompressible velocity field, a Poisson equation is solved, which can also be done in parallel. Conjugate gradient requires matrix multiplies and scalar operations. The matrix multiplies can be computed so that the vector result is computed partially on each processor, however, the data dependencies of the matrix row (the non-zero entries) must be synchronized with an MPI message before this. This corresponds to neighbors cells on other processors, so it is very similar to the advection ghost region. In addition, reductions for the dot product and convergence can be computed using MPI's reduce facilities.

One complexity of parallel incompressible solves is that the very commonly used Incomplete Cholesky Preconditioner cannot easily be computed in parallel. To avoid this, we form and compute the preconditioner independently on each processor. While this avoids the need for parallel application of the preconditioner, it does reduce the convergence rate near the boundary. For incompressible solutions we found this not to produce visual artifacts for unconverged results, however, in the case of implicit viscosity (see e.g. [125]), we did need to increase the maximum iterations. For SMP based parallelism a good option for computing the preconditioner in parallel is a red-black block decomposition of the LU forward and backward solves as in [78].

8.1.3 Particle Level Set

Parallelizing the Particle Level Set Method requires parallelizing the Level Set operations and the particle operations. For the level set operations, the level set equation is simply an advection equation which is solved in parallel with the methods above.

Redistancing the level set also requires a parallel implementation. The reinitialization equation [131], can be solved explicitly using the same techniques as advection, however, this method is typically much slower than the fast marching method (FMM) [136]. Unfortunately, FMM uses a heap to keep track of which distance to update next, which is not easy to parallelize. A fine-grain locking of a shared heap might produce good speedups if we were parallelizing for an SMP machine, but for distributed

memory this approach is less desirable. Thus, we consider domain decomposition approaches. [69] considered domain decomposition approaches to FMM, but they used a complicated rollback scheme when closer data became available. Instead, we use the fact that for fluids we typically only care about a small bandwidth of correct signed distance values. Thus, we can compute independent sequential fast marching on each processor as long as a large enough synchronized ghost region is used. In general, if we want k cells of correct sign distance bandwidth, we need only $\lceil k \rceil$ ghost cells duplicated from neighboring domains. This is because distance any distance characteristics beyond that ghost region could not reach the subdomain within the k cell distance. While this requires redundant computation in the boundaries, redundancy only scales with the square (surface) rather than the cube (volume). Incidentally, this approach does not use knowledge of the redistancing method, unlike the Hermann scheme, so we could also replace each serial fast marching operation with a non-pointer based scheme like the fast-sweeping method [182]. The parallelization of level set extrapolation uses exactly the same technique.

Particles must also be handled in parallel, which is also handled through domain decomposition. Particles belong to one and only one sub domain which handles their evolution using the interpolated velocity field (using the same ghost cells as level set advection used). After evolution, particles may land in another sub-domain, to which they are then transferred via MPI. However, escaped particles also must be used to modify the Eulerian level set. This is accomplished by temporarily synchronizing particles to neighboring domains if they are close enough to modify the level set equation. These particles are tracked separately so that they can be deleted from processors that do not own them after the modify step.

8.2 Solids Parallelism

In graphics, there has been much work on shared and distributed memory parallelism for cloth simulation (see e.g. [83, 153, 154, 90, 129, 181]). Here we outline the basic approach we take toward parallelizing individual parts of a solids solver.

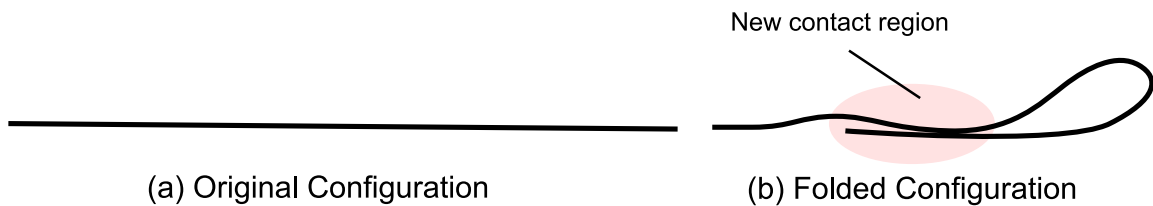


Figure 8.1: Cloth folding over itself, creating new data interdependencies.

8.2.1 Time Integration

Parallelizing time integration is one of the most important parts as, with increasing resolution, time integration begins to dominate simulation time. We spatially decompose our problem by partitioning nodes of the mesh into disjoint sets which are each assigned to a single processor. Mesh decomposition can be achieved in many ways, we typically use a simple recursive median split approach to give a power-of-two partitions with the same number of particles that are somewhat near to each other. Better approaches view the particle interdependencies as a graph, minimizing the number of edges connecting disparate partitions while at the same time considering spatial proximity for optimizing collision and contact communication. As a solids simulation proceeds, different particles might become close, and so one might repartition periodically (although we did not). See Figure 8.1 for an example where a cloth folds over and previously unconnected particles are now in contact.

Time integration is performed in parallel, by each processor having the full set of particles for the whole simulation, yet only computing forces for particles it owns. To evaluate forces, sometimes a processor will need time t^n data about particles owned by other processors. To determine what communications are necessary before force computation, each force helps build up a direct dependency graph between all the particles as a preprocess. This dependency graph is used to cache lists of where particles need to be transmitted before a force computation. A force element is computed on a given processor if it has at least one particle that needs the result of that force. For example, a linear spring between two particles on separate processors implies a data dependency, so before a force is computed each processor will get a

copy of the particle position and velocity of the particle it does not own. Then the force will be computed on both processors (obtaining the same result) and applied to their copies of the two particles even though each processor was only responsible for one of the particles.

Conjugate gradient is performed in parallel in a straightforward way. In particular, each processor has only a partial notion of the matrix (the forces it contributes to). The scalar quantities that conjugate gradient needs are obtained through parallel reduction operations using the MPI library.

Outputting data to disk is a simple matter of performing a gather to the root processor. Since we take many steps for every frame we produce, this serial process does not adversely limit scalability.

8.2.2 Repulsions and Collisions

Recall the time integration scheme that was modified for collisions and repulsions in Section 6.2. In step A and B of the outer loop we need to do searches for pairs of interacting objects so we use axis-aligned bounding box searches [158]. We synchronize the position data to every processor so each can construct a full hierarchy for points, segments, and triangles, which we use for doubly recursive traversals on every processor. Point/triangle interaction pairs are obtained by colliding the point hierarchy with the triangle hierarchy and edge/edge pairs are obtained by colliding the segment hierarchy against itself. We assign the detection of each potentially interacting pair to the processor of lowest index that owns one of the involved particles. Each processor searches for its assigned interaction pairs by pruning pairs of boxes whose expansion only contains interaction pairs assigned to other processors. Thus we avoid searching down branches of the Cartesian product tree if any interactions found would be owned by other processors.

Applying repulsions or collision responses must also be parallelized but it is important to make sure pair response is done in Gauss-Seidel ordering. That is, each pair should be processed seeing the newest data that is available, so the effect of any previously applied repulsion is used in any subsequent repulsion with which it

shares nodes. If response is done instead in Gauss-Jacobi order then impulses may be counted multiple times. While others [153] have have discussed parallelization of cloth, these papers have not discussed the importance of collision response ordering. Thus our parallel algorithm proceeds in two passes whenever we wish to apply a collision or repulsion set. We label interaction pairs (point/triangle or edge/edge) involving particles owned by different processors as *boundary pairs* and label those owned by only one processor as *internal pairs*. Using a flood-fill algorithm, each connected component of boundary pairs is processed separately (in parallel) in the first phase noting that a Gauss-Seidel ordering is still used within each connected component. Next in the second phase, modified particle velocities are sent to the processors that own the respective particles, and the remaining internal pairs are processed (again in Gauss-Seidel order) independently by the processor that contains them. See Figure 8.2 for an illustration of this application strategy. Note that this strategy always ensures the effective parallel ordering is equivalent to some serial Gauss-Seidel orderings. We note that repulsion pairs, boundary pairs and internal pairs are discovered and transferred to the appropriate processors in step A.1 and are used many times in step B.2 and B.8.

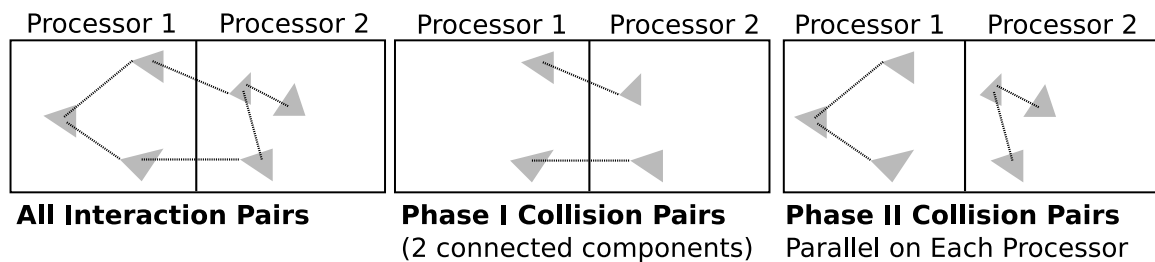


Figure 8.2: Parallel collision/repulsion pair Gauss-Seidel ordering. Phase I consists of each connected component of boundary pairs being processed (middle). Phase II computes all internal pairs using the new data from the boundary pair processing.

8.2.3 Solids Analysis

We have used the parallel solids extensively for large simulations of cloth, such as those that were presented in Section 6.6. In Figure 8.3, we show an example of parallel speedup. As you can see we obtain speedup up to 8 threads, but it declines rapidly. This is probably due to our slow gigabit interconnect between machines in our cluster. Modern multicore machines will soon have 16 cores per machine, making performance much better. In addition, connected component analysis could be performed on repulsion and collision boundary pairs so they would all need not be done in serial on one processor. In the future, we hope to perform hierarchical parallelism using data-level parallelism (SSE), thread-level parallelism (SMP), as well as distributed memory parallelism (MPI) together, so that we can optimally exploit all of the parallel opportunities that modern hardware provide. Such multi-tier parallelization approaches are becoming more important, and researchers have begun to study programming techniques for them (see e.g. [44]).

8.3 Conclusion

We have described parallel techniques used for solving both solids and cloth. As architecture evolves, algorithm designers will need to take increasing responsibility for algorithms that adapt well to common architectures. While our use of parallelism has not been entirely successful at increasing the performance of small problem sizes, we have found it useful for the simulation of high fidelity phenomena that was previously intractable.

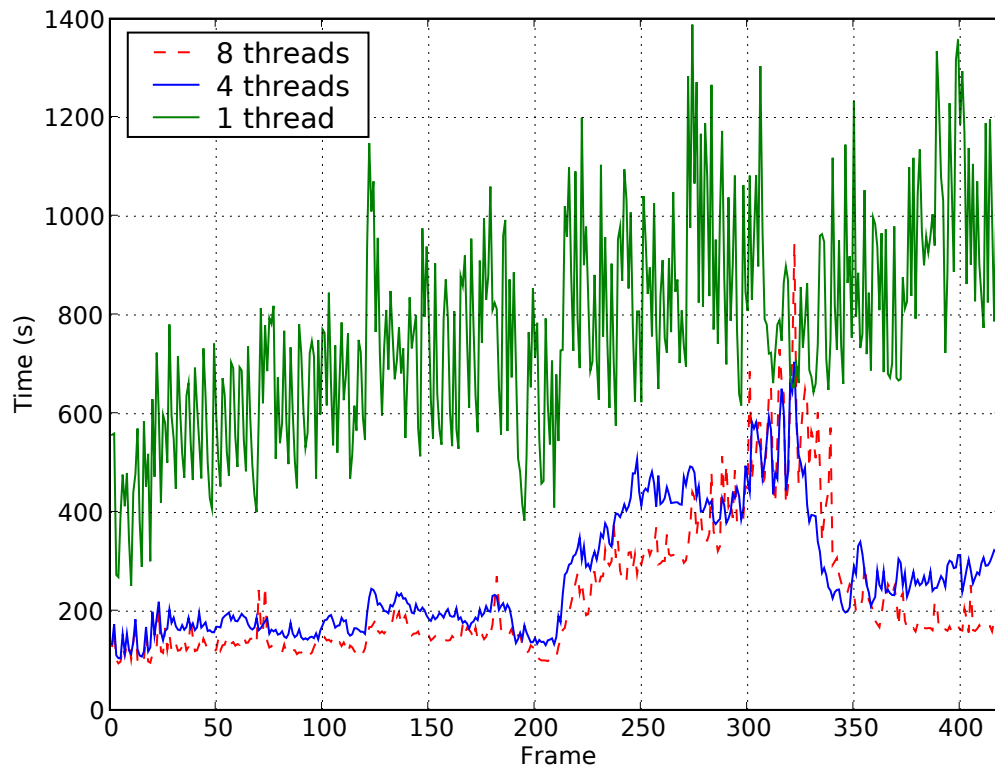


Figure 8.3: Comparison of performance of curtain and ball simulation (Figure 6.12) with increasing numbers of processors. We also ran this simulation with 16 threads, though there was not a significant speed up over 8 threads. This is most likely due to our slow gigabit ethernet interconnect.

Chapter 9

Conclusions

In this dissertation, several novel algorithms for solid and fluid simulation have been presented. While these approaches touched traditionally separate areas, they all followed a few common themes. In particular, the necessity to use the most appropriate primal representation for a phenomenon by using specific domain specific knowledge is most important. Nevertheless, even appropriate primal representations sometimes required augmentation by a complementing auxiliary representation. In Chapter 2, we used Lagrangian characteristics to gather future Eulerian data for the solution of advection equations. Then, in Chapter 3, we used vortex particles to track vorticity concentrations and reintegrated them into the grid. In Chapter 4, we used Lagrangian solids with ray tracing to define dynamic boundary conditions and one-sided interpolation for coupling thin objects to Eulerian fluids. Chapter 6, we used less accurate but fast repulsions together with more accurate but slower geometric collisions. In Chapter 7, we used volumetric tetrahedral methods for simulating hair segments to track torsion. Such hybridizations have become quite common and will continue to be, as researchers attempt to solve remaining simulation problems.

Besides hybridization, another theme is choosing algorithms not because they are high order, but because they can produce high fidelity results. High order methods are better than low order methods only in convergence, but since simulations are typically run coarsely, convergence is not near. In these cases, a method that is low order but has a low error constant, can do better than a high order method. For example,

we introduce a 2nd order advection scheme in Chapter 2, but we produce a coupled Lagrangian/Eulerian solver that can produce detailed simulations in Chapter 3. Also, high order accuracy does not help with discontinuous phenomena such as boundary conditions or collisions. In the cloth and hair examples, we used low order constitutive models so we could simulate very high resolution meshes and highly detailed collisions and interactions.

There are many other avenues of future work in physical simulation, and we have presented some of them in the individual chapters, so here we consider more general problems. The most obvious and probably most important future work is all-way coupling between solvers. The work of Chapter 4 was used for two-way coupling of fluids and solids in [60]. More recently, other work has considered two-way coupling of fluids with solids [128] and two-way coupling of deformable solids with rigid solids [137]. These works take steps in the right direction, but future work must go further by more tightly integrating time integration, collisions and all other parts of simulation algorithms. Lagrangian and Eulerian representations must be able to interact better and more seamlessly and collision handling must be fully general. Besides coupling, another obvious avenue of future work is increased interactive performance of simulation techniques while maintaining high quality results. This will be eased by the use of parallelism together with commodity chip multiprocessors, but like real-time rendering compared to offline rendering, this might require specialized algorithms. Of course, physical simulation can always benefit from additional resolution and fidelity, so additional work on highly turbulent flows, better collision algorithms, and better constitutive models, remains important.

All of the techniques presented in this thesis were demonstrated through simulations that were judged based on their visual plausibility. In particular, examples were shown that were plausible, that is belonging to the set of possible simulations in the world. This shows that we are simulating with a valid model and solving it sufficiently well for our purposes. That is to say, the simulator can produce a subset of the set of visually plausible phenomena. A much more challenging problem is obtaining the particular plausible situation subject to some additional constraints. Not only does this require a method that can choose the proper plausible simulation result, but it

requires that the simulator can produce all the desired plausible phenomena. Thus the quality of a simulator must be even better to allow this type of control. This perhaps will lead to a fully controllable and realistic virtual world.

Bibliography

- [1] J.D. Anderson. *Computational Fluid Dynamics: The Basics With Applications*. McGraw-Hill, 1995. New York, NY.
- [2] Kenichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. In *Comp. Graph. (Proc. ACM SIGGRAPH 92)*, volume 26, pages 111–120. ACM, 1992.
- [3] Yosuke Bando, Bing-Yu chen, and Tomoyuki Nishita. Animating hair with loosely connected particles. In *Comp. Graph. Forum (Eurographics Proc.)*, pages 411–418, 2003.
- [4] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.
- [5] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. SIGGRAPH 94*, pages 23–34, 1994.
- [6] D. Baraff and A. Witkin. Large steps in cloth simulation. In *ACM SIGGRAPH 98*, pages 43–54. ACM Press/ACM SIGGRAPH, 1998.
- [7] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22:862–870, 2003.
- [8] D. Benson. Computational methods in Lagrangian and Eulerian hydrocodes. *Comput. Meth. in Appl. Mech. and Eng.*, 99:235–394, 1992.
- [9] M. Berger and J. Olinger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.

- [10] M. Bergou, M. Wardetzky, D. Harmon, D. Zorin, and E. Grinspun. A quadratic bending model for inextensible surfaces. In *Proc. of Eurographics Symp. on Geometry Processing*, pages 227–230, 2006.
- [11] F. Bertails, T-Y. Kim, M-P. Cani, and U. Neumann. Adaptive wisp tree - a multiresolution control structure for simulating dynamics clustering in hair motion. *ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 207–213, 2003.
- [12] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frédéric Leroy, and Jean-Luc Lévêque. Super-helices for predicting the dynamics of natural hair. *ACM Trans. Graph.*, 25(3):1180–1187, 2006.
- [13] E. Boxerman and U. Ascher. Decomposing cloth. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 153–161, 2004.
- [14] D. E. Breen, D. H. House, and M. J. Wozny. Predicting the drape of woven cloth using interacting particles. *Comput. Graph. (SIGGRAPH Proc.)*, pages 365–372, 1994.
- [15] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, 2002.
- [16] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 28–36, 2003.
- [17] Joel Brown, Jean-Claude Latombe, and Kevin Montgomery. Real-time knot-tying simulation. *Vis. Comput.*, 20(2):165–179, 2004.
- [18] M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *Comput. Graph. (SIGGRAPH Proc.)*, pages 99–104, 1992.

- [19] M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23:377–384, 2004.
- [20] J. T. Chang, J. Jin, and Y. Yu. A practical model for hair mutual interactions. In *Proc. ACM SIGGRAPH Symp. on Comput. Anim.*, pages 77–80, 2002.
- [21] B. Choe, M.G. Choi, and H-S. Ko. Simulating complex hair with robust collision handling. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 153–160, 2005.
- [22] Byoungwon Choe and Hyeong-Seok Ko. A statistical wisp model and pseudo-physical approaches for interactive hairstyle generation. *IEEE Trans. on Vis. and Comput. Graph.*, 11(2):160–170, 2005.
- [23] K.-J. Choi and H.-S. Ko. Stable but responsive cloth. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:604–611, 2002.
- [24] M.-H. Choi, M. Hong, and S. Welch. Modeling and simulation of sharp creases. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.
- [25] A. Chorin. A numerical method for solving incompressible viscous flow problems. *J. Comput. Phys.*, 2:12–26, 1967.
- [26] J. M. Cohen and M. J. Molemaker. Practical simulation of surface tension flows. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.
- [27] F. Cordier, H. Seo, and N. Magnenat-Thalmann. Made-to-measure technologies for an online clothing store. *IEEE Comput. Graph. and Appl.*, 23(1):38–48, Jan 2003.
- [28] Georges-Henri Cottet and P. Poncet. Advances in direct numerical simulations of 3D wall-bounded flows by vortex-in-cell methods. *J. Comput. Phys.*, 193:136–158, 2003.

- [29] R. Courant, E. Issacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Comm. Pure and Applied Math*, 5:243–255, 1952.
- [30] B. Criswell, K. Derlich, and D. Hatch. Davy jones’ beard: rigid tentacle simulation. In *SIGGRAPH 2006 Sketches*, page 117, 2006.
- [31] L. Cutler, R. Gershbein, X.C. Wang, C. Curtis, E. Curtis, C. Curtis, E. Maigret, and L. Prasso. An art-directed wrinkle system for CG character clothing. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 2005.
- [32] P. Decaudin, D. Julius, J. Wither, L. Boissieux, A. Sheffer, and M.-P. Cani. Virtual garments: A fully geometric approach for clothing design. In *Comp. Graph. Forum (Eurographics Proc.)*, 2006.
- [33] M. Desbrun and M.-P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In R. Boulic and G. Hegron, editors, *Comput. Anim. and Sim. ’96 (Proc. of EG Wrkshp. on Anim. and Sim.)*, pages 61–76. Springer-Verlag, Aug 1996. Published under the name Marie-Paule Gascuel.
- [34] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Graph. Interface*, pages 1–8, 1999.
- [35] M. Drela and E. M. Murman. Prospects for Eulerian CFD analysis of helicopter vortex flows. In *American Helicopter Society Specialist Meeting, Arlington Texas*, Feb 1987.
- [36] T. Dupont and Y. Liu. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *J. Comput. Phys.*, 190/1:311–324, 2003.
- [37] T. Dupont and Y. Liu. Back and forth error compensation and correction methods for semi-Lagrangian schemes with application to level set interface computations. *Math. Comp.*, In Press., 2006.

- [38] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183:83–116, 2002.
- [39] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures*, 83:479–490, 2005.
- [40] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):736–744, 2002.
- [41] D. Enright, D. Nguyen, F. Gibou, and R. Fedkiw. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proc. 4th ASME-JSME Joint Fluids Eng. Conf.*, number FEDSM2003–45144. ASME, 2003.
- [42] O. Eitzmuß. *Animation of Surfaces with Applications to Cloth Modelling*. PhD thesis, Tübingen, 2002.
- [43] O. Eitzmuss, M. Keckeisen, and W. Strasser. A fast finite element solution for cloth modelling. In *Pacific Graph.*, pages 244–251, 2003.
- [44] K. Fatahalian, T. Knight, M. Houston, M. Erez, D. Horn, L. Leem, Ji P, M. Ren, A. Aiken, B. Dally, and P. Hanrahan. Sequoia: Programming for memory hierarchy. In *Proc. of Supercomputing*, 2006.
- [45] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In *Proc. of ACM SIGGRAPH 2001*, pages 15–22, 2001.
- [46] B. Feldman, J. O’Brien, and O. Arikan. Animating suspended particle explosions. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):708–715, 2003.
- [47] H. Felici and M. Drela. Eulerian/Lagrangian solution of 3-d rotational flows. In *AIAA 21st Fluid Dynamics, Plasma Dynamics and Lasers Conf.*, Jun 1990.
- [48] H. Felici and M. Drela. An Eulerian/Lagrangian coupling procedure for three-dimensional vortical flows. *AIAA J.*, (1993-3370), 1993.

- [49] H. Felici and M. Drela. Reduction of numerical diffusion in three-dimensional vortical flows using a coupled Eulerian/Lagrangian solution procedure. In *AIAA 24th Fluid Dynamics Conf.*, Jul 1993.
- [50] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proc. of ACM SIGGRAPH 2001*, pages 23–30, 2001.
- [51] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *Proc. of SIGGRAPH 97*, pages 181–188, 1997.
- [52] M. Gamito, P. Lopes, and M. Gomes. Two dimensional simulation of gaseous phenomena using vortex particles. In *Proc. of the 6th Eurographics Wrkshp. on Comput. Anim. and Sim.*, pages 3–15. Springer-Verlag, 1995.
- [53] O. Génevaux, A. Habibi, and J.-M. Dischler. Simulating fluid-solid interaction. In *Graph. Interface*, pages 31–38, June 2003.
- [54] T. G. Goktekin, A. W. Bargteil, and J. F. O’Brien. A method for animating viscoelastic fluids. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23:463–468, 2004.
- [55] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient simulation of inextensible cloth. *ACM Trans. Graph.*, 26(3):49, 2007.
- [56] N. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):991–999, 2005.
- [57] Mireille Grégoire and Elmar Schömer. Interactive simulation of one-dimensional flexible parts. In *Symp. on Solid and Physical Modeling*, pages 95–103, 2006.
- [58] E. Grinspun, A. Hirani, M. Desbrun, and P. Schröder. Discrete shells. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 62–67, 2003.

- [59] E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A simple framework for adaptive simulation. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:281–290, 2002.
- [60] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):871–878, 2003.
- [61] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):973–981, 2005.
- [62] Rajeev Gupta, Melanie Montagnoo, Pascal Volino, and Nadia Magnenat-Thalmann. Optimized framework for real time hair simulation. In *CGI Proc. 2006*, pages 702–710, 2006.
- [63] S. Hadap. Oriented strands: dynamics of stiff multi-body system. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 91–100, 2006.
- [64] S. Hadap, E. Bangarter, P. Volino, and N. Magnenat-Thalmann. Animating wrinkles on clothes. *Proc. of Visualization*, pages 175–523, 1999.
- [65] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Comput. Graph. Forum*, 20(3), 2001.
- [66] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. In *Comp. Graph. Forum (Eurographics Proc.)*, pages 329–338, 2001.
- [67] J. Hahn. Realistic animation of rigid bodies. *Comput. Graph. (Proc. SIGGRAPH 88)*, 22(4):299–308, 1988.
- [68] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids*, 8:2182–2189, 1965.
- [69] M. Herrmann. A domain decomposition parallelization of the fast marching method. *Center for Turbulence Research Annual Brief*, 2003.

- [70] M. Herrmann and G. Blanquart. Flux corrected finite volume scheme for preserving scalar boundedness in reacting large-eddy simulations. *AIAA J.*, 44(12):2879–2886, 2006.
- [71] Jeong-Mo Hong and Chang-Hun Kim. Animation of bubbles in liquid. *Comput. Graph. Forum (Eurographics Proc.)*, 22(3):253–262, 2003.
- [72] D. H. House and D. E. Breen, editors. *Cloth modeling and animation*. A. K. Peters, 2000.
- [73] C. Hughes, R. Grzeszczuk, E. Sifakis, D. Kim, S. Kumar, A. Selle, J. Chhugani, M. Holliman, and Y.-K. Chen. Physical simulation for animation and visual effects: Parallelization and characterization for chip multiprocessors. In *Intl. Symp. on Comput. Architecture*, 2007.
- [74] S. Huh and D. Metaxas. A collision resolution algorithm for clump-free fast moving cloth. In *Proc. of Comp. Graph. Intl.*, 2005.
- [75] S. Huh, D. N. Metaxas, and N. I. Badler. Collision resolutions in cloth simulation. In *Comput. Anim. IEEE*, 2001.
- [76] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 25(3):805–811, 2006.
- [77] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 131–140, 2004.
- [78] T. Iwashita and M. Shimasaki. Block red-black ordering method for parallel processing of iccg solver. In *Proc. of the 4th Intl. Symp. on High Performance Computing*. Springer Verlag, 2002.
- [79] G.-S. Jiang and D. Peng. Weighted ENO schemes for Hamilton-Jacobi equations. *SIAM J. Sci. Comput.*, 21:2126–2143, 2000.

- [80] S. Jimenez and A. Luciani. Animation of interacting objects with collisions and prolonged contacts. In B. Falcidieno and T. L. Kunii, editors, *Modeling in computer graphics—methods and applications*, Proc. of the IFIP WG 5.10 Working Conf., pages 129–141. Springer-Verlag, 1993.
- [81] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In *Comp. Graph. (Proc. ACM SIGGRAPH 90)*, pages 271–280. ACM, 1989.
- [82] M. Kang, R. Fedkiw, and X.-D. Liu. A boundary condition capturing method for multiphase incompressible flow. *J. Sci. Comput.*, 15:323–360, 2000.
- [83] M. Keckeisen and W. Blochinger. Parallel implicit integration for cloth animations on distributed memory architectures. *EG Symposium on Parallel Graphics and Visualization*, 2004.
- [84] B.-M. Kim, Y. Liu, I. Llamas, and J. Rossignac. Using BFEC for fluid simulation. In *Eurographics Workshop on Natural Phenomena 2005*, 2005.
- [85] B.-M. Kim, Y. Liu, I. Llamas, and J. Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Trans. on Vis. and Comput. Graph.*, In Press., 2006.
- [86] Doyub Kim and Hyeong-Seok Ko. Eulerian motion blur. In *Eurographics Workshop on Natural Phenomena 2007*, pages 39–46, 2007.
- [87] T.-Y. Kim and U. Neumann. Interactive multiresolution hair modeling and editing. *ACM Trans. Graph.*, 21(3):620–629, 2002.
- [88] N. Kondoh, A. Kunimatsu, and S. Sasagawa. Creating animations of fluids and cloth with moving characters. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.
- [89] A. Lamorlette and N. Foster. Structural modeling of flames for a production environment. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):729–735, 2002.

- [90] R. Lario, C. Garcia, M. Prieto, and F. Tirado. Rapid parallelization of a multi-level cloth simulator using openmp. In *Third European Workshop on OpenMP*, 2001.
- [91] Z. Li and M.-C. Lai. The immersed interface method for the Navier-Stokes equations with singular forces. *J. Comput. Phys.*, 171:822–842, 2001.
- [92] K. Lindsay and R. Krasny. A particle method and adaptive treecode for vortex sheet motion in three-dimensional. *J. Comput. Phys.*, 172:879–907, 2001.
- [93] L. Ling, M. Damodaran, and K. Gay. Aerodynamic force models for animating cloth motion in air flow. In *The Vis. Comput.*, pages 84–104, 1996.
- [94] Tom Lokovic and Eric Veach. Deep shadow maps. In *ACM SIGGRAPH 2000*, pages 385–392. ACM Press/ACM SIGGRAPH, 2000.
- [95] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids*, 35:995–1010, 2006.
- [96] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23:457–462, 2004.
- [97] R. MacCormack. The effect of viscosity in hypervelocity impact cratering. In *AIAA Hypervelocity Impact Conference*, 1969. AIAA paper 69-354.
- [98] Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. *ACM Trans. Graph.*, 22(3):780–791, 2003.
- [99] R. McDonnell, S. Dobbyn, S. Collins, and C. O’Sullivan. Perceptual evaluation of LOD clothing for virtual humans. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 2006.
- [100] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):449–456, 2004.

- [101] Mark Meyer, Gilles Debunne, Mathieu Desbrun, and Alan H. Barr. Interactive animation of cloth-like objects in virtual reality. *The Journal of Visualization and Computer Animation*, 12(1):1–12, 2001.
- [102] V. Mihalef, D. Metaxas, and M. Sussman. Animation and control of breaking waves. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 315–324, 2004.
- [103] C. Min and F. Gibou. A second order accurate projection method for the incompressible Navier-Stokes equation on non-graded adaptive grids. *J. Comput. Phys.*, 219:912–929, 2006.
- [104] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable*, pages 103–114, 2003.
- [105] Jonathan T. Moon and Stephen R. Marschner. Simulating multiple scattering in hair using a photon mapping approach. *ACM Trans. Graph.*, 25(3):780–791, 2006.
- [106] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Comput. Graph. (Proc. SIGGRAPH 88)*, 22(4):289–298, 1988.
- [107] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 154–159, 2003.
- [108] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. Interaction of fluids with deformable solids. *J. Comput. Anim. and Virt. Worlds*, 15(3–4):159–171, July 2004.
- [109] D. Nguyen, R. Fedkiw, and H. Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:721–728, 2002.
- [110] S. Oh, J. Ahn, and K. Wohn. Low damped cloth simulation. *Visual Computer*, 22(2), February 2006.

- [111] H. Okabe, H. Imaoka, T. Tomiha, and H. Niwaya. Three dimensional apparel CAD system. *Comput. Graph. (SIGGRAPH Proc.)*, pages 105–110, 1992.
- [112] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002. New York, NY.
- [113] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79:12–49, 1988.
- [114] Dinesh K. Pai. Strands: Interactive simulation of thin solids using cosserat models. In *Proc. of Eurographics*, volume 21 of *Comput. Graph. Forum*, pages 347–352. Eurographics Assoc., 2002.
- [115] D. Parks and D. Forsyth. Improved integration for cloth simulation. In *Proc. of Eurographics*, *Comput. Graph. Forum*. Eurographics Assoc., 2002.
- [116] C. Peskin. Flow patterns around heart valves: A numerical method. *J. Comput. Phys.*, 10:252–271, 1972.
- [117] C. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.
- [118] L. Petrovic, M. Henne, and J. Anderson. Volumetric methods for simulation and rendering of hair. Technical Report 06-08, Pixar, 2005.
- [119] R. Peyret and T. Taylor. *Computational methods for fluid flow*. Springer-Verlag, 1983. New York.
- [120] E. Plante, M.-P. Cani, and P. Poulin. Capturing the complexity of hair motion. *Graph. Models*, 64(1):40–58, january 2002.
- [121] P. Ploumhans, G. S. Winckelmans, J. K. Salmon, A. Leonard, and M. S. Warre. Vortex methods for direct numerical simulation of three-dimensional bluff body flows: Application to the sphere at $re=300$, 500, and 1000. *J. Comput. Phys.*, 178:427–463, 2002.

- [122] S. Premoze, T. Tasdizen, J. Bigler, A. Lefohn, and R. Whitaker. Particle-based simulation of fluids. In *Comp. Graph. Forum (Eurographics Proc.)*, volume 22, pages 401–410, 2003.
- [123] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graph. Interface*, pages 147–154, may 1995.
- [124] X. Provot. Collision and self-collision handling in cloth model dedicated to design garment. *Graph. Interface*, pages 177–89, 1997.
- [125] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 193–202, 2004.
- [126] N. Rasmussen, D. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22:703–707, 2003.
- [127] Clarence R. Robbins. *Chemical and physical behavior of human hair*. Springer-Verlag, New York, 1994.
- [128] A. Robinson-Mosher, T. Shinar, J. Grétarsson, J. Su, and R. Fedkiw. Two-way coupling of fluids to rigid and deformable solids and shells. *ACM Trans. Graph. (In Press)*, 2008.
- [129] S. Romero, L. Romero, and E.L. Zapata. Rapid parallelization of a multilevel cloth simulator using openmp. *EuroPar*, 2000.
- [130] R. E. Rosenblum, W. E. Carlson, and E. Tripp III. Simulating the structure and dynamics of human hair: modelling, rendering and animation. *J. Vis. and Comput. Anim.*, 2(4):141–148, 1991.
- [131] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM J. Num. Analysis*, 29:867–884, 1992.

- [132] A. Selle, R. Fedkiw, B.-M. Kim, Y. Liu, and J. Rossignac. An unconditionally stable maccormack method. *J. Sci. Comput.*, In Press., 2007.
- [133] A. Selle, M. Lentine, and R. Fedkiw. A mass spring model for hair simulation. *ACM Trans. Graph. (In Press)*, 2008.
- [134] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):910–914, 2005.
- [135] A. Selle, J. Su, G. Irving, and R. Fedkiw. Highly detailed folds and wrinkles for cloth simulation. *IEEE Trans. on Vis. and Comput. Graph. (In Press)*, 2008.
- [136] J. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci.*, 93:1591–1595, 1996.
- [137] T. Shinar, C. Schroeder, and R. Fedkiw. Two-way coupling of rigid and deformable bodies. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim. (In Review)*, 2008.
- [138] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock capturing schemes. *J. Comput. Phys.*, 77:439–471, 1988.
- [139] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw. Hybrid simulation of deformable solids. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 81–90, 2007.
- [140] J. Spillmann and M. Teschner. CoRDE: cosserat rod elements for the dynamic simulation of one-dimensional elastic object. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 209–217, 2007.
- [141] J. Stam. Stable fluids. In *Proc. of SIGGRAPH 99*, pages 121–128, 1999.
- [142] A. Staniforth and J. Cote. Semi-Lagrangian integration schemes for atmospheric models: A review. *Monthly Weather Review*, 119:2206–2223, 1991.

- [143] J. Steinhoff and D. Underhill. Modification of the Euler equations for “vorticity confinement”: Application to the computation of interacting vortex rings. *Phys. of Fluids*, 6(8):2738–2744, 1994.
- [144] J. Strain. Tree methods for moving interfaces. *J. Comput. Phys.*, 151:616–648, 1999.
- [145] John Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. SIAM Press, 2007.
- [146] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha. Fast proximity computation among deformable models using discrete Voronoi diagrams. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 25(3):1144–1153, 2006.
- [147] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys.*, 114:146–159, 1994.
- [148] J. Teran, S. Blemker, V. Ng, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 68–74, 2003.
- [149] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. Robust quasistatic finite elements and flesh simulation. *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 181–190, 2005.
- [150] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscoelasticity, plasticity, fracture. *Comput. Graph. (SIGGRAPH Proc.)*, pages 269–278, 1988.
- [151] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Comput. Graph. (Proc. SIGGRAPH 87)*, 21(4):205–214, 1987.
- [152] J. A. Thingvold and E. Cohen. Physical modeling with B-spline surfaces for interactive design and animation. *Comput. Graph. (SIGGRAPH Proc.)*, pages 129–137, 1992.

- [153] B. Thomaszewski and W. Blochinger. Parallel simulation of cloth on distributed memory architectures. *EG Symposium on Parallel Graphics and Visualization*, 2006.
- [154] B. Thomaszewski, S. Pabst, and W. Blochinger. Exploiting parallelism in physically-based simulations on multi-core processor architectures. *EG Symposium on Parallel Graphics and Visualization*, 2007.
- [155] B. Thomaszewski, M. Wacker, and W. Strasser. A consistent bending model for cloth simulation with corotational subdivision finite elements. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 2006.
- [156] G. Turk. Re-tiling polygonal surfaces. In *Comput. Graph. (Proc. ACM SIGGRAPH 92)*, pages 55–64. ACM, 1992.
- [157] U.S. National Library of Medicine. The visible human project, 1994. <http://www.nlm.nih.gov/research/visible/>.
- [158] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools*, 2(4):1–14, 1997.
- [159] P. Volino, M. Courchesne, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *Comput. Graph. (SIGGRAPH Proc.)*, pages 137–144, 1995.
- [160] P. Volino and N. Magnenat-Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In *Proc. of Eurographics*, volume 13 of *Comput. Graph. Forum*, pages C–155–166. Eurographics Assoc., 1994.
- [161] P. Volino and N. Magnenat Thalmann. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In D. Terzopoulos and D. Thalmann, editors, *Comp. Anim. and Simulation*, pages 55–65. Springer-Verlag, 1995.

- [162] P. Volino and N. Magnenat-Thalmann. Accurate collision response on polygonal meshes. In *Proc. of Comput. Anim.*, pages 154–163, 2000.
- [163] P. Volino and N. Magnenat-Thalmann. Implicit midpoint integration and adaptive damping for efficient cloth simulation. *Computer Animation and Virtual Worlds*, 16:163–175, 2005.
- [164] P. Volino and N. Magnenat-Thalmann. Resolving surface collisions through intersection contour minimization. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 25(3):1154–1159, 2006.
- [165] P. Volino and N. Magnenat-Thalmann. Simple linear bending stiffness in particle systems. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 101–105, 2006.
- [166] J. H. Walther and P. Koumoutsakos. Three-dimensional vortex methods for particle-laden flows with two-way coupling. *J. Comput. Phys.*, 167:39–71, 2001.
- [167] F. Wang, E. Burdet, A. Dhanik, T. Poston, and C.L. Teo. Dynamic thread for real-time knot-tying. *Eurohaptics Conf., 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, pages 507–508, 2005.
- [168] K. Ward, N. Galoppo, and M. Lin. Interactive virtual hair salon. *Presence: Teleoper. Virt. Environ.*, 16(3):237–251, 2007.
- [169] K. Ward, N. Galoppo, and M.C. Lin. A simulation-based vr system for interactive hairstyling. *Virt. Reality Conf., 2006*, pages 257–260, 2006.
- [170] Kelly Ward, Florence Bertails, Tae-Yong Kim, Stephen R. Marschner, Mari-Paule Cani, and Ming C. Lin. A survey on hair modeling: Styling, simulation and rendering. *IEEE Trans. on Vis. and Comput. Graph.*, 13(2):213–234, 2007.
- [171] Kelly Ward, Nico Galoppo, and Ming C. Lin. Modeling hair influenced by water and styling products. In *Proc. of Comput. Anim. and Social Agents (CASA)*, pages 207–214, 2004.

- [172] Kelly Ward, Nico Galoppo, and Ming C. Lin. Simulating and rendering wet hair. In *SIGGRAPH 2004 Sketches*, page 42. ACM Press, 2004.
- [173] Kelly Ward and Ming C. Lin. Adaptive grouping and subdivision for simulating hair dynamics. In *Pacific Graph.*, page 234, 2003.
- [174] Kelly Ward, Ming C. Lin, Joohee Lee, Susan Fisher, and Dean Macri. Modeling hair using level-of-detail representations. In *Proc. of Comput. Anim. and Social Agents (CASA)*, page 41, 2003.
- [175] R.F. Warming and R.M. Beam. Upwind second-order difference schemes and applications in aerodynamic flows. *AIAA J.*, 14(9):1241–1249, 1976.
- [176] X. Wei, Y. Zhao, Z. Fan, W. Li, S. Yoakum-Stover, and A. Kaufman. Blowing in the wind. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 75–85, 2003.
- [177] J. Weil. The synthesis of cloth objects. *Comput. Graph. (SIGGRAPH Proc.)*, pages 49–54, 1986.
- [178] J. Wejchert and D. Haumann. Animation aerodynamics. *Comput. Graph.*, 25(4):19–22, 1991.
- [179] L. Yaeger and C. Upson. Combining physical and visual simulation - creation of the planet jupiter for the film 2010. In *Proc. of SIGGRAPH 1986*, pages 85–93, 1986.
- [180] Yizhou Yu. Modeling realistic virtual hairstyles. In *Pacific Graph.*, pages 295–304, 2001.
- [181] F. Zara, F. Faure, and J.-M Vincent. Physical cloth animation on a pc cluster. In *Fourth Eurographics Workshop on Parallel Graphics and Visualisation*, 2002.
- [182] Hong-Kai Zhao. A fast sweeping method for eikonal equations. *Math. of Comp.*, 74:603–627, 2005.

- [183] L. Zhu and C. Peskin. Simulation of a flapping flexible filament in a flowing soap film by the immersed boundary method. *J. Comput. Phys.*, 179:452–468, 2002.