METHODS FOR THE PHYSICALLY BASED 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

Geoffrey Irving
June 2007

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.

(Ron 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.

(Leonidas Guibas)

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.

(Matthew West)

Approved for the University Committee on Graduate Studies.

Abstract

This thesis presents several techniques for the numerical simulation of solid and fluid phenomena, including elastoplastic volumetric solids, incompressible elastic solids, large bodies of water, and viscoelastic fluids.

We first present a method for modifying a finite element discretization to ensure robustness under arbitrarily large deformation, including situations with degenerate or inverted elements (Chapter 2). The algorithm is straightforward to implement and can be used with any material constitutive model, and for both volumetric solids and thin shells such as cloth. We also discuss improved time integration methods for collisions and friction with objects (Chapter 3). Since a primary application of these methods is to biological tissue, we next consider an extension to incompressible materials which avoids the locking problems normally associated with first order tetrahedral elements (Chapter 4) and works well in situations involving complex collision and contact.

Turning to fluids, we present a new adaptive scheme for water simulation which uses uniform cells near the surface to capture detail and coarsens away from the interface with tall, thin cells (Chapter 5). This allows us to take advantage of the flat structure of most large bodies of water to efficiently capture bulk motion while preserving three-dimensional surface effects. We coarsen with tall, thin cells (as opposed to octrees or AMR), because they maintain good resolution horizontally allowing for accurate representation of bottom topography.

Finally, we present a simple method for simulating viscoelastic fluid flow which tracks a rotated linear version of the strain rather than the more common quadratic metric tensor, and treats the rotational term in the evolution using rotations directly to ensure stability (Chapter 6).

Acknowledgments

To Weronika: you have been a wonderful source of love and delight as long as I have known you, and I know you will continue to be so in the future. I am very happy to have been able to share myself with you. Also, I would not have reached this point without the enthusiasm, curiosity, and perfectionism of my dad, and my mom in all her roles as friend, teacher, sparring partner, etc.

I will always be grateful to my advisor, Ron Fedkiw. He remains one of the most impressive people I have known, and while my four years with him have been difficult on a few occasions, I would choose it all again in a second. Not least, I want to thank him for assembling an amazing group of students. None of this work would have been possible without the collaboration of my co-authors, Joey Teran, Eftychis Sifakis, Frank Losasso, Eran Guendelman, Craig Schroeder, Tamar Shinar, Andrew Selle, and Jonathan Su. Thank you for math, soccer, and ping pong.

I want to thank the members of my reading committee, Ron Fedkiw, Matt West, and Leo Guibas, for all their support in the last stage of my graduate career, and Adrian Lew and Michael Kass for their participation in my oral committee.

The algorithms discussed in this dissertation form part of the PhysBAM simulation library, and it has been very fun to be involved in its growth and development. The ever expanding list of developers currently includes Josh Bao, Robert Bridson, Douglas Enright, Ron Fedkiw, Eran Guendelman, Frederic Gibou, Sergey Koltakov, Nipun Kwatra, Frank Losasso, Ian Mitchell, Neil Molino, Igor Neverov, Duc Nguyen, Nick Rasmussen, Avi Robinson-Mosher, Craig Schroeder, Andrew Selle, Tamar Shinar, Eftychios Sifakis, Jonathan Su, and Jerry Talton.

I owe a lot to the two people who started me out in graphics, Al Barr and Mathieu Desbrun. Al for his endless (?) enthusiasm, encouragement, and thesis organization advice, and Mathieu for a first glimpse into the beauty of simplices.

Most of my funding at Stanford was provided by an NSF Graduate Fellowship.

John Anderson at Pixar Animation Studios has been and remains a tremendous source of humor, knowledge, and insight on almost every subject. I have learned a lot from him and the other members of the Pixar research group. Thanks also to my other collaborators at Pixar, especially Martin Nguyen, Gordon Cameron, and Ryan Kautzman.

And, to Tamar: thank you for falling off your bike, and calling to tell me about it.

Contents

List of Figures

Chapter 1

Introduction

Physical simulation for computer graphics is in different ways both more and less demanding than simulation for computational physics and engineering. The latter places strict requirements on validation, in terms of convergence analysis, reproducibility, and comparison with experiment. This tends to focus effort on problems where convergence can be achieved, which often restricts results to two dimensions or simple, well-defined test cases. For a converged problem, degenerate configurations can typically be avoided through additional refinement, so accuracy in well-resolved regions can be more important than robustness.

In contrast, graphics applications emphasize detailed environments with complex time-varying boundary conditions and occasional nonphysical components. Detail is more important than accuracy: many wrinkles resolved with one or two elements each at the cost of incorrect stiffness is preferable to a few accurately resolved wrinkles. This focuses attention on the low resolution behavior of numerical schemes, where discretization errors can often be much larger than the size of a single element. From an algorithm design standpoint, the advantage of the graphics approach is that a detailed simulation with large numbers of elements but mostly underresolved phenomena is much more likely to explore degenerate configurations that can cause a scheme to fail than a converged test case.

This thesis presents several techniques for the physical simulation of solid and fluid phenomena, including elastoplastic volumetric solids, incompressible elastic solids, large bodies of water, and viscoelastic fluids. In each case we seek the simplest available scheme that captures the particular properties of the physical system we want to reproduce. This is often most easily accomplished by seeking schemes that are exact in special cases, such as the friction method in Chapter 3 which correctly reproduces the motion of a single particle on an inclined plane. When it is useful to depart from accuracy in favor of robustness or efficiency, we attempt to formulate our changes as convergent modifications to the original physics rather than inaccurate discretizations of the correct equations. For example, the inversion fixes in Chapter 2 first modify the underlying PDE to be meaningful in

all configurations, and apply a standard first order accurate discretization to the result.

In many cases, critical problems with these methods went unnoticed in simple tests, and appeared only after moving to large scale examples. Other problems arose only in combination with other techniques, such as the competition between incompressibility constraints and collisions in Chapter 4. Running these examples efficiently often requires compromises in the solution of numerical subproblems, such as large linear system tolerances or the use of floats instead of doubles to reduce memory costs, generating further errors and increasing the chances of finding algorithmic flaws. Note that the advantages of testing a method on complicated high resolution examples persist even if the final applications are much simpler, since even simple tests will discover problems if run sufficiently many times.

I believe the approaches to algorithm design followed in this thesis are a useful complement to rigorous analysis. The freedom and flexibility gained when this rigor is (at least temporarily) relaxed can be invaluable in the understanding and exploration of new methods, and as long as physical accuracy is always kept in mind rigor is often recoverable after the fact. Much of the effort in this work has been focused on the production of the largest, most complicated examples we could manage. While the algorithms discussed in the following chapters were essential to the creation of these examples, the opposite statement is also true: we could not have discovered the algorithms without running the examples.

Chapter 2

Robust Finite Element Simulation

The first step in the simulation of elastic deformable objects is a discretization of the internal constitutive model and resulting elastic forces. Even for graphics, it is important to be able to model the full range of possible constitutive behaviors, including biphasic response, anisotropy, and both active and passive components. At the same time, it is critical that the chosen discretization tolerates all forms of error, especially in situations involving collisions where parts of the mesh can be moved arbitrarily to avoid other objects. Any restriction on the validity of the discretization to nondegenerate or physically reasonable configurations severely constrains the development of other algorithms in the model.

The algorithm presented below, which appeared first in [66, 67], attempts to solve the robustness problem on a per element basis by heuristically choosing a direction along which an element with negative volume is considered to be inverted, and extending the constitutive model smoothly into this inverted regime. This was the first method to generalize the inversion techniques previously used for springs to the case of arbitrary constitutive models. While the local heuristic used does not always succeed in uninverting every element, it is guaranteed to produce valid, finite forces regardless of the configuration of the mesh. This approach has since been applied to muscle and flesh modeling for skeletal motion and face simulation, skin simulation for virtual surgery, and simulations of elastic characters and other deformable objects for films.

2.1 Introduction

Significant effort has been placed into making finite element simulation robust in the regime of large deformations, including the arbitrary Lagrangian-Eulerian (ALE) formulations pioneered by [56], continuous remeshing (see e.g. [19, 36] and the references therein), etc. However, as noted in [8], these approaches are often computationally intensive and difficult to implement, which has primarily limited their use to two spatial dimensions. Moreover, [8] points out that these difficulties often lead

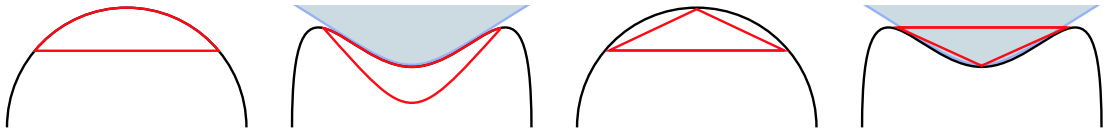

Figure 2.1: A highly curved region of an object is pushed inwards during a collision with a gray object (left), and a triangle used to represent this deformation is forced to invert (right).

authors to less optimal techniques such as element deletion. This not only degrades the accuracy of the simulation, but is unsuitable for graphics applications where disappearing tetrahedra cause visual artifacts.

Since constitutive models for real materials are meaningful only for uninverted material, standard finite element simulation algorithms fail as soon as a single tetrahedron inverts. For this reason, various authors have proposed techniques for untangling inverted meshes. For example, [130] used feasible set methods and optimization to untangle two-dimensional meshes, [35] extended various tetrahedron quality metrics intended for mesh smoothing to the case of inverted tetrahedra allowing untangling to occur simultaneously with optimization, etc. However, none of these techniques are guaranteed to work, and fail quite often in practice especially if the boundary of the mesh is also tangled. And the failure to untangle a single tetrahedron forces the simulation to fail for most real world constitutive models.

As pointed out by [36], element inversion can occur even if the vertex positions of the mesh are identical to their true continuum values. A common case of this is illustrated in Figure 2.1 where a triangle with three nodes on the boundary is forced to invert during a collision with another object. Even if an object as a whole deforms by only a small amount, say 10%, an individual tetrahedron may undergo severe deformation due to errors in the discrete representation of the continuous material. In fact, large deformation and inversion can arise even when simulating incompressible material, since one typically cannot conserve volume for each individual tetrahedron. Given that it is difficult or impossible to prevent inversion in all cases, we propose a simpler approach that allows elements to invert gracefully and recover.

If the particular material behavior and underlying physics is unimportant, there are several techniques for treating inversion. For mass-spring systems, altitude springs work surprisingly well [85] (see also the work on "Van Helsing" [74]). In fact, one can even add altitude springs to finite element models, but this changes the underlying partial differential equations and thus the behavior of the material (losing one of the key benefits of the finite element method as compared to mass-spring systems). Moreover, one cannot use constitutive models that lose meaning for inverting elements (which is most of them). Of course, one could switch from finite element methods to mass-spring methods for flat, degenerate, and inverted elements but this leads to force discontinuities (detrimental to implicit time integration or quasistatic simulation), visual artifacts such as popping,

etc. Various authors have proposed methods similar in spirit to altitude springs. [123] changes the underlying partial differential equation by adding a volume preservation term to penalize inversion. [87] first computes and removes the rotation from material to spatial coordinates, and then applies a linear model in the unrotated space. Although the linear model works well for inverted elements, it is severely limited in the types of materials it can approximate. In fact, we stress that a spring is the most common linear model that correctly accounts for rotation.

In this paper, we take a different approach motivated by the desire to continuously (or even smoothly) extend the finite element method so that it behaves gracefully for both degenerate and inverted elements, even for arbitrary constitutive models. This is especially important as a number of recent graphics publications have advocated the use of more advanced constitutive models to capture more realistic physical behavior. Moreover, many materials found in nature exhibit complex nonlinearities under large deformation, such as the biphasic nature of biological tissue and the anisotropic behavior of muscle. Our approach begins by computing a diagonalization of the deformation mapping in order to determine the "direction" along which a given tetrahedron is inverted. We then extend the constitutive model to the inverted regime using C^0, C^1 or higher continuity around the flat state, resulting in smooth behavior even in extreme situations. The resulting forces *always* act to restore the tetrahedron to its original shape, allowing objects to recover cleanly from flat or inverted configurations, as shown in Figure 2.2. We illustrate the generality and robustness of our approach with a number of simulations of objects undergoing large deformations, including nonlinear and anisotropic constitutive models, plasticity with and without control, both volumetric objects and thin shells, and fracture.

2.2 Previous Work

[122, 121, 120] pioneered deformable models in computer graphics including early work on plasticity and fracture. Finite element simulations have been used to model a hand grasping a ball [49], to simulate muscles [23], for virtual surgery [100], and to simulate data from the NIH visible human data set [138, 55]. [95] and [94] simulated brittle and ductile fracture respectively, while [137] coupled this work to explosions. Finite elements were also used for fracture in [88]. Other work includes the adaptive framework of [31], the rotation based approach in [86] and [87], the hybrid finite element free form deformation approaches in [20, 21], and the finite volume muscle models of [116]. Other interesting approaches to the simulation of deformable objects include [70, 69].

Many authors have worked to improve the robustness of mass-spring systems. [97] used a pseudopressure term in addition to edge springs, [14] used springs emanating from the barycenter of each tetrahedron to preserve volume, and [28] introduced altitude springs to prevent triangles from collapsing. [85] later improved this model and applied it in three spatial dimensions to the case of tetrahedral mesh generation. If altitude springs are used correctly, not only is inversion not a

problem, but the elements will work to uninvert. Unfortunately, spring systems do not allow the modeling of arbitrary constitutive models.

We also show examples of our method at work for the in-plane deformations of lower dimensional manifolds such as cloth and shells. Here, since triangles cannot invert in three spatial dimensions, our method is similar to the work of [37], except that they do not consider degenerate elements. For out-of-plane forces, we use the bending model of [18] (see also [50]), and for self-collisions we use the method of [17]. Moreover, for volumetric collisions we also use the method in [17] simply applied to the triangulated boundary surface of the tetrahedral mesh. Other interesting work on cloth and shells includes the implicit time integration of [4], the bending model of [25], the adaptive scheme of [51], and the self-collision untangling method of [5].

2.3 Measuring deformation

A deformable object is characterized by a time dependent map ϕ from material coordinates $\mathbf{X}$ to world coordinates $\mathbf{x}$. The stress at a given point $\mathbf{X}$ in the material depends only on the deformation gradient $\mathbf{F}(\mathbf{X}) = \partial\mathbf{x}/\partial\mathbf{X}$ of this mapping. Since we are using a purely Lagrangian framework, all mappings are based in material space. We restrict ourselves to constant strain tetrahedral elements, and therefore work with a fixed tetrahedral mesh in material space. In this case, ϕ becomes a piecewise linear map completely determined by its values at the vertices of the mesh, and $\mathbf{F}$ is a constant 3×3 matrix in each tetrahedron.

Since [116] proved that their constant strain tetrahedra finite volume method was *exactly* the finite element method for this case (see also [115]), we adopt their geometrically intuitive notation. For simplicity, consider two spatial dimensions where each element is a triangle. We define edge vectors for each triangle as $\mathbf{d}_{m_1} = \mathbf{X}_1 - \mathbf{X}_0$, $\mathbf{d}_{m_2} = \mathbf{X}_2 - \mathbf{X}_0$, $\mathbf{d}_{s_1} = \mathbf{x}_1 - \mathbf{x}_0$, $\mathbf{d}_{s_2} = \mathbf{x}_2 - \mathbf{x}_0$, and construct 2×2 matrices $\mathbf{D}_m$ with columns $\mathbf{d}_{m_1}$ and $\mathbf{d}_{m_2}$, and $\mathbf{D}_s$ with columns $\mathbf{d}_{s_1}$ and $\mathbf{d}_{s_2}$. Then we have $\mathbf{F} = \mathbf{D}_s\mathbf{D}_m^{-1}$ within this triangle. In three dimensions, $\mathbf{D}_m$ and $\mathbf{D}_s$ become 3×3 matrices, and $\mathbf{F} = \mathbf{D}_s\mathbf{D}_m^{-1}$ still holds. Since the tetrahedral mesh is fixed in material coordinates, $\mathbf{D}_m^{-1}$ is constant and can be precomputed for efficiency. More importantly, as long as the initial tetrahedral mesh is reasonable, $\mathbf{D}_m$ is well-conditioned, and therefore $\mathbf{F} = \mathbf{D}_s\mathbf{D}_m^{-1}$ is well-defined and finite regardless of the current state of the object. Furthermore, $\mathbf{F}$ contains all the information about the deformation of each tetrahedron, since we can recover $\mathbf{D}_s$ via $\mathbf{D}_s = \mathbf{F}\mathbf{D}_m$. In particular, we can use $\mathbf{F}$ to detect whether elements are inverted by checking the sign of $\det \mathbf{F}$.

If the material is isotropic, we can save storage space by performing a QR-decomposition of $\mathbf{D}_m$ and storing only the upper triangular part, as noted in [88]. This corresponds to rotating material space, and therefore has no effect on an isotropic material. This optimization can be performed for an anisotropic model by rotating the anisotropic terms via the rotation from the QR-decomposition.

The next step is usually to define the Green strain $\mathbf{G} = 1/2(\mathbf{F}^T\mathbf{F} - \mathbf{I})$, and compute stress

and forces based on $\mathbf{G}$. We do not do this, however, since $\mathbf{G}$ is invariant with respect to all orthogonal transformations, including reflection, and is therefore incapable of detecting inversion. Furthermore, $\mathbf{G}$ is already nonlinear in the deformation, and it is therefore more difficult to interpret the large deformation behavior of a constitutive model based on $\mathbf{G}$ than one based on $\mathbf{F}$, which is linearly related to deformation. Thus, for the remainder of this paper, we make the (nonrestrictive) assumption that the constitutive model is written explicitly in terms of $\mathbf{F}$.

2.4 Force computation

Suppose the constitutive model is given as a first Piola-Kirchhoff stress $\mathbf{P}$, i.e. a mapping from area-weighted normals in material space to traction vectors in world space. The force on a node i due to a single tetrahedron incident to it is $\mathbf{g}_i = -\mathbf{P}\left(A_1\mathbf{N}_1 + A_2\mathbf{N}_2 + A_3\mathbf{N}_3\right)/3$, where $A_j\mathbf{N}_j$ are the area weighted normals (in material coordinates) of the faces of the tetrahedron incident to node i. Since these do not change during the simulation, we can precompute a vector $\mathbf{b}_i$ such that $\mathbf{g}_i = \mathbf{P}\mathbf{b}_i$. We optimize the computation by calculating $\mathbf{g}_0$ as $\mathbf{g}_0 = -(\mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3)$, and compactly express the other $\mathbf{g}_i$ as $\mathbf{G} = \mathbf{P}\mathbf{B}_m$, where $\mathbf{G} = (\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3)$ and $\mathbf{B}_m = (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$. Note that since $\mathbf{B}_m$ is constant, the nodal forces are linearly related to $\mathbf{P}$. Therefore, the key to obtaining robust forces in face of large deformation is an accurate calculation of $\mathbf{P}$.

If a constitutive model is given in terms of a Cauchy stress σ or second Piola-Kirchhoff stress $\mathbf{S}$, we can easily convert to a first Piola-Kirchhoff stress via the formulas $\mathbf{P} = \mathbf{F}\mathbf{S}$ and $\mathbf{P} = J\sigma\mathbf{F}^{-T}$ where $J = \det\mathbf{F}$. Alternatively, one could rewrite the force formula $\mathbf{G} = \mathbf{P}\mathbf{B}_m$ directly in terms of the other stresses as $\mathbf{G} = \sigma\mathbf{B}_s$ or $\mathbf{G} = \mathbf{F}\mathbf{S}\mathbf{B}_m$. Unlike the first Piola-Kirchhoff case where obtaining a valid $\mathbf{P}$ is sufficient to obtain robust forces, computing a valid σ or $\mathbf{S}$ is not enough. For example, if the tetrahedron is a single point, $\mathbf{F}$ and thus $\mathbf{G} = \mathbf{F}\mathbf{S}\mathbf{B}_m$ are identically zero. Moreover, $\mathbf{B}_s$ (the analogue to $\mathbf{B}_m$ in world or spatial coordinates) depends on the area weighted normals of the deformed, possibly degenerate tetrahedra. And thus $\mathbf{B}_s$ and $\mathbf{G} = \sigma\mathbf{B}_s$ are identically zero as well. That is, there are no restorative forces in either instance. Therefore, we write all constitutive models in terms of $\mathbf{P}$ before force computation.

2.5 Diagonalization

Since rigid body rotations do not change the physics of a deformable object, the stress $\mathbf{P}$ satisfies $\mathbf{P}(\mathbf{U}\mathbf{F}) = \mathbf{U}\mathbf{P}(\mathbf{F})$ for any rotation $\mathbf{U}$ (where $\mathbf{P}(\mathbf{F})$ denotes function application). Furthermore, if we temporarily assume an isotropic constitutive model, $\mathbf{P}$ is invariant under rotations of material space, i.e. $\mathbf{P}(\mathbf{F}\mathbf{V}^T) = \mathbf{P}(\mathbf{F})\mathbf{V}^T$. Therefore, if we diagonalize $\mathbf{F}$ via rotations $\mathbf{U}$ and $\mathbf{V}$ to obtain $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$, $\mathbf{P}$ becomes

$$\mathbf{P} = \mathbf{P}(\mathbf{F}) = \mathbf{U}\mathbf{P}(\hat{\mathbf{F}})\mathbf{V}^T = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T \tag{2.1}$$

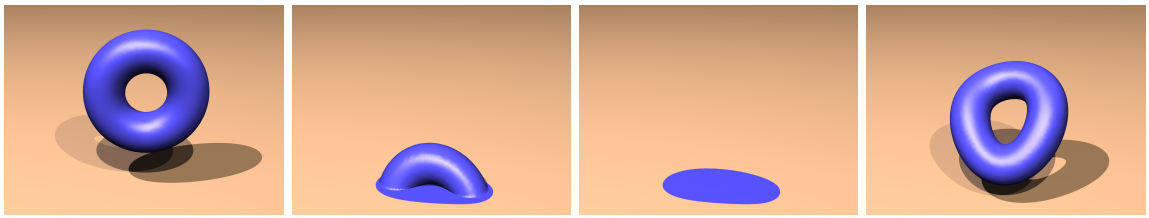

Figure 2.2: A torus with zero strength collapses into a puddle. When the strength is increased, the torus recovers.

where a hat superscript denotes the corresponding rotated quantity. Since the elastic energy of an isotropic material is invariant under world and material rotations, it can depend only on the invariants of $\mathbf{F}$, or equivalently on the entries of the diagonalization $\hat{\mathbf{F}}$ (see e.g. [12]). Therefore, the gradient of the energy, $\hat{\sigma}$, will also be diagonal. Moreover, since the three stresses are related via $\hat{\sigma} = (1/J)\hat{\mathbf{P}}\hat{\mathbf{F}}^T$ and $\hat{\mathbf{P}} = \hat{\mathbf{F}}\hat{\mathbf{S}}$, the diagonalization of $\mathbf{F}$ actually results in the simultaneous diagonalization of all three stresses. In particular, $\hat{\mathbf{P}}$ in (2.1) is diagonal for an isotropic constitutive model. For an anisotropic constitutive model, a diagonal $\hat{\mathbf{F}}$ does not result in a diagonal $\hat{\mathbf{P}}$. However, this is not restrictive, and we show examples of anisotropic constitutive models in Section 2.6.1.

The diagonalization of $\mathbf{F}$ is not unique, however. While the ordering of the entries of the diagonal matrix $\hat{\mathbf{F}}$ is unimportant, the signs of the entries determine whether the tetrahedron is inverted in a particular direction. The standard SVD convention of choosing all nonnegative entries works only when $\det \mathbf{F} \geq 0$. When $\det \mathbf{F} < 0$, the signs of the entries must be chosen carefully in order to guarantee that the forces act to uninvert the tetrahedron. In this case, $\hat{\mathbf{F}}$ has either one or three negative entries. We heuristically assume that each tetrahedron is as uninverted as possible, and thus we assume that only one entry (not three) is negative. Moreover, the entry with the smallest magnitude is chosen to be negative. This is motivated by the geometric fact that an inverted tetrahedron can be uninverted by moving any one node across the plane of the opposite face, and it is most efficient to choose the node that is closest to the opposite face.

We compute the correct diagonalization by finding any diagonalization and correcting the signs. When doing this, we must be careful to ensure that the final $\mathbf{U}$ and $\mathbf{V}$ are pure rotations, i.e., $\det \mathbf{U} = \det \mathbf{V} = 1$. This is because deformable objects are not invariant under reflections of material or world space, and (2.1) does not hold if either $\mathbf{U}$ or $\mathbf{V}$ is a reflection. We compute the SVD of $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$ as follows. First we form the normal equations $\mathbf{F}^T\mathbf{F} = \mathbf{V}\hat{\mathbf{F}}\mathbf{U}^T\mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T = \mathbf{V}\hat{\mathbf{F}}^2\mathbf{V}^T$. Then we rearrange to obtain an eigenproblem, $\mathbf{F}^T\mathbf{F}\mathbf{V} = \mathbf{V}\hat{\mathbf{F}}^2$ for the symmetric positive semidefinite $\mathbf{F}^T\mathbf{F}$. Here, $\mathbf{V}$ is an orthogonal matrix of eigenvectors and $\hat{\mathbf{F}}^2$ is a diagonal matrix with nonnegative entries. Robust computation of eigensystems for 3×3 matrices (even with repeated or zero eigenvalues) is a solved problem. And since it is relevant to rigid body simulations, it has received a lot of attention. Note that if $\mathbf{V}$ is a reflection with $\det \mathbf{V} = -1$ we can simply multiply a column of $\mathbf{V}$ by -1 to

make $\mathbf{V}$ a rotation with $\det \mathbf{V} = 1$. The entries of $\hat{\mathbf{F}}$ are then determined by taking the square root of the diagonal elements of $\hat{\mathbf{F}}^2$, and $\mathbf{U}$ can be found via $\mathbf{U} = \mathbf{F}\mathbf{V}\hat{\mathbf{F}}^{-1}$ for well-shaped elements. However, if a diagonal entry of $\hat{\mathbf{F}}$ is near zero, which is the case for flat tetrahedra, we do not use this formula for the corresponding column of $\mathbf{U}$, but instead take it to be orthogonal to the other columns. For example, in the extreme case where the tetrahedron is a single point and $\mathbf{F} = \mathbf{0}$, we choose $\mathbf{U} = \mathbf{I}$. Finally, for inverted tetrahedra with $\det \mathbf{F} < 0$, we have $\det \mathbf{U} = -1$ implying that $\mathbf{U}$ is a reflection. This is removed by negating the minimal element of $\hat{\mathbf{F}}$ and the corresponding column of $\mathbf{U}$. Figure 2.2 illustrates degeneracy and inversion handling for a torus mesh. Moreover, we have tested our approach for a variety of degenerate configurations, such as when a tetrahedron collapses to a single point or line, and the method always leads to robust recovery from inversion. Figure 2.7 shows similar results for a hexahedral mesh.

2.5.1 Other Rotations

Typically, authors use a polar decomposition to remove the world rotation of a tetrahedron producing a symmetric $\mathbf{F}_s$ with $\mathbf{F} = \mathbf{Q}\mathbf{F}_s$. To recover from inversion, one must be careful to control the signs of the eigenvalues of $\mathbf{F}_s$ as in the diagonalization case. However, we know of no way to do this without first computing the full diagonalization $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$, and forming the polar decomposition via $\mathbf{Q} = \mathbf{U}\mathbf{V}^T$, $\mathbf{F}_s = \mathbf{V}\hat{\mathbf{F}}\mathbf{V}^T$. Polar decomposition was used by [37] for cloth simulation and [87] for volumetric solids, but neither showed how to correctly handle inverting or degenerate elements. However, for the cloth case, we note that triangles do not invert in three spatial dimensions (although they can become degenerate). Both tetrahedra in three spatial dimensions and triangles in two spatial dimensions can invert and become degenerate.

Alternatively, one could attempt to remove the world rotation with a QR-decomposition, i.e. $\mathbf{F} = \mathbf{Q}\mathbf{F}_r$ with $\mathbf{F}_r$ an upper triangular matrix. However, any stress which depends linearly on $\mathbf{F}_r$ will be anisotropic in a mesh dependent way, since it is not invariant under rotations of material space. To see this, it suffices to note that if $\mathbf{V}$ is a rotation of material space, then $\mathbf{F}\mathbf{V}^T = \mathbf{Q}\mathbf{F}_r\mathbf{V}^T$, and $\mathbf{F}_r\mathbf{V}^T$ is not upper triangular. Therefore, QR-decomposition is inadvisable even if physical accuracy is not a requirement. Moreover, this is a problem with any method for removing rotations, such as [86], that does not use $\mathbf{Q} = \mathbf{U}\mathbf{V}^T$ for the world rotation.

Note that our approach departs from the typical goal of determining (or approximating) the rotation from material space to world space, i.e. $\mathbf{Q}$ from the polar decomposition. Instead, we look for *two* rotations $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{U}^T$ and $\mathbf{V}^T$ rotate the world and material spaces, respectively, to a space where the deformation gradient is a diagonal matrix. This is preferable to the space obtained using $\mathbf{Q}$ in which the deformation gradient is a more complex symmetric matrix.

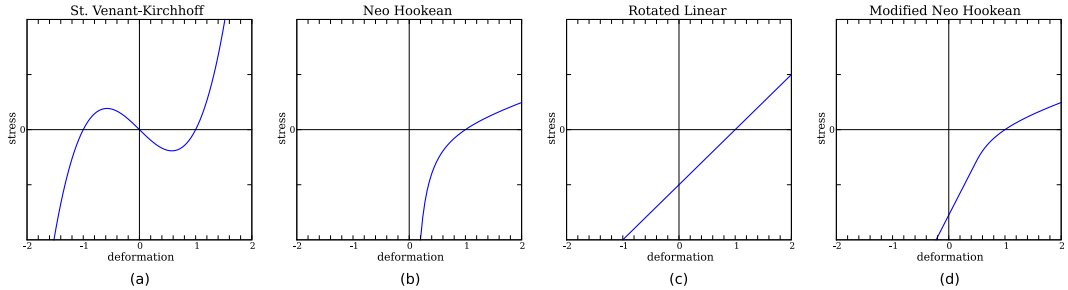

Figure 2.3: The relationship between the first Piola-Kirchhoff stress $\hat{\mathbf{P}}$ and the deformation gradient $\hat{\mathbf{F}}$ for various constitutive models. Standard constitutive models fail for inversion (a, b), but models extended with linear extrapolation are robust in all configurations (c, d).

2.6 Constitutive Models

Once we have carefully diagonalized $\mathbf{F}$, we can extend our constitutive models to behave reasonably under inversion. The diagonal setting makes this quite simple.

If St. Venant-Kirchhoff material is compressed beyond a certain point, it gets weaker and weaker as the compression increases, and the stress drops to zero as the object becomes flat. Moreover, if an element inverts, the forces act to keep the element inverted. See Figure 2.3a. As noted in [12], this makes the St. Venant-Kirchhoff model completely useless for modeling large deformations. [95] noted these difficulties, but dismissed them since they were simulating rigid materials. However, as discussed previously, stiff or incompressible objects may still have inverted tetrahedra due to discretization error, especially on the coarse grids common in the computer graphics community.

In order to alleviate the problems with the St. Venant-Kirchhoff model, various authors (e.g. [100]) have proposed adding a pseudo-pressure term to prevent element inversion. In fact, the classical neo-Hookean constitutive model already does this as shown in Figure 2.3b. The singularity at the origin means that infinite energy is required to completely flatten a tetrahedron, and as long as the equations for this constitutive model are accurately simulated, inversion is prevented. However, preventing inversion also prevents the handling of situations where inversion is the desired, correct response, as in Figure 2.1. Moreover, since the forces become arbitrarily large, the system can become arbitrarily stiff and difficult to integrate, making it difficult to handle situations such as that shown in Figure 2.9 where a volumetric Buddha model is pulled through rigid, interlocking gears.

To avoid the unnecessary stiffness associated with the neo-Hookean constitutive model, we modify the constitutive model near the origin to remove the singularity by linearizing at a given compression limit. Moreover, as shown in Figure 2.3d, we extend the model past the origin into the inverted regime in order to obtain valid forces for inverted elements. These forces act to uninvert the element. Note that since we have removed both spatial and material rotations by diagonalizing, the modified model is automatically rotation invariant and isotropic.

The major strength of the diagonal setting is that these modifications can be applied to arbitrary constitutive models. This is quite natural, since the diagonal setting is also commonly used in the experimental determination of material parameters. The resulting model is identical to the physical model most of the time, and allows the simulation to continue if a few tetrahedra invert. Furthermore, our extensions provide C^0 or C^1 continuity around the flat case, which avoids sudden jumps or oscillations which might effect neighboring elements.

While it may seem nonphysical to modify a constitutive model for inversion handling, most constitutive models lose accuracy long before inversion occurs. It is exceedingly difficult to measure material response in situations of extreme compression, so constitutive models are often measured for moderate deformation and continued heuristically down to the flat cases. Given that some accuracy loss is unavoidable when tetrahedra are nearly flat, it is preferable to provide smooth, consistent handling of inversion in order to avoid unnecessary corruption of the more meaningful parts of the simulation.

If a specific qualitative material behavior is desired but the exact quantitative model is less important, we can use the diagonal setting to construct a suitable constitutive model. For example, most biological material is soft under small deformation, but becomes stiffer as the deformation increases. A simple model capturing this behavior is given by choosing threshold values for compression and elongation, specifying the slope of the stress curve outside these threshold values and at the undeformed state, and using a cubic spline to interpolate between them. This model, equipped with a linear pressure component, was used for the simulations of the volumetric Buddha model shown in Figures 2.10 and 2.9, and the hexahedral simulation in Figure 2.8. Note that any isotropic constitutive model expressed in diagonal form will automatically preserve angular momentum, since if $\hat{\mathbf{P}}$ is diagonal, $\hat{\mathbf{S}} = \hat{\mathbf{F}}^{-1}\hat{\mathbf{P}}$ is symmetric (see e.g. [12]). For the torus puddle, hexahedron puddle, and plastic sphere simulations (see Figures 2.2 and 2.5), where the focus is on degeneracy and plasticity, respectively, we used the simple rotated linear model $\hat{\mathbf{P}} = 2\mu(\hat{\mathbf{F}} - \mathbf{I}) + \lambda \operatorname{tr}(\hat{\mathbf{F}} - \mathbf{I})$ depicted in Figure 2.3c.

Once we have computed the diagonalized stress $\hat{\mathbf{P}}$, the force computation becomes

$$\mathbf{G} = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T\mathbf{B}_m = \mathbf{U}\hat{\mathbf{P}}\hat{\mathbf{B}}_m \tag{2.2}$$

where $\hat{\mathbf{B}}_m = \mathbf{V}^T\mathbf{B}_m$ can be computed and stored if the rotation is fixed for multiple force computations, as in some versions of Newmark time integration (see Section 2.9).

2.6.1 Anisotropy

If the constitutive model includes anisotropic components, it is no longer invariant under rotations of material space. However, we can continue to fully diagonalize $\mathbf{F}$, and rotate the anisotropic terms using $\mathbf{V}$. Since we still work with a diagonal $\hat{\mathbf{F}}$, the large deformation behavior of the constitutive

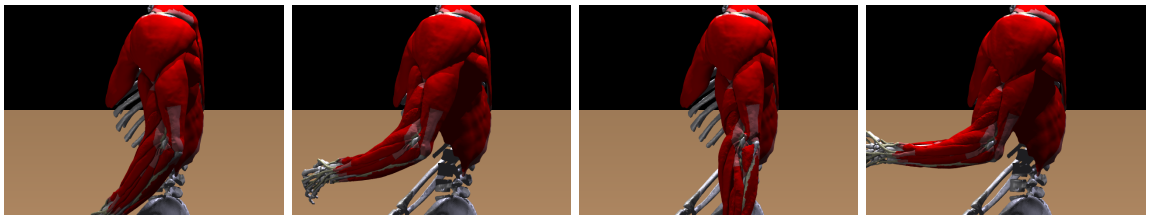

Figure 2.4: A simulation of muscles driven by a key-framed skeleton. The muscle is represented with a transversely isotropic constitutive model, and the strength along the fiber direction in the muscle is based on activation levels.

model is still apparent and easy to modify to handle inversion. For example, if the material is stronger in a certain material direction $\mathbf{a}$, we diagonalize $\mathbf{F}$ and use $\mathbf{V}^T\mathbf{a}$ in the computation of $\hat{\mathbf{P}}$. $\hat{\mathbf{P}}$ is no longer a diagonal matrix, but we can still compute forces using (2.2). When constructing anisotropic constitutive models that allow inversion, we write $\hat{\mathbf{P}}$ as a diagonal matrix plus $\hat{\mathbf{F}}$ times a symmetric matrix for the anisotropic terms. Then $\hat{\mathbf{S}} = \hat{\mathbf{F}}^{-1}\hat{\mathbf{P}}$ is symmetric (preserving angular momentum) as required.

We illustrate the handling of anisotropy with an example simulation of skeletal muscle in the upper limb (see Figure 2.4). We use a nonlinear transversely-isotropic quasi-incompressible constitutive model. See [118] for more details. This is an intricate region of the body articulated with complex joints in the shoulder, elbow and wrist. Inaccuracy in the joint models and motion data leads to skeletal configurations that are incompatible with the musculature creating boundary conditions that degenerately deform muscles and tendons leading to spurious element inversion. However, these configurations often only occur in limited regions of the mesh and only for brief moments during a given motion. Our algorithm allows simulations to progress past these temporary problems by letting elements invert and then later recover.

2.6.2 Damping

Damping forces can be implemented by rotating the velocity gradient $\dot{\mathbf{F}}$ by the same $\mathbf{U}$ and $\mathbf{V}$ used to diagonalize $\mathbf{F}$, computing the damping stress $\hat{\mathbf{P}}$ in the rotated frame, and computing the force exactly as for the elastic case. Note that the rotated velocity gradient will in general not be diagonal.

As in the case of anisotropic elastic forces, a damping model will only preserve angular momentum if $\hat{\mathbf{P}}$ can be expressed as $\hat{\mathbf{F}}\hat{\mathbf{S}}$, with $\hat{\mathbf{S}}$ symmetric. This was not a problem for anisotropy since the anisotropic terms are usually not important for flat or inverted elements. However, in order to prevent visually unpleasant oscillations, we do not want the damping forces to disappear for flat tetrahedra. For example, the analogous damping model to the rotated linear constitutive model, $\hat{\mathbf{P}} = \beta(\hat{\dot{\mathbf{F}}} + \hat{\dot{\mathbf{F}}}^T) + \alpha\,\mathrm{tr}(\hat{\dot{\mathbf{F}}})$, does not preserve angular momentum unless $\hat{\mathbf{F}}$ is a uniform scaling. However, since the angular momentum errors are small around the undeformed state, and highly

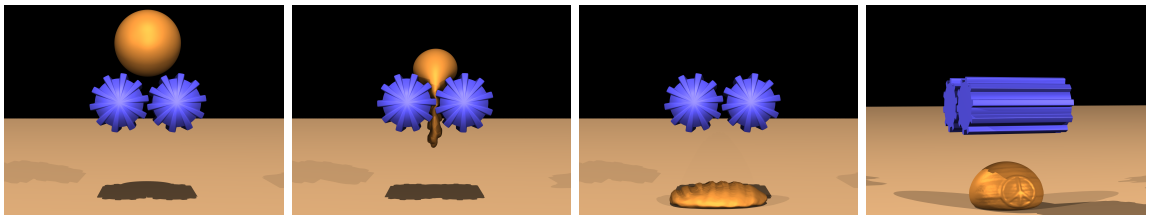

Figure 2.5: A plastic sphere controlled towards a flattened disk shape is pulled through rigid interlocking gears (left). A more obvious example of plasticity control (right).

deformed tetrahedra are usually interacting with other objects, we have not found this lack of conservation to be visually noticeable. In simulations where more physical accuracy is desired, we use a correct damping model for moderate deformations and a more robust but nonphysical model for the few flat or inverted tetrahedra.

2.6.3 Plasticity

We represent plastic deformation with a multiplicative decomposition of the deformation $\mathbf{F} = \mathbf{F}_e\mathbf{F}_p$, where $\mathbf{F}_p$ represents the permanent plastic deformation and $\mathbf{F}_e$ the elastic deformation (see e.g. [12] or [3]). The multiplicative formulation allows a complete separation between plastic flow and elastic forces, and makes constraints such as volume preservation simple to enforce. In contrast, the additive plasticity formulation of [94] does not support true incompressibility, though this might not be a significant problem for graphics applications. Note that if the elastic constitutive model is isotropic, the rotational part of $\mathbf{F}_p$ is arbitrary, e.g. we can choose $\mathbf{F}_p$ to be symmetric.

We restrict ourselves to rate-independent plasticity models, and use the return mapping algorithm to transfer deformation from the elastic part $\mathbf{F}_e$ to the plastic part $\mathbf{F}_p$ whenever a yield criterion on $\mathbf{F}_e$ is exceeded. The details of the computation of plastic flow are as follows. Compute the trial elastic deformation $\mathbf{F}_{e,trial} = \mathbf{F}\mathbf{F}_p^{-1}$, and find the diagonalization $\mathbf{F}_{e,trial} = \mathbf{U}\hat{\mathbf{F}}_{e,trial}\mathbf{V}^T$. If a yield criterion on $\hat{\mathbf{F}}_{e,trial}$ is exceeded, project back onto the yield surface producing a new diagonal matrix $\hat{\mathbf{F}}_{e,proj}$. Compute the trial plastic deformation $\mathbf{F}_{p,trial} = \hat{\mathbf{F}}_{e,proj}^{-1}\mathbf{U}^T\mathbf{F}$ (dropping $\mathbf{V}$ since rotations of $\mathbf{F}_p$ are unimportant). If $\mathbf{F}_{p,trial}$ exceeds a separate limit criterion, project it back onto the limit surface producing the final $\mathbf{F}_p$. Compute and store $\mathbf{F}_p^{-1}$ for future use.

This structure supports an arbitrary plastic yield criterion while still ensuring that the plastic deformation does not become too extreme. This is important, since the time step required for stability depends on the conditioning of $\mathbf{F}_p$. In particular, $\mathbf{F}_p$ should never invert. This is easily done by diagonalizing $\mathbf{F}_{p,trial}$ and clamping its entries to a given range around 1 to ensure a well-conditioned $\mathbf{F}_p$.

2.7 Controlling Plasticity

Various authors, such as [44, 103, 128], have considered controlling physics based simulations. The ability to control a simulation alleviates the need for laborious parameter tuning to achieve a desired effect, and makes possible animations which could not be achieved through physical accuracy alone. In the context of plasticity, we can use the plastic limiting projection step to control the plastic deformation toward any desired state without sacrificing realism. To do this, we compute a goal deformation $\mathbf{F}_{p,goal}$ at the beginning of the simulation. In the plastic projection step, we are given a tentative plastic flow from the old deformation $\mathbf{F}_p$ to the trial deformation $\mathbf{F}_{p,trial}$. In order to always move towards $\mathbf{F}_{p,goal}$, we choose the final plastic deformation to be the point on the segment from $\mathbf{F}_p$ to $\mathbf{F}_{p,trial}$ which is closest to $\mathbf{F}_{p,goal}$. This computation is actually performed on the logarithms of each $\mathbf{F}_p$ after removing the world rotation. Since the mapping from rest to goal state will rarely preserve volume locally, volume preservation should not be used during the elastic projection step.

Allowing some flexibility in the plastic flow allows the deformation to pick up additional fine detail not present in the goal state. For example, Figure 2.5 shows a plastic sphere pulled through interlocking gears. The sphere is controlled towards the flattened disk shape. In particular, the goal state does not include the teeth marks present in the final state of the sphere. A more obvious example of control is shown in Figure 2.5. Both examples used the yield criterion $\|\log \mathbf{F}_e\| \leq \gamma$.

2.8 Thin Shells and Cloth

The diagonalized framework is readily extended to handle the in-plane behavior of triangles for modeling thin shells and cloth (see Figure 2.10). Here, $\mathbf{F}$ is a 3×2 matrix decomposed as $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$ where $\mathbf{U}$ is a 3×2 matrix with orthonormal columns, $\hat{\mathbf{F}}$ is a 2×2 diagonal matrix, and $\mathbf{V}$ is a 2×2 rotation matrix. Everything else follows in a straightforward manner.

Inversion does not occur for freely moving thin shells and cloth, since an "inverted" triangle is indistinguishable from a triangle that has been rotated 180° out of plane. However, when triangles degenerate to lines or points special care *is* needed. Moreover, when a shell approximates a two dimensional surface such as during surface mesh generation (see e.g. [85]), "inversion" can occur. That is, a triangle can be tested for inversion by considering the sign of the dot product between its face normal and a known approximation to the surface normal at the center of the triangle. If this sign is negative, the triangle can be considered inverted, and the signs of the entries of $\hat{\mathbf{F}}$ can be corrected as before. Thus, the triangle acts to uninvert by flipping the direction of its face normal.

For bending forces, we use the formulation of [18], which is similar to that of [50]. The bending model is independent of the in-plane model, and in-plane plasticity is analogous to the three dimensional case. To allow plastic bending, we apply the plastic flow algorithm to the rest angles between each pair of adjacent triangles. An example of a shell simulation showing both in-plane and bending plasticity is shown in Figure 2.6.

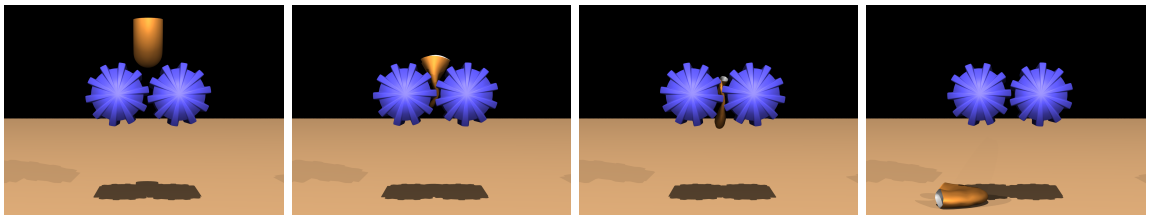

Figure 2.6: Half of a torus shell simulated with in-plane and bending plasticity (3.5k triangles).

2.9 Time Integration and Collision Handling

We use the Newmark time integration scheme of [18] with explicit integration for the elastic forces and implicit integration for the damping forces. Treating only the damping forces implicitly removes the strict quadratic time step restriction required by fully explicit schemes without introducing the extra artificial damping characteristic of fully implicit schemes. As most damping models are linear in the velocities with a positive definite, symmetric Jacobian, the implicit integration can be implemented using a fast conjugate gradient solver.

We modify [18]'s scheme slightly to improve the handling of rigid body collisions. Specifically, we use the velocity from the last implicit update as input to the rigid body collision algorithm, and use constraints in the velocity update to prevent motion in the direction normal to the rigid body for points experiencing a collision. The resulting algorithm to move from step n to $n+1$ is as follows:

1. $\tilde{v}^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, \tilde{v}^{n+1/2})$

2. $\tilde{x}^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$

3. Process rigid body collisions using $\tilde{x}^{n+1}$ and v^n, producing final positions x^{n+1} and modified velocities $\tilde{v}^n$.

4. $v^{n+1} = \tilde{v}^n + \Delta t \left(a(t^n, x^n, \tilde{v}^n) + a(t^{n+1}, x^{n+1}, v^{n+1}) \right) /2$

Note that the last line is exactly the trapezoidal rule applied to the velocities. This algorithm supports a variable time step with second order accuracy and monotone behavior. Since the positions change only in lines 2 and 3 of the algorithm, we can compute $\mathbf{F}$ and its diagonalization only once per time step after step 3. Plastic flow is also computed at this time. With this optimization, the cost of diagonalization becomes *negligible* compared to the cost of the implicit velocity updates.

The rigid body collision processing is based on the algorithm of [18]. We represent rigid bodies as implicit surfaces, which simplifies collision detection. Each such node is projected to the surface of the object, and its normal velocity is set to that of the object if it is not already moving away

from it. We incorporate friction by changing the relative tangential velocity $v_{T,rel}$ to

$$v_{T,rel}^{new} = \max\left(0, 1 - \mu\frac{\Delta v_N + \Delta x_N/\Delta t}{|v_{T,rel}|}\right) v_{T,rel}$$

where Δx_N and Δv_N are the changes in position and normal velocity from the projection step. The $\Delta x_N/\Delta t$ term ensures that the particle will experience the correct friction for the change in position imparted by the object. This term was not considered in [18].

Any node involved in a collision is flagged, and its normal velocity is held fixed during the final trapezoidal rule step. Enforcing normal velocities of colliding particles via constraints during the velocity update further increases the stability of the collision scheme, since it allows a nonlocal response to collision. This strategy is similar to that proposed in [4], where rigid body collisions were implemented within a fully implicit scheme via constraints in the conjugate gradient solver.

Since projecting points to the surface of an object tends to crush tetrahedra, the ability to handle flat or inverted tetrahedra is essential to enable the use of reasonable time steps. Also, since the rigid body collision algorithm is applied to surface vertices only, not surface triangles, it is useful to apply the algorithm to the interior points as well as the surface points, to prevent small rigid bodies from slipping between surface points into the interior of the object. The importance of this increases for very soft objects, as very soft surface triangles can easily expand and pass around even moderately sized obstacles.

For self-collisions we extract the boundary surface and apply the cloth collision algorithm of [17]. This algorithm is applied "outside" of the time integration algorithm outlined above. Although a surface-only collision algorithm does not prevent the interior of the object from extending outside its boundary, our method has no difficulty with this inversion and only the surface is needed for rendering.

Subsequent to the original publication of this work, the time integration and friction handling methods were significantly improved. In particular, the $\Delta x_N/\Delta t$ added above should be considered an obsolete approximation to the more accurate handling of friction due to forces during the implicit solve, and is not present in the more recent algorithm. These improved methods are discussed in Chapter 3.

2.10 Hexahedra and Other Elements

While we have presented our method for constant strain tetrahedral elements, it can be easily generalized to arbitrary element types by applying the modifications to $\mathbf{P}$ separately at each quadrature point. To do this, consider an isoparametric element parameterized by ξ with n_e vertices, and let $N_i(\xi)$ be the basis function associated with node i (see [62]). Given the nodal values of a variable

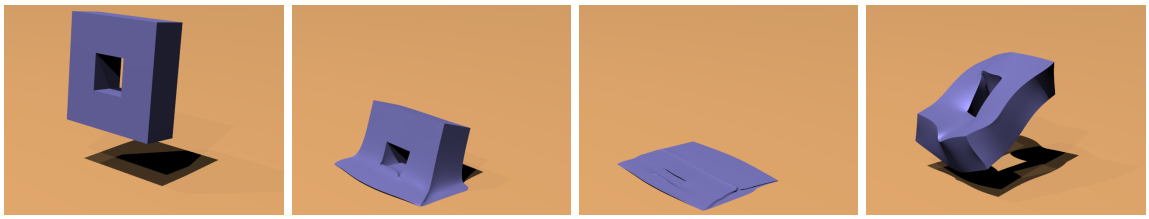

Figure 2.7: A hexahedral mesh collapses into a puddle and recovers.

$\mathbf{x}_i$, the interpolated values of $\mathbf{x}$ are

$$\mathbf{x}(\xi) = \sum_{i=1}^{n_e} \mathbf{x}_i N_i(\xi)$$

Using this and the chain rule allows us to compute the deformation gradient as

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \frac{\partial \mathbf{x}}{\partial \xi}\left(\frac{\partial \mathbf{X}}{\partial \xi}\right)^{-1} = \sum_{i=1}^{n_e} \mathbf{x}_i \frac{\partial N_i(\xi)}{\partial \xi}\left(\sum_{i=1}^{n_e} \mathbf{X}_i \frac{\partial N_i(\xi)}{\partial \xi}\right)^{-1}.$$

For simplicity we assemble the spatial positions of the element vertices in a $3 \times n_e$ matrix $\mathbf{D}_s = [\mathbf{x}_1, ..., \mathbf{x}_{n_e}]$, and similarly for material positions, $\mathbf{D}_m = [\mathbf{X}_1, ..., \mathbf{X}_{n_e}]$. Additionally, we assemble the derivatives $\frac{\partial N_i}{\partial \xi}$ in a $n_e \times 3$ matrix $\mathbf{H} = [\frac{\partial N_1}{\partial \xi}^T, \cdots, \frac{\partial N_{n_e}}{\partial \xi}^T]^T$. With these conventions, $\mathbf{F}$ can be written as $\mathbf{F} = \mathbf{D}_s\mathbf{H}(\xi)\left(\mathbf{D}_m\mathbf{H}(\xi)\right)^{-1}$.

For example, basis functions for hexahedral elements are determined by trilinearly interpolating nodal values throughout the primitive element $[-1, 1]^3$. The associated interpolating functions are

$$N_{4(i-1)+2(j-1)+k}(\xi) = \frac{(1 + (-1)^i\xi_1)(1 + (-1)^j\xi_2)(1 + (-1)^k\xi_3)}{8}$$

where $i, j, k = 1, 2$. Note that these elements do not yield a compact expression for the deformation gradient as in the case of tetrahedral elements. Additionally, the deformation gradient typically needs to be evaluated at 8 different quadrature points within each element making hexahedral elements considerably more expensive than tetrahedral elements.

Given a first Piola-Kirchhoff stress $\mathbf{P}$ at each ξ, an element's contribution to the finite element force on one of its nodes $\mathbf{x}_a$ is given as

$$\mathbf{f}_a^e = \int_{\Omega_m^e} \mathbf{P}\frac{\partial N_a}{\partial \mathbf{X}}^T d\mathbf{X} = \int_{\Omega_i} \mathbf{P}\frac{\partial N_a}{\partial \mathbf{X}}^T \left|\frac{\partial \mathbf{X}}{\partial \xi}\right| d\xi$$

where Ω_m^e represents the element in material space and Ω_i represents the ideal element. This integral

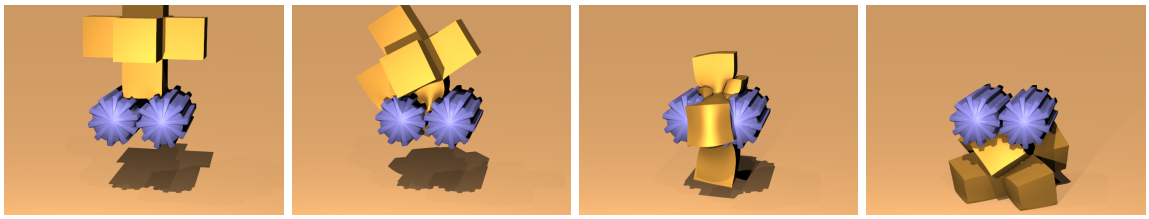

Figure 2.8: A hexahedralized volume pulled between rigid interlocking gears (56k hexahedra).

can be approximated using quadrature as

$$\mathbf{f}_a^e = \sum_{g=1}^{n_g} \mathbf{P}_g \left(\frac{\partial N_a}{\partial \mathbf{X}}\right)_g^T \left|\left(\frac{\partial \mathbf{X}}{\partial \xi}\right)_g\right| W_g = \sum_{g=1}^{n_g} \mathbf{P}_g (\mathbf{D}_m \mathbf{H}_g)^{-T} \left(\frac{\partial N_a}{\partial \xi}\right)_g^T \left|\left(\frac{\partial \mathbf{X}}{\partial \xi}\right)_g\right| W_g$$

where n_g is the number of quadrature points, $\mathbf{P}_g$ is the first Piola-Kirchhoff stress at a quadrature point g, and W_g is the weight associated with quadrature point g. The second equality comes from $\frac{\partial N_a}{\partial \mathbf{X}} = \frac{\partial N_a}{\partial \xi}\left(\frac{\partial \mathbf{X}}{\partial \xi}\right)^{-1} = \frac{\partial N_a}{\partial \xi}(\mathbf{D}_m \mathbf{H})^{-1}$. The element's contribution to *all* its nodal forces can be compactly represented as $\mathbf{G} = [\mathbf{f}_1^e, \cdots, \mathbf{f}_{n_e}^e] = \sum_{g=1}^{n_g} \mathbf{G}_g$ where

$$\mathbf{G}_g = \mathbf{P}_g (\mathbf{D}_m \mathbf{H}_g)^{-T} \mathbf{H}_g^T \left|\left(\frac{\partial \mathbf{X}}{\partial \xi}\right)_g\right| W_g = \mathbf{P}_g \mathbf{B}_{mg}$$

using the definition of $\mathbf{H}$. As before, the $3 \times n_e$ matrix $\mathbf{B}_{mg}$ is constant, and the nodal forces are linearly related to $\mathbf{P}$. Therefore, we can make the element robust by computing $\mathbf{F}$ at each quadrature point ξ_g and evaluating $\mathbf{P}_g = \mathbf{P}(\mathbf{F}_g)$ with the same inversion corrected version of $\mathbf{P}(\mathbf{F}_g)$ used for tetrahedra.

2.11 Examples

We used the algorithm of [85] to generate the tetrahedral meshes used in this paper. Even without preconditioning in the CG solver, computation times were generally under 20 minutes per frame for the largest meshes. Of course, coarser meshes can be simulated in just a few minutes a frame. For example, the torus simulation in Figure 2.2 ran at around 10 to 20 seconds per frame with the 115k element mesh, and .5 to 1 second per frame with an 11k element mesh. All the simulations involved large numbers of inverted elements: a typical frame from the Buddha simulation in Figure 2.9 had about 29k inverted tetrahedra out of a total of 357k tetrahedra, or about 8% of the mesh.

The Buddha with cape example in Figure 2.10 was simulated in two layers, with one-way coupling from the Buddha to the cloth using the collision processing algorithm from Section 2.9. We used the exact triangulated surface geometry of the Buddha in order for the cloth to resolve the many

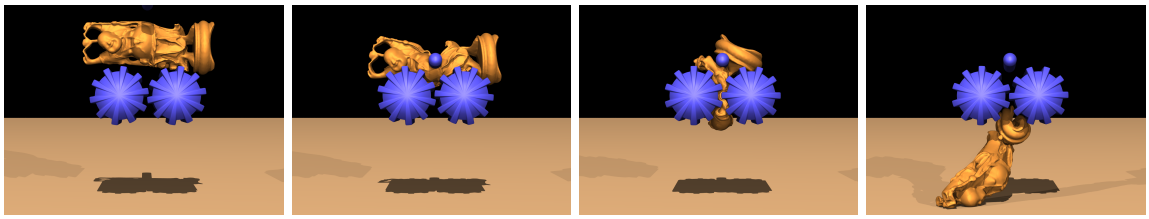

Figure 2.9: A volumetric Buddha model is pushed down with a cylinder and pulled between rigid interlocking gears, then recovers its shape elastically (300k tetrahedra).

features of the Buddha mesh. To evaluate the Buddha as an implicit surface at a cloth vertex $\mathbf{v}$, we find the closest point $\mathbf{p}$ to $\mathbf{v}$ on the Buddha surface (which may lie on a vertex, edge, or face) and define the "normal" at $\mathbf{v}$ to be in the direction from $\mathbf{v}$ to $\mathbf{p}$ or $\mathbf{p}$ to $\mathbf{v}$, whichever points outwards.

The hexahedral meshes shown in Figures 2.7 and 2.8 were cut out of regular cubic grids in the obvious way. For self-collision handling and rendering, we divide each boundary quadrilateral into two triangles. This adds no new degrees of freedom, and does not effect force computation.

2.12 Comparison of Tetrahedral and Hexahedral Elements

Given the ability to robustly handle inverted elements, it becomes possible to use all vertex degrees of freedom in order to represent deformation without danger of breaking the simulation. Therefore, we can compare the efficiency of tetrahedral and hexahedral elements simply by comparing the effort required to evaluate the force on each vertex. This can be further broken down into the number of quadrature points per vertex and the cost of each quadrature point. A standard cubic grid of hexahedra has approximately one element per vertex, and 8 quadrature points per element, or about 8 quadrature points per vertex. Dividing each hexahedron into 5 tetrahedra with a Freudenthal cut produces 5 quadrature points per vertex. The regular BCC tetrahedral lattice used in [85] has 4 tetrahedra for each face in a cubic grid and doubles the number of vertices by adding the center of each cube, resulting in 6 quadrature points per vertex. In both of these examples, the number of quadrature points per vertex is larger for hexahedral meshes than for tetrahedral meshes. Since the work required per quadrature point is also larger for hexahedra due to the use of 8×3 matrices, tetrahedral elements are significantly faster per vertex than hexahedral elements.

2.13 Discussion

We stress here that element inversion does not imply that mass or even volume is necessarily lost. In fact, there are precedents for our approach in the finite element literature. When considering

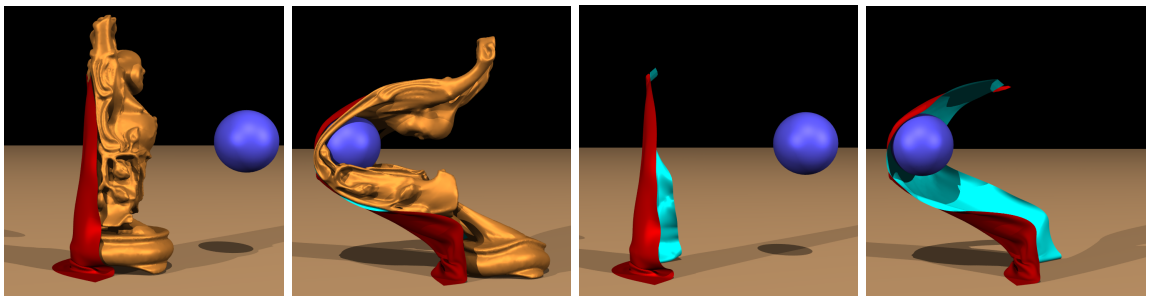

Figure 2.10: A deformable Buddha with a cape undergoing large deformation when hit by a ball (left). The same with the Buddha removed to illustrate the deformation (right). (Cape - 84k triangles, Buddha - 357k tetrahedra)

incompressible or nearly incompressible materials, linear tetrahedral elements suffer from severe volumetric and strain locking precluding their use. The problem occurs because tetrahedra have limited flexibility under the constraint that their volume has to be preserved (or almost preserved), especially when they are connected into a mesh of elements that all have this same volume preservation constraint. [13] partially alleviate this problem by allowing material to flow between elements so that the overall volume can be preserved without preserving the volume of each individual element. Another very interesting approach is the F-bar approach of [30] (see also [101]). The F-bar approach defines a modified deformation gradient $\bar{\mathbf{F}}$, which replaces the volume change in each element with the average volume change over a patch of several elements. This is similar in spirit to composite element approaches, see e.g. [53, 127]. The F-bar approach allows individual elements to change volume as much as they want as long as the volume of the patch is preserved, significantly alleviating difficulties with locking.

Element inversion can be viewed in the context of incompressibility and locking. Restricting individual linear tetrahedra to a non-inverted state is too restrictive for some deformations, and it would be better to consider a patch of elements as in the F-bar technique. That is, one could imagine a method that allows an individual tetrahedron to invert as long as a larger patch of tetrahedra that contain the inverted element does not invert. Then mass (or volume) is not lost, but just transferred to other tetrahedra in the patch.

2.14 Constant vs. Linear Extrapolation and Indefiniteness

A previous version of this chapter suggested modifying a constitutive model for inversion by either linear or constant extrapolation, i.e., clamping the stress at some maximum value. Practical experience with various types of inversion fixing indicates that linear extrapolation into the inverted regime is much more robust than constant extrapolation. In tangled situations where the inversion

heuristic becomes incoherent across adjacent tetrahedra, a force defined by constant extrapolation can occasionally cause the mesh to further invert and expand linearly without bound. This is because a tetrahedron that grows more and more inverted feels no additional force back towards a correct state. Forces defined by linear extrapolation into inversion do not exhibit this phenomenon, since an expanding inverted tetrahedron produces an increasing force which will eventually overwhelm the external forces causing it to invert.

Finally, we note that for standard linear extrapolation to be robust, the diagonal stress derivative $\partial\hat{\mathbf{P}}/\partial\hat{\mathbf{F}}$ must be positive definite. For some models, such as the rotated linear and cubic spline models discussed in Section 2.6, this is true everywhere and the corresponding energy is globally convex (ignoring rotations). Unfortunately, some physically correct models exhibit indefiniteness for certain values of $\hat{\mathbf{F}}$. For example, a neo-Hookean material has a stress of

$$\hat{\mathbf{P}} = \mu(\hat{\mathbf{F}} - \hat{\mathbf{F}}^{-1}) + \lambda \log(\det \hat{\mathbf{F}})\hat{\mathbf{F}}^{-1}$$

with stress differential given by

$$\delta\hat{\mathbf{P}} = \mu(\delta\hat{\mathbf{F}} + \hat{\mathbf{F}}^{-2}\delta\hat{\mathbf{F}}) + \lambda(\hat{\mathbf{F}}^{-1} : \delta\hat{\mathbf{F}})\hat{\mathbf{F}}^{-1} - \lambda \log(\det \hat{\mathbf{F}})\hat{\mathbf{F}}^{-2}\delta\hat{\mathbf{F}}.$$

The rightmost term $-\lambda \log(\det \hat{\mathbf{F}})\hat{\mathbf{F}}^{-2}$ is negative definite if $\det \hat{\mathbf{F}} > 1$, and the entire derivative $\partial\hat{\mathbf{P}}/\partial\hat{\mathbf{F}}$ is indefinite if $\det \hat{\mathbf{F}}$ is sufficiently large. If we linearly extrapolate stress from a point where the derivative is indefinite, the resulting quadratic energy will have no minimum and the material will again expand without bound.

One approach to avoid this problem is to project the stress derivative to be positive definite or semidefinite before extrapolating, which is straightforward since $\partial\hat{\mathbf{P}}/\partial\hat{\mathbf{F}}$ is a 3×3 symmetric matrix. This is similar but simpler than the indefiniteness fixes used in [117], since here we do not need to consider nondiagonal variations of $\hat{\mathbf{F}}$. An alternative approach is to choose a threshold surface where the stress derivative is always positive definite. In the case of neo-Hookean, it suffices to choose a threshold surface given by $\det \hat{\mathbf{F}} = \gamma$ where $\gamma < 1$ is the desired failure threshold, since indefiniteness occurs only if $\det \hat{\mathbf{F}} > 1$. Both of these approaches produce reasonable behavior for fairly extreme deformations, but neither is completely robust in all cases.

Chapter 3

Time Integration and Friction

The fully implicit or semi-implicit time integration schemes often used for solid simulation derive their performance from an efficient implicit handling of stiff forces. The resulting systems must be linearized for fast solution, and typically use only a single Newton iteration per time step. Linearization is sufficient to capture smooth behavior from internal elastic forces, but cannot capture fundamentally discontinuous phenomena such as contact separation and static friction. These phenomena are critical to the behavior of deformable solids, especially for cloth where static friction is responsible for maintaining folds. This has led many authors to resort to ad hoc methods for static friction, such as the zero length springs of [71] or the dynamic sticking constraints of [18].

In this chapter we present an improved friction algorithm which correctly accounts for the friction due to constraint forces during linear system solves. This allows an accurate treatment of both static and dynamic friction without compromising the stability or efficiency of the implicit integration. Comparisons between the new algorithm and the algorithm of [18] are shown for a single particle or tetrahedron sliding down an inclined plane, and for a piece of cloth rolling with static friction. This work appeared originally as part of [109].

3.1 Time Integration

We modify the semi-implicit integration from Section 2.9 slightly by using $x^{n+1/2}$ for the final trapezoidal step, implementing this trapezoidal step as a backward Euler step followed by extrapolation, and adding additional steps for self-repulsions. One step of the modified scheme proceeds as follows:

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. $\tilde{x}^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$

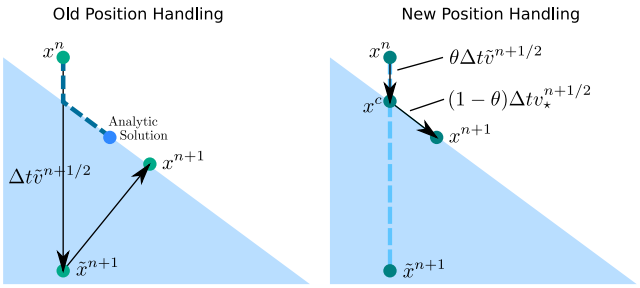

Figure 3.1: Particles are depicted falling towards an inclined plane. (Left) A simple level set adjustment to position incorrectly projects in the normal direction. (Right) Our improved method first finds the collision point and then steps to the final position.

4. Collide with 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

Here Δt is the time step, $a(t, x, v)$ is the acceleration and $x^{n+1/2} = (x^n + x^{n+1})/2$. Step 1 is a backward Euler solve to obtain a temporary velocity, which is subsequently modified with self-repulsions in step 2, before being used to advance the cloth positions forward in time in step 3. Then in step 4 our novel cloth-object collision algorithm is applied, obtaining the final position x^{n+1} and a velocity $v_\star^n$. Next, we apply a backward Euler solve in step 5 followed by an extrapolation in step 6, which are equivalent to applying the trapezoidal rule to velocity but are significantly better conditioned than the standard formulation (see [111]). Finally, in step 7 we modify this final velocity to obtain the appropriate cloth-object friction as dictated by our new cloth-object collision algorithm and subsequently apply self-repulsions in step 8.

3.2 Object Collisions and Friction

We use a level set based collision algorithm similar to that presented in [18] to process each point's position $\tilde{x}^{n+1}$ and velocity v^n as follows. If the signed distance $\phi(\tilde{x}^{n+1}) < 0$, then 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 3.1 (left). For the sake of exposition, we 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

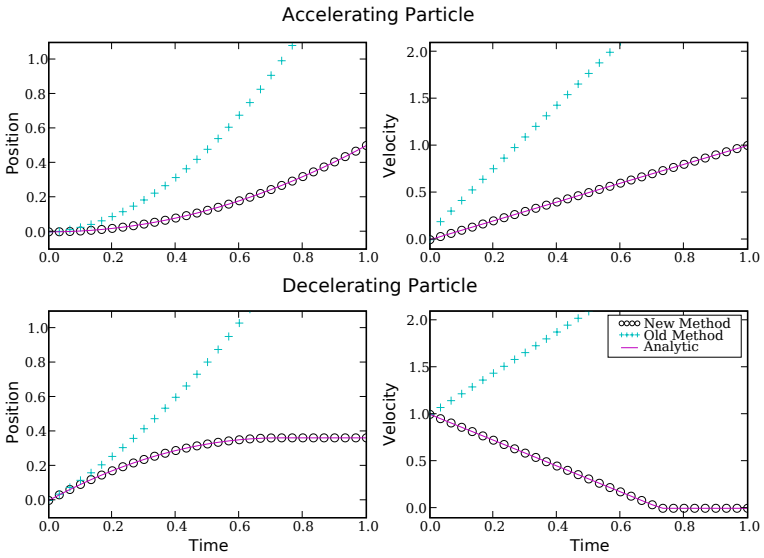

Figure 3.2: Plot of displacement and tangential velocity over time for a single particle moving down an inclined plane. Our result is coincident with the analytic result for both accelerating and decelerating particles, while the previous method accelerates too fast in both cases.

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. 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 $v_\star = v_{\star N} N + v_{\star T}$. In terms of step 4 of our time integration method $v_\star^n = \Gamma(v^n)$.

The two sources of inaccuracy in this old 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 [18] 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 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 $\theta = \phi(x^n)/(\phi(x^n) - \phi(\tilde{x}^{n+1}))$. 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 time step after

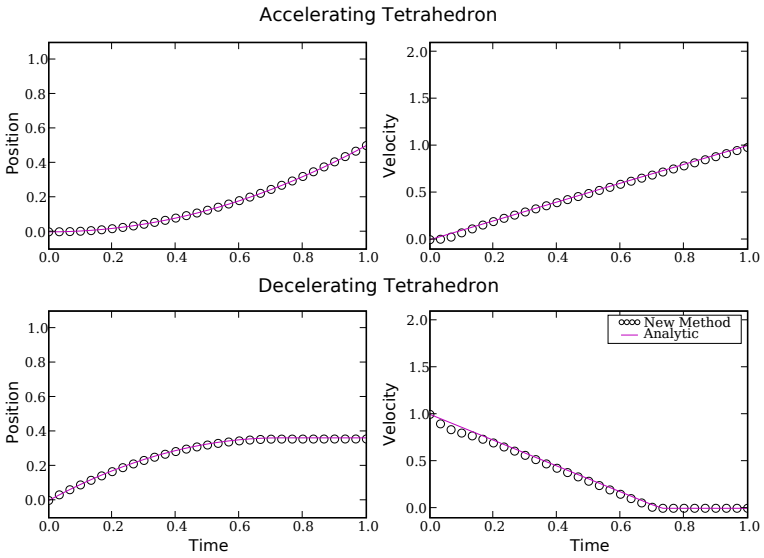

Figure 3.3: A single tetrahedron slides down an inclined plane, matching the analytic solution for both accelerating and decelerating test cases.

the collision as shown in Figure 3.1 (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 5 and 6).

While steps 1 to 3 of the time integration scheme are used to obtain the correct position, steps 5 and 6 are used to update the velocity which also must be corrected for collisions. Specifically, 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 colliding points constrained to $v_{\star N}^n$. Step 7 determines the net normal force applied by the projections in the trapezoidal rule, and subsequently applies the friction due to this normal force. 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 linear system for the velocity update would be

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

where F_i is the velocity independent force, $F_d \tilde{v}^{n+1}$ is the linear velocity dependent damping force, and P projects away the normal component of each colliding point with $I - NN^T$. We rewrite this equation as

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

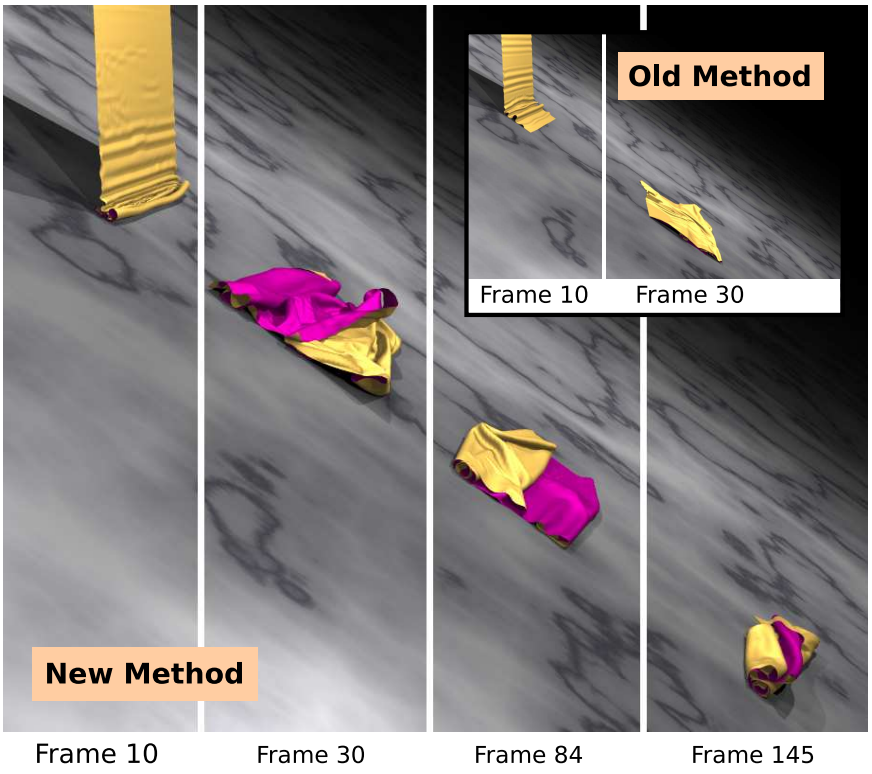

Figure 3.4: A piece of cloth with initial tangential velocity falls on an inclined plane with high coefficient of friction, and undergoes static friction causing it to roll. Cloth simulated by the old method slips immediately on contact with the ground and moves faster (incorrectly) down the inclined plane.

where

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

contains the normal velocity that was removed by projection. As stated above, $\Gamma(v)$ removes the inward normal component of the velocity and applies the resultant friction. Thus, we can mimic conjugate gradient's projection of Δv while adding the appropriate friction by evaluating $v^{n+1} = \Gamma(\tilde{v}^{n+1} + \max(0, \Delta v^T N)N)$. Although we considered backward Euler for the sake of exposition, for the trapezoidal rule that we use the final velocity is $v^{n+1} = \Gamma(\tilde{v}^{n+1} + \max(0, (2\Delta v)^T N)N)$. Here Δv is computed with a half time step of backward Euler, and the subsequent extrapolation doubles this effect resulting in an extra factor of 2 in the formula.

To test our new algorithm we consider a single particle sliding down an inclined 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. Figure 3.2 compares the old 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 3.3 the same test repeated with a tetrahedron that uses non-trivial damping forces. Figure 3.4 shows a more complicated example where cloth rolls with static friction. We used a simple mass-spring constitutive model with edge springs as well as bending springs that connect unshared vertices of adjacent triangles for internal forces in the cloth, and the self-collision algorithm of [109].

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.

Chapter 4

Volume Conserving Deformable Objects

The primary application of deformable solid simulation in computer graphics is to the modeling of biological tissue for elastic characters. Since biological tissue is composed primarily of water, and this water is mostly confined within cells and cannot move freely through the material (at least on short time scales), these tissues are highly locally incompressible. At the same time, tissues such as muscles, skin, and fat resist deviatoric stress relatively weakly (at least for small deformations).

There are two primary difficulties in modeling incompressible materials. First, unless the material is rigid, the equations of motion are infinitely stiff, and the time integration scheme must be carefully chosen in order to enforce the volume constraint without sacrificing efficiency or introducing oscillations into the deviatoric motion. Second, the spatial discretization must avoid volumetric locking, where volume constraints alias with purely deviatoric motion and artificially increase overall stiffness.

This chapter presents an algorithm which avoids these difficulties without the need for multiple quadrature points or stabilization techniques and does not depend on any particular time discretization. This work appeared previously in [65]. In contrast to previous works, which were mostly limited to simple test cases and convergence analyses, we present a technique for stable coupling between collisions and volume preservation and show examples involving complex collision and contact.

4.1 Introduction

Recently virtual humans have received increased attention for modeling stunt doubles, virtual surgery, etc. When modeling virtual humans, one needs to consider shape changes dictated by muscles, skin, fat, and other organs. These soft biological tissues are highly incompressible and

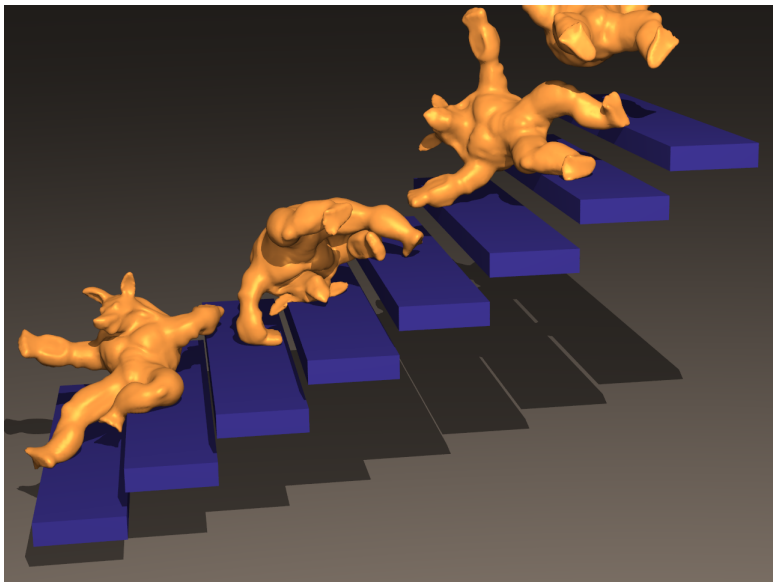

Figure 4.1: An incompressible elastic armadillo falls down a flight of stairs preserving its volume to an accuracy of 0.1%.

involve complicated constitutive models including anisotropy and both active and passive components. Notably, volume in biological tissues is conserved locally, and it is insufficient to only conserve the total volume. Besides realistic modeling of tissues for virtual humans, volume preservation is important in its own right. [77] states, "The most important rule to squash and stretch is that, no matter how squashed or stretched out a particular object gets, its volume remains constant."

Although our interest is in physically based simulation, constant volume deformations are also of interest in shape modeling, e.g. [2, 131]. Several authors have proposed methods that conserve total but not local volume, e.g. [104, 105], and [58] proposed an ad hoc method to address the "undesirable behaviors" caused by conserving only total volume.

A number of authors have considered approximate local volume preservation using simple spring-like forces, e.g. [28, 89, 14, 85, 123]. In the area of finite element simulation, [100] added a volume preserving force to each tetrahedron, a technique similar to the notion of quasi-incompressibility (see [112]) which has been used extensively in finite element simulations of muscle tissue [134, 118]. See also [102, 32]. The problem with mass-spring and quasi-incompressible formulations is that they only provide a force towards volume preservation, and therefore volume is not preserved in the presence of competing forces. This can be alleviated to some extent by increasing the stiffness of the volume-preserving forces, but this competes with and can overwhelm the other forces in the model.

In fluid dynamics, volume preservation is addressed by decomposing a vector field into the gradient of the pressure plus a divergence-free part and subsequently discarding the gradient (see e.g. [40]). By introducing this pressure variable, one discards compressible motions while retaining those

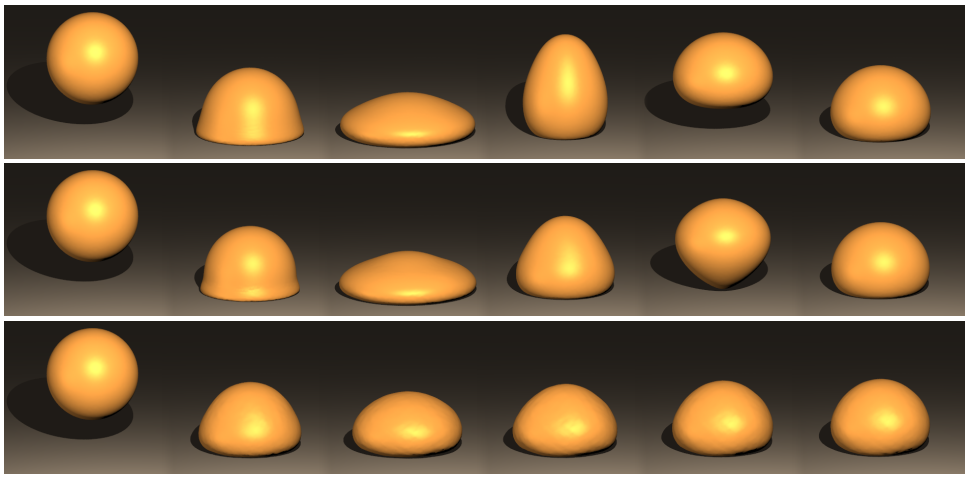

Figure 4.2: An elastic sphere dropped on the ground. (Top) Enforcing the volume of each one-ring using our method maintains correct volume within 1%. (Middle) Using standard finite element forces with a Poisson's ratio of .45 results in a maximum volume loss of over 15%. (Bottom) Increasing the Poisson's ratio to .499 reduces the maximum volume loss to 2% but causes severe locking of the sphere's degrees of freedom hindering deformation.

orthogonal to volume change. [92] proposed a fluid dynamics approach to incompressible deformable solids, but used artificial compressibility (requiring ad hoc volume adjustments) rather than fully divergence-free velocities and did not consider constitutive models or elastic forces in the object's interior. We take a fluid dynamics approach to deformable solids as well, introducing a pressure variable into a standard finite volume approximation. Others, such as [107], have used an independent pressure variable for similar purposes. Besides using the pressure to obtain a divergence-free fluid velocity, we also project the positions to exactly conserve volume avoiding error accumulation (note that Eulerian fluids do not have a position variable). Similar approaches are currently receiving attention in the computational mechanics literature, see e.g. [33, 93, 76, 9, 10, 106, 27]. In contrast to most of these works, we present a simple technique independent of any particular constitutive model or time integration scheme so that it is easily integrated into any finite element solver. Moreover, we show how to integrate this incompressibility constraint with other possibly competing constraints, in particular object contact and self-contact.

4.2 Time Discretization

Regardless of the time integration scheme, our goal is to make the velocity divergence free as well as to update the positions in a manner that moves the nodes to maintain constant volume in one-rings. When processing collisions or treating volume errors, some methods can only modify the position via adjustments to the velocity. Unfortunately, this requires $O(1)$ velocities to make $O(\Delta x)$

changes in position (since $\Delta t \sim \Delta x$). Thus, we propose a method that treats errors in position and velocity separately, ensuring that volume errors due to collisions or other phenomena can be corrected without introducing oscillations. Note that structural integration methods such as [76, 75] do not have this property: in order to quickly correct $O(\Delta x)$ errors in volume they must produce $O(1)$ velocities.

For concreteness, we give the particular time integration scheme used for our examples below, with the two additional steps required for incompressibility highlighted, and then explain these new steps in detail. We used a modified version of the semi-implicit Newmark scheme of [18] (see [109] for details). A step of size Δt from (x^n, v^n) to (x^{n+1}, v^{n+1}) proceeds as follows:

1. $v_\star^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^{n+1/2}, x^n, v_\star^{n+1/2})$
2. $\tilde{v}^{n+1/2} = v_\star^{n+1/2} + \gamma_x$ **(to correct positions)**
3. Modify $\tilde{v}^{n+1/2}$ with elastic and inelastic self-repulsions
4. $\tilde{x}^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$
5. Collide with objects to obtain x^{n+1} and $v_\star^n$
6. $\tilde{v}^n = v_\star^n + \gamma_v$ **(to correct velocities)**
7. $v^{n+1/2} = \tilde{v}^n + \frac{\Delta t}{2} a(t^{n+1/2}, x^{n+1/2}, v^{n+1/2})$
8. $v^{n+1} = 2v^{n+1/2} - \tilde{v}^n$
9. Modify v^{n+1} for inelastic self-repulsions and friction

where $x^{n+1/2} = (x^n + x^{n+1})/2$ in step 7 is the average of the initial and final positions and $a(t, x, v)$ is the acceleration due to all forces except collisions and incompressibility. Step 1 is a backward Euler solve to obtain a velocity for use in the position update, and we adjust this velocity in step 2 so that step 4 corrects the volume in each one-ring. After colliding with kinematic objects in step 5, steps 7 and 8 advance the velocity field forward in time, and we apply a correction in step 6 to make the velocity field divergence free before time evolution.

If we temporarily ignore collisions, steps 2 and 4 combine to form $x^{n+1} = x^n + \Delta t v_\star^{n+1/2} + \Delta t \gamma_x$. This formula is valid for any time integration scheme that computes x^{n+1} from x^n, by defining $v_\star^{n+1/2} = (x^{n+1} - x^n)/\Delta t$. The final volumes should equal the initial volumes, i.e. $V(x^{n+1}) = V(x^0)$. Substituting for x^{n+1} and linearizing gives $V(x^n) + \Delta t \, \mathbf{div}\, v_\star^{n+1/2} + \Delta t \, \mathbf{div}\, \gamma_x = V(x^0)$, where **div** is the volume-weighted divergence (see Section 4.3). Similar to the typical pressure correction in fluids, we use $\tilde{v}^{n+1/2} = v_\star^{n+1/2} - \nabla \hat{p}/\rho$, so $\gamma_x = -\nabla \hat{p}/\rho$ where $\hat{p} = \Delta t p$ is the scaled pressure. Thus, we have $V(x^n) + \Delta t \, \mathbf{div}\, v_\star^{n+1/2} - \Delta t \, \mathbf{div}\, M^{-1} \, \mathbf{grad}\, \hat{p} = V(x^0)$ where **grad** is the volume-weighted

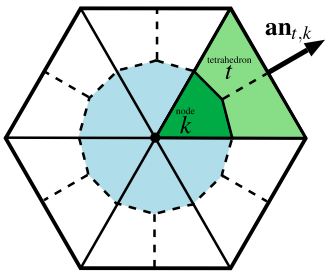

Figure 4.3: A one-ring with highlighted region belonging to the center node.

gradient (see Section 4.3) and M is the diagonal mass matrix. Rearranging into standard Poisson equation form, we have

$$-\,\mathbf{div}\, M^{-1}\,\mathbf{grad}\,\hat{p} = -\,\mathbf{div}\, v_\star^{n+1/2} - (V(x^n) - V(x^0))/\Delta t \tag{4.1}$$

which can be solved for $\hat{p}$, and then $\gamma_x = -M^{-1}\,\mathbf{grad}\,\hat{p}$. Note that **div** is $n \times 3n$, M is $3n \times 3n$, **grad** is $3n \times n$, $v_\star^{n+1/2}$ is a $3n$-vector, and $\hat{p}$, $V(x^n)$, and $V(x^0)$ are n-vectors. Finally, we stress that (4.1) corresponds to a single step of Newton's method applied to the equation $V(x^{n+1}) = V(x^0)$.

We correct the velocity to be divergence free in step 6, although this can be executed at any point in the algorithm since it is a static projection. Taking the divergence of step 6 and setting $\nabla \cdot \tilde{v}^n = 0$ yields $0 = \nabla \cdot v_\star^n + \nabla \cdot \gamma_v$, where γ_v is also defined as $\gamma_v = -\nabla\hat{p}/\rho$. Similar to (4.1) we obtain

$$-\,\mathbf{div}\, M^{-1}\,\mathbf{grad}\,\hat{p} = -\,\mathbf{div}\, v_\star^{n+1/2} \tag{4.2}$$

which we can solve for $\hat{p}$ and subsequently correct the velocity via $\tilde{v}^n = v_\star^n - M^{-1}\,\mathbf{grad}\,\hat{p}$. The difference between (4.1) and (4.2) is that (4.2) computes a divergence-free velocity whereas (4.1) adds an extra term to obtain a non-zero divergence (similar to [41]) in order to correct any drift in volume.

Although the velocity projection is always stable, small time steps and significant volume errors can lead to difficulties as all the missing volume is recovered at once. We alleviate this by introducing a minimum volume recovery time scale $\Delta\tau$ and clamping the last term in (4.1) such that its magnitude is no larger than $V(x^0)/\Delta\tau$ in any given time step.

4.3 Spatial Discretization

A mesh with n nodes has $3n$ degrees of freedom, and enforcing a volume constraint for each tetrahedron typically results in more than $4n$ constraints (the number of tetrahedra) making the system heavily overconstrained resulting in locking as shown in Figure 4.2 (bottom). We avoid locking by

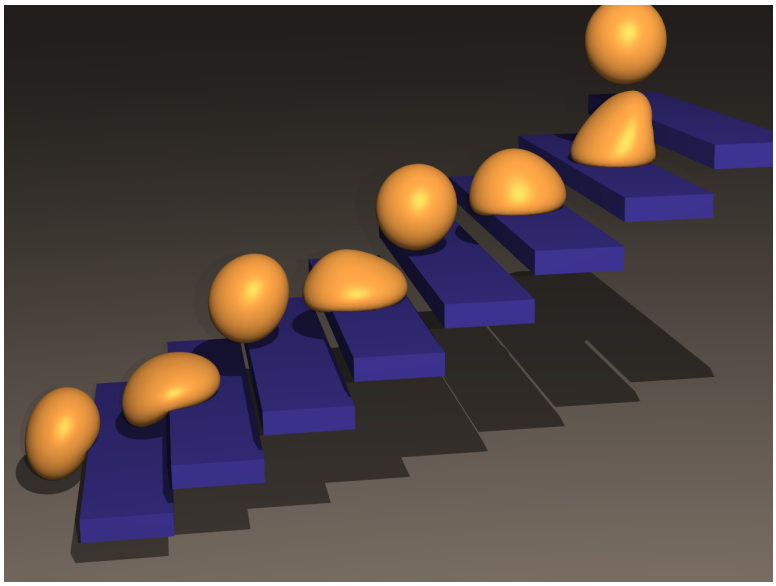

Figure 4.4: An incompressible elastic sphere falls down a flight of stairs illustrating rigid body collisions and contact.

enforcing incompressibility on one-rings, i.e. on composite elements centered at each node, as shown in Figure 4.3 (the blue region). This approach adds only n constraints. Composite elements have proven useful in a number of scenarios, see e.g. [127, 13, 101, 30]. Note that the spatial discretization derived below is identical to the average pressure element in [11].

We use a standard finite volume discretization with all information colocated on the nodes of the mesh as in [116]. Let p_0 to p_3 and $\mathbf{x}_0$ to $\mathbf{x}_3$ be the pressures and positions of the four vertices of a tetrahedron. Define $P = \begin{pmatrix} p_1 - p_0 & p_2 - p_0 & p_3 - p_0 \end{pmatrix}$, $\mathbf{D} = \begin{pmatrix} \mathbf{x}_1 - \mathbf{x}_0 & \mathbf{x}_2 - \mathbf{x}_0 & \mathbf{x}_3 - \mathbf{x}_0 \end{pmatrix}$, and $\dot{\mathbf{D}} = \begin{pmatrix} \mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 & \mathbf{v}_3 - \mathbf{v}_0 \end{pmatrix}$, and let V be the volume of the tetrahedron, $\mathbf{an}_k$ the outward-facing area-weighted normal opposite vertex k, and $\mathbf{B} = V\mathbf{D}^{-T} = -\begin{pmatrix} \mathbf{an}_1 & \mathbf{an}_2 & \mathbf{an}_3 \end{pmatrix}/3$. The linearly interpolated velocity field is $\mathbf{v}(\mathbf{x}) = \dot{\mathbf{D}}\mathbf{D}^{-1}(\mathbf{x} - \mathbf{x}_0) + \mathbf{v}_0$, from which $\nabla \cdot \mathbf{v}(\mathbf{x}) = \mathrm{tr}(\dot{\mathbf{D}}\mathbf{D}^{-1}) = \mathbf{D}^{-T} : \dot{\mathbf{D}}$.

The total volume-weighted divergence over the one-ring centered at node k is

$$(\mathbf{div}\,\mathbf{v})_k = \frac{1}{4}\int_{R(k)} \nabla \cdot \mathbf{v}\, d\mathbf{x} = \frac{1}{4}\sum_{t \in R(k)} V_t \mathbf{D}_t^{-T} : \dot{\mathbf{D}}_t = \frac{1}{4}\sum_{t \in R(k)} \mathbf{B}_t : \dot{\mathbf{D}}_t$$

where $R(k)$ is the set of tetrahedra incident on k, and the fact that the divergence of the velocity field is constant on each tetrahedron allows us to assign $1/4$ of the tetrahedral volume to each of its incident nodes.

We construct our gradient operator to be the negative transpose of the divergence operator so

that (4.1) and (4.2) result in symmetric positive definite systems allowing for fast iterative techniques such as conjugate gradient. Thus, we want $\langle \nabla p, \mathbf{v} \rangle = \langle p, -\nabla \cdot \mathbf{v} \rangle$, that is $\int_\Omega \nabla p \cdot \mathbf{v}\, d\mathbf{x} + \int_\Omega p \nabla \cdot \mathbf{v}\, d\mathbf{x} = \int_\Omega \nabla \cdot (p\mathbf{v})\, d\mathbf{x} = \int_{\partial\Omega} p\mathbf{v} \cdot dS = 0$. This holds, for example, in the case of Dirichlet boundary conditions ($p = 0$ on the boundary). Note that we cannot define gradient using an analogue of the formula used to define divergence, since the resulting forces would not conserve momentum near the boundary.

In order to define **grad**, we assume that the pressure field is zero outside the object (noting that it is straightforward to relax this restriction). We then partition the pressure field into $p = \sum_{t \in R(k)} p_t$ where p_t is a pressure field that agrees with p in t and is identically zero elsewhere. This allows us to restrict attention to a single tetrahedron, since the linearity of the gradient operator gives $(\mathbf{grad}\, p)_k = \sum_{t \in R(k)} (\mathbf{grad}\, p_t)_k$. Each tetrahedron in $R(k)$ makes a contribution to $(\mathbf{div}\, \mathbf{v})_k$ of the form $(\mathbf{div}\, \mathbf{v})_k = \frac{1}{4}\mathbf{B} : \dot{\mathbf{D}} = -\frac{1}{12}\sum_{j=1}^{3} (\mathbf{v}_j - \mathbf{v}_0) \cdot \mathbf{an}_j = -\frac{1}{12}\sum_{j=0}^{3} \mathbf{v}_j \cdot \mathbf{an}_j$. Thus, we can interpret the divergence operator on a single tetrahedron as a 4×12 matrix $-\mathbf{div} = \frac{1}{12}\begin{pmatrix} \mathbf{N} & \mathbf{N} & \mathbf{N} & \mathbf{N} \end{pmatrix}^T$ where $\mathbf{N} = \begin{pmatrix} \mathbf{an}_0^T & \mathbf{an}_1^T & \mathbf{an}_2^T & \mathbf{an}_3^T \end{pmatrix}^T$.

We can now write gradient as a 12×4 matrix $\mathbf{grad} = \frac{1}{12}\begin{pmatrix} \mathbf{N} & \mathbf{N} & \mathbf{N} & \mathbf{N} \end{pmatrix}$ so that the contribution of a single tetrahedron to the gradient at a node k is $(\mathbf{grad}\, p)_k = \frac{1}{12}\mathbf{N}_k \sum_{j=0}^{3} p_j = \frac{1}{3}\mathbf{an}_k \overline{p}$, where $\overline{p}$ is the average pressure of the four vertices of the tetrahedron. Summing over $R(k)$ gives

$$(\mathbf{grad}\, p)_k = \frac{1}{3} \sum_{t \in R(k)} \overline{p}_t \mathbf{an}_{t,k}$$

where $\overline{p}_t$ is the average pressure in t and $\mathbf{an}_{t,k}$ is the area-weighted normal of the face opposite node k. This equation is exactly the standard FVM force for a Cauchy stress of $\overline{p}_t$ (see [116]) and can be computed by forming $\mathbf{G} = -\mathbf{B}_t \overline{p}_t$ and distributing the columns of $\mathbf{G}$ to the nodes (where one node gets the negation of the sum of the columns). In particular our volume preservation forces conserve momentum for each tetrahedron independent of other tetrahedra, since the net force on a tetrahedron is $\sum_k \mathbf{F}_k = -\sum_k (\mathbf{grad}\, p)_k = -\frac{1}{3}\overline{p}\sum_k \mathbf{an}_k = 0$. Angular momentum is also conserved per tetrahedron, since the torque is $\tau = \sum_k (\mathbf{x}_k - \mathbf{c}) \times \mathbf{F}_k = [\sum_k (\mathbf{x}_k - \mathbf{x}_0) \times \mathbf{F}_k] + [(\mathbf{x}_0 - \mathbf{c}) \times \sum_k \mathbf{F}_k]$. The second term is zero since $\sum_k \mathbf{F}_k = 0$, and replacing $\mathbf{F}_k$ with $-(\mathbf{grad}\, p)_k$ makes the first term equal to $-\frac{1}{3}\overline{p}\sum_k (\mathbf{x}_k - \mathbf{x}_0) \times \mathbf{an}_k$, which is zero by Jacobi's identity.

4.4 Collisions and Contact

While steps 2 and 6 of the time integration algorithm work to preserve incompressibility, steps 5, 3, and 9 add additional constraints for object collisions and self-collisions. In practice, we have noticed that the blind application of our algorithm can cause serious artifacts due to these competing constraints, resulting in unusably tangled surfaces. Therefore, we incorporate both object contact and self-contact constraints into our incompressible Poisson equations. Despite being important

Figure 4.5: An incompressible elastic armadillo drops onto the ground illustrating self-collisions and contact.

for robust behavior in the presence of collisions, this coupling is largely undiscussed by previous authors who focused primarily on integrating incompressibility into their particular time or space discretization schemes.

Step 5 sets the position and velocity of particles to respect collisions with the object, and the conjugate gradient solver used in step 7 incorporates constraints in the normal direction to maintain the correct normal velocity for colliding particles, i.e. $\mathbf{n}^T\Delta\mathbf{v} = 0$ where $\mathbf{n}$ is the local unit normal to the collision body and $\Delta\mathbf{v}$ is the change in velocity due to conjugate gradient. We incorporate a similar constraint into the Poisson equations solved in steps 2 and 6 of the algorithm, stressing that this is a linear constraint of the form $\mathbf{c}^T\Delta\mathbf{v} = 0$. Self-contact can similarly be written as linear constraints of the form $\mathbf{c}^T\Delta\mathbf{v} = 0$. Constraining the relative velocity of a point and triangle to not change yields

$$\mathbf{n}^T(\Delta\mathbf{v}_p - w_1\Delta\mathbf{v}_1 - w_2\Delta\mathbf{v}_2 - w_3\Delta\mathbf{v}_3) = 0$$

where w_i are the barycentric weights of the point on the triangle interacting with particle p and $\mathbf{n}$ is the triangle's normal. Constraining the relative velocity of interacting points in an edge-edge pair yields

$$\mathbf{s}^T((1-\alpha_1)\Delta\mathbf{v}_1 + \alpha_1\Delta\mathbf{v}_2 - (1-\alpha_2)\Delta\mathbf{v}_3 - \alpha_2\Delta\mathbf{v}_4) = 0$$

where α_i are positions of the interacting points along the segments and $\mathbf{s}$ is the shortest vector between the interacting segments. Self-contact constraints are generated for each point-triangle and edge-edge pair currently in close proximity (these correspond to the repulsion pairs in [17]). Note that the ability to set the velocity before the Poisson solve and guarantee no changes during the solve is equivalent to using a Neumann boundary condition on the pressure.

First consider a single constraint, i.e. a single point-object interaction, point-triangle interaction,

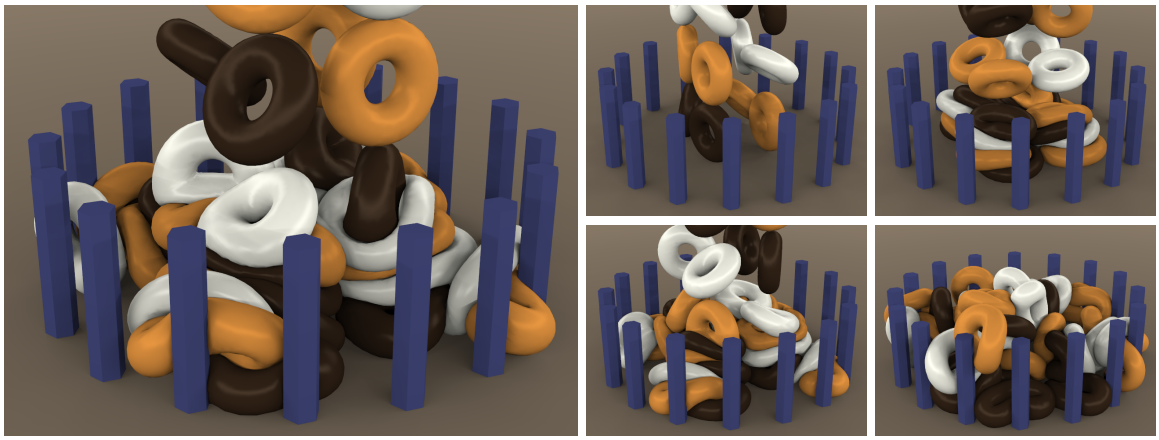

Figure 4.6: 40 incompressible elastic tori fall into a pile illustrating complex collision and contact. Object contact and self-contact are represented as linear constraints during each Poisson solve. Each torus maintains correct volume to within 0.5%, even for those on the bottom of the pile.

or edge-edge interaction. We project out any constraint violating contribution by redefining $\gamma = -\mathbb{P}M^{-1}\,\mathbf{grad}\,\hat{p}$ altering the left-hand sides of (4.1) and (4.2) to $-\,\mathbf{div}\,\mathbb{P}M^{-1}\,\mathbf{grad}\,\hat{p}$, where $\mathbb{P}$ projects a change in velocity using an impulse $\mathbf{j}$ defined by $\mathbb{P}\Delta\mathbf{v} = \Delta\mathbf{v} + M^{-1}\mathbf{j}$. The impulse $\mathbf{j}$ can be found by minimizing the kinetic energy $\mathbf{j}^T M^{-1}\mathbf{j}/2$ subject to $\mathbf{c}^T\mathbb{P}\Delta\mathbf{v} = 0$ using the objective function $\mathbf{j}^T M^{-1}\mathbf{j}/2 + \lambda(\mathbf{c}^T\Delta\mathbf{v} + \mathbf{c}^T M^{-1}\mathbf{j})$. Differentiating with respect to $\mathbf{j}$ and setting to zero gives $\mathbf{j} = -\mathbf{c}\lambda$, and substituting this into the constraint equation yields $\lambda = (\mathbf{c}^T M^{-1}\mathbf{c})^{-1}\mathbf{c}^T\Delta\mathbf{v}$. Thus, $\mathbb{P}\Delta\mathbf{v} = (\mathbf{I} - M^{-1}\mathbf{c}(\mathbf{c}^T M^{-1}\mathbf{c})^{-1}\mathbf{c}^T)\Delta\mathbf{v}$ and

$$\mathbb{P} = \mathbf{I} - M^{-1}\mathbf{c}(\mathbf{c}^T M^{-1}\mathbf{c})^{-1}\mathbf{c}^T.$$

Note that $\mathbb{P}M^{-1}$ is symmetric positive semidefinite with exactly one zero eigenvalue.

In the case of many constraints $\mathbf{C}^T\Delta\mathbf{v} = \begin{pmatrix}\mathbf{c}_1\cdots\mathbf{c}_n\end{pmatrix}^T\Delta\mathbf{v} = 0$ applying the projections in simple Gauss-Seidel order gives $\mathbb{P}_n\cdots\mathbb{P}_1 M^{-1}$ which is only symmetric if none of the constraints overlap. For example, this is violated whenever point-triangle or edge-edge pairs share vertices since the corresponding $\mathbb{P}_i{}'s$ do not commute. One might attempt to alleviate this problem by avoiding sequential application and applying all constraints at once, but this would require inversion of the $n \times n$ matrix $\mathbf{C}^T M^{-1}\mathbf{C}$ appearing in $\mathbb{P} = \mathbf{I} - M^{-1}\mathbf{C}(\mathbf{C}^T M^{-1}\mathbf{C})^{-1}\mathbf{C}^T$ which is prohibitively expensive for complex scenarios with dynamic constraints. Instead, we apply the projections in alternating forward and backward Gauss-Seidel sweeps using the symmetric positive semidefinite matrix

$$\mathbb{P}M^{-1} = (\mathbb{P}_1\cdots\mathbb{P}_n\cdots\mathbb{P}_1)^q M^{-1}$$

where q is a small integer. Applying a single unsatisfied projection $\mathbb{P}_i$ strictly reduces the energy, so

this iteration is stable and always converges to the correct constraint satisfying velocity. In practice, we found that using only $q = 4$ iterations reduced the constraint violating components by 1-2 orders of magnitude on average. Since the pressure system is itself solved to only 1% accuracy, and the alternating sweeps ensure symmetry of the full matrix regardless of convergence, this was sufficient for all our examples. These projections increase the cost of solving for pressure only moderately since there are typically many fewer collisions than vertices and each $\mathbb{P}_i$ can be applied in constant time. Since object contacts and point-triangle contacts have more coherent normals and are typically better behaved than edge-edge contacts, we bias unconverged results towards the former by placing them first (and last) in the ordering.

The final matrix $-\mathbf{div}\,\mathbb{P}M^{-1}\mathbf{grad}\,\hat{p}$ is always symmetric positive semidefinite, but can become singular in cases with large numbers of constraints. Since conjugate gradient breaks down for singular matrices, we instead use MINRES, an alternative Krylov space method which requires only symmetry of the matrix. MINRES required significantly fewer iterations than conjugate gradient even in nonsingular cases (though this could change with preconditioning). For example, Figure 4.2 averaged 72 iterations with conjugate gradient and 34 iterations with MINRES (with small additional cost per iteration). Thorough description and analysis of MINRES, conjugate gradient, and related solvers for singular or nearly singular systems can be found in [26].

4.5 Discussion

In our first attempt to incorporate contact constraints into the incompressible solve, we enforced only point-triangle constraints and ignored edge-edge constraints. This resulted in extremely poor behavior, and provided a useful illustration of the importance of including both types of collisions. In the absence of edge-edge constraints, colliding surfaces would start out in smooth contact but then tangle with a piece of one surface "burrowing" into a hole in the other. This is because point-triangle constraints alone are not sufficient to prevent all possible interpenetrations of two triangulated surfaces. While geometric collisions in the outer time integration loop prevented actual interpenetration, the tangled objects unnaturally stuck together when pulled apart.

Although the volume function V is already smooth and well-behaved through inversion, we also initially experimented with applying additional inversion fixes to V, **div**, and **grad** analogous to those used for deviatoric forces in Chapter 2. To do this, we chose upper and lower thresholds for each entry of the diagonal deformation gradient $\hat{\mathbf{F}}$, and applied linear extrapolation to the volume function V. We then evaluated **div** and **grad** based on this modified volume function by clamping $\mathbf{B}$ within these thresholds. Note that the resulting modified forces still preserve volume since these properties hold for each tetrahedron independently. In practice, however, the intrinsic inversion properties of the true volume were more reliable than our extrapolated version, so this modification generally hindered robustness. In particular, the total volume of an object (or subregion of an

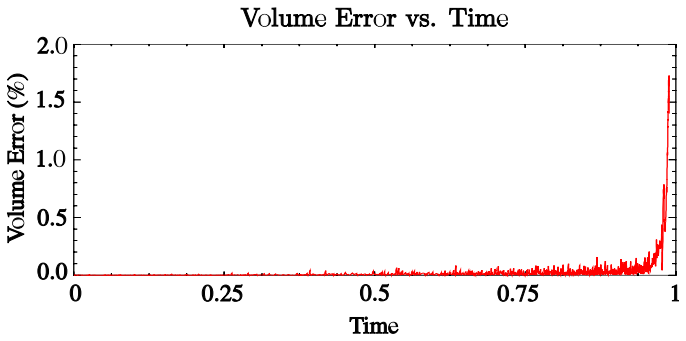

Figure 4.7: Volume error, in percent, as a sphere is pressed between plates. The plates are just touching the sphere at time 0 and move towards each other with constant velocity until they meet at time 1.

object) measured based on each tetrahedron depends only on the volume of the surface independent of inverted tetrahedra in the interior, a property which fails for the modified volume function.

4.6 Examples

We used the method of [66] for internal deviatoric finite element forces in all our examples. When necessary, we used a minimum volume recovery time scale of one-fifth of a frame. Figure 4.2 shows a comparison of our method against a standard finite volume discretization using a 104k element mesh. Using a 3GHz Xeon machine, the computational cost was 18 s/frame for our method, 25 s/frame with Poisson's ratio .45, and 3.4 min/frame with Poisson's ratio .499. Similarly the simulation time for Figure 4.4 was 34 s/frame. The armadillo simulations in Figures 4.1 and 4.5 were both under 4 min/frame with a 112k element mesh. The simulation in Figure 4.6 took an average of 15 min/frame for 40 12k element meshes (500k elements total) with approximately 65% the time spent in the two Poisson solves due to the complexity of the contact constraints.

As a stress test for our method, we pressed an incompressible sphere with 22k elements between two kinematic plates (similar to an example in [58]). The minimum volume recovery time scale was not used (i.e. we set $\tau = 0$). The sphere was successfully compressed to 1.1% of its original thickness before numerical error forced the time step to zero (all computations were performed in single precision). The total volume error remained below 1.7% throughout the simulation, and was lower than 0.1%, 0.5%, and 1% until the sphere reached 13%, 2.3%, and 1.4% of its original thickness, respectively. A plot of volume error vs. time is shown in Figure 4.7. For this simulation we modified the time integration scheme to enforce contact constraints during both backward Euler solves (steps 1 and 7) instead of only in step 7, so that the volume correction in step 2 used the correct collision-aware velocities for particles in contact with the plates. The use of uncorrected velocities as input to volume correction would have caused significant degradation for this simulation due to

the high tension involved. This modification was not necessary for the other examples, which is fortunate since enforcing contact constraints in both solves typically causes sticking artifacts during separation.

4.7 Conclusion

We proposed a novel technique for enforcing local incompressibility in deformable solids drawing ideas from computational fluid dynamics. We benefit from the simplicity and flexibility of tetrahedra while avoiding the pitfalls of locking by enforcing volume preservation over one-rings instead of individual tetrahedra. We augmented our method to incorporate both object contact and self-contact constraints into the incompressible solve to alleviate problems with conflicting constraints. The method is trivially adapted for triangles and thin shells.

Chapter 5

Coupled Two and Three Dimensional Water Simulation

With current hardware, reasonable simulations of elastic solids can typically be performed in a few seconds to a minute per frame, with only moderate memory usage. This is because the bulk of the motion is determined by the geometry of the rest state together with relatively low spatial frequency motion, such as from a skeleton. While more resolution always adds more detail, the spatial frequency of motion required is often not much higher than this base geometry.

In contrast, large scale fluid phenomena exhibit detail at all visible resolutions, and the full resolution of this level of detail is well beyond currently available computational power and memory. Some of this detail can be recovered with various postprocesses such as deep water texturing or one-way coupled spray particles, but increasing the resolution of the base simulation will always produce a significant increase in quality. The following chapter describes an adaptive fluid simulation scheme specifically designed for these large scale problems. This work was first published in [64].

5.1 Introduction

Scenes involving stormy seas, sudden floods or cascading rapids provide some of the most spectacular visual effects shots in feature films (e.g. "The Day After Tomorrow" [68]), and are invariably expensive whether they are simulated via computer or hundreds of thousands of gallons of real water. Since water is opaque at large scales, its appearance is governed by a surface layer while the interior flow is "visible" only through its effect on the surface.

Fully three-dimensional Navier-Stokes solvers produce stunning results, but do not scale well to large bodies of water when the simulation is carried out on a uniform Cartesian grid. Although the situation can be improved significantly by coarsening away from the surface with an octree as in

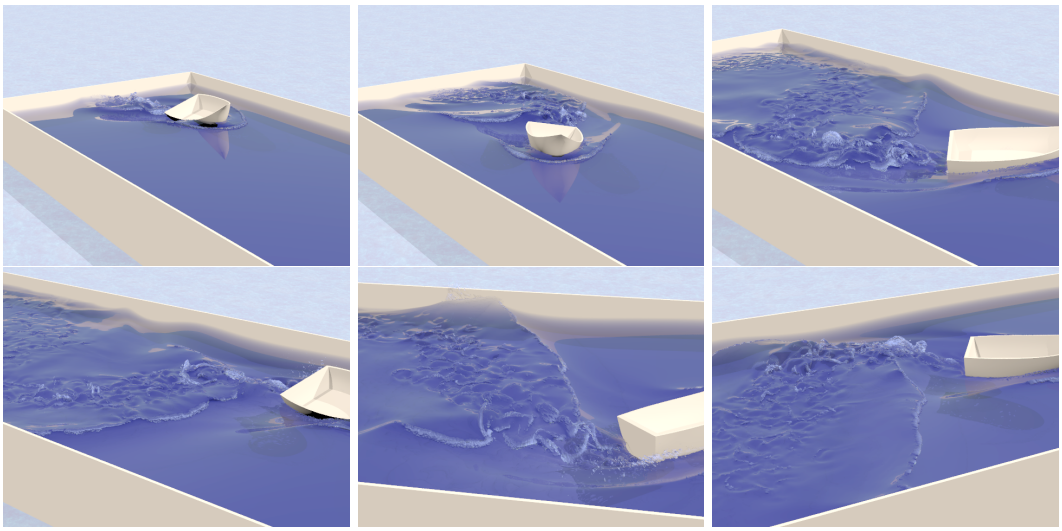

Figure 5.1: Simulation of a wake behind a kinematically scripted boat (1500×300 horizontal resolution).

[79], there are two major drawbacks to this approach. First, the current octree approaches do not make full use of the highly accurate MAC grid method as in [34], instead relying too much on nodal values for interpolation resulting in increased numerical dissipation. Of course, this can be fixed by coarsening the octree away from the interface, and otherwise applying a standard MAC solver in the uniform cells near the water surface. Second, and more importantly (especially since we propose no remedy), large octree cells cannot represent bottom topography. Even if one refined near the bottom of the domain, the large octree cells in the middle of the water filter out horizontal detail making the surface simulation unaware of any rich structure below. In contrast, methods based on height fields (such as the shallow water equations [73, 96]) use tall cells with detailed refinement in the horizontal directions, thus capturing the effects of complex bottom topography rather well. Unfortunately, these methods do not support overturning or other interesting three-dimensional behavior.

Thus, we place the following requirements on our method. First, we need detailed three-dimensional behavior near the interface, down to the depth at which turbulent motion directly affects the look of the surface. This "optical depth" for simulation will usually be greater than the visible optical depth, since interesting turbulence will cause the visible water to continuously mix with the water in a larger layer underneath. Second, the large unseen region of the liquid should be represented as efficiently as possible without losing plausible bulk motion and important details such as bottom topography. The first requirement is satisfied by using a state-of-the-art, uniform MAC grid Navier-Stokes solver near the interface (as in [34]). This also allows for the addition of any other technique that works on a uniform grid, such as vortex particles [108]. Outside of this surface layer, we maintain the same resolution in the horizontal directions but coarsen in the vertical

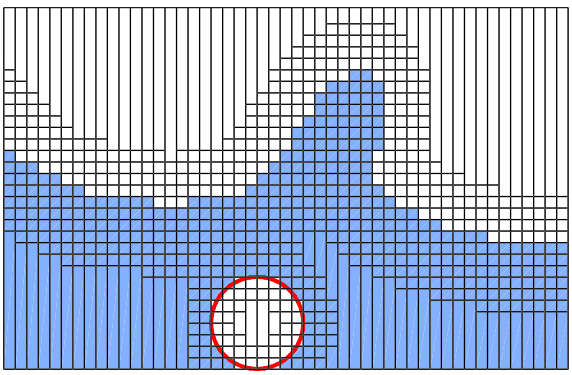

Figure 5.2: We use uniform cells near objects and within a specified optical depth of the surface, and coarsen with tall cells elsewhere.

direction obtaining tall thin cells that reach down to the bottom of the domain (satisfying the second requirement). Since the effects of bottom topography diminish with increasing depth, the method is most useful in the shallow water regime.

One advantage of this hybrid approach is that it reduces to exactly the standard MAC discretization under full refinement. In contrast, the octree method of [79] suffers from increased dissipation due to repeated back and forth averaging even when fully refined. While this dissipation was partially reduced in [52] using FLIP rather than PIC averaging (see also [139]), our experience with uniform grids indicates that the full MAC method still produces higher quality results.

Although a grid of this type could be represented with pointer structures (as in [135]) connecting all the tall and short cells together in the required fashion, this could prove rather inefficient. Luckily, we can leverage recent work on Run-Length-Encoded (RLE) grids by [61, 15, 136, 91, 59, 60] (note that [29] applied RLE to volumes and [16] suggested RLE level sets). Their fluid simulator solves the level set on a narrow band near the interface, but uses a uniform MAC discretization inside the liquid for the fluid solver with long cells allowed only in the air. This is analogous to previous local level set and narrow band methods such as [99] and [1]. Thus, their method improves the cost of the level set algorithm and removes the need to store and check grid cells deep inside the air, but does not reduce the time spent solving for advection and pressure in the fluid itself, which is typically most of the computational cost. In contrast, we use tall cells in the water, which greatly reduces the computational cost outside the band of cells near the interface. This requires novel methods both for discretizing tall cells full of liquid as well as for coupling these cells to the standard uniform cells. Our tall cells only occur far from the interface, and thus do not participate in the particle level set method. However, we do require methods for advecting the velocity field and solving for the pressure in order to make the velocity divergence free. Note that all our tall cells are vertical as is typical for shallow water methods, because it is reasonable to assume linear pressure variation in the vertical direction but not in the horizontal directions.

5.2 Previous Work

The three-dimensional Navier-Stokes equations were popularized with the work of [45, 113, 40]. These equations were combined with methods for simulating a water surface in [43, 44, 42], and we use the particle level set approach of [34] as the method of choice in the three-dimensional surface layer of water. [7] used a conservative height field fluid model in an interactive painting system. Other work on water and liquids includes viscoelastic fluids [48], solid-fluid coupling [22, 52], control [83, 110], contact angles [133], sand [139], and two phase flow [57].

There are a number of two-dimensional techniques for large bodies of water including the popular methods for deep water [46, 98, 82, 129, 125]. We refer the interested reader to the recent work of [54] and course notes of [124] and [6]. Of course, the deep water equations ignore bottom topography and do not resolve fully three-dimensional phenomena at the surface. Another interesting technique consists of solving the two-dimensional Navier-Stokes equations for the horizontal velocities, and then using the pressure to define a height field as in [24]. [126] also solved the two-dimensional Navier-Stokes equations for the horizontal velocity in streams, but used a noise function for the vertical velocity. [90] proposed a simpler stream model using a two-dimensional Laplace equation for the bulk flow. Finally, a few authors have tackled large bodies of water with the three-dimensional Navier-Stokes equations, e.g. focusing on splash and foam in [114] and on breaking waves in [84].

5.3 Grid Structure

We simulate on a uniform two-dimensional horizontal grid of vertical columns that contain both uniform cells and tall cells that replace collections of uniform cells, e.g. see Figure 5.2. The tall and uniform cells can be arbitrary in each column, and in particular we split tall cells at the ground to match bottom topography (Figure 5.4). In a uniform MAC grid, scalars are stored at cell centers while velocity components are stored on their respective faces (Figure 5.3a). Level set values are only required in uniform cells near the interface, whereas pressure and velocity are needed everywhere. Tall cells contain two pressure values corresponding to the cell centers of the uppermost and bottommost uniform cells that the tall cell replaces (Figure 5.3b). Pressure values can be interpolated to the centers of the other uniform cells replaced by a tall cell using vertical linear interpolation.

Later, we will calculate horizontal pressure derivatives for every pressure sample (Figure 5.3c), and thus we colocate horizontal velocities with them (Figure 5.3e). Figure 5.3e outlines the control volume around each *minimal* face noting that minimal faces between tall cells may contain two horizontal velocities (as shown in the figure). Between these two velocities, we interpolate horizontal velocities on the uniform faces replaced by this minimal face using linear interpolation consistent with the pressure and its derivatives. Vertical pressure derivatives and vertical velocities are shown in Figure 5.3d and 5.3f (also colocated) along with outlined control volumes. The linear pressure

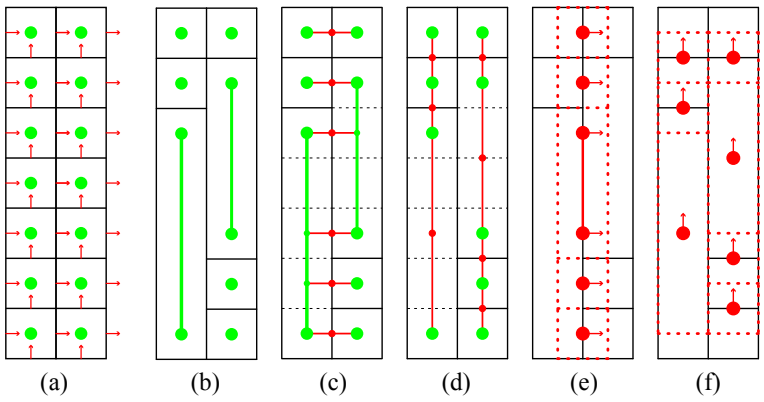

Figure 5.3: (a) pressure and velocity for a uniform MAC grid (b) pressure in uniform and tall cells (c) horizontal pressure derivatives (d) vertical pressure derivatives (e) horizontal velocities colocated with horizontal pressure derivatives (f) vertical velocities colocated with vertical pressure derivatives.

profile in the tall cell dictates that all uniform grid vertical faces replaced by the face at the center of this cell should have the same vertical velocity.

5.3.1 Refinement and Coarsening

When we change the structure of tall and uniform cells, the velocity needs to be interpolated to the new grid. The vertical velocities at the center of new tall cells are set to the average value of all the vertical faces replaced by the tall cells, whereas those for uniform cells are defined as their interpolated values. This conserves vertical momentum. For horizontal velocities, the difficulties occur when a new minimal face replaces more than one uniform face. For example, let $u(j_a), u(j_a + 1), \ldots, u(j_b)$ be a sequence of velocity values on a minimal face. We desire bottom and top values u_a and u_b such that linear interpolation between them best approximates the sequence. The least squares error is

$$E = \frac{1}{2} \sum_{j=j_a}^{j_b} \left(\frac{j_b - j}{j_b - j_a} u_a + \frac{j - j_a}{j_b - j_a} u_b - u(j) \right)^2$$

Differentiating with respect to u_a and u_b and setting the derivatives to zero gives

$$\sum_{j=j_a}^{j_b} \frac{(j_b - j)^2}{j_b - j_a} u_a + \frac{(j_b - j)(j - j_a)}{j_b - j_a} u_b = \sum_{j=j_a}^{j_b} (j_b - j) u(j)$$

$$\sum_{j=j_a}^{j_b} \frac{(j_b - j)(j - j_a)}{j_b - j_a} u_a + \frac{(j - j_a)^2}{j_b - j_a} u_b = \sum_{j=j_a}^{j_b} (j - j_a) u(j)$$

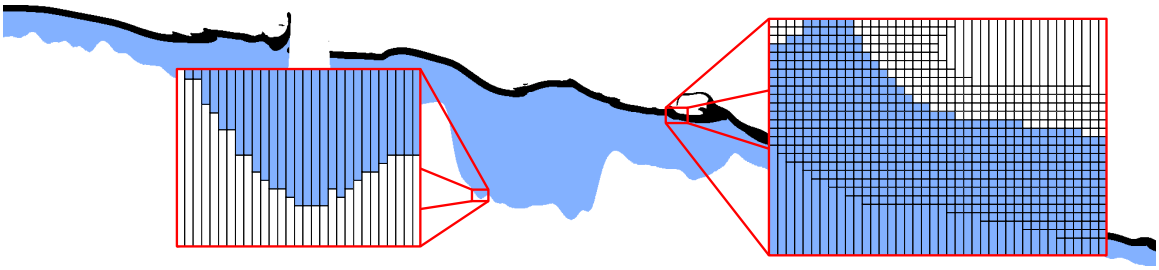

Figure 5.4: A cross-section of the grid from a river simulation showing tall cells used to represent bottom topography.

Using the formulas for sums of linear and quadratic sequences, these equations simplify into

$$\frac{n+1}{6}\begin{pmatrix} 2n+1 & n-1 \\ n-1 & 2n+1 \end{pmatrix}\begin{pmatrix} u_a \\ u_b \end{pmatrix} = \sum_{j=j_a}^{j_b}\begin{pmatrix} j_b - j \\ j - j_a \end{pmatrix} u(j) \tag{5.1}$$

where $n = j_b - j_a$. If some subsequence of the $u(j)$'s come from a minimal face's linear profile, the same formulas can compute that portion of the right-hand side sum in constant time. This 2×2 system is trivially inverted to find u_a and u_b. As shown in [60], the entire process of building a new grid and transferring data between grids takes linear time.

We use a one grid cell band of uniform cells around moving objects, so that no special tall cell treatment is required for one-way or two-way coupling with solids (though we implemented only one-way coupling so far). In addition, we define an optical depth down to which we want to preserve as much detail as possible and use uniform cells within that distance from the interface. We emphasize that the optical depth required for a given problem does not shrink under refinement, rather it is a true length scale related to the three-dimensional physics we wish to capture.

5.3.2 Refinement Analysis and Comparison

The effects of varying the optical depth are illustrated in Figure 5.5. With a sufficient optical depth, satisfactory results are obtained (middle). Otherwise, we fail to resolve enough of the three-dimensional flow structure and obtain nonphysical results (right). These simulations used 4 processors in a 2×2 grid (see Section 5.6), and took 250, 116, and 59 seconds per frame, respectively. Note that the fully refined splash (left) uses essentially the same method as [60].

Since the computational cost of a tall cell is independent of its height, all the tall cells together are equivalent in cost to a few extra layers of uniform cells regardless of the depth of the water. That is, if a certain optical depth of uniform cells is needed, then another layer of cells provides the extra computation needed for our approach. It is unlikely that another adaptive method such as an octree would make this layer more efficient. In particular, we can double the depth of the

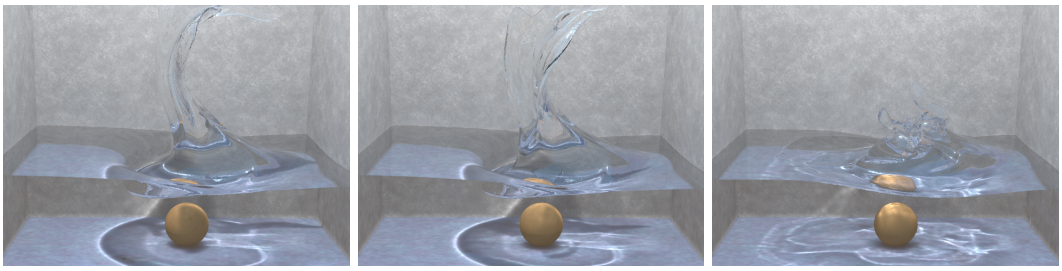

Figure 5.5: Three balls splashing into water (300×200 horizontal resolution). The fully refined case (left) and the 1/4 refined case (middle) are quite similar. However, quite different results are obtained if one doesn't refine enough (right, 1/16 refined).

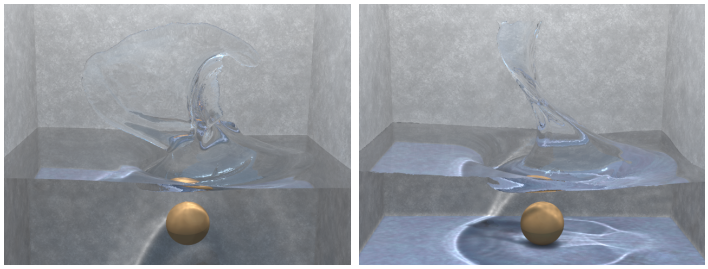

Figure 5.6: (Left) The same optical depth as Figure 5.5 (middle), but with twice the water depth. While this would be twice as expensive with a uniform grid, there is almost no cost increase using our approach. (Right) An octree simulation with two levels of coarsening away from the interface ($312 \times 208 \times 208$ effective resolution).

water without incurring the factor of two slowdown characteristic of uniform grids or previous RLE techniques (Figure 5.6 (left)). Moreover, the resulting splash is qualitatively different from the shallow simulation, which shows the importance of water depth on the bulk flow. The deeper simulation took 140 seconds a frame due to the increased surface area compared to Figure 5.5 (middle).

Figure 5.6 (right) shows a serial octree simulation of the same splash. Although Figure 5.5 (middle) takes the same amount of total computation (460 seconds per frame), our method is readily parallelized leading to faster turnaround and larger simulations. Excessive computational costs kept us from refining the octree to the same resolution *throughout* the entire optical depth region, and we instead were forced to coarsen away from the typical three grid cell band, which made the simulation significantly more viscous. This highlights an interesting aspect of our approach, in that we combine our tall cells with a *uniform* grid whereas it is probably better to combine them with an *adaptive* octree grid. In fact, this is straightforward to implement by placing individual octrees in each of our uniform cells just as was done in [78].

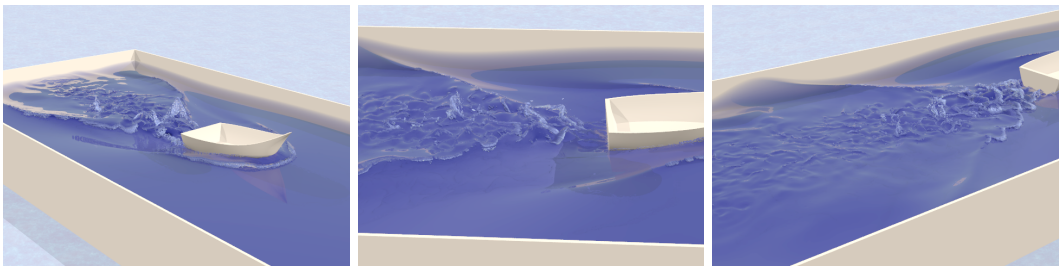

Figure 5.7: The boat moving along a straight path (1500×300 horizontal resolution).

5.4 Uniform Three-Dimensional Method

The inviscid, incompressible Navier-Stokes equations for mass and momentum conservation are

$$\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p/\rho = \mathbf{g} \tag{5.2}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{5.3}$$

where $\mathbf{u} = (u, v, w)$ is the velocity, p is the pressure, ρ is the density, and $\mathbf{g}$ is the acceleration due to gravity. We use a standard, uniform MAC grid fluid solver in the band of uniform cells, together with the particle level set method of [34]. Both the level set and velocity field are advected with semi-Lagrangian advection [113] in this band. Since the level set is only defined near the interface, its semi-Lagrangian rays will interpolate from other uniform cells. In contrast, velocity is defined everywhere, and some semi-Lagrangian rays will ask for information from tall cells. These velocities will instead be updated with the same algorithm used to update velocities in tall cells (see Section 5.5.1). Note that the pressure is solved for globally, coupling the uniform and tall cells together (see Section 5.5.2). In situations where additional turbulence is desired, we use the vortex particle approach of [108]. Figures 5.1 and 5.7 show simulations with vortex particles seeded behind a boat to represent the turbulence generated by a propeller.

5.5 Adaptive Two-Dimensional Method

In this section, we address the advection of velocity on all the tall cells and on the uniform cells that were not updated with semi-Lagrangian advection because they were too close to the tall cells. We also address solving for the pressure on an arbitrary collection of tall and uniform cells. These discretizations will be constructed by summing uniform finite volume discretizations over tall cells using linear or constant basis functions, and can be considered discrete finite volume/finite element methods.

5.5.1 Advection

We use first order accurate conservative upwinding for all tall cells and any uniform cells that are not updated with the semi-Lagrangian method. Since the *number* of cells that need to be updated with this method is a small subset of the total cells (even though it can be most of the volume), the efficiency of this approach is unimportant. Thus, we chose the conservative method to preserve momentum in these *very* tall cells that are likely to have large truncation errors. Moreover, conservative schemes are quite popular for the two-dimensional nonlinear shallow water equations, which have many similarities with our tall cells. We emphasize that conservation does not cure large truncation errors: it only makes the results more physically plausible.

We first put the velocity terms into conservation form using $\nabla \cdot \mathbf{u} = 0$ to transform (5.2) into

$$\mathbf{u}_t + \nabla \cdot (\mathbf{u}\mathbf{u}^T) + \nabla p/\rho = \mathbf{g}.$$

Ignoring the pressure and force terms, we focus on the u component of the velocity

$$u_t + (u^2)_x + (vu)_y + (wu)_z = 0.$$

If we place a control volume around each face as shown by the dotted lines in Figure 5.3e, then this equation can be rewritten to indicate that the velocity is updated based on the fluxes across the control volume faces, i.e.

$$u^* = u^n + \frac{\Delta t}{\Delta x \Delta y \Delta z} \sum_{f=1}^{6} \pm F_f \tag{5.4}$$

where the sign depends on which side of the control volume a flux is on. The fluxes are computed by averaging the velocities to the control volume faces, and then using these average velocities to decide on the upwind direction. The flux itself is constructed by multiplying the upwind component of the velocity we are advecting, u_{up}, with the average values for the other components v_{av} and w_{av}:

$$F_u = \Delta y \Delta z u_{up}^2 \qquad F_v = \Delta x \Delta z v_{av} u_{up} \qquad F_w = \Delta x \Delta y w_{av} u_{up} \tag{5.5}$$

The v and w velocity components are treated similarly.

While this algorithm is straightforward on a uniform grid, a few modifications are required for tall cells. For motivation, we point out that we could refine the tall cells into a uniform grid, apply the uniform grid method just discussed, and then re-coarsen. Although this would be inefficient to implement directly, our approach achieves exactly this result in an efficient manner. To simulate this process of refining, advecting, and coarsening, we consider each pair of adjacent minimal faces one at a time, apply (5.5) and (5.4) to the entire velocity profile to produce a piecewise linear or quadratic u^* on each side, and use (5.1) to reduce u^* back to a constant or linear profile. The fact that reducing each flux contribution to u^* separately has the same result as treating them all at

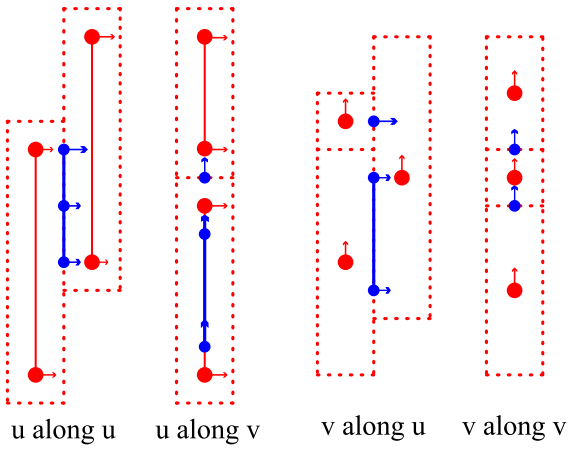

Figure 5.8: Advection fluxes (small blue circles) on minimal control volume faces between adjacent velocity control volumes.

once follows from the linearity of (5.1). As shown in Figure 5.8, the structure of the flux profile differs depending on which component is being advected and in which direction.

For example, given two minimal u faces whose control volumes intersect along the x direction, the averaged velocity profile u_{av} will be linear. If u_{av} has constant sign, F_u has a single quadratic profile (u_{up} is linear) on the control volume face. If u_{av} changes sign, we split the control volume face into two sections with constant sign and handle each section separately. Advection of u along the other horizontal dimension w is similar.

Advecting u along v between distinct minimal u faces is simple since the intersection is a single uniform control volume face. However, we must also account for flux between the top and bottom samples of a linear profile. Averaging v to the middle control volume face produces a constant v_{av} at every virtual control volume face, so the flux profile F_v will be linear. The update of u^* can be simplified by noting that the net flux into every virtual uniform face except for the first and last is constant, since it is the difference of consecutive terms in a linear sequence.

Advection of the other horizontal velocity w is similar to that of u. Advection of v along u or w is the same as for u except that the flux profiles are linear instead of quadratic, and there is no need to consider flux inside middle v faces since they are constant.

First order upwinding requires a CFL condition on the time step for stability, namely

$$\Delta t \max\left(\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y} + \frac{|w|}{\Delta z}\right) \leq 1.$$

If a larger time step is desired for the other parts of the algorithm, the upwinding phase of advection can be subcycled for little extra cost since it is only needed for a small fraction of the velocities.

After advection, we refine/coarsen the grid. This is done before making the velocity field divergence free, because velocity interpolation does not preserve discrete incompressibility.

There are two issues with this advection method which we would like to resolve with future work. First, instabilities sometimes appear near sharp changes in bottom topography such as the walls of the canyon in Figure 5.9. These can occur because the final u^* values are not affine combinations of the starting velocities, since averaging to control volume faces breaks discrete incompressibility. We avoided these instabilities by computing the total weight used for each u^* value, and dividing by this weight if u^* would be larger than an affine combination. Note that these weights can be computed by applying the same advection algorithm to the constant field 1 (i.e., substitute 1 for $\mathbf{u}_{up}$ everywhere). This fix is far from satisfactory, especially since the lack of discrete incompressibility causes no problems for uniform grids. Second, while a first order method is sufficient for bulk motion, it would be interesting to generalize the ENO and WENO schemes typically used for shallow water to the case of tall cells to reduce numerical dissipation.

5.5.2 Laplace Equation

After advection, we solve for the pressure and make the velocities divergence free via

$$\begin{aligned} \nabla \cdot (\nabla p/\rho) &= \nabla \cdot \mathbf{u}^*/\Delta t \\ \mathbf{u}^{n+1} &= \mathbf{u}^* - \Delta t \nabla p/\rho \end{aligned}$$

for the entire collection of uniform and tall cells at once. Since variable density flows do not have approximately linear vertical pressure profiles, we assume that the density is spatially constant and can be moved to the right hand side, i.e. $\nabla \cdot \nabla p = \rho \nabla \cdot \mathbf{u}^*/\Delta t$.

The component of the discrete pressure gradient on a MAC grid face between two MAC grid cells p_1 and p_2 is $\partial p/\partial l = (p_2 - p_1)/\Delta l$ where $l = x$, y, or z. This readily generalizes to arbitrary configurations of tall cells, since we determined the placement and structure of the velocities with pressure gradients in mind (Figure 5.3c,d). For the horizontal derivative, say in the x direction, consider two adjacent tall cells extending from j_1 to j_2 and j_3 to j_4, respectively, intersecting in a minimal face from $j_a = \max(j_1, j_3)$ to $j_b = \min(j_2, j_4)$. If the corresponding pressures are p_1, p_2, p_3, p_4, then the component of the discrete gradient on the face between them is a linear profile with values $(p_x)_a$ and $(p_x)_b$ given by interpolating pressures to j_a and j_b in each cell and applying standard central differencing, e.g.

$$(p_x)_a = \frac{1}{\Delta x}\left(\frac{j_4 - j_a}{j_4 - j_3}p_3 + \frac{j_a - j_3}{j_4 - j_3}p_4 - \frac{j_2 - j_a}{j_2 - j_1}p_1 - \frac{j_a - j_1}{j_2 - j_1}p_2\right)$$

and similarly for $(p_x)_b$. The derivative between vertically adjacent pressure samples p_1 at j_1 and p_2 at j_2 is simply $p_y = (p_2 - p_1)/((j_2 - j_1)\Delta y)$.

The discrete volume weighted divergence of a uniform cell can be written as

$$V(\nabla \cdot \mathbf{u}) = \sum_{f=1}^{6} \pm u_f A_f$$

where the $\pm$ sign is chosen based on which face is being considered, and A_f is the area of the face. We generalize this equation to tall cells by computing the flux into each virtual uniform cell in the tall cell, dividing each flux between the bottom and top samples in the cell, and adding up the contributions. *We divide the virtual uniform fluxes using the same fractions used to interpolate pressures to the given virtual uniform cell in order to ensure the symmetry of the final system.* For example, given a tall cell from j_1 to j_2 and a minimal face from j_a to j_b with velocities u_a and u_b, the total flux contribution from the face given to the upper portion of the cell is

$$\sum_{j=j_a}^{j_b} \frac{j - j_1}{j_2 - j_1} \left(\frac{j_b - j}{j_b - j_a} u_a + \frac{j - j_a}{j_b - j_a} u_b \right) \Delta y \Delta z. \tag{5.6}$$

while

$$\sum_{j=j_a}^{j_b} \frac{j_2 - j}{j_2 - j_1} \left(\frac{j_b - j}{j_b - j_a} u_a + \frac{j - j_a}{j_b - j_a} u_b \right) \Delta y \Delta z.$$

is the contribution to the lower portion of the cell. The vertical flux contribution is always $v \Delta x \Delta z$.

We can now combine the discrete volume weighted divergence and gradient operators to discretize the Laplacian. For the horizontal direction, to get the contribution from the minimal face from j_a to j_b to the top portion of the cell, we substitute $(p_x)_a$ and $(p_x)_b$ for u_a and u_b in (5.6) and collapse the nested interpolation to obtain

$$\frac{\Delta y \Delta z}{\Delta x} \sum_{j=j_a}^{j_b} \frac{j - j_1}{j_2 - j_1} \left(\frac{j_4 - j}{j_4 - j_3} p_3 + \frac{j - j_3}{j_4 - j_3} p_4 - \frac{j_2 - j}{j_2 - j_1} p_1 - \frac{j - j_1}{j_2 - j_1} p_2 \right)$$

and similarly

$$\frac{\Delta y \Delta z}{\Delta x} \sum_{j=j_a}^{j_b} \frac{j_2 - j}{j_2 - j_1} \left(\frac{j_4 - j}{j_4 - j_3} p_3 + \frac{j - j_3}{j_4 - j_3} p_4 - \frac{j_2 - j}{j_2 - j_1} p_1 - \frac{j - j_1}{j_2 - j_1} p_2 \right)$$

is the contribution to the bottom portion of the cell. Note that the coefficient of p_1 in the first equation (which is for p_2) is identical to the coefficient of p_2 in the second equation (which is for p_1). Since all symmetric pairs of contributions to the discretization matrix are identical, the final result is symmetric. The coefficients relating p_3 to p_4 are handled similarly. Moreover, the cross terms relating p_1 to p_3, p_1 to p_4, p_2 to p_3 and p_2 to p_4 are also similarly treated. For the vertical terms, we obtain $(\Delta x \Delta z / \Delta y)(p_2 - p_1)/(j_2 - j_1)$ once again resulting in identical contributions for symmetric

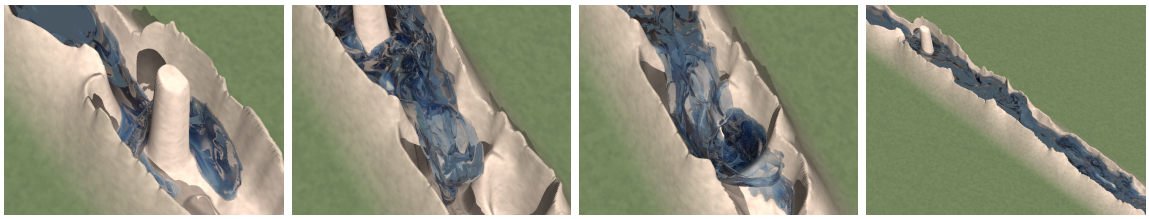

Figure 5.9: Simulation of a river filling a canyon (2000×200 horizontal resolution).

terms. The resulting negative semidefinite linear system can be treated in the same manner as a typical MAC grid discretization. In particular, we can use preconditioned conjugate gradient, incomplete Cholesky preconditioners, and Neumann and Dirichlet boundary conditions anywhere in tall or uniform cells.

5.6 Examples

We implemented our method in parallel using MPI by decomposing the horizontal domain into a two-dimensional grid and giving each processor one fixed rectangular piece of the domain. Before every stage of the fluid solve that requires neighboring data, we fill ghost cells on the sides of each processor from the boundary data of its neighboring processors. In the pressure solve, each processor constructs an incomplete Cholesky preconditioner for the portion of the grid that it owns, and participates in a global PCG solve with a block diagonal preconditioner formed from the incomplete Cholesky blocks on each processor. Extending the serial version of the algorithm to use MPI was relatively straightforward since the domain decomposition is applied only along uniform horizontal dimensions, and the use of tall cells for the bulk of the fluid significantly reduced the communication bandwidth required. All simulations were run on a cluster of 4 processor 2.6 GHz Opteron machines.

The boat simulations in Figures 5.1 and 5.7 used 16 processors in an 8×2 grid. The optical depth was 1/5 the water depth and the cost was approximately 3 minutes a frame. For the boat examples we used vortex particles [108] and rendered removed negative particles as in [52]. Figure 5.9 shows a river simulation with varying bottom topography using 20 processors in a 20×1 grid. The computational cost was approximately 25 minutes a frame, and the optical depth is illustrated in Figure 5.4. Memory consumption was not a significant issue in any of the simulations.

5.7 Conclusion

We have presented a novel method for the simulation of large bodies of water by combining two-dimensional and three-dimensional simulation techniques. The bulk of the water volume is represented with tall cells similar to a height field method, and a surface layer of water is simulated with a

state-of-the-art, fully three-dimensional Navier-Stokes free surface solver. This algorithm works well for capturing detailed surface motion and for representing detailed bottom topography. Our general philosophy is to use the best available method in the surface layer where we expect interesting detail, and to coarsen the mesh away from the interface for efficiency.

Like shallow water, our method only admits computational gains for flows heavily dominated by gravity, where a large portion of the water is in near vertical equilibrium against the ground. In general, we believe that the best approach for any simulation is to start with a uniform grid that captures the interesting flow features, coarsen vertically using our method in areas where linear pressure profiles are sufficient, and then refine uniform cells using an octree in areas where more resolution is desirable. Finally, we note that the effects of bottom topography on the surface diminish with increasing depth, and in the deep water limit the effects are negligible. However, we are interested in problems in the shallow water regime, where tall cells are almost exclusively used.

Chapter 6

Viscoelastic Fluid Simulation

[48] incorporated viscoelastic effects into an incompressible fluid solver by adding an advected elastic strain tensor to the Navier-Stokes equations. Unfortunately, their formulation ignores the rotation of the strain tensor due to local rotations of the fluid. This chapter presents a small modification to their method which restores rotation invariance. The extra terms are discretized using exact rotations to ensure their stability. We illustrate the difference between these two models with a simple spinning rectangle test, and also show a more complicated example which uses the correction. The modified model is unfortunately not hyperelastic, but this is not a serious concern since most applications of viscoelastic fluids in graphics involve significant plastic flow for large deformations.

6.1 Previous Work

[48] introduced viscoelastic fluid simulation to graphics based on the work of [47], which presented algorithms for simulation of non-Newtonian fluids in two dimensions. [47] stored diagonal components of the stress tensor in the centers of cells, and stored off-diagonal components at nodes (which correspond to edges in three dimensions). This results in very compact definitions for divergence of stress and gradient of velocity analogous to the discretization of the Laplace equation on a MAC grid. Unfortunately, discretization of the rotation terms require colocated tensors, so the stress must be averaged back and forth between colocated and non-colocated representations each time step. Once in colocated form, matrix exponentials were used to advance the stress forward in time based on the velocity gradient.

[48] simplified this model by ignoring the rotation terms and treating stretch linearly, which is equivalent to a small strain assumption on the deformation. The resulting model avoids the need for averaging between colocated and non-colocated tensors, but is incorrect for large rotation. This error was noted in [81], which presented a method for simulating viscoelastic flow using SPH. They show that the rod climbing effect characteristic of viscoelastic fluids is lost if the rotation terms are

ignored. A much simpler example of these errors is given below.

In the CFD community, much effort has been applied to the numerical instabilities associated with high Weissenberg numbers (see [132] for a detailed survey). Recently, [38, 39, 63] have partially alleviated this problem by discretizing the stress evolution using logarithmic variables and treating rotation and stretch terms separately. Their method treats all stress evolution terms in logarithmic form, converting back only when necessary to apply viscoelastic forces.

6.2 Strain Rotation

[48] simulated viscoelastic flow using purely linear stress and strain. Assuming constant viscosity μ and constant elastic modulus E, the equations in the interior of the fluid are

$$\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} + \frac{\nabla p}{\rho} = \frac{\mu}{\rho} \nabla \cdot \nabla \mathbf{u} + \frac{E}{\rho} \nabla \cdot \boldsymbol{\epsilon} + \frac{\mathbf{f}}{\rho} \tag{6.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{6.2}$$

where $\mathbf{u}$ is the velocity, p is the pressure, ρ is the density, $\boldsymbol{\epsilon}$ is the strain (a symmetric 3×3 matrix at each point), and $\mathbf{f}$ is the external force. If we let $\boldsymbol{\epsilon}' = \boldsymbol{\epsilon} - \operatorname{tr} \boldsymbol{\epsilon}/3$ be the deviatoric part of the strain, the linear strain evolution follows

$$\boldsymbol{\epsilon}_t + \mathbf{u} \cdot \nabla \boldsymbol{\epsilon} = \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^T}{2} - \alpha \boldsymbol{\epsilon}' \max \left(0, 1 - \frac{\gamma}{\|\boldsymbol{\epsilon}'\|} \right) \tag{6.3}$$

where γ is the plastic yield and α is the rate of plastic flow. This equation is correct for small strain, but does not account for the rotation of the strain tensor due to the rotation of the fluid. This results in errors such as that shown in Figure 6.1 (dotted line), where the rest state of a spinning object is not preserved.

The simplest way to restore rotation invariance to this model is to add an additional term to (6.3) which rotates the strain tensor at the current rate of rotation. Since the rate of rotation of the fluid is $\left(\nabla \mathbf{u} - \nabla \mathbf{u}^T \right) /2$ and $\boldsymbol{\epsilon}$ rotates on both the left and the right, the new equation is

$$\boldsymbol{\epsilon}_t + \mathbf{u} \cdot \nabla \boldsymbol{\epsilon} = \frac{\nabla \mathbf{u} - \nabla \mathbf{u}^T}{2} \boldsymbol{\epsilon} - \boldsymbol{\epsilon} \frac{\nabla \mathbf{u} - \nabla \mathbf{u}^T}{2} + \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^T}{2} - \alpha \boldsymbol{\epsilon}' \max \left(0, 1 - \frac{\gamma}{\|\boldsymbol{\epsilon}'\|} \right) \tag{6.4}$$

This corresponds to a corotational Maxwell model for strain, and was used in [81]. (6.4) is the simplest strain evolution law that satisfies rotation invariance, and is attractive in that all nonlinearity is confined to the rotation terms. As shown in Figure 6.1 (solid line), the additional terms remove the errors associated with rotating objects.

Unfortunately, as discussed in [119], the corotational Maxwell model is not hyperelastic (i.e. there is no corresponding strain energy function) even with no plastic flow. In particular, a deformation

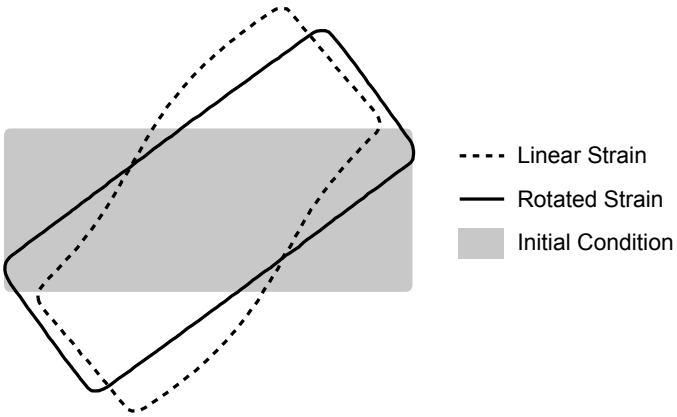

Figure 6.1: Comparison of linear and rotation-corrected strain for a spinning rectangle. Incorrect treatment of rotation in the linear model results in a distorted final state (200×200 grid).

of the fluid can result in zero strain without returning to its rest state. The easiest way to see this is to note that in the absence of convection, rotation, and plastic flow, both (6.3) and (6.4) reduce to $\boldsymbol{\epsilon}_t = \nabla \mathbf{u}$ which implies that $\boldsymbol{\epsilon}(t)$ depends only on the average value of $\nabla \mathbf{u}$. In contrast, the evolution of the deformation gradient is given by $\mathbf{F}_t = \nabla \mathbf{u} \mathbf{F}$ which depends multiplicatively on $\nabla \mathbf{u}$. Since symmetric matrix multiplication is not commutative, $\mathbf{F}(t)$ necessarily depends on the order of occurrence of different values of $\nabla \mathbf{u}$, so $\boldsymbol{\epsilon}$ cannot be a function of $\mathbf{F}$. The lack of hyperelasticity is not a serious problem in many applications, which involve significant plastic flow. Note that if hypoelasticity is unacceptable for a given application and one of the fully nonlinear models is used, the entire strain update should be performed in logarithmic space for stability as in [38].

6.3 Discretization

We discretize (6.4) in time using operator splitting, evaluating all terms of (6.4) using $\mathbf{u}^n$ to advance $\boldsymbol{\epsilon}^n$ to $\boldsymbol{\epsilon}^{n+1}$. The force due to strain in (6.1) is then computed as $E \nabla \cdot \boldsymbol{\epsilon}^{n+1}/\rho$. Given the frozen velocity field $\mathbf{u}^n$, we use semi-Lagrangian for the advection term $\mathbf{u} \cdot \nabla \boldsymbol{\epsilon}$ followed by exact integration of each of the rotation, stretch, and plastic flow terms. This amounts to a leapfrog discretization of the elastic response, since $\mathbf{u}^n$ is used to compute $\boldsymbol{\epsilon}^{n+1}$ which is used to compute $\mathbf{u}^{n+1}$. A single strain time step proceeds as follows:

1. $\boldsymbol{\epsilon}_1 = \text{Advect}(\boldsymbol{\epsilon}^n, \Delta t \mathbf{u}^n)$
2. $\boldsymbol{\epsilon}_2 = e^{\Delta t A} \boldsymbol{\epsilon}_1 e^{-\Delta t A}$, where $A = (\nabla \mathbf{u} - \nabla \mathbf{u}^T)/2$
3. $\boldsymbol{\epsilon}_3 = \boldsymbol{\epsilon}_2 + \Delta t (\nabla \mathbf{u} + \nabla \mathbf{u}^T)/2$
4. $\boldsymbol{\epsilon}^{n+1} = \boldsymbol{\epsilon}_3 - \boldsymbol{\epsilon}_3' \min(1, \Delta t \alpha) \max(0, 1 - \gamma / \|\boldsymbol{\epsilon}_3'\|)$

All components of strain are colocated in cell centers to allow pointwise matrix multiplication. The component derivatives in $\nabla \mathbf{u}$ and $\nabla \cdot \boldsymbol{\epsilon}$ are evaluated via central differencing, noting that the stencil size is either Δx or $2\Delta x$ depending on which dimensions are involved due to the MAC layout of velocities. The linearized elastic sound speed of this system is $\sqrt{\rho/E}$, so the CFL condition for the viscoelastic terms is $\Delta t \leq \Delta x \sqrt{\rho/E}$.

Note that the choice of $\boldsymbol{\epsilon}^{n+1}$ to compute $\mathbf{u}^{n+1}$ is important for conditional stability in the case of low viscosity. If $\boldsymbol{\epsilon}^n$ is used, the time discretization reduces to forward Euler for zero viscosity, which is unconditionally unstable. More generally, the behavior of the two schemes can be analyzed by considering the ODE for a single linear mode with frequency ω and overdamping fraction d. This gives $\ddot{x} + 2d\omega\dot{x} + \omega^2 x = 0$. Using $\boldsymbol{\epsilon}^n$ to update $\mathbf{u}^{n+1}$ corresponds to the integration scheme

$$\begin{aligned} x^{n+1} &= x^n + \Delta t v^{n+1} \\ v^{n+1} &= v^n - \omega^2 \Delta t x^n - 2d\omega \Delta t v^{n+1} \end{aligned}$$

which is stable iff $\Delta t < 2d/\omega$. This vanishes as $d \to 0$. In contrast, using $\boldsymbol{\epsilon}^{n+1}$ gives

$$\begin{aligned} x^{n+1} &= x^n + \Delta t v^{n+1} \\ v^{n+1} &= v^n - \omega^2 \Delta t x^{n+1} - 2d\omega \Delta t v^{n+1} \end{aligned}$$

which is stable iff $\Delta t < 2(d + \sqrt{d+1})/\omega$.

Analytic boundary conditions for strain are Dirichlet ($\boldsymbol{\epsilon} = 0$) for free surfaces or Neumann ($\mathbf{n} \cdot \nabla \boldsymbol{\epsilon} = 0$) for no-slip kinematic objects such as walls. For the advection step, strain should be extrapolated both into walls and across the interface so that information is not accidentally pulled from outside the fluid due to discretization error. When evaluating $\nabla \cdot \boldsymbol{\epsilon}$ for force computation, the correct Dirichlet conditions should be used near the interface.

6.4 Examples

Figure 6.1 shows the final state of a $1/2 \times 1/5$ m rectangle of fluid with an initial angular velocity of 5 s^{-1}, density $\rho = 1000$ kg/m^2, viscosity $\mu = 25$ kg/s, and elastic modulus $E = 1000$ kg/s^2. The bar stops rotating due to incorrect boundary conditions for viscosity, where the two velocity components are coupled physically but are solved using independent heat equations with Dirichlet boundary conditions for efficiency. These errors are discussed in [80], and the correct flux is derived in [72]. Note that under more dramatic deformation, neither model would return to the rest state due to hypoelasticity. Finally, Figure 6.2 shows a multiphase fluid simulation from [80] with different viscoelasticities in different phases.

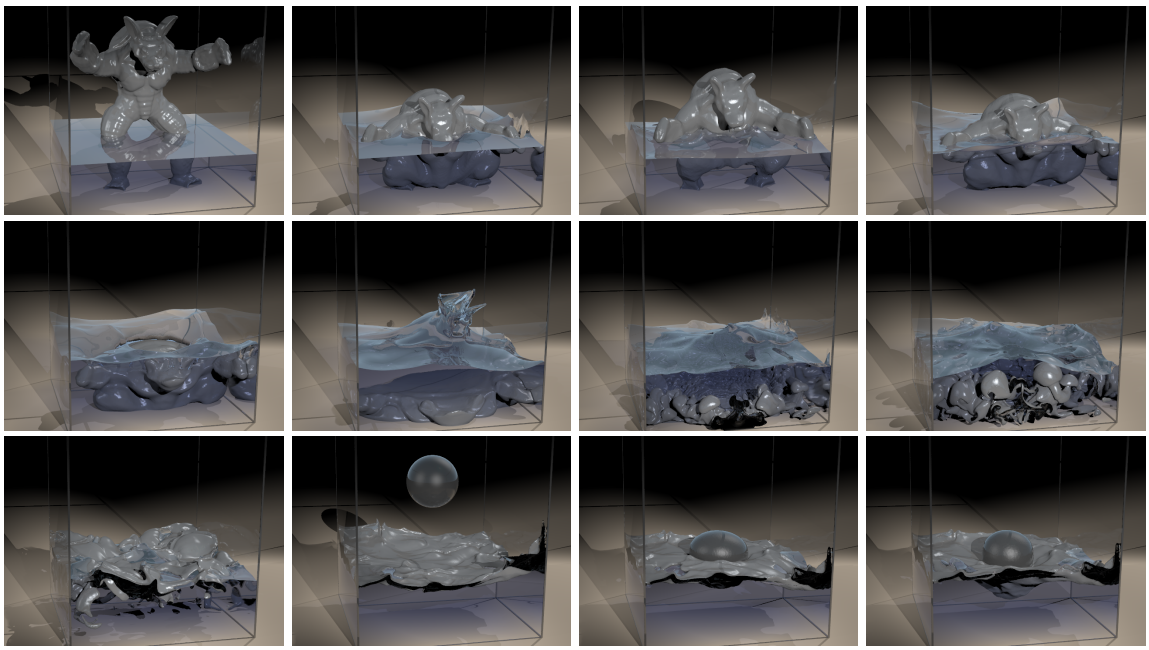

Figure 6.2: An armadillo that starts out viscoelastic, becomes viscous and more dense than the water, then inviscid and lighter than the water, and finally viscoelastic again before another viscoelastic liquid is dropped onto it ($250 \times 275 \times 250$ grid, 4 phases).

Bibliography

[1] D. Adalsteinsson and J. Sethian. A fast level set method for propagating interfaces. *J. Comput. Phys.*, 118:269–277, 1995.

[2] A. Angelidis, M.P. Cani, G. Wyvill, and S. King. Swirling-sweepers: constant volume modeling. *Graph. Models*, 68(4):324–32, 2006.

[3] F. Armero and E. Love. An arbitrary Lagrangian-Eulerian finite element method for finite strain plasticity. *Int. J. Num. Meth. Eng.*, 57:471–508, 2003.

[4] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proc. SIGGRAPH 98*, pages 43–54, 1998.

[5] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22:862–870, 2003.

[6] D. Baraff, A. Witkin, M. Kass, and J. Anderson. Physically based modeling (a little fluid dynamics for graphics). In *SIGGRAPH Course Notes*. ACM, 2003.

[7] W. Baxter, J. Wendt, and M. Lin. Impasto: A realistic, interactive model for paint. In *Proc. of Non-Photorealistic Anim. and Rendering*, pages 45–56, 2004.

[8] G. Bessette, E. Becker, L. Taylor, and D. Littlefield. Modeling of impact problems using an H-adaptive, explicit Lagrangian finite element method in three dimensions. *Comput. Meth. in Appl. Mech. and Eng.*, 192:1649–1679, 2003.

[9] I. Bijelonja, I. Demirdžić, and S. Muzaferija. A finite volume method for large strain analysis of incompressible hyperelastic materials. *Int. J. Num. Meth. Eng.*, 64:1594–1609, 2005.

[10] I. Bijelonja, I. Demirdžić, and S. Muzaferija. A finite volume method for incompressible linear elasticity. *Comput. Meth. Appl. Meth. Eng.*, 195:6378–90, 2006.

[11] J. Bonet and A. Burton. A simple average nodal pressure tetrahedral element for incompressible and nearly incompressible dynamic explicit applications. *Comm. Num. Meth. Eng.*, 14:437–449, 1998.

[12] J. Bonet and R. Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge University Press, Cambridge, 1997.

[13] B. Boroomand and B. Khalilian. On using linear elements in incompressible plane strain problems: a simple edge based approach for triangles. *Int. J. Num. Meth. Eng.*, 61:1710–1740, 2004.

[14] D. Bourguignon and M. P. Cani. Controlling anisotropy in mass-spring systems. In *Eurographics*, pages 113–123. Eurographics Assoc., 2000.

[15] D. Breen, R. Fedkiw, K. Museth, S. Osher, G. Sapiro, and R. Whitaker. Level sets and PDE methods for computer graphics. In *SIGGRAPH Course Notes*. ACM, 2004.

[16] R. Bridson. *Computational Aspects of Dynamic Surfaces*. PhD thesis, Stanford University, June 2003.

[17] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:594–603, 2002.

[18] 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.

[19] G. Camacho and M. Ortiz. Adaptive Lagrangian modelling of ballistic penetration of metallic targets. *Comput. Meth. in Appl. Mech. and Eng.*, 142:269–301, 1997.

[20] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:586–593, 2002.

[21] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. A multiresolution framework for dynamic deformations. In *ACM SIGGRAPH Symp. on Comput. Anim.*, pages 41–48. ACM Press, 2002.

[22] 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.

[23] D. Chen and D. Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. *Comput. Graph. (SIGGRAPH Proc.)*, pages 89–98, 1992.

[24] J. Chen and N. Lobo. Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations. *Comput. Graph. and Image Processing*, 57:107–116, 1994.

[25] K.-J. Choi and H.-S. Ko. Stable but responsive cloth. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:604–611, 2002.

[26] S.-C. Choi. *Iterative Methods for Singular Linear Equations and Least-Squares Problems*. PhD thesis, Stanford University, 2006.

[27] B. Cockburn, D. Schötzau, and J. Wang. Discontinuous Galerkin methods for incompressible elastic materials. *Comput. Meth. Appl. Meth. Eng.*, 195:3184–3204, 2006.

[28] L. Cooper and S. Maddock. Preventing collapse within mass-spring-damper models of deformable objects. In *5th Int. Conf. in Central Europe on Comput. Graph. and Vis.*, 1997.

[29] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Comput. Graph. (SIGGRAPH Proc.)*, pages 303–312, 1996.

[30] E. de Souza Neto, F. Andrade Pires, and D. Owen. F-bar-based linear triangles and tetrahedra for finite strain analysis of nearly incompressible solids. Part I: formulation and benchmarking. *Int. J. Num. Meth. Eng.*, 62:353–383, 2005.

[31] G. Debunne, M. Desbrun, M. Cani, and A. Barr. Dynamic real-time deformations using space and time adaptive sampling. In *Proc. SIGGRAPH 2001*, volume 20, pages 31–36, 2001.

[32] M. Desbrun and M.-P. Gascuel. Animating soft substances with implicit surfaces. In *Proc. SIGGRAPH 95*, pages 287–290, 1995.

[33] J.E. Dolbow and A. Devan. Enrichment of enhanced assumed strain approximation for representing strong discontinuities: Addressing volumetric incompressibility and the discontinuous patch test. *Int. J. Num. Meth. Eng.*, 59:47–67, 2004.

[34] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):736–744, 2002.

[35] J. Escobar, E. Rodríguez, R. Montenegro, G. Montero, and J. González-Yuste. Simultaneous untangling and smoothing of tetrahedral meshes. *Comput. Meth. in Appl. Mech. and Eng.*, 192:2775–2787, 2003.

[36] H. Espinosa, P. Zavattieri, and G. Emore. Adaptive FEM computation of geometric and material nonlinearities with application to brittle failure. *Mech. Materials*, 29:275–305, 1998.

[37] O. Etzmuss, M. Keckeisen, and W. Strasser. A fast finite element solution for cloth modelling. In *Pacific Graph.*, pages 244–251, 2003.

[38] R. Fattal and R. Kupferman. Constitutive laws for the matrix-logarithm of the conformation tensor. *J. Non-Newtonian Fluid Mech.*, 123(2-3):281–285, 2004.

[39] R. Fattal and R. Kupferman. Time-dependent simulation of viscoelastic flows at high weissenberg number using the log-conformation representation. *J. Non-Newtonian Fluid Mech.*, 126(1):23–37, 2005.

[40] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In *Proc. of ACM SIGGRAPH 2001*, pages 15–22, 2001.

[41] B. Feldman, J. O'Brien, and O. Arikan. Animating suspended particle explosions. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):708–715, 2003.

[42] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proc. of ACM SIGGRAPH 2001*, pages 23–30, 2001.

[43] N. Foster and D. Metaxas. Realistic animation of liquids. *Graph. Models and Image Processing*, 58:471–483, 1996.

[44] N. Foster and D. Metaxas. Controlling fluid animation. In *Comput. Graph. Int.*, pages 178–188, 1997.

[45] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *Proc. of SIGGRAPH 97*, pages 181–188, 1997.

[46] A. Fournier and W. Reeves. A simple model of ocean waves. In *Comput. Graph. (Proc. of SIGGRAPH 86)*, volume 20, pages 75–84, 1986.

[47] M. Gerritsma. *Time dependent numerical simulations of a viscoelastic fluid on a staggered grid*. PhD thesis, Rijksuniversiteit Groningen, 1996.

[48] 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.

[49] J.-P. Gourret, N. Magnenat-Thalmann, and D. Thalmann. Simulation of object and human skin deformations in a grasping task. *Comput. Graph. (SIGGRAPH Proc.)*, pages 21–30, 1989.

[50] 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.

[51] E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A simple framework for adaptive simulation. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:281–290, 2002.

[52] 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.

[53] Y. Guo, M. Ortiz, T. Belytschko, and E. Repetto. Triangular composite finite elements. *Int. J. Num. Meth. Eng.*, 47:287–316, 2000.

[54] D. Hinsinger, F. Neyret, and M.-P. Cani. Interactive animation of ocean waves. In *ACM SIGGRAPH Symp. on Comput. Anim.*, pages 161–166, 2002.

[55] G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs. An implicit finite element method for elastic solids in contact. In *Proc. of Comput. Anim.*, pages 136–146, 2001.

[56] C. Hirt, A. Amsden, and J. Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *J. Comput. Phys.*, 135:227–253, 1974.

[57] J.-M. Hong and C.-H. Kim. Discontinuous fluids. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):915–920, 2005.

[58] M. Hong, S. Jung, M.H. Choi, and S. Welch. Fast volume preservation for a mass-spring system. *IEEE Comput. Graph. and Appl.*, 26:83–91, September 2006.

[59] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Gigantic deformable surfaces. In *SIGGRAPH 2005 Sketches & Applications*. ACM Press, 2005.

[60] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph.*, 25(1):1–24, 2006.

[61] B. Houston, M. Wiebe, and C. Batty. RLE sparse level sets. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[62] T. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice Hall, 1987.

[63] M. Hulsen, R. Fattal, and R. Kupferman. Flow of viscoelastic fluids past a cylinder at high weissenberg number: stabilized simulations using matrix logarithms. *J. Non-Newtonian Fluid Mech.*, 127(1):27–39, 2005.

[64] 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.

[65] G. Irving, C. Schroeder, and R. Fedkiw. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph. (SIGGRAPH Proc.) (in press)*, 26(3), 2007.

[66] 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.

[67] G. Irving, J. Teran, and R. Fedkiw. Tetrahedral and hexahedral invertible finite elements. *Graph. Models*, 68(2):66–89, 2006.

[68] J. Iversen and R. Sakaguchi. Growing up with fluid simulation on "The Day After Tomorrow". In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[69] D. James and K. Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22:879–887, 2003.

[70] D. James and D. Pai. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:582–585, 2002.

[71] 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.

[72] M. Kang, R. Fedkiw, and X.-D. Liu. A boundary condition capturing method for multiphase incompressible flow. *J. Sci. Comput.*, 15:323–360, 2000.

[73] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *Comput. Graph. (Proc. of SIGGRAPH 90)*, volume 24, pages 49–57, 1990.

[74] R. Kautzman, A. Maiolo, D. Griffin, and A. Bueker. Jiggly bits and motion retargetting: Bringing the motion of Hyde to life in Van Helsing with dynamics. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[75] L. Kharevych, W. Yang, Y. Tong, E. Kanso, J. Marsden, P. Schröder, and M. Desbrun. Geometric variational integrators for computer animation. *ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 43–51, 2006.

[76] S. Lahiri, J. Bonet, J. Peraire, and L. Casals. A variationally consistent fractional time-step integration method for incompressibility and nearly incompressible Lagrangian dynamics. *Int. J. Num. Meth. Eng.*, 59:1371–95, 2005.

[77] R. Lasseter. Principles of traditional animation applied to 3D computer animation. *Comput. Graph. (SIGGRAPH Proc.)*, pages 35–44, 1987.

[78] F. Losasso, R. Fedkiw, and S. Osher. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids*, 35:995–1010, 2006.

[79] 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.

[80] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw. Multiple interacting liquids. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 25(3):812–819, 2006.

[81] H. Mao and Y.-H. Yang. Particle-based non-Newtonian fluid animation with heating effects. Technical Report TR06-12, Univ. of Alberta, 2006.

[82] G. Mastin, P. Watterberg, and J. Mareda. Fourier synthesis of ocean scenes. *IEEE Comput. Graph. Appl.*, 7(3):16–23, 1987.

[83] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Trans. Graph. (SIGGRAPH Proc.)*, pages 449–456, 2004.

[84] 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.

[85] 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.

[86] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *ACM SIGGRAPH Symp. on Comput. Anim.*, pages 49–54, 2002.

[87] M. Müller and M. Gross. Interactive virtual materials. In *Graph. Interface*, pages 239–246, May 2004.

[88] M. Müller, L. McMillan, J. Dorsey, and R. Jagnow. Real-time simulation of deformation and fracture of stiff materials. In *Comput. Anim. and Sim. '01*, Proc. Eurographics Wrkshp., pages 99–111. Eurographics Assoc., 2001.

[89] L. P. Nedel and D. Thalmann. Real time muscle deformations using mass-spring systems. In *Proc. Comput. Graph. Int.*, pages 156–165, 1998.

[90] F. Neyret and N. Praizelin. Phenomenological simulation of brooks. In *Comput. Anim. and Sim. '01*, Proc. Eurographics Wrkshp., pages 53–64, 2001.

[91] M. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Accepted to SIAM J. Scientific Comput.*, 2005.

[92] D. Nixon and R. Lobb. A fluid-based soft-object model. *IEEE Comput. Graph. Appl.*, 22(4):68–75, 2002.

[93] E. Oñate, J. Rojek, R. Taylor, and O. Zienkiewicz. Finite calculus formulation for incompressible solids using linear triangles and tetrahedra. *Int. J. Num. Meth. Eng.*, 59:1473–1500, 2004.

[94] J. O'Brien, A. Bargteil, and J. Hodgins. Graphical modeling of ductile fracture. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:291–294, 2002.

[95] J. O'Brien and J. Hodgins. Graphical modeling and animation of brittle fracture. In *Proc. SIGGRAPH 99*, volume 18, pages 137–146, 1999.

[96] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *Comput. Anim. '95*, pages 198–205, apr 1995.

[97] B. Palmerio. An attraction-repulsion mesh adaption model for flow solution on unstructured grids. *Comput. and Fluids*, 23(3):487–506, 1994.

[98] D. Peachey. Modeling waves and surf. In *Comput. Graph. (Proc. of SIGGRAPH 86)*, volume 20, pages 65–74, 1986.

[99] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-based fast local level set method. *J. Comput. Phys.*, 155:410–438, 1999.

[100] G. Picinbono, H. Delingette, and N. Ayache. Nonlinear and anisotropic elastic soft tissue models for medical simulation. In *IEEE Int. Conf. Robot. and Automation*, 2001.

[101] F. Andrade Pires, E. de Souza Neto, and D. Owen. On the finite element prediction of damage growth and fracture initiation in finitely deforming ductile materials. *Comput. Meth. in Appl. Mech. and Eng.*, 193:5223–5256, 2004.

[102] J. Platt and A. Barr. Constraint methods for flexible models. *Comput. Graph. (SIGGRAPH Proc.)*, pages 279–288, 1988.

[103] J. Popović, S. Seitz, M. Erdmann, Z. Popović, and A. Witkin. Interactive manipulation of rigid body simulations. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 19:209–217, 2000.

[104] E. Promayon, P. Baconnier, and C. Puech. Physically-based deformations constrained in displacements and volume. In *Eurographics*, volume 15. Eurographics Assoc., 1996.

[105] S. Punak and J. Peters. Localized volume preservation for simulation and animation. In *Poster, SIGGRAPH Proc.* ACM, 2006.

[106] J. Rojek, E. O nate, and R. Taylor. CBS-based stabilization in explicit solid dynamics. *Int. J. Num. Meth. Eng.*, 66:1547–68, 2006.

[107] S. H. Roth, M. Gross, M. H. Turello, and S. Carls. A Bernstein-Bézier based approach to soft tissue simulation. *Comput. Graph. Forum (Proc. Eurographics)*, 17(3):285–294, 1998.

[108] 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.

[109] A. Selle, J. Su, G. Irving, and R. Fedkiw. Highly detailed folds and wrinkles for cloth simulation. In review, 2007.

[110] L. Shi and Y. Yu. Taming liquids for rapidly changing targets. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 229–236, 2005.

[111] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw. Hybrid simulation of deformable solids. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim. (in press)*, 2007.

[112] J. Simo and R. Taylor. Quasi-incompressible finite elasticity in principal stretches: continuum basis and numerical examples. *Comput. Meth. in Appl. Mech. and Eng.*, 51:273–310, 1991.

[113] J. Stam. Stable fluids. In *Proc. of SIGGRAPH 99*, pages 121–128, 1999.

[114] T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki. Realistic animation of fluid with splash and foam. *Comp. Graph. Forum (Eurographics Proc.)*, 22(3):391–400, 2003.

[115] G. Taylor, C. Bailey, and M. Cross. A vertex-based finite volume method applied to nonlinear material problems in computational solid mechanics. *Int. J. Num. Meth. Eng.*, 56:507–529, 2003.

[116] 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.

[117] 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.*, 2005.

[118] J. Teran, E. Sifakis, S. Salinas-Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *IEEE Trans. on Vis. and Comput. Graph.*, 11(3):317–328, 2005.

[119] T. Tervoort, R. Smit, W. Brekelmans, and L. Govaert. A constitutive equation for the elasto-viscoplastic deformation of glassy polymers. *Mech. of Time-Dependent Materials*, 1:269–291, 1998.

[120] D. Terzopoulos and K. Fleischer. Deformable models. *The Vis. Comput.*, 4(6):306–331, 1988.

[121] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscoelasticity, plasticity, fracture. *Comput. Graph. (SIGGRAPH Proc.)*, pages 269–278, 1988.

[122] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Comput. Graph. (Proc. SIGGRAPH 87)*, 21(4):205–214, 1987.

[123] M. Teschner, B. Heidelberger, M. Müller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. In *Proc. Comput. Graph. Int.*, pages 312–319, 2004.

[124] J. Tessendorf. Simulating ocean water. In *SIGGRAPH 2002 Course Notes #9 (Simulating Nature: Realistic and Interactive Techniques)*. ACM Press, 2002.

[125] S. Thon, J.-M. Dischler, and D. Ghazanfarpour. Ocean waves synthesis using a spectrum-based turbulence function. In *Comput. Graph. Int.*, pages 65–74, 2000.

[126] S. Thon and D. Ghazanfarpour. A semi-physical model of running waters. *Comput. Graph. Forum (Proc. Eurographics)*, 19:53–59, 2001.

[127] P. Thoutireddy, J. Molinari, E. Repetto, and M. Ortiz. Tetrahedral composite finite elements. *Int. J. Num. Meth. Eng.*, 53:1337–1351, 2002.

[128] A. Treuille, A. McNamara, Z. Popovic, and J. Stam. Keyframe control of smoke simulations. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22:716–723, 2003.

[129] P. Ts'o and B. Barsky. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Trans. Graph.*, 6(3):191–214, 1987.

[130] P. Vachal, R. Garimella, and M. Shashkov. Untangling of 2D meshes in ALE simulation. *J. Comput. Phys.*, 196:627–644, 2004.

[131] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 25(3):1118–1125, 2006.

[132] K. Walters and M. Webster. The distinctive CFD challenges of computational rheology. *Int. J. Num. Meth. in Fluids*, 43(5):577–596, 2003.

[133] H. Wang, P. Mucha, and G. Turk. Water drops on surfaces. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):921–929, 2005.

[134] J. Weiss, B. Maker, and S. Govindjee. Finite-element implementation of incompressible, transversely isotropic hyperelasticity. *Comput. Meth. in Appl. Mech. and Eng.*, 135:107–128, 1996.

[135] R. T. Whitaker. A level-set approach to 3D reconstruction from range data. *Int. J. Comput. Vision*, 29(3):203–231, 1998.

[136] M. Wiebe and B. Houston. The tar monster: Creating a character with fluid simulation. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[137] G. Yngve, J. O'Brien, and J. Hodgins. Animating explosions. In *Proc. SIGGRAPH 2000*, volume 19, pages 29–36, 2000.

[138] Q. Zhu, Y. Chen, and A. Kaufman. Real-time biomechanically-based muscle volume deformation using FEM. *Comput. Graph. Forum*, 190(3):275–284, 1998.

[139] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):965–972, 2005.