SIMULATION OF COUPLED RIGID AND DEFORMABLE SOLIDS AND MULTIPHASE FLUIDS

A DISSERTATION
SUBMITTED TO THE PROGRAM IN SCIENTIFIC COMPUTING
AND COMPUTATIONAL MATHEMATICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Tamar Shinar
July 2008

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

(Ronald Fedkiw) Principal Adviser

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

(Scott Delp)

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.

(Charbel Farhat)

Approved for the University Committee on Graduate Studies.

Abstract

This thesis presents methods for the physically-based simulation of various solid and fluid phenomena, including coupled rigid and deformable bodies, multiphase incompressible flow, and rigid and deformable volumes or shells coupled to incompressible flows.

First, we develop a hybrid deformable solid simulation framework for combining mesh-based and point-based representations of deformable solids. Kinematic relationships between sets of points or simulation objects are defined by introducing hard bindings, providing a variety of capabilities such as the simulation of meshes with T-junctions and improved collision resolution. Soft bindings are then introduced, allowing for additional degrees of freedom while using spring-like forces to target an underlying state. We further apply this framework to partially couple rigid and deformable bodies.

We then develop a framework for the full two-way coupling of rigid and deformable bodies, which is achieved with both a unified time integration scheme as well as individual two-way coupled algorithms at each point of that scheme. Thus we do not require specialized methods for dealing with stability issues or interleaving parts of the simulation. We maintain the ability to treat the key desirable aspects of rigid bodies (e.g. contact, collision, stacking, and friction) and deformable bodies (e.g. arbitrary constitutive models, thin shells, and self-collisions). In addition, our simulation framework supports more advanced features such as proportional derivative controlled articulation between rigid bodies. This not only allows for the robust simulation of a number of new phenomena, but also directly lends itself to the design of deformable creatures with proportional derivative controlled articulated rigid skeletons that interact in a life-like way with their environment.

Next, we extend the particle level set method for the geometric representation of three or more fluid regions. The method uses a separate level set function and a separate set of particles for each region. A novel projection algorithm is used to uniquely define the interface at any given point, providing a dictionary for translating the vector-valued multiple level set function into the standard single-valued level set representation. We use this method to simulate multiple interacting liquids with varying densities and viscosities, viscoelastic properties, surface tension forces and surface reactions such as combustion.

Finally, we present a novel scheme for the two-way coupling of incompressible fluids and rigid or deformable volumetric solids or thin shells. The scheme uses an Eulerian representation for the fluid and a Lagrangian representation for the solid and enforces the no-slip boundary condition at the solid/fluid interface in a momentum conserving way. The coupling is treated implicitly, and the resulting linear system is symmetric indefinite.

Acknowledgments

I would like to thank my parents, Ruth and Joseph, for their constant support and encouragement, for striving to give me exceptional educational opportunities, and for creating an environment that encouraged learning and achievement. I would also like to thank my dear siblings, Anat, Yaeli, and Ron.

I would like to thank my advisor, Ron. I very much enjoyed being a part of the research environment that he facilitates and learning from his fresh and unique perspective. I am also grateful for his encouragment and his high expectations of me and for the many opportunities he has helped to give me. Thank you. It has been a rich time and I feel I will continue to draw lessons from it going forward.

Ron's research group has been a very fun and lively place to be a part of. I would like to acknowledge my coauthors, Frank Losasso Petterson, Andrew Selle, Eftychis Sifakis, Geoffrey Irving, Jeong-Mo Hong, Craig Schroeder, Avi Robinson-Mosher, Jon Gretarsson, and Jon Su. I am grateful to have had the opportunity to work very closely with and learn a lot in my collaboration with Craig. I also very much enjoyed and benefited from my close collaborations with Frank, Avi, Eftychis and Jeong-Mo. I would also like to acknowledge the other past and present members of the group. It was fun to work and interact with them: Eran Guendelman, Rachel Weinstein, Nipun Kwatra, Michael Lentine, Kevin Der, Igor Neverov, Joseph Teran, Zhaosheng Bao, and Robert Strzodka. I would like to thank Joey for helping me with my next plans.

I would like to thank my very good friend, Katja. She has been a very welcome and unexpected source of happiness in my life at Stanford. I would also like to thank my friend

Nathan for his steadfast support and no-nonsense approach. I would like to thank my friend and sister, Yaeli, for her love and support. Thanks also to my friend Geoffrey, for the gifts that he's given me and for our interaction.

I would like to acknowledge the SCCM/ICME program and the people involved for giving me the opportunity to study at Stanford and contributing to my education here. Special thanks to Gene Golub and Michael Saunders for their help and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Physically-based simulators employ a wide variety of geometric representations and algorithmic techniques for a multitude of materials and phenomena. For example, complex fluid motion is often simulated using Eulerian methods, while Lagrangian methods are preferred for solids. And while mesh-based methods are effective for modeling the elastic deformation of solid materials, collisions between objects may be efficiently processed in a particle-based fashion. Simulations of materials such as rigid bodies, deformable solids, cloth, smoke and free surface flows have achieved impressive results using such disparate techniques.

The physically-based simulation of coupled interactions between different materials is also of great interest with numerous applications in science, engineering, medicine, and entertainment. In developing such a system, it is desirable to use the most advantageous representations and techniques for each individual material or phenomenon rather than choose a least common denominator and sacrifice quality. The advantages of heterogeneity, combined with the complexity of individual systems or even lack of access to source code, often lead to a partitioned approach to coupling. In this approach, the individual subsystems are interleaved using the results of the previous step as boundary conditions for the next. The main problem with such an approach is that the straightforward implementation treats the interaction forces explicitly, degrading the stability of the simulation. Various

techniques can be employed to improve the straightforward treatment including predictor steps and specialized stabilization techniques. Although in theory the subsystems can be iterated to achieve fully implicit coupling, this is often intractable, and in practice it can be difficult to devise partitioned approaches that have good stability properties for highly complex interactions. An alternative to the partitioned approach is the strongly coupled or monolithic approach where the equations for the various materials are combined into one system and solved simultaneously allowing for fully implicit integration of interaction forces. The increased stability this provides often enables significantly larger time steps, resulting in increased efficiency and a broader variety of phenomena that can be tractably simulated. The challenge in this approach lies in maintaining the desirable representations and techniques for each individual material while unifying their potentially disparate time integration methods.

This thesis presents several methods for the simulation of coupled phenomena emphasizing a unified approach and implicit treatment of coupling forces. In Chapter 2, an embedding framework is developed which allows for multiple geometric representations of deformable solids taylored to different aspects of the computation. This framework has application in both rigid/deformable solid coupling as well as in solid/fluid coupling. In particular, although the fluid and solid use different geometric representations, embedded points collocated with fluid variables can be introduced for the solid, allowing coupling forces to be applied to both the fluid and solid at these points. Chapter 3 develops a fully unified approach to the two-way coupling of rigid and deformable bodies, allowing for complex interactions while maintaining important features for the individual phenomena. Chapter 4 treats the geometric problem of the interface representation of multiple fluid regions using a hybrid particle/level set based method, and discusses the simulation of various physical interactions at fluid/fluid boundaries. Finally, Chapter 5 presents a method for two-way coupling of rigid or deformable solids or shells to incompressible fluids. This approach draws inspiration from the rigid body contact problem, introducing an unknown impulse that enforces the contact constraint in a momentum-conserving way. The material presented in this thesis is based on the previously published works [52] and [78] as well the works to appear [73] and [69].

Chapter 2

Hybrid Solids

The hybrid solids simulation framework introduces the concepts of hard and soft bindings and first appeared in [78]. Hard bindings are used to embed a point in a simulation mesh, rigid body, or point cloud. The embedded point is fully constrained based on its embedding, and forces or impulses applied to it are propagated to the degrees of freedom of the simulation in a physically consistent way. Soft bindings relax the hard constraint, transforming the embedded particle into a true degree of freedom while targeting an underlying state. The framework is shown to facilitate a variety of phenomena, including simulation of meshes with T-junctions, dynamic point sampling for improved collision resolution, resolution of nonphysical conflicting constraints, subelement elasticity and plasticity, and methods for fracture and cutting. We furthermore extend the framework to enable the implicit integration of forces between rigid and deformable bodies. We also note that the hard binding principles are used in coupling an Eulerian fluid to a Lagrangian solid in Chapter 5.

2.1 Introduction

Mesh-based methods require the generation of an initial simulation mesh, which can be conforming (e.g. [1, 56]), or non-conforming when used in conjunction with an embedded simulation technique (e.g. [11, 12, 55, 57, 61]) in which case a simple cube or BCC background mesh can be used. Generation of a conforming simulation mesh is typically non-trivial and is best suited to applications that require no changes to the initial mesh. However, if the mesh needs to be adapted on the fly, e.g. for fracture [63], remeshing can be prohibitively expensive and can introduce poor quality elements. [55] proposed a partial solution that uses embedded simulation technology to fracture tessellated objects without continuous remeshing, allowing elements to be modeled as partially full of material. Limitations, including the restriction that edges cannot be fractured twice, prevent this method from being completely general. Collision processing can often require more surface resolution than is present in the initial simulation mesh. While some have considered adaptive frameworks [12,17,31], it is wasteful to refine the full volumetric mesh in the vicinity of the boundaries solely for collisions. Although not as efficient as mesh-based methods for the simulation of elastic deformation, point-based methods provide added flexibility making them attractive for applications involving fracture, virtual surgery, resampling for collision handling, etc., see e.g. [58, 59, 66, 81, 88].

Starting with either a conforming or non-conforming mesh, we propose a method for embedding an arbitrary point in this mesh. We call this a *hard binding*. To derive the relationships between physical quantities of the embedded point and the mesh, we start by considering a hypothetical refinement of the mesh that resolves the embedded point. We compute internal finite element forces for these hypothetical subelements and illustrate how to redistribute these forces to the parent mesh. The mass of the embedded particle is redistributed to the parent mesh as well. This redistribution is shown to be independent of the chosen refinement. Notably, this approach results in automatic and natural handling of T-junctions, as masses and forces of T-junction nodes can be redistributed to the parent mesh. This specific handling of T-junctions is similar to that in [53]. We then generalize this approach to arbitrary embeddings. Hard bound particles have their mass and any forces

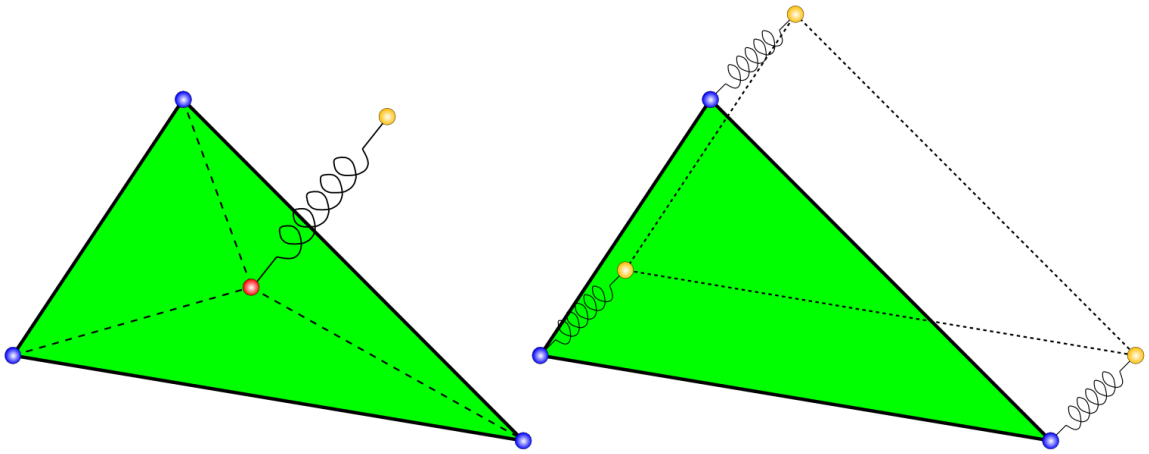


Figure 2.1: Parent particles (blue), hard bound target locations (red) and soft bound particles (yellow). Soft bound duplication of a hard bound target location (left), and three soft bound duplications of parent particles (right).

or impulses applied to them redistributed to the parents in a natural fashion while maintaining the notion of an *effective mass* so that they can fully participate in various numerical algorithms.

A hard binding constrains an embedded particle to its barycentrically determined location (a similar heuristic was given in [28]). Thus, hard bound particles are not particles at all (i.e. they do not possess any degrees of freedom), but merely target locations that live on the parent mesh. In order to transition to a fully particle-centric simulation framework, we create the notion of a *soft binding*. A soft binding is an abstract connection between a real particle with full degrees of freedom and a hard binding target location enabling full two-way interaction. Soft bound particles are free to interact with each other and the parent mesh constituting a fully particle-centric framework. Soft bindings can be created between any particle and any target location even duplicating the original degrees of freedom in the mesh itself (see Figure 2.1). The soft binding mechanism is responsible for the two-way interaction between the particle-based system and the mesh-based framework. Although one could implement this connection using simple springs, we have designed a more sophisticated soft binding interface which is notably fully implicit allowing one to stiffen the two-way interaction to the point where the soft bound particle always lies on its

target location up to numerical precision, without stability issues or additional time step restrictions.

Novel contributions of our work include the tight integration of hard binding constraints and unconditionally stable binding spring forces into the semi-implicit Newmark time integration scheme, lag-free duplication of degrees of freedom through soft bindings by force transfer from parent particles, and integration of rigid/deformable coupling into the Newmark scheme and conjugate gradient solver. We present these contributions as part of a broad hybrid simulation framework building on simple, physically motivated principles. We demonstrate the features of this framework with examples that include dynamically adapting the surface sampling density for collisions, duplicating parent mesh particles to resolve conflicting constraints caused by the nonphysical pinching of cloth, the facilitation of fracture and cutting algorithms, and an extension of our framework to two-way interactions with rigid bodies.

2.2 Hard Bindings

The simulation mesh is subject to internal forces, from e.g. finite elements, as well as external forces from gravity, friction, collisions, etc., and we want these forces to act on the hard bound particles as well. We motivate our approach for propagating forces to the parent mesh by considering a refinement that resolves all the hard bound particles. Figure 2.2 shows three hard bound particles along with an associated refinement. Although this refinement is not unique, it turns out that the propagation of physical properties from the hard bound particles to the parent mesh is independent of the tessellation used and can be performed without explicitly refining the parent element. Internal forces are defined on the subelements in standard fashion, but must be mapped to the parent particles since the hard bound particles are not free to move independently.

Conservative forces can be defined in terms of the gradient $\mathbf{f} = -\partial\Psi(\mathbf{x})/\partial\mathbf{x}$ of the potential energy $\Psi(\mathbf{x})$. The potential energy of the parent triangle is $\Psi = \sum\Psi_i$, where Ψ_i is the potential energy of subtriangle t_i. Each Ψ_i is naturally defined in terms of the positions

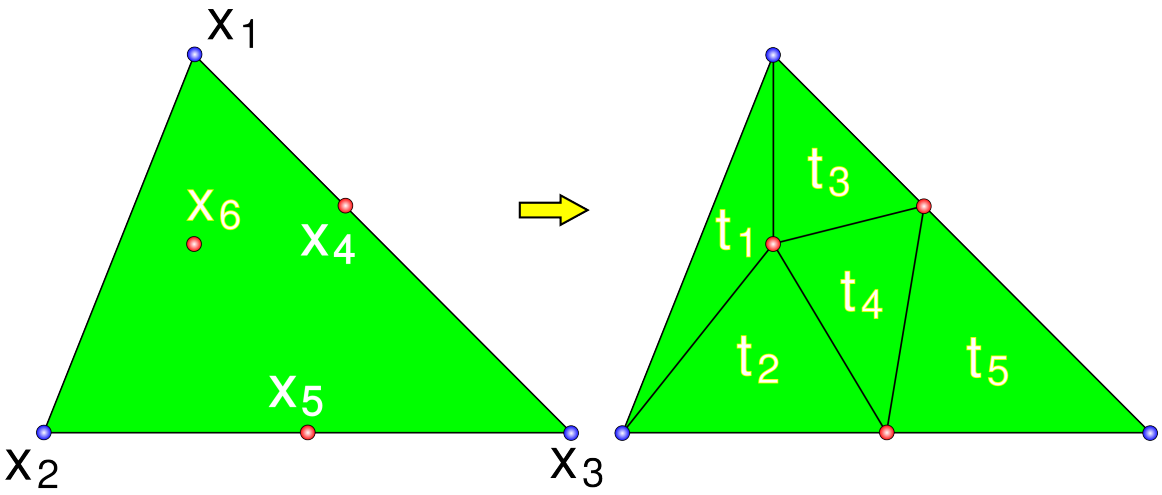


Figure 2.2: Resolution of hard bound particles (depicted in red) via refinement of the parent element.

of the vertices of triangle t_i rather than the positions of the parent particles, although the former are fully determined by the latter through the binding constraint. Let $\mathbf{x}_i$ denote the positions of the vertices of triangle t_i, e.g. $\mathbf{x}_2$ is composed of the positions of particles $\{x_2, x_5, x_6\}$ in Figure 2.2. The parent triangle vertices $\{x_1, x_2, x_3\}$ are represented by $\mathbf{x}$. Using this notation, the three forces on the parent particles are

$$\mathbf{f} = -\sum_i \frac{\partial \Psi_i}{\partial \mathbf{x}} = -\sum_i \left(\frac{\partial \mathbf{x}_i}{\partial \mathbf{x}}\right)^T \frac{\partial \Psi_i}{\partial \mathbf{x}_i} = \sum_i \left(\frac{\partial \mathbf{x}_i}{\partial \mathbf{x}}\right)^T \mathbf{f}_i \tag{2.1}$$

where $\mathbf{f}_i = -\partial \Psi_i / \partial \mathbf{x}_i$ are the three vertex forces computed on each subtriangle t_i. The Jacobian $\partial \mathbf{x}_i / \partial \mathbf{x}$ is constant for each triangle t_i and contains the barycentric coordinates of $\mathbf{x}_i$ with respect to the vertices $\mathbf{x}$ of the parent triangle. Equation (2.1) can be extended to the computation of elastic force differentials for implicit or quasistatic time integration via

$$\delta \mathbf{f}|_{\delta \mathbf{x}} = \sum_i \left(\frac{\partial \mathbf{x}_i}{\partial \mathbf{x}}\right)^T \delta \mathbf{f}_i|_{\delta \mathbf{x}_i} \tag{2.2}$$

Equation (2.2) is obtained by taking the directional derivative of (2.1), assuming that the Jacobian $\partial \mathbf{x}_i / \partial \mathbf{x}$ is constant (i.e. the binding constraint is linear), which is always the case for the barycentric embeddings used here. Nonlinear binding constraints would result in

additional terms in Equation (2.2).

From equations (2.1) and (2.2) we can write the net force and force differential on a single parent particle as

$$f_k = \sum_{t_i} \sum_{x_j \in t_i} w_k^j f_j^i \quad \text{and} \quad \delta f_k|_{\delta \mathbf{x}} = \sum_{t_i} \sum_{x_j \in t_i} w_k^j \, \delta f_j^i|_{\delta \mathbf{x}_i} \tag{2.3}$$

where w_k^j is the barycentric weight of particle j with respect to particle k of the parent triangle and f_j^i is the force on particle j from subtriangle t_i. In practice we implement Equation (2.3) by accumulating all forces on both parent and hard bound particles from all their incident elements and subsequently distributing the force on each hard bound particle to its parents weighted by its barycentric weights. Note that the final force distribution does not depend on the tessellation used but only on the barycentric weights of the hard bound particles. We also note that an alternate derivation is possible using virtual work as in Section 5.3.2.

We use Equation (2.3) for velocity dependent damping forces as well noting that it preserves the symmetry and definiteness of the linear damping forces allowing for our semi-implicit time integration framework. We consider the linear damping model $\bar{\mathbf{f}} = \bar{\mathbf{G}}\bar{\mathbf{v}}$ where $\bar{\mathbf{G}}$ is symmetric negative definite, and the force $\bar{\mathbf{f}}$ and velocity $\bar{\mathbf{v}}$ refer to *all* particles, including hard bound particles. The velocities of this extended set of particles are expressed in terms of the velocities of the parent particles, $\bar{\mathbf{v}} = \mathbf{W}\mathbf{v}$. Using this notation, Equation (2.1) reduces to $\mathbf{f} = \mathbf{W}^T\bar{\mathbf{f}} = \mathbf{W}^T\bar{\mathbf{G}}\mathbf{W}\mathbf{v} = \mathbf{G}\mathbf{v}$ where $\mathbf{G} = \mathbf{W}^T\bar{\mathbf{G}}\mathbf{W}$ preserves both symmetry and negative definiteness of the original damping matrix $\bar{\mathbf{G}}$. A similar derivation shows that the distribution scheme for force differentials given in equation (2.3) preserves symmetry and negative definiteness of the elastic stiffness matrix, which is an important property for schemes that use an implicit or quasistatic treatment of elastic forces as well.

After forces have been computed and distributed to the parent particles, accelerations are computed via $\mathbf{a} = \mathbf{M}^{-1}\mathbf{f}$ where $\mathbf{M}$ is diagonal in a typical lumped mass formulation. We store the lumped masses of the parent particles in a vector $\mathbf{m}$, which is the diagonal of $\mathbf{M}$. In analogy to our force derivation, we write the mass vector as the gradient of total momentum

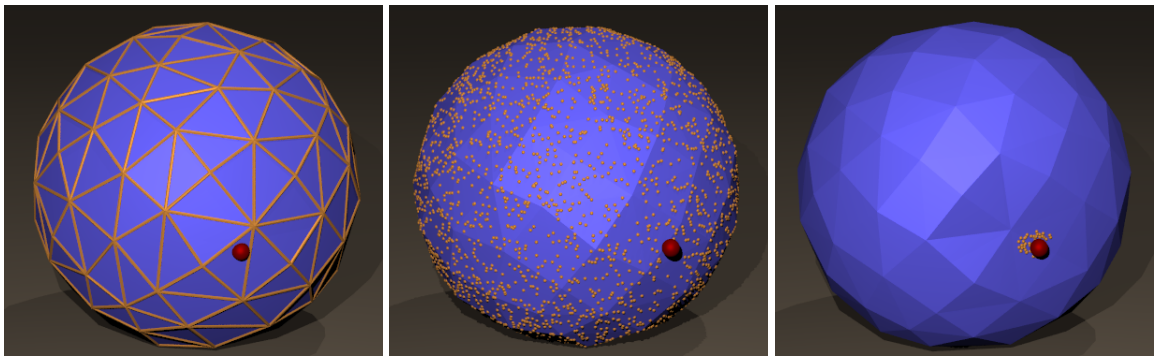


Figure 2.3: (Left) A coarse deformable ball is penetrated by a kinematic sphere. (Center) Sprinkling hard bound particles on the surface of the sphere allows one to resolve the collisions without requiring refinement of the tetrahedral simulation mesh. (Right) Hard bound particles can be dynamically added and removed based on proximity to collision objects.

with respect to velocity, i.e. $\mathbf{m} = \partial P / \partial \mathbf{v}$. Let $\bar{\mathbf{m}}$ be the masses of all particles. As is typical, one could compute the initial mass of hard bound particles by refining the parent mesh as in Figure 2.2 and assigning one third the mass of each triangle to its vertices, although any scheme for assigning mass (including uniform mass) is allowed. The total momentum is then given as $P = \bar{\mathbf{m}}^T \bar{\mathbf{v}}$, thus $\mathbf{m} = \frac{\partial}{\partial \mathbf{v}} P = \frac{\partial}{\partial \mathbf{v}} \left(\bar{\mathbf{m}}^T \bar{\mathbf{v}} \right) = \frac{\partial}{\partial \mathbf{v}} \left(\bar{\mathbf{m}}^T \mathbf{W} \mathbf{v} \right) = \mathbf{W}^T \bar{\mathbf{m}}$ indicating that masses of hard bound particles should be redistributed to the parents based on the barycentric weights, i.e. the same as for force redistribution. More generally in a non-lumped mass formulation, the mass matrix is given by the Hessian of the kinetic energy as $\mathbf{M} = \mathbf{W}^T \bar{\mathbf{M}} \mathbf{W}$ (see [74]).

Although hard bound particles have no mass or momentum, an *effective mass* is useful for many numerical algorithms. We define the effective mass as the ratio of an applied force to the resultant acceleration of the hard bound particle, i.e. $f_e = m_e a_e$. Denoting the binding weights by w_i, the applied force is distributed to the parent particles via $f_i = w_i f_e$ and the acceleration of the bound particle is $a_e = \sum w_i a_i$. Combining these equations gives

$$\frac{f_e}{m_e} = a_e = \sum w_i a_i = \sum w_i \frac{f_i}{m_i} = f_e \sum \frac{w_i^2}{m_i} \Rightarrow \frac{1}{m_e} = \sum \frac{w_i^2}{m_i} \tag{2.4}$$

The effective mass is used in the determination of stability restrictions for forces defined on

the hard bound particle, e.g. if a hard bound particle is connected to a spring, the effective mass would be used to compute the harmonic mass.

One often needs to apply impulses to hard bound particles, e.g. for collisions. An impulse j_e applied to a hard bound particle is the result of a force f_e acting on the particle for an infinitesimal interval Δt, i.e. $j_e = f_e \Delta t$, and from Equation (2.3) we obtain $j_i = w_i j_e$. Furthermore, changes in velocity are governed by $\Delta v_i = j_i/m_i = w_i(m_e/m_i)\Delta v_e$, and displacements follow $\Delta x_i = w_i(m_e/m_i)\Delta x_e$ assuming that $\Delta x_e = \Delta t v_e$.

Figure 2.3 illustrates the utility of hard bindings in processing collisions. A typical approach to collision processing (due to its simplicity) is to collide the points (possibly only the surface points) of a deformable object with implicitly represented collision geometry, and we collide each tetrahedral mesh node from the sphere's surface with the ground and the kinematically controlled red sphere. While ground collisions are sufficiently resolved, the kinematically controlled red sphere passes directly through the deformable object, missing any potential collisions with surface particles. Although one might adaptively refine tetrahedra near the colliding red sphere, this increases the total tetrahedra that need to be simulated and the smaller edge lengths lead to a stiffer time step restriction. Our hard binding framework allows us to simply sprinkle particles on the surface of the sphere at a higher density than the tetrahedral mesh for the sake of collisions. This can be done statically or even adaptively based on proximity to collision objects.

2.2.1 T-Junctions

Figure 2.4 (left) depicts four coarse triangles undergoing up to two levels of non-graded red refinement leading to a number of T-junctions. Structured adaptive meshing schemes such as [56] resolve these T-junctions via a combination of red and green refinement. This produces both more elements increasing computational cost as well as lower quality green elements which adversely affect the time step restriction. Additional savings may be realized by exploiting the regularity of adaptive red-only refinements where all elements in

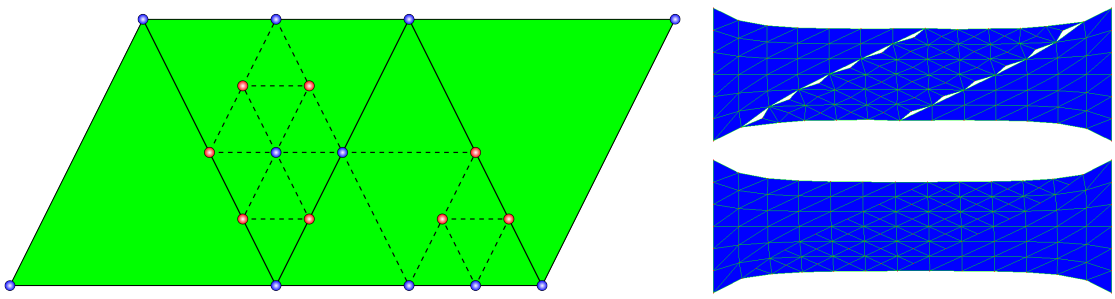


Figure 2.4: Left: Two levels of non-graded red refinement leading to T-junctions (red). The T-junctions do not have a complete one-ring of triangles. Our framework treats the T-junction nodes as hard bound particles (red) embedded in the parent mesh (blue). Right: An elastic sheet meshed with T-junctions is quasistatically stretched without bindings (top) and using hard bindings (bottom).

the mesh are identical up to rotation and scaling requiring only the storage of a scale factor per element to encode the rest state. For every T-junction, we compute its parents by recursively tracking the endpoints of refined segments. This leads to the barycentric embedding of hard bound particles on segments or triangles in two spatial dimensions, and on segments, triangles or tetrahedra in three spatial dimensions. Figure 2.4 (right) illustrates that the straightforward simulation of a T-junction mesh leads to gaps in the mesh (top), while treating T-junction particles as hard bound particles properly constrains them to their incident edges (bottom).

2.2.2 Arbitrary Embeddings

Although our framework is described in the context of a parent simulation mesh outfitted with a number of embedded particles that will be extended into a fully particle-centric simulation framework (in section 2.3), here we give an example to illustrate that our framework is more general than this. That is, we switch from the notion of a parent simulation mesh with an embedded set of particles to a parent point cloud simulation system with an embedded mesh. In the context of free-form deformations, one might use a nearest neighbor approach to compute a mesh-free discretization of internal forces on the point cloud, and

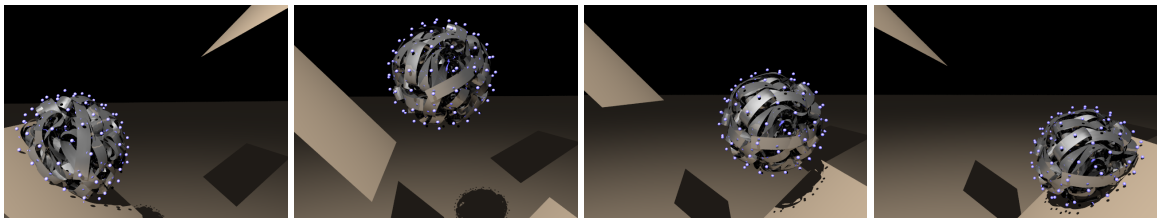


Figure 2.5: A ribbon is kinematically embedded in a point cloud. Instead of using a standard mesh-free discretization of the point cloud, we endow the ribbon with a constitutive model which is used to compute elastic forces that are subsequently mapped to the point cloud. The first approach penalizes point cloud deformation and has no direct knowledge of deformation artifacts within the ribbon, while our approach directly penalizes distortions in the ribbon (2.3 sec/frame).

subsequently move the embedded mesh kinematically. Of course, this can lead to potentially severe distortions of the embedded mesh, especially since the point cloud sampling of deformation is very different from that perceived by the embedded mesh. This can be alleviated by computing the internal forces on the embedded mesh itself, as we do in the elastic ribbon shown in figure 2.5, and then mapping the resultant forces to the point cloud. Using these forces, we still time integrate the point cloud particles and kinematically enslave the embedded mesh, but the deformation of space is now sampled more adequately for the purpose of simulating the embedded mesh. Note that the kinematic motion of the ribbon is dictated by the k closest parent particles of the point cloud.

2.3 Soft Bindings

The hard bound particle framework is limited because the hard bound particles are kinematically constrained to the parent mesh rather than possessing real degrees of freedom. Thus, specialized algorithms would be required for collision processing, etc. We overcome these issues by introducing the notion of a *soft binding* which couples a new, fully simulation capable particle with an existing parent particle or hard bound particle target location as illustrated in Figure 2.1. We can then ignore hard bound particles for the purpose of collisions and apply collision processing uniformly to parent particles and soft bound

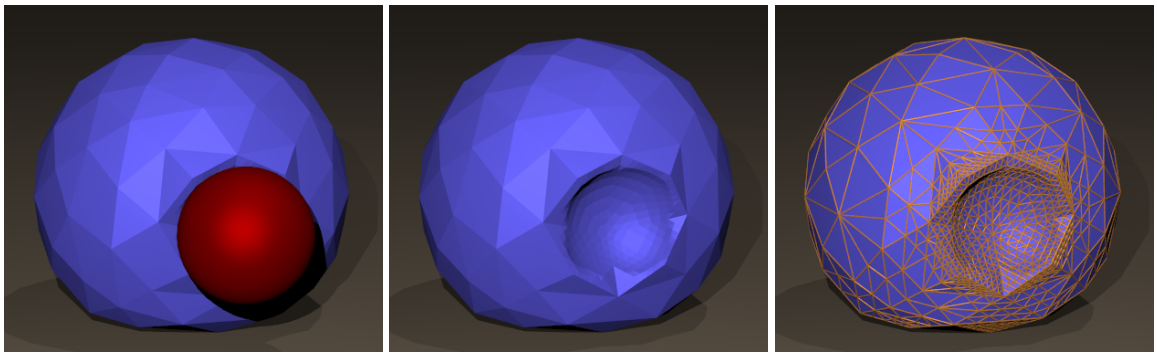


Figure 2.6: (Left) A coarse deformable ball collides with a kinematic sphere. (Middle) Soft bindings enable subtetrahedron elasticity in response to the collision. (Right) The dynamically refined surface mesh.

particles.

Under no external influence such as collisions or soft bound particle intrinsic forces, soft bound particles should follow their target positions. This is achieved by applying the force that leads to a matching acceleration between the soft bound particle and its target, i.e. the soft bound particles *inherit* elasticity and similar forces from the parent mesh. If the acceleration of a hard bound particle is $a_e = \sum w_i a_i = \sum w_i (f_i/m_i)$, the force that is applied to the soft bound particle to match this acceleration is $f_s = m_s \sum w_i (f_i/m_i)$ where m_s is the mass of the soft bound particle. Note that we only use this process to map elastic forces, since applying it to damping forces would compromise the symmetry of the damping matrix in our implicit time integration scheme.

The mass of each soft bound particle is set to the *effective mass* of its target (which is just the mass for parent particles). This duplication of mass does not change the effective total mass of the object as measured by applied forces, because our mapping of forces from the parent mesh to the soft bound particles properly accelerates those particles. However, a mass assignment stemming from a consistent physical principle would be desirable.

The coupling strategy between soft bound particles and their targets depends on whether short term or long term drift is desired. Therefore we give separate algorithms for short term and long term drift below.

2.3.1 Synchronized Coupling

The most straightforward way to transfer information from the target particle to the soft bound particle is by directly copying the target state. At first glance, this appears to nullify the desired properties of soft bound particles effectively reducing them to hard bound particles. However, in contrast with hard bound particles which are kept consistent with their target state at all times, soft bound particles are synchronized to their target state at specific points of the time integration loop allowing drift between two subsequent synchronizations. Apart from collision processing, additional forces may also be used to cause drift in soft bound particles, e.g. a soft bound copy of the surface of a volumetric object could be endowed with shell elastic forces. We subsequently propagate any drift from their target locations back to their parents, prior to the next synchronization of soft bindings with their target state.

To propagate drift from a soft bound particle to its target, we compute the discrepancy in position $\Delta x = x - x_e$ and velocity $\Delta v = v - v_e$, where x_e and v_e are the position and velocity of the (possibly hard bound) target particle. If the target particle is part of the parent mesh these increments are directly applied to its position and velocity, while if it is hard bound we use the formulas in section 2.2 to propagate these increments to the hard bound particle's parent particles.

2.3.2 Coupling Through Binding Springs

The synchronized coupling scheme only allows for short term drift of the soft bound particles limiting their added simulation capabilities. If long term drift is desired, we instead employ a spring-like force between the soft bound particle and its respective target particle (similar in spirit to a PD controller). This *binding spring* force is

$$f^b = -f_e^b = -k(x - x_e) - b(v - v_e). \tag{2.5}$$

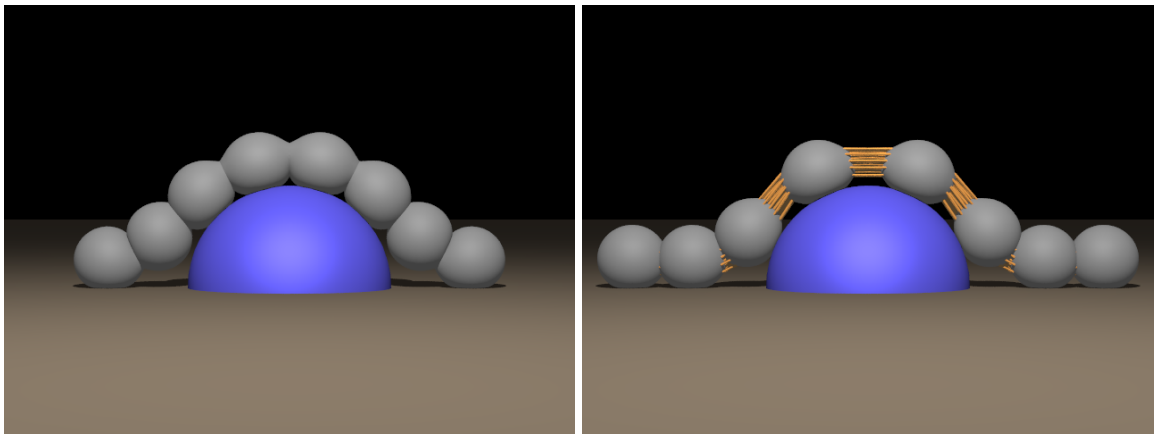


Figure 2.7: (Left) Several spheres stitched together using stiff implicit binding springs. (Right) Lowering the binding spring stiffness leads to drift. The binding springs are depicted as orange cylinders (30 sec/frame).

Note that the binding spring has zero rest length and thus linearizing about the rest state results in a multidirectional (as opposed to the typical directional) damping term. We will capitalize on this fact in our time integration scheme. Binding springs more readily allow for full simulation capabilities in the soft bound particles. One no longer needs to explicitly propagate information from the soft bound particles to the parents or to synchronize the soft bound particles, but rather the soft bound particles and parents are now implicitly coupled in a two-way fashion via these binding springs. We emphasize that the important properties of soft bindings depend on the combination of binding springs with force mapping, and would not be achieved through springs alone. Note that a limitation of our approach is that damping forces are not mapped from parents to soft bound particles (to preserve symmetry). Even in the absence of collisions, this leads to drift that has to be corrected via the binding springs.

In Figure 2.6 a coarse tetrahedralized volume has its surface mesh adaptively and dynamically refined based on proximity to collision geometry, and the soft bound particles present at every vertex of the surface mesh provide for subtetrahedron deformation.

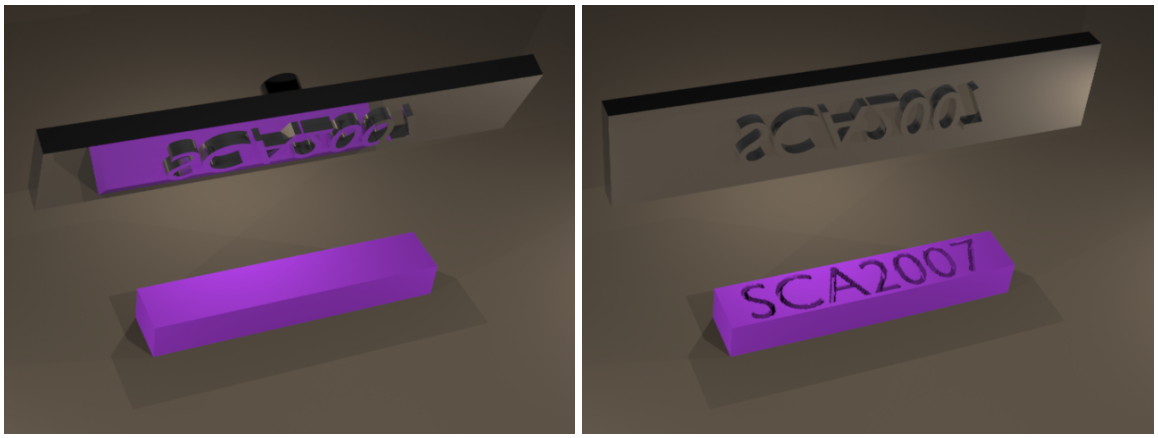


Figure 2.8: Allowing the hard bound target positions to drift, one can model subtetrahedron plasticity resulting from collisions of soft bound particles (3 sec/frame).

2.3.3 Time Integration

Our time integration scheme is a variant of the semi-implicit modified Newmark integration scheme of [42]. The deformable object is evolved from time t^n to t^{n+1} as follows:

1. $\tilde{\mathbf{v}}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^n, \tilde{\mathbf{v}}^{n+\frac{1}{2}})$
2. $\tilde{\mathbf{x}}^{n+1} = \mathbf{x}^n + \Delta t\tilde{\mathbf{v}}^{n+\frac{1}{2}}$
3. Process collisions $(\tilde{\mathbf{x}}^{n+1}, \mathbf{v}_n) \rightarrow (\mathbf{x}^{n+1}, \tilde{\mathbf{v}}_n)$
4. * $\mathbf{v}^{n+1} = \tilde{\mathbf{v}}^n + \frac{\Delta t}{2}\left(\mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^{n+\frac{1}{2}}, \tilde{\mathbf{v}}^n) + \mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^{n+\frac{1}{2}}, \mathbf{v}^{n+1})\right)$

where $\mathbf{x}^{n+1/2} = (\mathbf{x}^n + \mathbf{x}^{n+1})/2$. Our scheme deviates from that of [42] in that the trapezoidal velocity update in step 4^* uses $\mathbf{x}^{n+1/2}$. This substitution retains second order accuracy and allows for fully implicit integration of our binding spring forces. Under the common assumption that the velocity dependent forces are linear, we can rewrite step 4^* as

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

The version of trapezoidal rule in step 4^* can suffer from loss of precision when the two acceleration terms are rather large but of opposite sign, whereas steps 4 and 5 use a robust backward Euler update followed by extrapolation. That is, in the limit of a large time step the first acceleration term in step 4^* pushes a positive coefficient c of an eigenvector toward $-\infty$ while the second acceleration term brings that value back from $-\infty$ to $-c$, possibly failing due to loss of precision. However, in the same large time step limit, step 4 damps c to 0 and step 5 robustly extrapolates c through 0 to $-c$.

The time step restriction imposed by a single binding spring is $\Delta t < (b + \sqrt{b^2 + 4\mu k})/k$ where μ is the reduced mass. As is typical, this drives the time step to zero as the spring constant is increased, and thus we propose a fully implicit treatment of the binding springs leveraging the fact that our binding springs are fully linear in both position and velocity (which is not the case for nonzero rest length springs).

Equation (2.5) can be written in matrix form as $\mathbf{f}^b = -\mathbf{K}\mathbf{x} - \mathbf{B}\mathbf{v}$, where the stiffness and damping matrices $\mathbf{K}$ and $\mathbf{B}$ are symmetric positive semi-definite. We apply our time integration scheme to this force, with the modification that in step 1 we use the half time step position $\tilde{\mathbf{x}}^{n+1/2}$, making this step fully implicit:

$$\begin{aligned} \tilde{\mathbf{v}}^{n+\frac{1}{2}} &= \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}^b(t^{n+\frac{1}{2}}, \tilde{\mathbf{x}}^{n+\frac{1}{2}}, \tilde{\mathbf{v}}^{n+\frac{1}{2}}) && (2.6) \\ &= \mathbf{v}^n + \frac{\Delta t}{2}\left(-\mathbf{M}^{-1}\mathbf{K}\tilde{\mathbf{x}}^{n+\frac{1}{2}} - \mathbf{M}^{-1}\mathbf{B}\tilde{\mathbf{v}}^{n+\frac{1}{2}})\right). && (2.7) \end{aligned}$$

Substituting $\tilde{\mathbf{x}}^{n+1/2} = \frac{1}{2}(\mathbf{x}^n + \tilde{\mathbf{x}}^{n+1}) = \mathbf{x}^n + \frac{\Delta t}{2}\tilde{\mathbf{v}}^{n+1/2}$, where the last equality comes from step 2, gives

$$\left(\mathbf{I} + \frac{\Delta t}{2}\mathbf{M}^{-1}\mathbf{B} + \frac{\Delta t^2}{4}\mathbf{M}^{-1}\mathbf{K}\right)\tilde{\mathbf{v}}^{n+\frac{1}{2}} = \mathbf{v}^n - \frac{\Delta t}{2}\mathbf{M}^{-1}\mathbf{K}\mathbf{x}^n \qquad (2.8)$$

which is a true backward Euler step implicit in both position and velocity with no time step restriction. We emphasize that Equation (2.6) only replaces step 1 in our time integration loop, and step 4 still requires the use of $\mathbf{x}^{n+1/2}$. Our modifications to [42] are crucial for unconditional stability, e.g. using $\mathbf{x}^{n+1}$ instead of $\mathbf{x}^{n+1/2}$ in step 4 leads to a time step restriction of $\Delta t < (b + \sqrt{b^2 + 8\mu k})/2k$ (see [74]). We stress that this implicit treatment of elastic forces is only used on the binding springs and that all other elastic forces are treated

Figure 2.9: Cloth pinched between two spheres. Conflicting constraints are resolved by allowing drift of duplicate surfaces using soft bindings (2.8 min/frame, 150×150 mesh).

explicitly in order to more accurately resolve dynamic motion.

2.4 Results

We used critical damping for the soft binding springs in our examples. The stiffness parameters were set experimentally. Figure 2.7 shows eight deformable spheres stitched together using implicit binding springs as described in section 2.3.2. On the left the binding springs are made sufficiently stiff to retain coincidence of the connecting points without incurring additional time step restrictions. The binding springs are visible on the right where their stiffness is reduced. In Figure 2.8 we begin with a coarse deformable block. We then create a duplicate copy of the top surface, which is refined and hard bound to the block. Next, we use binding springs to attach a soft bound duplicate of this refined surface for collision with a stamp. When collisions stretch these binding springs to a length beyond a prescribed threshold the hard bound target particles are moved in material space (and a new barycentric embedding is computed) to bring the length back down to this threshold, analogous to typical plastic yield. We are careful not to move any hard bound target particle outside the material.

Using a variant of the virtual node algorithm [55], we cut a coarse cube mesh consisting of 320 tetrahedra into 289 sticks as shown in Figure 2.10. Each stick is composed of a number of tetrahedra that are only partially filled with material and the polygonization of the material surface is augmented with soft bindings to resolve self-collisions and object collisions. See [75] which is enabled by our new hybrid simulation framework.

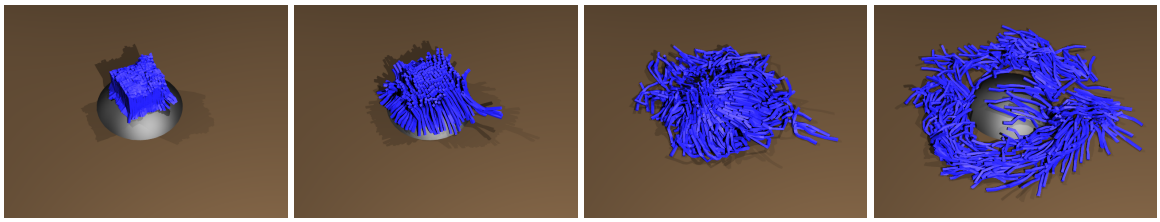


Figure 2.10: A coarse 320 tetrahedra cube mesh is cut into 289 sticks. Object collisions and self-collisions are resolved by the soft bound embedded surface.

Figure 2.9 shows cloth pinched between two kinematically controlled spheres producing contradictory collision constraints (see [5]). For each particle in the cloth mesh that is pinched between the two spheres, we create two duplicate soft bound particles which each collide with only one of the two spheres. Collisions with the sphere pull these soft bound particles away from their hard bound target locations on the cloth surface resulting in binding spring forces that pull the cloth mesh in both directions equilibrating in a steady state. In the absence of any other forces, the soft bound particles would slide along the sphere in an attempt to minimize the length of the underlying binding springs. However, we map the mesh-based constitutive model forces of the cloth to the soft bound particles making them behave in a fashion similar to their hard bound counterparts, i.e. resisting stretching. Figure 2.9 (lower right) is a visualization of the effect of mapping the constitutive model forces from the hard bound target locations to their soft bound counterparts. That is, it is similar to creating a mesh which connects all the drifted soft bound particles to each other and to the parent mesh as depicted schematically by the orange wireframe. In fact, one could envision this as simulating three distinct meshes: the parent mesh given by the cloth itself, the orange wireframe mesh, and a second wireframe mesh which agrees with the bottom half of the orange wireframe but is concave up elsewhere resolving collisions with the top sphere. However, we do not need to construct these duplicate meshes, and instead automatically inherit these properties from the parent simulation mesh.

Figure 2.11 depicts the dynamic simulation of a muscle-driven facial model for speech articulation from [76]. The surface geometry is embedded in a non-graded adaptive red-only BCC mesh with 135K tetrahedral elements, an 85% reduction from the mesh size in [77] enabled by the features of the binding framework. Hard bindings are used to simulate

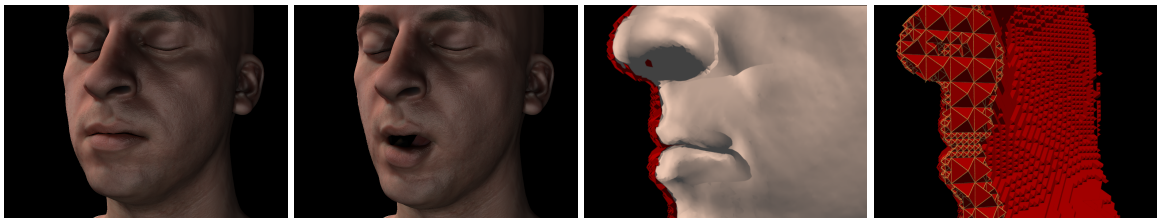


Figure 2.11: An adaptive red-only BCC mesh for embedded simulation of a muscle-driven face model. Hard bindings are used for T-junctions, and soft bindings are used for collisions and boundary conditions (18 min/frame).

the resulting T-junctions, while soft bound particles are used to model the high resolution surface and also used for collisions, self-collisions and boundary conditions. Since the simulation mesh does not conform to the geometry of the cranium and jawbone, we use soft bound particles on the high-resolution surface to enforce bone attachments as well. A higher stiffness was used to attach the embedded free skin surface to its hard bound counterpart, and a lower stiffness was used for the connections between soft-bound bone attachments and their hard bound targets.

2.5 Extensions to Rigid Body Coupling

Various authors have considered the two-way coupling of rigid and deformable bodies using a variety of schemes, see e.g. [3, 43, 50, 64]. We present preliminary results showing that our hybrid framework can be applied to these types of problems as well.

We consider the rigid body frame (x_c, q), twist (v_c, ω) and inertia tensor $I(q) = R(q)I_0R(q)^T$, where I_0 is a diagonal inertia tensor and $R(q)$ is the rotation matrix associated with the current orientation. To facilitate our hybrid formulation, we define a *rigid body particle* with the same state as a rigid body. A rigid body hard binding is defined by embedding a particle onto a fixed object space location r_0. Consequently, the kinematic state of the bound particle is given as $x_e = x_c + r(q)$ and $v_e = v_c + \omega \times r(q)$ where $r(q) = R(q)r_0$ is the world space displacement of the embedded particle from the center of mass of the rigid body. We can write $v_e = W(q)(v_c, \omega)$ with the aid of the interpolation matrix $W(q) = (\mathbb{I} \;\; r(q)^{*T})$ where

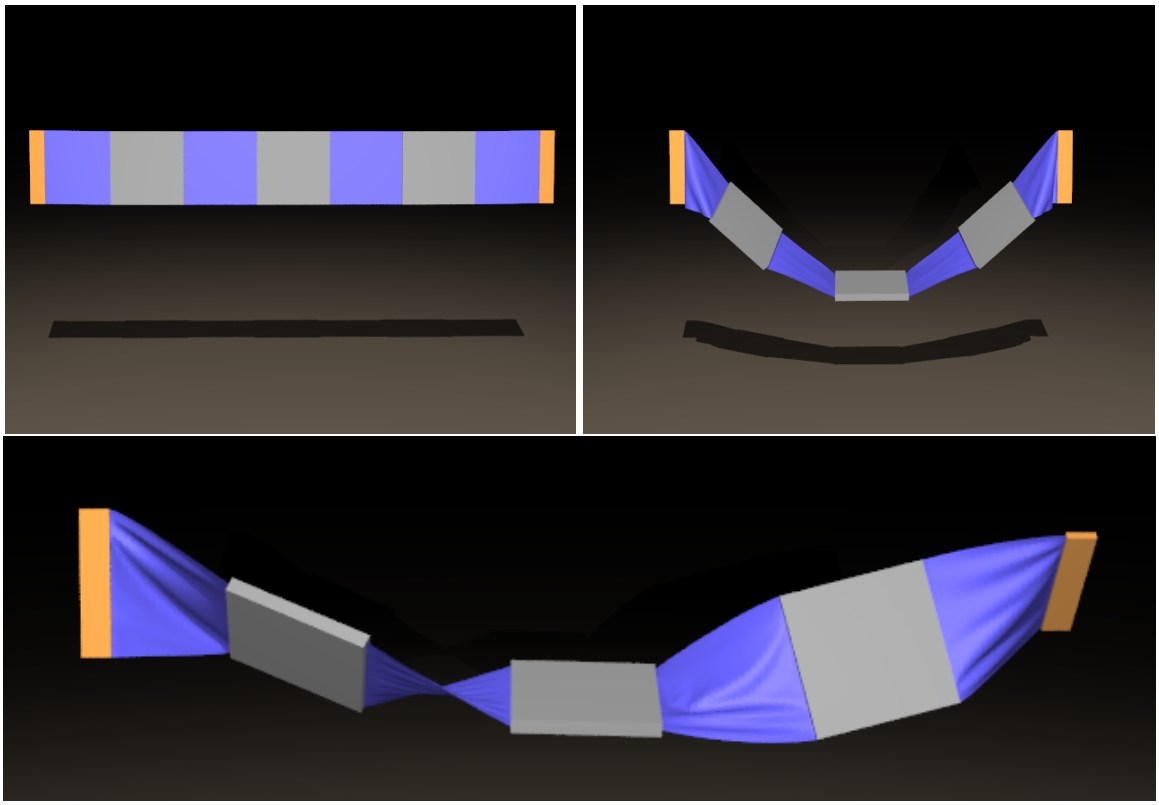


Figure 2.12: Coupled simulation of rigid plates and cloth sheets, where cloth particles have been hard bound to the rigid plates. Each of the three rigid plates as well as the two constrained rigid bodies depicted in orange are represented by single rigid body particles.

$\mathbb{I}$ is the 3×3 identity matrix and $r(q)^*$ is the cross product matrix. Although x_e cannot be written in similar form, a linearized change in the frame of the rigid body particle admits a similar mapping, i.e. $\Delta x_e = W(q)(\Delta x_c, \Delta q)$, where Δx_c is the displacement of the center of mass and Δq is the vector whose cross product matrix is the linearized rotation satisfying $R(q(t+\Delta t)) = R(q(t)) + \Delta q^* + O(\Delta t^2)$.

Applying a force f_e to the hard bound particle results in a wrench $(f_c, \tau) = W(q)^T f_e$ where $f_c = f_e$ is applied to the center of mass and $\tau = r(q) \times f_e$ is the torque. Next consider a velocity-dependent damping force $f_e = G_e v_e$ on a hard bound particle, which results in $(f_c, \tau) = W(q)^T G_e W(q)(v_c, \omega) = G(v_c, \omega)$ where $G = W(q)^T G_e W(q)$ maps twists to wrenches directly and retains the symmetry and definiteness of the damping matrix G_e. Similarly elastic wrench differentials are given by $(\Delta f_c, \Delta\tau) = W(q)^T K_e W(q)(\Delta x_c, \Delta q) =$

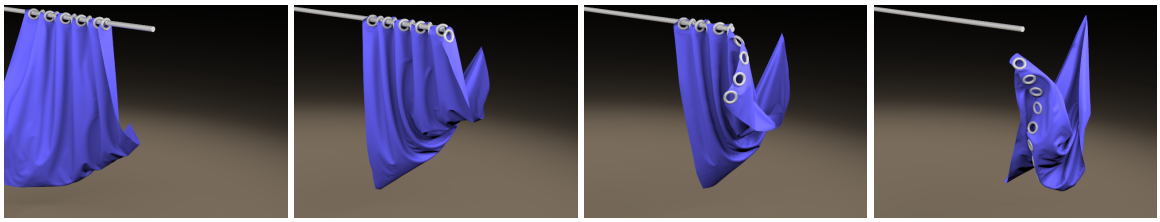


Figure 2.13: Simulation of a cloth curtain with a number of particles hard bound on rigid rings that are used to suspend the cloth from a curtain rod. Each of the ten rings is modeled with a single rigid body particle.

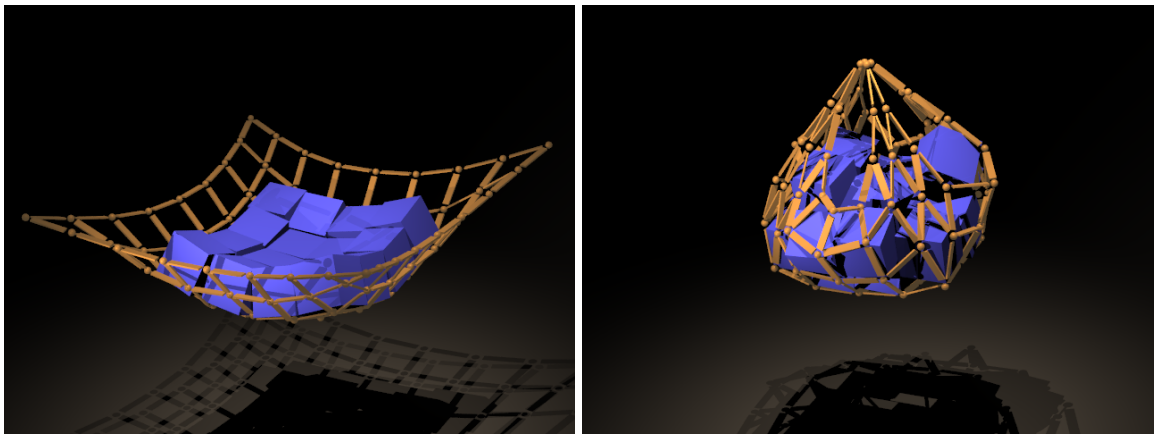


Figure 2.14: A net is constructed using binding springs to create simple point joints between hard bound particles on different rigid bodies.

$K(\Delta x_c, \Delta q)$ where $K = W(q)^T K_e W(q)$ is the stiffness matrix expressed in terms of the rigid body particle and $K_e = \partial f_e / \partial x_e$ is the stiffness matrix expressed in terms of the hard bound particle. We use these results to treat rigid body particles in the same fashion as other parent particles in our time integration scheme. Note that the definition of the global damping matrix $\mathbf{G} = \mathbf{W}^T \bar{\mathbf{G}} \mathbf{W}$ and global stiffness matrix $\mathbf{K} = \mathbf{W}^T \bar{\mathbf{K}} \mathbf{W}$ trivially extends to the case of rigid body bindings, by incorporating the interpolation matrix $W(q)$ into the global matrix $\mathbf{W}$ used in these equations. Finally, the global mass matrix $\mathbf{M}$ is augmented with the diagonal entry $\text{diag}(m\mathbb{I}, I)$ for each rigid body particle.

2.5.1 Coupled Rigid/Deformable Simulation

[78] described a preliminary algorithm for coupling the rigid body particles to a state-of-the-art rigid body simulation system. The scheme interleaves the rigid body time integration with the hybrid deformable solids simulator augmented with rigid body particles as follows:

1. Copy the state of the rigid bodies to the rigid body particles
2. Compute $\tilde{\mathbf{v}}^{n+\frac{1}{2}}$ as in section 2.3.3
3. Compute $\tilde{\mathbf{x}}^{n+1}$ using $\tilde{\mathbf{x}}_{\mathbf{c}}^{n+1} = \mathbf{x}_{\mathbf{c}}^{n} + \Delta t \tilde{\mathbf{v}}_{\mathbf{c}}^{n+\frac{1}{2}}$ and $\tilde{\mathbf{q}}^{n+1} = \mathbf{q}(\Delta t\, \omega^{n+\frac{1}{2}})\mathbf{q}^{n}$
4. Process collisions for deformable objects only
5. Compute $\mathbf{v}^{n+1}$ using steps 4 and 5 from section 2.3.3
6. Copy *momentum only* from the rigid body particles to the rigid bodies
7. Separately, time integrate and collide the rigid bodies

This last step evolves the rigid bodies forward in time using a standard scheme as in [32], and the two-way coupling is integrated into the standard rigid body solver using steps 1 through 6 where only the effects of momentum are preserved. Figure 2.14 shows the use of implicitly integrated binding springs to enforce a simple point joint articulation between rigid body hard bound particles. Figures 2.12 and 2.13 illustrate two-way coupling between cloth and rigid bodies. In Chapter 3, the interleaved rigid/deformable coupling described above is extended to a fully two-way coupled rigid/deformable time integration scheme allowing for more complex scenarios involving contact, stacking, friction, and articulation.

Chapter 3

Two-Way Coupling of Rigid and Deformable Bodies

In Chapter 2, a preliminary coupling of rigid and deformable bodies was demonstrated which interleaves the rigid body and deformable solid simulators. While this approach allows for some two-way interactions, it treats collisions and contact in a one-way fashion and generally does not allow for the two-way interaction of the full capabilities of the rigid and deformable simulators. In this chapter (based on [73]), we develop a framework for the full two-way coupling of rigid and deformable solids, which is achieved with both a unified time integration scheme as well as individual two-way coupled algorithms at each point of that scheme. Thus we do not require specialized methods for dealing with stability issues or interleaving parts of the simulation. We maintain the ability to treat the key desirable aspects of rigid bodies (e.g. contact, collision, stacking, and friction) and deformable bodies (e.g. arbitrary constitutive models, thin shells, and self-collisions). In addition, our simulation framework supports more advanced features such as proportional derivative controlled articulation between rigid bodies. This not only allows for the robust simulation of a number of new phenomena, but also directly lends itself to the design of deformable creatures with proportional derivative controlled articulated rigid skeletons that interact in a life-like way with their environment.

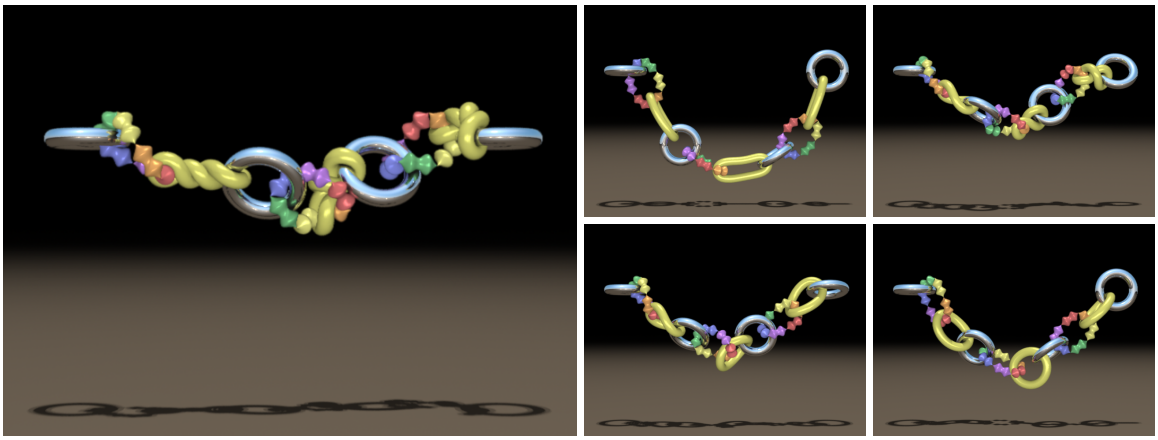


Figure 3.1: Chain composed of both rigid and deformable tori, as well as loops of articulated rigid bodies. The leftmost torus is static, and the rightmost torus is kinematically spun to wind and unwind the chain.

3.1 Introduction

A number of authors have proposed methods for simulating two-way coupling between rigid and deformable bodies, see e.g. [3, 43, 50, 64, 78]. Although coupling deformable and rigid bodies is interesting from the standpoint of simulating new phenomena as demonstrated by those authors, it is also quite interesting from the standpoint of designing creatures. Typically creatures are designed as articulated rigid bodies with some sort of joint or muscle control with notable examples being Luxo Jr. [89], athletes from the 1996 Summer Olympics [35], and the virtual stunt man of [19]. However, creatures are more life-like when their internal skeleton can be used to drive a deformable exterior, such as the tentacles for Davy Jones [16, 85]. The difficulty with wrapping a rigid skeleton with a deformable exterior is that the creature can only interact with its environment if environmental forces deform the exterior, and the deformable exterior subsequently applies forces to the rigid interior. That is, modeling deformable creatures with rigid skeletons requires two-way rigid/deformable interaction [25].

For the two-way coupling of physical phenomena, there are two common approaches. One approach is to start with the best methods for each phenomenon and subsequently design

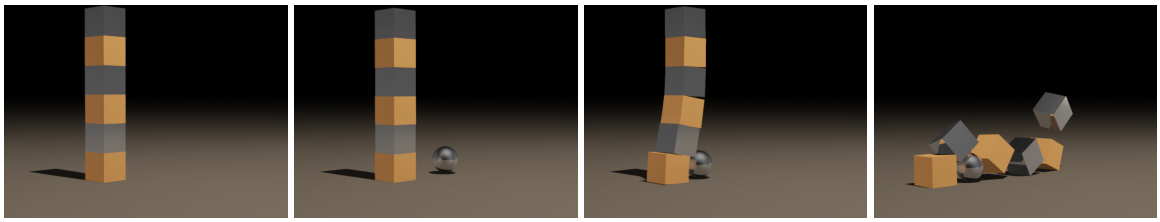


Figure 3.2: Silver rigid boxes and orange deformable boxes are arranged to form a stack, which is knocked down by a rigid ball.

methods for linking these disparate simulation techniques together. Typically interleaving is necessary, where each simulation is run using the results of the previous one in an alternating one-way coupled fashion. This leads to stability issues as one system cannot adequately predict what the other system will do–similar to explicit time integration. Various ad hoc tricks can be applied to increase the stability making the system semi-implicit (e.g. solving for the effect of damping forces between rigid and deformable bodies implicitly, and subsequently mapping the momentum back to the rigid body simulator [78]). Additionally, specialized techniques for increasing the stability of partitioned approaches have been investigated in the computational physics literature (see e.g. [44], [67]). Still, stability issues remain. The other approach is to design a fully two-way coupled system, and there are essentially two ways to accomplish this. One could use the same type of simulation for both types of phenomena (e.g. SPH for both solids and liquids [60] or cloth as an articulated net of rigid bodies [8]). Unfortunately, this typically leads to inferior methods for at least one of the two phenomena being simulated. For example, one could imagine extending the mass-spring simulation framework to the rigid limit and simulating rigid bodies with very stiff springs. We take the alternative approach and fully hybridize the simulation frameworks in a manner that maintains the ability to use the more advanced techniques for each physical phenomenon. That is, we want to retain the ability of our rigid body framework to accurately model contact, collision, stacking, friction, articulation, proportional derivative (PD) control, etc., and our deformable object framework to handle arbitrary constitutive models, finite elements, masses and springs, volumes or thin shells, contact, collision, self-collision, etc. In addition, we would like to maintain stability and avoid ad hoc methods for interleaving the simulations.

Full two-way coupling of state of the art rigid body and deformable object time integration schemes, though seemingly straightforward, is more difficult and subtle than it appears. Robust rigid body systems tend to be sensitive to the time integration scheme, because a minor change in the scheme can, for example, cause blocks to tumble rather than slide down an inclined plane. In contrast, deformable object Newmark schemes are far less sensitive and allow more freedom in the way collisions are processed but utilize separate position and velocity updates as well as implicit solves. Our approach demonstrates that we can fully integrate these disparate methods without losing their individual capabilities. Notably, this can be done by coupling together standard rigid body and standard deformable body simulation systems using our newly proposed techniques instead of requiring one to design a unique simulation system from scratch.

We propose a unified time integration scheme, where the rigid and deformable bodies are integrated forward together in every manner, not only with position and velocity evolution, but such that collision, contact, interpenetration resolution, etc. are all handled at the fine-grained level with fully two-way coupled algorithms. On the rigid body side, we were guided by the time integration scheme proposed by [32], which proposes a clean separation of contact and collision and handles both simple and more complex scenarios ranging from a block sliding down an inclined plane to large-scale stacking behaviors. We also wanted our rigid body simulator to include the effects of articulation and internally torque controlled joints [86, 87]. On the deformable side, we required a Newmark-style algorithm as in [78], which cleanly separates the evolution of position and velocity. The deformable algorithm should at least be implicit on the damping forces as in [9] but could also be fully implicit as in [4]. We propose such a time integration scheme, which evolves every object synchronously using fully coupled algorithms at each step.

We describe in detail the steps of our unified time integration scheme below. We note, however, that the approach does not depend on our particular choice of algorithms for the various subsystems but rather serves as a proof of concept of the benefits of a unified approach. Other choices of specific algorithms, e.g., for contact and collision, can be substituted. Much of what we incorporate is optional, including the incompressibility of [41], the articulation of [87], the PD control of [86], and the bindings of [78]. We include them

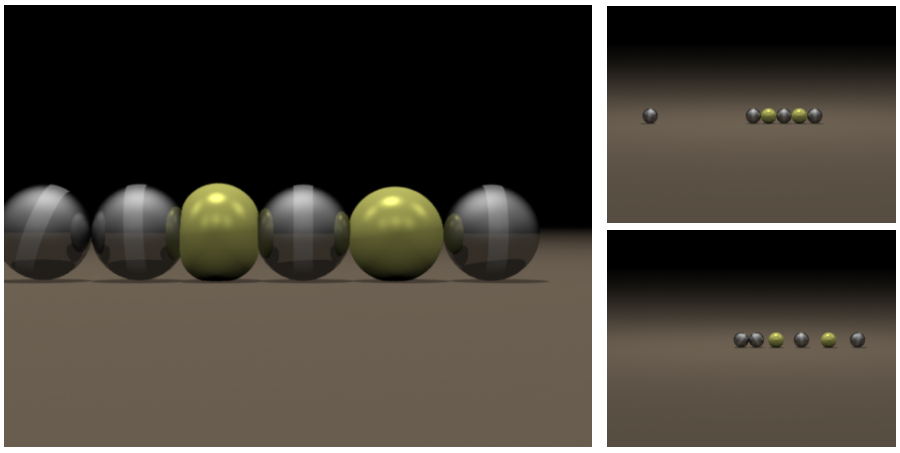


Figure 3.3: A row of silver rigid balls and yellow deformable balls are impacted by a rigid ball. The yellow balls are modeled as deformable and incompressible materials.

to demonstrate the breadth of phenomena that can be incorporated.

3.2 Time Integration

The basic structure of our time integration scheme is that of a Newmark method. Newmark methods are characterized by separate position and velocity updates. In particular, the velocity used in the position update can be distinct from the final velocity, and we take advantage of this to treat the position and velocity updates differently. For example, one might add constraint violating components to the velocity during the position update to correct drift, while projecting out these components in the final velocity update (e.g. as in [41]). Moreover, maintaining the Newmark structure allows us to incorporate a wide range of algorithms from computational mechanics. For example, we can enforce the incompressibility of materials, deal with contact and collisions, self-collisions, friction, etc. For background on the Newmark family of methods see e.g. [38].

We begin by outlining our time integration scheme as composed of five major steps:

I. Advance velocity $\mathbf{v}^n \rightarrow \tilde{\mathbf{v}}^{n+\frac{1}{2}}$

II. Apply collisions $\mathbf{v}^n \rightarrow \hat{\mathbf{v}}^n$, $\tilde{\mathbf{v}}^{n+\frac{1}{2}} \rightarrow \hat{\mathbf{v}}^{n+\frac{1}{2}}$

III. Apply contact and constraint forces $\hat{\mathbf{v}}^{n+\frac{1}{2}} \rightarrow \mathbf{v}^{n+\frac{1}{2}}$

IV. Advance positions $\mathbf{x}^n \rightarrow \mathbf{x}^{n+1}$ using $\mathbf{v}^{n+\frac{1}{2}}$, $\hat{\mathbf{v}}^n \rightarrow \overline{\mathbf{v}}^n$

V. Advance velocity $\overline{\mathbf{v}}^n \rightarrow \mathbf{v}^{n+1}$

Adhering to the time integration scheme proposed in [32], we cleanly separate position and velocity updates from contact and collision. Collisions are done with time t^n velocities so that objects in contact do not bounce, and contacts are performed only on a subset of pairs processed by collisions so that one does not inaccurately apply contact forces to objects before they collide. The algorithmic ordering of collisions followed by contact followed by position updates followed by velocity updates is exactly as in [32] except for the initial advancement of velocities to $\mathbf{v}^{n+1/2}$. In fact, their algorithm would have been more accurate if it had done this, as opposed to using the full Δt for the predictive velocities, obtaining $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^n + \Delta t^2 \mathbf{a}$. [86] took special measures to deal with this inaccuracy, changing parameters to get the desired solution. Using $\mathbf{v}^{n+1/2}$ as in the Newmark scheme automatically alleviates this issue.

Each of the five major steps of the time integration scheme is further described in the five sections that follow.

3.3 Step I: Advance Velocity

Typically the first step in a Newmark iteration scheme is to predict the velocity that will be used to update the positions. We accomplish this as follows:

1. Advance velocities $\mathbf{v}_\star^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^n, \mathbf{v}_\star^{n+\frac{1}{2}})$
2. Apply volume correction $\mathbf{v}_\star^{n+\frac{1}{2}} \rightarrow \mathbf{v}_{\star\star}^{n+\frac{1}{2}}$
3. Apply self-repulsions $\mathbf{v}_{\star\star}^{n+\frac{1}{2}} \rightarrow \tilde{\mathbf{v}}^{n+\frac{1}{2}}$

Figure 3.4: Ten rigid balls and ten deformable tori fall on a cloth trampoline. The trampoline is constructed by embedding the cloth in a kinematically controlled torus, which tosses the objects after they have landed and then allows them to settle.

Here and in the remainder of the chapter we use starred variables to denote intermediate states that do not carry over from section to section. When using semi-implicit time integration with implicit, linear damping forces as in [10], step 1 requires the solution of a linear system, which we solve using the conjugate gradient method. One could instead use the fully implicit method from [4]. In that case step 1 becomes $\mathbf{v}_\star^{n+1/2} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}(t^{n+1/2}, \mathbf{x}^{n+1/2}, \mathbf{v}_\star^{n+1/2})$, where $\mathbf{x}^{n+1/2}$ is replaced with $\mathbf{x}^{n+1/2} = \mathbf{x}^n + \frac{\Delta t}{2}\mathbf{v}_\star^{n+1/2}$. The resulting nonlinear equation is subsequently solved via Newton-Raphson iteration. During the solve, the acceleration in step 1 is projected to satisfy equality constraints such as joint velocity equality constraints, but we avoid applying constraints that may lead to undesirable sticking artifacts at this stage.

If we are simulating incompressible solids, as in Figure 3.3, step 2 adjusts the velocities such that local volume errors will be corrected when positions are advanced [41]. One could also make the velocity field inextensible as in [30]. Step 3 applies self-repulsions as in [9].

While we include steps 2 and 3 here, the only essential part of Step I is to advance the velocity to time $t^{n+1/2}$ as required by the Newmark structure.

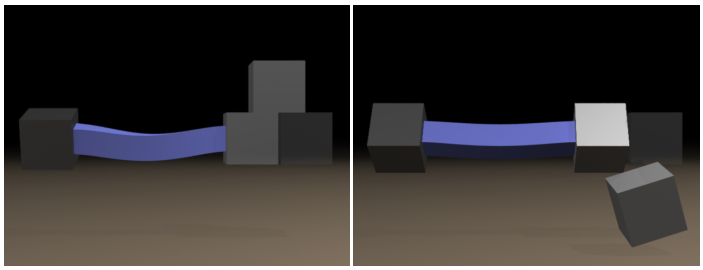


Figure 3.5: The four silver boxes are rigid, and the deformable blue bar is attached to the boxes using the embedding framework of [78]. The far left box is kinematically rotated, causing the bar to twist and subsequently turn the box to its right, which is attached to the static box to the right of it by a twist joint. The free box on top falls off as the bar is twisted.

3.4 Step II: Collisions

Once the time $t^{n+1/2}$ velocities have been computed, we correct them for rigid/rigid, rigid/deformable and deformable/deformable collisions. As in [32], collisions are processed before contact to allow elastically colliding rigid bodies to bounce. The main goal of this step is to adjust velocities for collisions, and other collision algorithms may be used. The steps we use for our collision processing are

1. Process collisions $\mathbf{v}^n \rightarrow \mathbf{v}^n_\star$
2. Apply post-stabilization $\mathbf{v}^n_\star \rightarrow \hat{\mathbf{v}}^n$
3. Re-evolve velocities $\hat{\mathbf{v}}^{n+\frac{1}{2}} = \hat{\mathbf{v}}^n + (\tilde{\mathbf{v}}^{n+\frac{1}{2}} - \mathbf{v}^n)$

For collision detection, all positions are evolved forward in time to look for interferences between pairs of rigid/rigid, rigid/deformable, or deformable/deformable bodies using the update formula $\mathbf{x}^{n+1}_\star = \mathbf{x}^n + \Delta t \tilde{\mathbf{v}}^{n+\frac{1}{2}}$ (see Section 3.4.1 for rigid body orientations). As in [32], collisions are processed using $\mathbf{v}^n$ so that objects in contact do not bounce even when they have nonzero coefficient of restitution. For details on rigid/rigid collisions, see [32]. Rigid/deformable collisions are processed by iteratively colliding each particle of the deformable body against the rigid body. Each rigid/particle pair is processed using the rigid/rigid collision algorithm treating the particle as a rigid body (with an infinite inertia

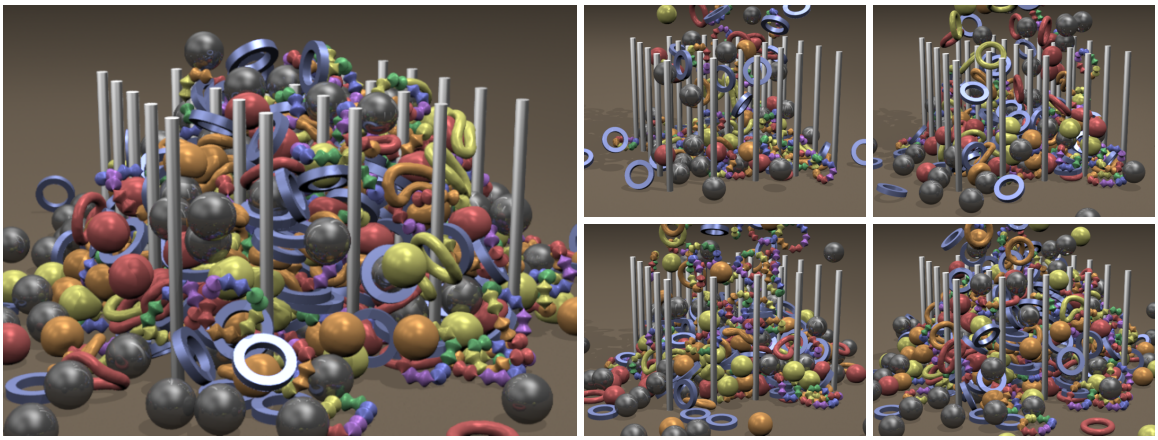


Figure 3.6: 1600 bodies are dropped on a 5×5 grid of static rigid pegs to form a pile. There are 160 colored deformable balls, 160 colored deformable tori, 160 rainbow-colored articulated loops with six rigid bodies each, 160 silver rigid balls and 160 blue rigid rings.

tensor) and a zero coefficient of restitution. For deformable/deformable pairs, one could use tetrahedral collisions as in [82], however we instead use the self-collisions of [9] after the position update as described in Section 3.6.

In Gauss-Seidel fashion, we iteratively process the collisions one pair at a time and re-update the post-collision position of each body via $\mathbf{x}_\star^{n+1} = \mathbf{x}^n + \Delta t(\mathbf{v}_\star^n + (\tilde{\mathbf{v}}^{n+1/2} - \mathbf{v}^n))$ where $\tilde{\mathbf{v}}^{n+1/2} - \mathbf{v}^n$ includes the forces added in step 1 of Section 3.3. If no collision was applied, then $\mathbf{v}_\star^n = \mathbf{v}^n$, and the position is unchanged. Note that the positions computed here are only used to resolve collisions, and are subsequently discarded. We also explicitly save a list of all pairs processed so that contact later considers only those pairs already processed by collisions. This ensures that newly colliding objects will have an opportunity to bounce before being processed for contact. After processing collisions, we apply the post-stabilization algorithm of [87] to ensure that the collision velocities do not violate joint constraints. Finally, step 3 applies the effects of collisions $\hat{\mathbf{v}}^n - \mathbf{v}^n$ to the time $t^{n+1/2}$ velocities $\tilde{\mathbf{v}}^{n+1/2}$.

3.4.1 Second Order Rigid Body Evolution

Given a rigid body in a certain orientation with a certain kinetic energy and angular momentum, the number of states the rigid body is allowed to evolve to is limited by conservation of both energy and momentum. Not every orientation will conserve both quantities, and typical simulators only conserve momentum, allowing kinetic energy to rise or fall. [87] used the first order update $q^{n+1} = \hat{q}(\Delta t\,\omega)q^n$ where $\hat{q} = (\cos(|\omega|/2), \sin(|\omega|/2)\omega/|\omega|)$ is the unit quaternion which rotates by $|\omega|$ around the vector ω. To reduce errors in kinetic energy, we instead propose using the second order update $q^{n+1} = \hat{q}(\Delta t\,\omega + (1/2)\Delta t^2 I^{-1}(L \times \omega))q^n$. A straightforward Taylor expansion can be used to verify that this formula is second order accurate, and we note that it essentially uses a time $t^{n+1/2}$ angular velocity ω similar to Newmark methods. The Taylor expansion is simplified by observing that $\hat{q}(\omega)$ is equivalent to e^{ω^*}, where ω^* is the cross product matrix of ω.

3.5 Step III: Contact and Constraint Forces

The main purpose of this section is contact and contraint processing. Articulation and PD are optional, and other contact processing algorithms could be substituted. The steps used for the contact and constraint forces stage are

1. Compute contact graph using $\hat{\mathbf{v}}^n$ and $\hat{\mathbf{v}}^{n+\frac{1}{2}}$
2. Apply PD to velocities $\hat{\mathbf{v}}^{n+\frac{1}{2}} \rightarrow \mathbf{v}_\star^{n+\frac{1}{2}}$
3. Apply contact and pre-stabilization $\mathbf{v}_\star^{n+\frac{1}{2}} \rightarrow \mathbf{v}^{n+\frac{1}{2}}$

We compute a contact graph for the rigid bodies similar to [32] which also includes the order in which articulated rigid bodies will be processed for pre-stabilization as in [87]. We also apply PD control, see [86]. Proceeding in the order determined by the contact graph, each rigid body is processed in turn for contact with all rigid and deformable bodies it interpenetrates. For rigid/rigid pairs, we perform contact as in [32]. Rigid/deformable pairs are treated in the same manner as they were treated for collisions, since we always

treat deformable body collisions inelastically. This process is iterated as in [32]. If one were using tetrahedral collisions for deformable/deformable pairs [82], they would also be processed at this stage. We instead apply the self-collisions of [9] after the position update as described in Section 3.6 to resolve deformable/deformable contact. The stack in Figure 3.2 demonstrates the effectiveness of the two-way contact algorithm, standing upright until being struck at its base by a rigid ball.

3.5.1 Improved Pre-stabilization

[87] performed pre-stabilization on orientations by solving a quaternion equation of the form $f(\mathbf{j}_\tau) = \mathbf{q}_p(\mathbf{j}_\tau) - \mathbf{q}_c(\mathbf{j}_\tau) = 0$ using Newton-Raphson iteration. To alleviate issues with quaternion subtraction, we instead solve $\mathbf{g}(\mathbf{j}_\tau) = \mathbf{q}_p\mathbf{q}_c^{-1} = (\pm 1, \mathbf{0})$. $\mathbf{g}(\mathbf{j}_\tau)$ consists of both a scalar and vector part, and it turns out to be enough to solve the nonlinear equation that sets the vector part equal to $\mathbf{0}$.

3.6 Step IV: Advance Positions

In the preceeding steps, we made all the desired adjustments to the velocities. The purpose of Step IV is then to evolve the bodies to their final positions, which we do as follows:

1. Advance positions $\hat{\mathbf{x}}^{n+1} = \mathbf{x}^n + \Delta t\, \mathbf{v}^{n+\frac{1}{2}}$
2. Interpenetration resolution $\hat{\mathbf{x}}^{n+1} \rightarrow \mathbf{x}^{n+1}$
3. Post-stabilization $\hat{\mathbf{v}}^n \rightarrow \overline{\mathbf{v}}^n$

Since our rigid/rigid and rigid/deformable contact algorithms do not guarantee that the bodies are completely interpenetration free, we apply an interpenetration resolution method (similar in spirit to [2]), as described in detail below. Finally, since steps 1 and 2 changed rigid body orientations, rigid body angular velocities also changed, and thus post-stabilization of velocities for articulated rigid bodies must be applied.

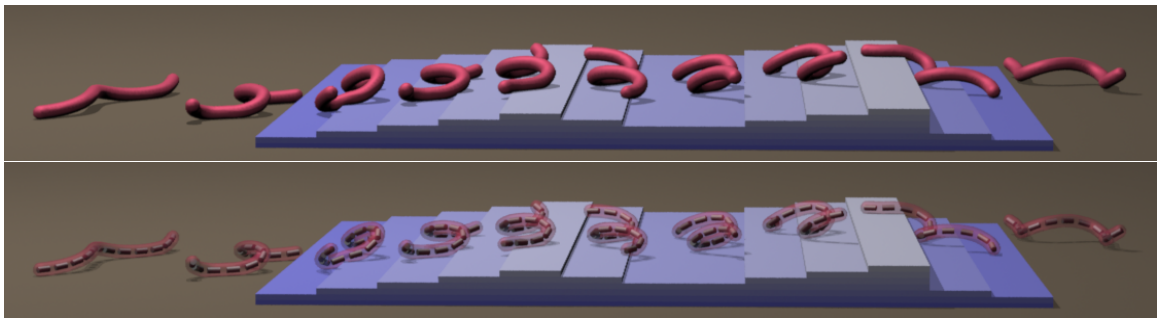


Figure 3.7: A snake constructed by embedding a 12-joint articulated rigid body skeleton in a deformable body. The snake sidewinds up and down stairs using PD-control on its skeleton.

3.6.1 Interpenetration Resolution

One might consider an approach to resolving penetration that operates by iterating pairwise algorithms. This approach suffers from the *messenger problem*, where a large amount of momentum must be exchanged through low-capacity messengers. Consider a row of colliding objects, with the outer two objects being massive and the objects between them being light. The collision must be resolved by moving the massive outer bodies, which requires the exchange of a large impulse. Because the objects in between are light, they are unable to transfer large impulses without obtaining high velocities. Each collision processing step is only able to transfer an impulse across the chain one body at a time, so a large number of iterations are required to complete the momentum transfer.

With this in mind, we propose a novel method to remove small interpenetrations between bodies. Rather than process pairwise as was done for the contact and collision algorithms, we consider one central object (either a rigid body or a particle of a deformable body) at a time and push out every outer object (rigid or deformable) intersecting it simultaneously in a fully two-way coupled fashion. This helps address the messenger problem by allowing a single step to transfer impulses two links at a time instead of one and by avoiding the problem entirely when there is only one such light body. This approach is not easily applied to the full contact and collision problems, primarily because of the complications introduced by friction. The central body c exchanges an impulse $\mathbf{j}_o$ with an outer body o resulting in a

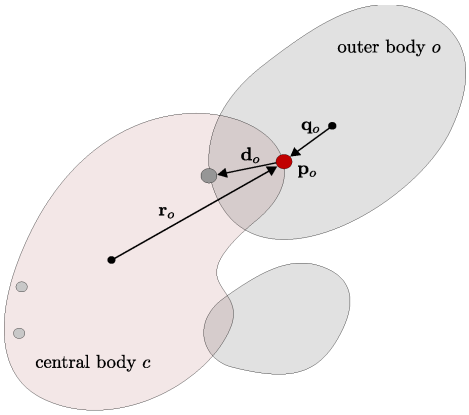


Figure 3.8: One step of the interpenetration resolution algorithm processes a central body along with all the interpenetrating outer bodies.

change in relative velocity at the pair's intersection point of

$$m_c^{-1}\mathbf{j} + \mathbf{r}_o^{*T} I_c^{-1}\mathbf{j}_\tau - K_o\mathbf{j}_o = \mathbf{v}_o \tag{3.1}$$

where $K_o = m_o^{-1}\delta + \mathbf{q}_o{}^T I_o^{-1}\mathbf{q}_o$, $\mathbf{r}_o$ and $\mathbf{q}_o$ are the vectors from the center of masses of body c and body o to the point of application of $\mathbf{j}_o$, respectively, and $\mathbf{j} = -\sum_o \mathbf{j}_o$ and $\mathbf{j}_\tau = -\sum_o \mathbf{r}_o^* \mathbf{j}_o$ are the net linear and angular impulses applied to body c. Solving Equation (3.1) for $\mathbf{j}_o$ and plugging the result into the expressions for $\mathbf{j}$ and $\mathbf{j}_\tau$ gives the symmetric 6×6 system

$$\begin{pmatrix} m_c\delta + \sum_o K_o^{-1} & \sum_o K_o^{-1}\mathbf{r}_o^{*T} \\ \sum_o \mathbf{r}_o^* K_o^{-1} & I_c + \sum_o \mathbf{r}_o^* K_o^{-1}\mathbf{r}_o^{*T} \end{pmatrix} \begin{pmatrix} m_c^{-1}\mathbf{j} \\ I_c^{-1}\mathbf{j}_\tau \end{pmatrix} = \begin{pmatrix} \sum_o K_o^{-1}\mathbf{v}_o \\ \sum_o \mathbf{r}_o^* K_o^{-1}\mathbf{v}_o \end{pmatrix}. \tag{3.2}$$

Once Equation (3.2) is solved for the net impulse $\mathbf{j}$ and $\mathbf{j}_\tau$ on the central body, each $\mathbf{j}_o$ is obtained from substitution into Equation (3.1). To achieve a desired separation distance $\mathbf{d}_o$, we take $\mathbf{v}_o = \mathbf{d}_o/\Delta\tau$ and integrate positions with the velocity change due to the computed impulses for a pseudo-time of $\Delta\tau$. The above procedure is iterated over the bodies and is both more accurate and converges faster than an analogous pairwise method.

Equations (3.1) and (3.2) apply generally to both rigid bodies and deformable particles, with the simplification that $\mathbf{r}_o = \mathbf{0}$ for a central deformable particle (reducing Equation (3.2) to a 3×3 system) and that $\mathbf{q}_o = \mathbf{0}$ for an outer deformable particle. When the central body

is static or kinematic (having infinite inertia) the equations for the outer bodies decouple as the first two terms in Equation (3.1) vanish. When any outer bodies are static or kinematic the system must be decomposed into the degrees of freedom determined by the static bodies and the remaining degrees of freedom corresponding to the nullspace of the equations for the static bodies. Afterwards, the system can be reassembled and solved. For more details see [71].

3.7 Step V: Advance Velocity

The second half of the Newmark time integration scheme is the velocity update, which we do as follows:

1. Make incompressible $\overline{\mathbf{v}}^n \rightarrow \mathbf{v}_\star^n$
2. $\mathbf{v}_\star^{n+1} = \mathbf{v}_\star^n + \Delta t\, \hat{\mathbf{a}}(t^{n+\frac{1}{2}}, \frac{1}{2}(\mathbf{x}^n + \mathbf{x}^{n+1}), \frac{1}{2}(\mathbf{v}_\star^n + \mathbf{v}_\star^{n+1}))$
3. $\mathbf{v}_{\star\star}^{n+1} = \mathbf{v}_\star^n + \Delta t\, \mathbf{a}(t^{n+\frac{1}{2}}, \frac{1}{2}(\mathbf{x}^n + \mathbf{x}^{n+1}), \frac{1}{2}(\mathbf{v}_\star^n + \mathbf{v}_\star^{n+1}))$
4. Apply constraints: post-stabilization, PD control, contact, post-stabilization and self-repulsions $\mathbf{v}_{\star\star}^{n+1} \rightarrow \mathbf{v}^{n+1}$

We project the velocity field for incompressibility before our velocity evolution as in [41]. In step 2, we use a variant of trapezoid rule to solve for the velocities. In practice, we break this step into a backward Euler solve followed by extrapolation as in [78]. We note that one could instead use backward Euler, which introduces more damping. We discuss steps 2-4 in more detail below.

3.7.1 Constrained Solve

The linear system in step 2 is solved using conjugate gradients. We include as many linear constraints as possible in the conjugate gradient solve, accounting for post-stabilization of joints between rigid bodies and two-way contact. This is desirable because it allows

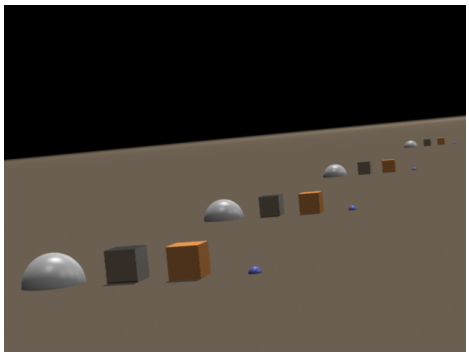


Figure 3.9: Objects sliding down an inclined plane with friction. From left to right: analytic solution, rigid box, deformable box and a single particle.

the effects of constraints to propagate instantly during the implicit solve. Note that $\hat{\mathbf{a}}$ has been used instead of $\mathbf{a}$ in step 2 above to indicate that these forces are modified during the conjugate gradient solve by applying projections to satisfy constraints as in [4].

3.7.2 Final Velocities

As shown in [4], the unprojected residual of the converged system in step 2 gives the net normal force due to all constraints. [4, 72] use the unprojected residual to compute frictional forces due to contact between particles and a single immovable body. They are able to postprocess each particle independently based on the normal force it feels because they enforce contact in a one-way fashion. That is, computing the frictional forces on a particle results in a change in the particle's tangential velocity relative to the body, but does not affect the body's velocity. Thus, the particle/body interactions are decoupled, with the body acting as though it had infinite inertia.

In contrast to [4, 72], the interactions we compute are fully two-way. Thus, we cannot postprocess constrained particles for friction independently after the solve, since these constraints are not decoupled. Furthermore, the residual only yields the net force, so we would not be able to untangle the effects of multiple interactions. Therefore, we propose a novel framework of incorporating the constraints into conjugate gradients only for the purpose of determining damping forces (and elastic forces if they are treated implicitly) and apply

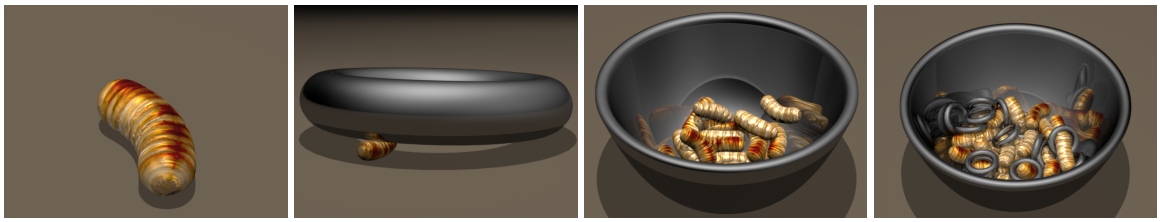


Figure 3.10: Maggots constructed by embedding a two-joint PD-controlled articulated rigid body skeleton into a deformable body. From left to right: a maggot on the ground, a maggot trapped under a large rigid body, 20 maggots dropped into a bowl wriggling and interacting with each other, and 20 maggots interacting with 20 rigid tori.

friction, contact, etc. as a postprocess. Step 3 uses the result of step 2 to compute the acceleration $\mathbf{a}$ as opposed to the projected acceleration $\hat{\mathbf{a}}$. Thus, all of the projections in step 2 only help to determine what velocity $\mathbf{v}_\star^{n+1}$ will be used to evaluate the damping forces in step 3. This approach produces accurate friction as illustrated in Figure 3.9.

An issue in our approach is that the postprocess of the velocities for constraint forces does not allow the damping forces to act on forces applied during the postprocess until the next time step. Thus, one needs to realize that there are undamped velocities, e.g. when computing a CFL condition. In addition, those velocities may get damped based on a smaller time step than the time step in which they were applied. We have nevertheless found that we can work around this by simply moving the postprocesses after applying explicit forces but before the conjugate gradient solve (the postprocesses still see explicit forces but not implicit forces and will thus be less accurate). We note that this was only done for the large pile example, and all other examples were run as described above.

3.7.3 Enforcing Constraints in the Solve

We enforce the linear constraints in the conjugate gradient solve in a momentum-conserving way. Let the constraints be written as $\mathbf{C}^T\mathbf{v} = 0$, where $\mathbf{C}$ is a matrix of coefficients and $\mathbf{v}$ is a vector containing the linear and angular velocities of all particles and rigid bodies. If M^{-1} is the block-diagonal mass matrix, the projection to be applied is $\delta - M^{-1}\mathbf{C}(\mathbf{C}^T M^{-1}\mathbf{C})^{-1}\mathbf{C}^T$, where δ is the identity matrix. This general form for the projection matrix holds for both

Figure 3.11: A deformable fish constructed by embedding a four-joint PD-controlled articulated rigid body skeleton. The fish is dropped on the ground and flops back and forth interacting with its environment.

rigid bodies and individual particles. When there are multiple constraints, we iterate these projections using forward and backward sweeps to ensure our projection step is symmetric even if it has not fully converged as in [41]. The resulting matrix is always symmetric positive semidefinite and can be solved with conjugate gradients provided the residual is properly projected at each iteration.

An alternative approach would be to solve the augmented system (or KKT system), which is symmetric and indefinite. This system was solved efficiently in [2] for the case of explicitly integrated applied forces and a loop-free set of constraints. In the case of loops, such as those introduced by loops of articulated rigid bodies or implicitly integrated damping forces, this approach is no longer efficient. Another possible approach is to solve the KKT system with an iterative Krylov solver for indefinite systems, avoiding the projections at the cost of potentially inferior convergence characteristics.

Post-stabilization The post-stabilization algorithm of [87] is included in our conjugate gradient solve to project the rigid body velocities in a momentum-conserving fashion so that they satisfy joint constraints. Since step 3 discards these velocities, they are reapplied in step 5.

Rigid/rigid contact We incorporate rigid/rigid contact constraints into the solve by creating joints just prior to the solve and removing them afterwards. In this fashion, we are able to use the articulated rigid body post-stabilization algorithm for projecting contact

constraints during the conjugate gradient solve. When computing contact (during step 3 of Section 3.5) we record all points on the surface of one rigid body processed for contact against another. For each such point, we use its time t^{n+1} location and the level set normal from the other body at that point. We then construct a joint that only constrains motion in the normal direction and leaves the other two prismatic degrees of freedom and all angular degrees of freedom unconstrained.

Rigid/deformable contact Rigid/deformable contact projection is performed in a manner similar to interpenetration resolution as described in Section 3.6.1. We process a rigid body against all particles in contact with it simultaneously. However, we restrict impulses to the normal direction $\mathbf{n}_o$ during this contact processing phase. We target a relative change $\mathbf{v}_o$ that will cancel out the relative velocities in the normal direction of the particle and the body at the intersection point $\mathbf{p}_o$. If the central body is kinematic, the particles are given an impulse in the normal direction that cancels the relative normal velocity between the particle and the rigid body. Otherwise we apply impulses $j_o\mathbf{n}_o$ (where j_o is a scalar) to each outer body c at $\mathbf{p}_o$ so that

$$\mathbf{n}_o^T(m_c^{-1}\mathbf{j}+\mathbf{r}_o^{*T}I_c^{-1}\mathbf{j}_\tau - K_o j_o \mathbf{n}_o) = \mathbf{n}_o^T\mathbf{v}_o. \tag{3.3}$$

Using $L_o = \mathbf{n}_o\mathbf{n}_o^T(\mathbf{n}_o^T K_o \mathbf{n}_o)^{-1}$, we can express these equations in the form

$$\begin{pmatrix} m_c\delta + \sum_o L_o & \sum_o L_o\mathbf{r}_o^{*T} \\ \sum_o \mathbf{r}_o^* L_o & I_c + \sum_o \mathbf{r}_o^* L_o \mathbf{r}_o^{*T} \end{pmatrix} \begin{pmatrix} m_c^{-1}\mathbf{j} \\ I_c^{-1}\mathbf{j}_\tau \end{pmatrix} = \begin{pmatrix} -\sum_o L_o\mathbf{v}_o \\ -\sum_o \mathbf{r}_o^* L_o \mathbf{v}_o \end{pmatrix} \tag{3.4}$$

where the net linear and angular impulses applied to the central body are $\mathbf{j} = -\sum_o j_o\mathbf{n}_o$ and $\mathbf{j}_\tau = -\sum_o \mathbf{r}_o^* j_o \mathbf{n}_o$. This 6×6 system is symmetric positive definite and can be solved to obtain $\mathbf{j}$ and $\mathbf{j}_\tau$, from which j_o is obtained by substitution into Equation (3.3).

Self-repulsions Self-repulsions apply forces between edge-edge and point-triangle pairs to help avoid collisions [9]. In the conjugate gradient solve, we use the projection algorithm proposed in [41] to enforce these constraints.

3.8 Examples

Figure 3.5 illustrates that we can use the embeddings of [78] in our fully two-way coupled approach. Note that [78] was unable to handle two-way collisions and contact and more generally used an interleaved, semi-implicit approach to rigid/deformable coupling. Figure 3.1 demonstrates the robustness of the algorithm which allows stable two-way interaction between many types of competing constraints. Figure 3.4 demonstrates the ability of the two-way contact and collision algorithms to handle both volumetric objects and thin shells. Figure 3.6 demonstrates the ability of the algorithm to handle simulations with large numbers of two-way coupled bodies. One of the major applications of two-way rigid/deformable coupling is the simulation of skeleton-controlled deformable objects that can interact with their environment as shown in Figures 3.7, 3.10 and 3.11. Table 3.1 provides timing information for all of the examples in this chapter. Most of the examples were fast enough that we simply ran them with conservative time steps, whereas the large pile was expensive enough to warrant performance tuning.

3.9 Conclusions

We propose a novel time integration scheme for two-way coupling rigid and deformable bodies that retains the strengths of deformable object simulators and rigid body simulators. We build upon existing algorithms for rigid/rigid and deformable/deformable interaction and where necessary propose new fully-coupled algorithms. The resulting scheme handles two-way coupled contact, collision, stacking, friction, articulation, and PD control. We use our framework to simulate life-like creatures that interact with their environment.

	Ave. Time Per Frame	Ave. Substeps Per Frame
Twisting Chain (Figure 3.1)	92.2 sec†	123
Stack (Figure 3.2)	10.0 sec	35
Row of Spheres (Figure 3.3)	35.3 sec	94
Deformable Bar (Figure 3.5)	4.1 sec	51
Trampoline (Figure 3.4)	14.5 sec	32
Large Pile (Figure 3.6)	40 min	14
Snake (Figure 3.7)	1.8 sec	38
Friction (Figure 3.9)	6.4 sec	2515^{+}
Single Maggot (Figure 3.10)	0.6 sec	30
Maggot and Ring (Figure 3.10)	7.7 sec	100
Bowl of Maggots (Figure 3.10)	19.1 sec	30
... with Rings (Figure 3.10)	62.2 sec	59
Fish (Figure 3.11)	52.8 sec	117

Table 3.1: This table contains average times per frame and number of substeps required per frame for each example. † The individual times varied substantially between frames, depending strongly on the degree of stress. $^{+}$ This test was run as a convergence test with an artificially low time step.

Chapter 4

Multiple Interacting Liquids

This chapter presents an extension of the particle level set method for the geometric representation of three or more fluid regions, and first appeared in [52]. The method uses a separate level set function and a separate set of particles for each region. A novel projection algorithm is used to uniquely define the interface at any given point, providing a dictionary for translating the vector-valued multiple level set function into the standard single-valued level set representation. We use this method to simulate multiple interacting liquids with varying densities and viscosities, viscoelastic properties, surface tension forces and surface reactions such as combustion.

4.1 Introduction

Earlier works on fluid simulation focused on single phase flows such as smoke [22, 24, 80] or free surface flows such as water [18, 23]. More recently, researchers have considered more complex phenomena including fire [48, 62], bubbles [37], viscoelasticity [29], etc. Although the level set method allows for the simulation of two distinct fluids, such as fuel and products [62] or water and air [37], it does not handle the complex phenomena associated with the interactions of more than two fluids. We propose a novel method that

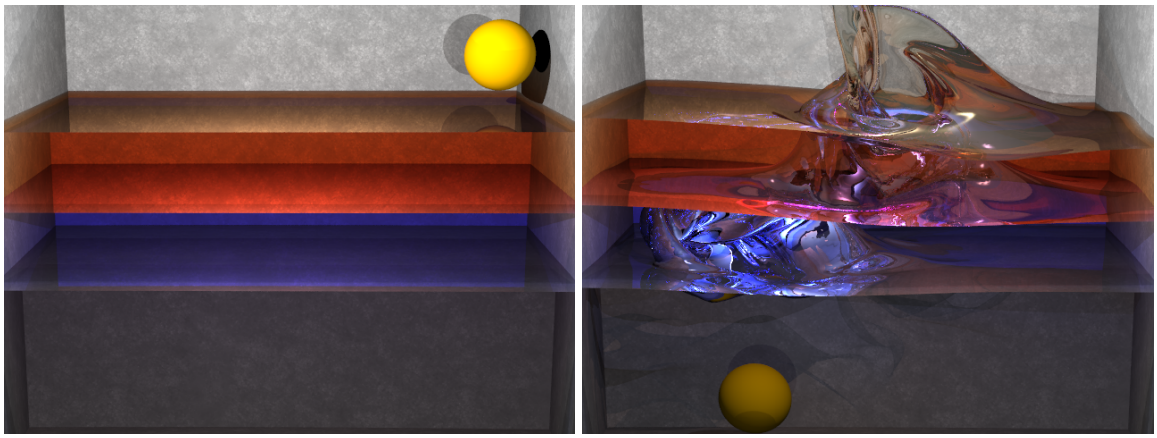


Figure 4.1: A kinematically controlled sphere splashing into a multi-layer pool ($300 \times 300 \times 200$ grid, 4 phases).

allows one to simulate multiple (more than two) liquids (and gases), including complex interactions between the different fluids.

A few researchers have begun to tackle the difficulties associated with using multiple level sets (without particles) to represent multiple regions. One approach is to use a different level set for each region as in [54, 70, 79]. Each level set is independently evolved forward in time leading to contradictions in the representation of the interface, which are resolved via projection (or slowly, via a penalty method as in [92]) eliminating points that have been classified as inside more than one region or not inside any region. The other main approach uses n level sets to represent up to 2^n regions [84]. For example, two level sets can be used to represent four regions classified via all possible sign combinations (i.e. "$++$", "$+-$", "$-+$" and "$--$"). This approach intrinsically removes the need for projection, but typically suffers from biasing artifacts especially where more than two regions intersect. So while it is quite useful in computer vision especially when there are *many* regions, it has not enjoyed similar success in physics-based applications where the number of distinct materials is typically small enough that it is not inefficient to use a separate level set for each allowing for a non-biased simulation of the underlying physics. In fact, level set methods are typically only applied in a lower dimensional band near the interface making them cheap as compared to the physical equations that need to be solved everywhere. Moreover, regardless of the number of level sets used, the particle level set method still requires one

set of particles for each region, so its cost remains unchanged. Finally, we note that all these approaches are predated by [26], which proposed a method for removing overlaps of implicitly represented deformable objects, although gaps between objects were not addressed making the work inapplicable to fluids where vacuum regions need to be properly addressed.

At each point in the domain, we have a vector $\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \ldots, \phi_n(\vec{x}))$. Since the individual level set functions (the ϕ_i's) will generally give contradictory geometric information, a consistent approach to interpreting the vector-valued level set function is needed. We do this by creating a level set "dictionary" that translates between the vector $\vec{\phi}$ and the traditional single level set representation at each point in the domain. Our novel projection method makes this translation straightforward for both theoretical and practical purposes (e.g. allowing the incorporation of previous level set simulation techniques). Moreover, our method is purely geometric and thus does not interfere with the underlying physics. It also preserves the signed distance property of the various level set functions (unlike for example [54]).

Our method provides for the straightforward simulation of multiple liquids (and air) with varying densities, viscosities, or viscoelastic properties. We also consider complex interactions between fluids such as surface tension forces and reactions (e.g. the burning of a premixed fuel as in [62]). Such interactions typically involve discontinuous material properties across the interface, e.g. pressure jumps due to surface tension. [62] and [37] advocated using the ghost fluid method (GFM) to avoid the visual errors associated with nonphysically smearing out these discontinuities. We propose a novel paradigm that automatically detects when discontinuous information is combined across (any number of) interfaces, computes jump conditions and ghost values "on the fly," and returns appropriate values. This reduces the memory requirements associated with storing ghost values for multiple region interactions, and furthermore makes the implementation of the algorithms straightforward. We note that it also simplifies the treatment of complex solid objects (see e.g. [33]).

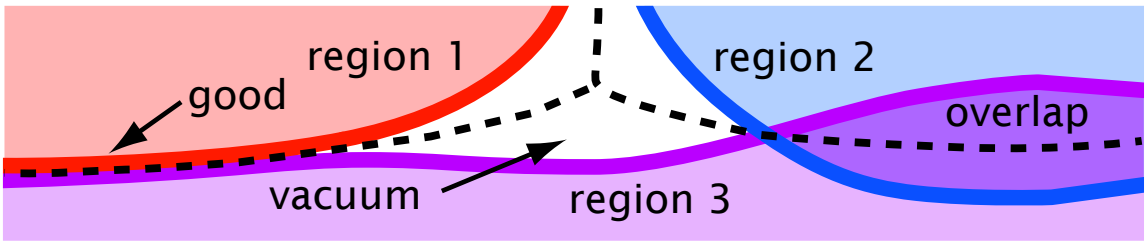


Figure 4.2: Each of the three regions is independently evolved in time, after which the interface locations do not agree. There are vacuums where all ϕ_i are positive, and overlaps where more than one ϕ_i is negative. The dotted black line shows the new interface locations after our projection step.

4.2 Multiple Level Sets

Each level set function is independently evolved in time, after which the interface locations do not agree (because of numerical errors) as shown for example in Figure 4.2. We propose a novel method for fixing the level set functions removing overlaps and vacuums while preserving an accurate interface location.

4.2.1 Projection Method

We first make the following observations about an arbitrary vector $\vec{\phi}$ of level set values at a point $\vec{x}$. (O1) If ϕ_j is the smallest element, $\vec{x}$ is in region j. This assigns $\vec{x}$ to the region it is deepest inside when it is inside more than one region (overlap), or the region it is closest to when it is outside every region (vacuum). (O2) If O1 holds and ϕ_k is the second smallest element, only ϕ_j and ϕ_k are needed to locally represent the interface. Basically, $\vec{x}$ is in region j, and the closest point on an interface lies between region j and region k. Region k is the region $\vec{x}$ is closest to not counting the region it is in.

Given these observations, we desire the following properties for numerical robustness and backward compatibility with the standard single level set function for two phases. (P1) If ϕ_j is the smallest element, it is the only negative element and its magnitude represents the distance to the interface. This is consistent with observation 1, but also makes ϕ_i a signed

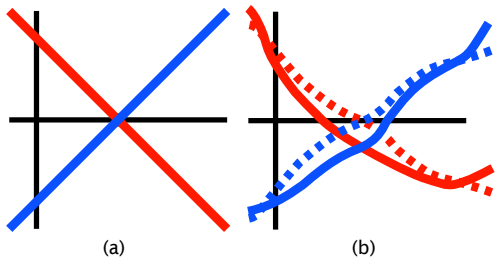


Figure 4.3: (Left) Two level sets initialized so that properties 1 and 2 hold. (Right) After evolving in time, we obtain the solid lines with overlap (both negative in the middle). The dotted lines show an example result after projection. Not only has the overlap been removed, but the interface location is preserved.

distance function in region i for all i. Moreover, it removes overlaps since only one ϕ_i is negative, and removes vacuums since the smallest ϕ_i *is* negative. (P2) If P1 holds and ϕ_k is the second smallest element, $\phi_k = -\phi_j$. This is consistent with observation 2, and it makes the level set for the region a point is closest to but not inside a signed distance function as well. Moreover, all other ϕ_i are positive and bigger than ϕ_k and not relevant.

These observations and properties are consistent with the standard single level set function methodology. A standard single level set function ϕ can be broken into two separate functions ϕ_1=ϕ and $\phi_2 = -\phi$, and be shown to satisfy the above observations and properties. This readily gives us a dictionary that translates between $\vec{\phi}$ and ϕ. That is, once we take an arbitrary level set vector $\vec{\phi}$ and project it to satisfy properties 1 and 2, we can use ϕ_j and ϕ_k as if they were ϕ_1 and ϕ_2 in the appropriate order. The only discrepancy lies in how the exact interface is handled where $\phi_j = \phi_k = 0$. For the standard level set function, this is equivalent to $\phi_1 = \phi_2 = \phi = 0$ and is typically nominally assigned to the negative level set, i.e. we define the regions via $\phi \leq 0$ and $\phi > 0$. This is equivalent to assigning the point to region 1 when ϕ_1 and ϕ_2 are both zero. To extend this to multiple level sets, we assign a point where both ϕ_j and ϕ_k are zero to region j or k depending on whether $j < k$.

We illustrate our method in one spatial dimension. Figure 4.3a shows two level set functions that satisfy properties 1 and 2, and Figure 4.3b shows a property violating version after evolving in time. Based on property 1, the interface location is defined as the point where the minimum ϕ_i changes from one level set to the other. This is the location where

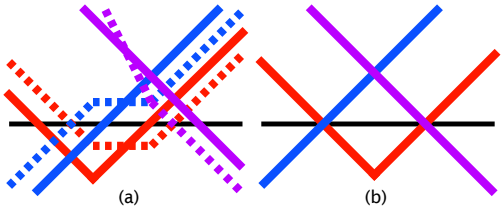


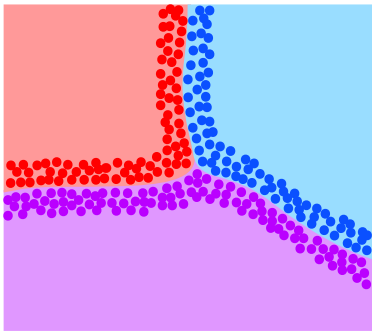
Figure 4.4: (Left) The solid lines have overlap on the left (two ϕ_i negative) and a vacuum on the right (all ϕ_i positive). The average of the smallest two level sets at any point is subtracted from $\vec{\phi}$ to obtain the dotted lines. Not only has the overlap and vacuum been removed, but the smallest ϕ_i is preserved and negative at each point preserving interface locations and inside/outside information. (Right) Results after reinitializing each level set to a signed distance function.

the two level sets intersect in the figure, and we want our projection method to preserve the interface location to avoid biasing. Thus, our projection method *computes the average value of the two level set functions and subtracts this average from both of them.* At points where the two level sets intersect, their average equals their individual values, and thus subtracting off their averages sets them both to zero preserving the interface location. Otherwise, at points where one level set is larger than the other, subtracting their average makes them the same magnitude but opposite sign preserving the region a given point is inside. The result is shown as dotted lines in the figure. If both level sets are reinitialized to signed distance functions, we obtain the result shown in Figure 4.3a which satisfies all desired properties. *The same projection method can be generalized to an arbitrary vector of level sets by subtracting the average of the smallest two ϕ_i from all of the ϕ_i.* An example of this is shown in Figure 4.4.

Notably our method is unbiased preserving signed distance information. For example, consider Figure 4.3a and Figure 4.4b where the level sets are all signed distance functions to begin with. Because property 2 holds, the two smallest ϕ_i are equal and opposite in sign making their average identically zero at every point. Thus, subtracting the average leaves all the the ϕ_i unchanged preserving signed distance. This surprisingly simple algorithm has all the properties we desire. Notably, [54] is similar in spirit to our own, but while they preserve the interface location and inside/outside information, they do not preserve signed distance thus introducing biasing into the algorithm.

For two regions (hence two level sets) the result produced by our method is identical to that of the traditional single level set (or particle level set) method. If the two level set functions at time n are negatives of each other, i.e. $\phi_1^n = -\phi_2^n$, after advection we still have $\phi_1^* = -\phi_2^*$ because they are evolved with the same method. Then projection leaves ϕ_1^* and ϕ_2^* unchanged, since their identically zero average is subtracted off. This is also true for more than two regions away from multiple junctions (e.g., triple points).

4.2.2 Particle Level Set Method



Each level set has an associated set of particles that are seeded near the boundary of its interior region as shown in the figure to the right. Following the standard particle level set algorithm, for each level set function we rebuild ϕ^- using that level set's particles, and rebuild ϕ^+ using all the particles from all other regions. For efficiency, we ignore particles that are far from the interface of the region in question. Typically, particles are used to correct the level set function both after advection and after reinitialization. We apply our projection method to every grid point after each of these particle correction steps. Then for each grid point, all geometric information can be computed from the level set function that is negative at that point. When level set values are needed in between grid points, we interpolate $\vec{\phi}$ to that location and apply our projection method on the fly to find the resulting negative ϕ_i. Note that the first projection step is important because reinitialization preserves the interface location of each level set individually, but not their intersections which correspond to the pre-projected interface locations. The second projection removes any numerical drift introduced by reinitialization, and we note that a method such as [54] could not be used for this step because it does not preserve signed distance.

Advancing the incompressible velocity field to the next time step and particle advection are the most expensive parts of our algorithm. These steps depend mostly on the fluid volume and surface area, respectively, rather than the number of regions. The cost of advecting and

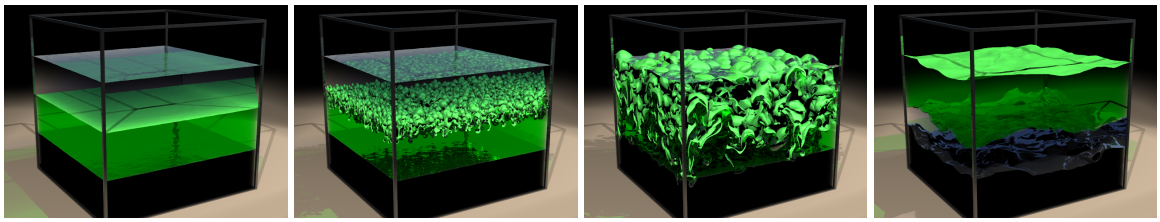


Figure 4.5: Rayleigh-Taylor instability (300^3 grid, 4 phases).

reinitializing the level set functions remains fixed at twice the usual cost, since two level sets are updated locally near each interface instead of one. Thus, the cost of updating the level set function scales with the surface area just as in the standard single level set method regardless of the number of regions.

4.3 Multiple Liquids

We model the fluids using the incompressible Navier-Stokes equations

$$\nabla \cdot \vec{u} = 0 \tag{4.1}$$

$$\vec{u}_t + (\vec{u} \cdot \nabla)\vec{u} + \nabla p/\rho = (\nabla \cdot \tau)/\rho + \vec{f} \tag{4.2}$$

where $\vec{u} = (u, v, w)$ is the velocity, ρ is the density, τ is the viscous stress tensor, and $\vec{f}$ accounts for body forces, e.g. gravity, vorticity confinement, etc. For simplicity, we first consider the inviscid case. First, an intermediate velocity field $\vec{u}^\star$ is computed

$$(\vec{u}^\star - \vec{u}^n)/\Delta t + (\vec{u}^n \cdot \nabla)\vec{u}^n = \vec{f} \tag{4.3}$$

using a semi-Lagrangian advection scheme as in [80]. Next, we compute the pressure via

$$\nabla \cdot (\nabla p/\rho) = \nabla \cdot \vec{u}^\star/\Delta t. \tag{4.4}$$

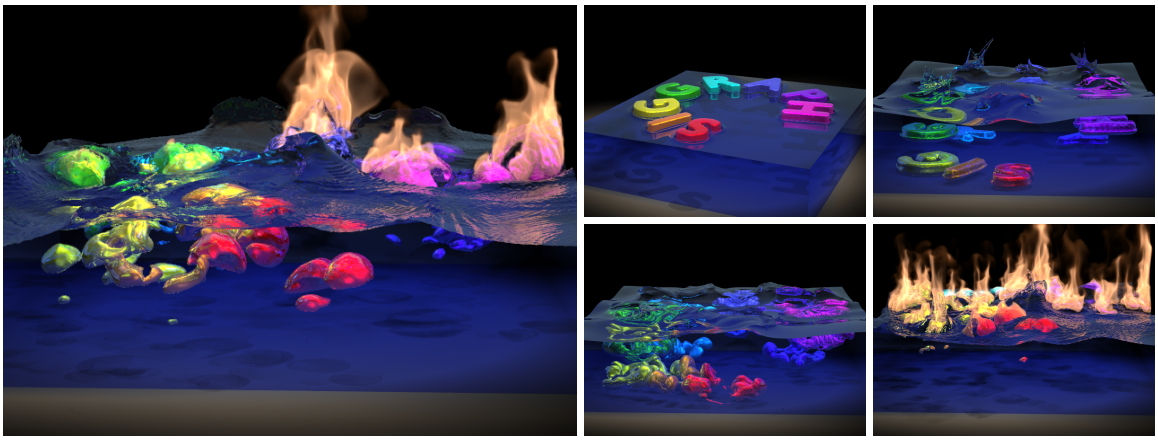


Figure 4.6: Viscous letters splash into a pool of water, then change into low density inviscid fuel bubbling up and burning when they hit the surface ($350 \times 200 \times 350$ grid, 10 phases).

and use it to make the velocity field divergence free

$$(\vec{u}^{n+1} - \vec{u}^{\star})/\Delta t + \nabla p/\rho = 0. \tag{4.5}$$

4.3.1 Poisson Equation

We follow the method of [62]. For multiple fluid regions, equation (4.4) is a Poisson equation with discontinuous coefficients. The equation is separable so we can consider each dimension independently. A standard second order accurate discretization of the left hand side in one spatial dimension at a grid node i is

$$\left(\beta_{i+1/2}(p_{i+1} - p_i)/\Delta x - \beta_{i-1/2}(p_i - p_{i-1})/\Delta x\right)/\Delta x \tag{4.6}$$

where $\beta = 1/\rho$. For inviscid flow, [45] showed that the flux in equation (4.4) is continuous across the interface satisfying

$$\beta^- p_x^- = \beta^+ p_x^+ \tag{4.7}$$

where the $-$ and $+$ superscripts represent values from different sides of the interface. Thus if ρ (and hence β) varies across the interface then so must p_x. Consider the case where an

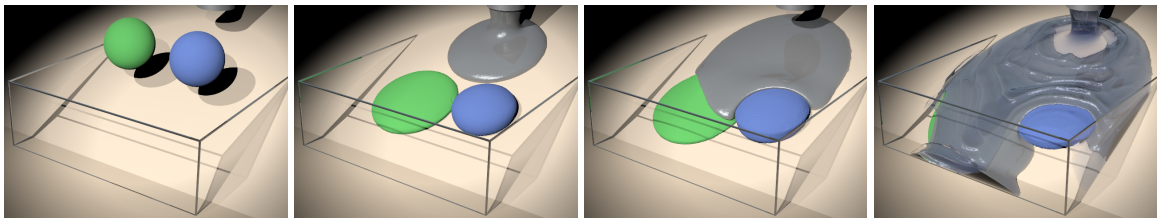


Figure 4.7: Different viscosity liquids interacting on an inclined plane. ($300 \times 150 \times 240$ grid, 5 phases).

interface lies between nodes x_i and x_{i+1}. We define $\theta = |\phi(x_i)|/(|\phi(x_i)| + |\phi(x_{i+1})|)$ and approximate equation (4.7) with one-sided differences as

$$\beta^-(p_I - p_i)/(\theta\Delta x) = \beta^+(p_{i+1} - p_I)/((1-\theta)\Delta x) \tag{4.8}$$

and solve for the interface pressure $p_I = (\theta\beta^+ p_{i+1} + (1-\theta)\beta^- p_i)/(\theta\beta^+ + (1-\theta)\beta^-)$ which can be substituted into either the left or the right hand side of equation (4.8) to obtain $\hat{\beta}(p_{i+1} - p_i)/\Delta x$ where $\hat{\beta} = (\beta^-\beta^+)/(\theta\beta^+ + (1-\theta)\beta^-)$. Thus, the discontinuity between grid nodes i and $i+1$ is readily handled by replacing $\beta_{i+1/2}$ with $\hat{\beta}$ in equation (4.6). $\beta_{i-1/2}$ is treated similarly.

4.3.2 Viscosity

The viscous stress tensor for incompressible flow is $\tau = \mu(\nabla\vec{u} + (\nabla\vec{u})^T)$. As discussed in [68], a spatially constant μ (within each region) implies that $\nabla \cdot \tau = \mu\Delta\vec{u}$. Renaming $\vec{u}^{n+1}$ in equation (4.5) to be $\vec{u}^{\star\star}$, we next solve the three systems of linear equations given by

$$\vec{u}^{\star\star\star} = \vec{u}^{\star\star} + \Delta t\nabla \cdot (\nu\nabla\vec{u}^{\star\star\star}) \tag{4.9}$$

where $\nu = \mu/\rho$. Note that we moved ρ under the divergence operator, under the assumption that it is spatially constant in each region. Since the viscosity to density ratio is discontinuous across the interface, we replace ν with $\hat{\nu}$ for differences that cross the interface in the same manner as β is adjusted to $\hat{\beta}$ when solving equation (4.4). Then, we again solve for the pressure and make the flow divergence free using $\vec{u}^{\star\star\star}$ in place of $\vec{u}^{\star}$ in equations

(4.4) and (4.5).

Each component of equation (4.9) should conserve momentum, so we require a unique flux between every two velocity values. The physically correct flux in the incompressible flow context is rather complicated (see its derivation in [45]). However, considering equation (4.9) in isolation admits a simple approximation which [37] showed was sufficient for visual accuracy. In isolation, the flux is given by $\nu\nabla\vec{u}$, and we assume $\nu^-\nabla\vec{u}^- = \nu^+\nabla\vec{u}^+$ which is not actually true.[1] Instead, correction terms should be added for stencils that cross the interface, but these terms couple the u, v, and w diffusion equations together making them difficult to solve with a fully implicit method. For example, see [68] where variable viscosity couples the three equations together.[2] They explicitly add correction terms before solving for the velocity implicitly. We could take a similar approach allowing for a larger time step than a fully explicit method, but it is still less efficient than a fully implicit method. [45] showed that the viscosity jump causes a jump in the pressure and its derivatives as well[3], but [37] showed that these too can be ignored for graphical purposes.

4.3.3 Viscoelasticity

[29] incorporated viscoelastic effects by adding $(\mu_e/\rho)\nabla\cdot\varepsilon$ to the Navier-Stokes equations, where μ_e is the elastic modulus and ε is the elastic strain tensor evolved in time via $\varepsilon_t + \vec{u}\cdot\nabla\varepsilon = (\nabla\vec{u} + (\nabla\vec{u})^T)/2 - \varepsilon_t^{Plastic}$. They solved this last equation by first using semi-Lagrangian advection, and then incorporating the right hand side which is the total strain rate minus the plastic strain rate. [40] points out that this ignores the rotation of the strain tensors, yielding incorrect results when the fluid rotates. Thus, [40] proposes rotating the strain tensor by the curl of the velocity field after the advection step. This is accomplished by computing an explicit rotation matrix in the center of each cell, and using it to rotate the strain tensor also stored in the center of each cell. This has enabled high quality detailed viscoelastic fluid computations, see e.g. Figure 4.12.

[1] [45], equation (30) gives the jump across the interface
[2] [68], equations (5)-(7)
[3] [45], equations (19) and (32)-(34)

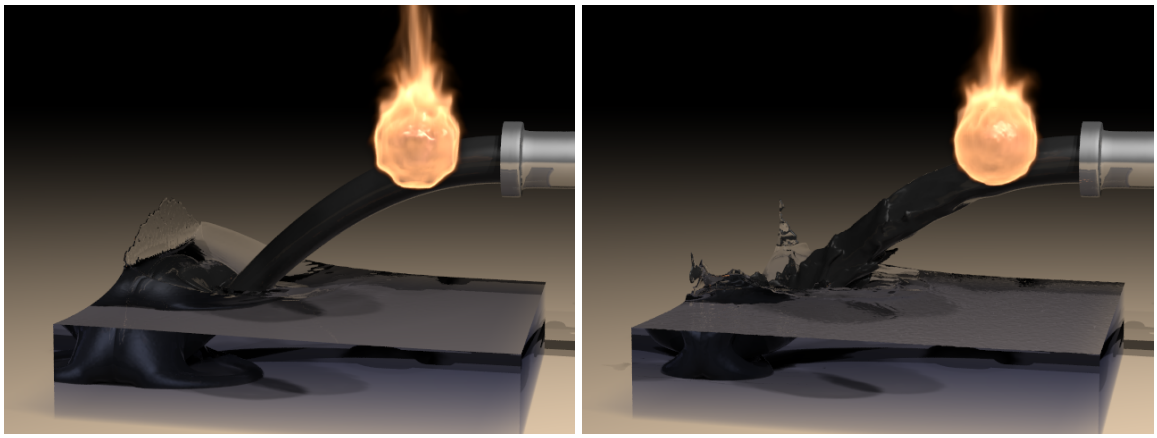


Figure 4.8: (Left) One-way coupling from liquid to air. (Right) Two-way coupling of liquid and air. Note the surface ripples and the unstable stream of liquid in the fully coupled simulation.

4.4 Adding Air or Empty Regions

Often times, only the liquid region is of interest and the gas flow can be ignored. Our system allows for the standard treatment of this by setting an entire region to be empty, and subsequently extrapolating velocity into that region from the liquid regions and using Dirichlet boundary conditions during the pressure solve (see e.g. [18]). However, when the gas flow is important, our method trivially extends to simulate gas regions just as though they were other liquid regions. This allows for straightforward incorporation of smoke, fire, and even reactive gases as in [39]. Besides empty regions and air regions, there is yet a third way to model non-liquid regions. The animator may wish to simulate air, but not have the air affect the liquid. This requires one-way coupling from liquid to air, but not vice versa. Figure 4.8 shows one way coupling (left) as compared to two-way coupling (right). One-way coupling is accomplished by using extrapolated velocities from the liquid to the gas as boundary conditions for the liquid region, while gas advection is carried out normally. In addition, these extrapolated velocities are used to overwrite the gas velocity at any grid points that become liquid as the interface moves. The Poisson equation is first solved for the liquid region setting Dirichlet boundary conditions in the gas so that it has no effect (as is usual for empty regions), and then a second Poisson equation is solved for

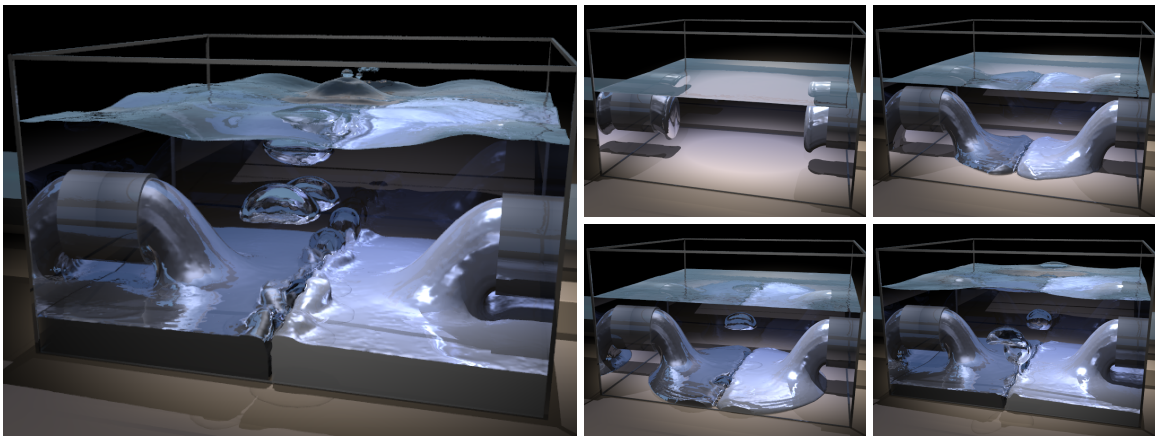
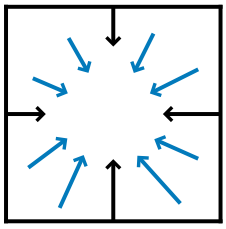


Figure 4.9: Two submerged liquids meeting and reacting to create air (150^3 grid, 4 phases).

the gas using Neumann fixed velocity boundary conditions in the liquid so that it properly drives the gas.

In free surface flow, the air region is modeled as empty allowing it to vanish. Thus, characteristics coalesce as shown in the single grid cell in the figure to the right. Eventually, liquid rushes into the cell from all sides, and the air should disappear. However, air particles faithfully follow these characteristics ending up trapped in the center of the cell. Since this cell should be a sink for air, we simply delete the air particles as they approach the center of the sink. Note that these sinks are easily detected by finding local minima level set values in empty regions.



4.5 Surface Tension

The ghost fluid method (GFM) of [21] uses the physically correct interfacial jump conditions to define ghost values for discontinuous quantities which are then incorporated into finite difference or interpolation stencils. [37] used the GFM to discretize the jumps in pressure caused by surface tension effects, and [36][4] showed that the method produces far

[4]page 36

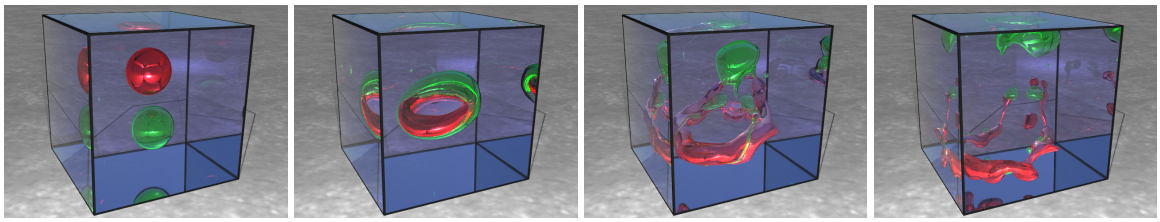


Figure 4.10: Two drops with high surface tension collide. Green has low density, red high density (350^3 grid, 3 phases).

better visual results than a smeared out delta function approach. Surface tension causes a jump in pressure across the interface equal to $\sigma\kappa$, where σ is a surface tension coefficient (defined pairwise for the regions) and $\kappa = -\nabla\cdot(\nabla\phi/|\nabla\phi|)$ is the interface curvature. Consider the case where an interface lies between x_i and x_{i+1}. We adjust p_{i+1} in equation (4.6) to account for the jump in pressure,

$$\left(\hat{\beta}((p_{i+1}+\sigma\kappa_\Gamma)-p_i)/\Delta x-\beta_{i-1/2}(p_i-p_{i-1})/\Delta x\right)/\Delta x$$

where $\kappa_\Gamma = \theta\kappa_{i+1}+(1-\theta)\kappa_i$ is computed with respect to the region containing x_i, and $\beta_{i+1/2}$ has been replaced by $\hat{\beta}$ as explained in section 4.3.1. The $(\hat{\beta}\sigma\kappa_\Gamma)/\Delta x^2$ term can be moved to the right hand side, so that the resulting matrix is unaffected allowing for the use of fast symmetric linear system solvers such as the preconditioned conjugate gradient method.

4.6 Surface Reactions

Our method incorporates surface reactions allowing one material to turn into another. One example of this is the work on fire by [62], but our framework allows for a more generalized treatment, e.g. Figure 4.9 shows two materials (submerged beneath a third) coming together and reacting to form a fourth.

[62] used the GFM to model fire where the expansion of fuel into products admits jumps

Figure 4.11: Oil pouring into water, then catching on fire. Note that the fiery ball is a separate phase of fluid, and that it deforms into the shape of a droplet as it falls (200^3 grid, 4 phases).

in both velocity and pressure.[5] As in the surface tension case, the pressure jump is incorporated directly into the Poisson equation. The velocity jump needs to be handled whenever information is combined from different sides of the interface. They implement this by storing two velocity fields, one for fuel and one for products, that inherently store the jump conditions. While this only doubles their storage, the storage requirements would scale linearly with the number of different materials. We instead compute ghost values on the fly by generalizing the concept of a scalar or vector field to encapsulate the application of the jump condition. Our implementation wraps the data in a *lookup* class that maintains a state variable indicating the region for which values are being looked up. For example, when using semi-Lagrangian advection to update a face velocity in region i, we create a lookup class instance and set its internal state to indicate that any queried values should be returned with respect to region i. Then the lookup class ensures that any data used to construct the ray or interpolate is retrieved with the proper jump conditions already applied. In this manner, existing interpolation and discretization code is generalized with relative ease to account for discontinuities. The lookup classes can also be used to incorporate object intersections in the same manner as jump discontinuities. For example, for thin objects, a nested lookup class can be used to check for object intersections and return the appropriate

[5] See equations (2) and (3) for the jump conditions

object ghost velocity as described in [33], simplifying object interaction implementations significantly.

[62] used only the normal component of the velocity to advect the fuel level set, which is sufficient for their WENO scheme that was applied without particles. We instead use semi-Lagrangian advection for the level set equation, and this requires the tangential component of velocity as well in order to properly trace characteristics. Since the tangential component is continuous across the interface, we form the normal component as usual and simply add the tangential component from the local fluid velocity. The tangential component is required for particle advection as well.

4.7 Examples

To demonstrate the effectiveness of our approach, we simulated a number of examples that range in resolution from 150^3 to 350^3 on a number of 4 processor Opteron machines. The computational cost for the examples range from 5 to 50 minutes per frame. Surface tension was the main cause for the examples with slower simulation times. We augmented a standard ray tracer to use the same projection based querying of the level set functions that the simulation uses, since rendering each level set independently can lead to multiple intersections per interface (from numerical error).

Figure 4.10 depicts two drops suspended in liquid. The lower drop has a lower density than the surrounding fluid and the upper drop has a higher density. Both drops have surface tension. Figure 4.1 depicts a kinematic sphere splashing into a number of liquids. The air region is simulated as a Dirichlet region, and the liquids are of increasing density from top to bottom. Figure 4.5 shows four layers of fluid where the lower middle liquid is lighter than the top middle liquid. This causes a Rayleigh-Taylor instability as the two liquids switch places.

Figure 4.7 shows a number of different fluids on an inclined plane. The liquid with the highest viscosity is blue, then green, then silver, and finally the clear water is simulated as

inviscid. Figure 4.12 depicts a viscoelastic armadillo in a pool of water. The viscoelastic property is then removed making it viscous only, then the viscosity is turned off and the density is turned down making it bubble up to the surface. The former armadillo is then changed back to viscoelastic and a newly introduced viscoelastic liquid is dropped on top of it.

Figure 4.11 depicts flammable oil being released into a tank of water. The oil rises to make a layer on the surface, which is then ignited. This example uses a temperature based ignition model where the temperature in actively burning regions is T_{max}, but a lower $T_{ignition}$ is needed to cause ignition of surrounding fluid. Figure 4.9 shows two viscous liquids that react with each other creating a third (air) that bubbles up to the surface. In Figure 4.6, eight letters with various high densities and viscosities splash into a pool of water and sink to the bottom. The air is treated as an empty region. In the second part of the simulation, the letters are changed to be low density fuels with surface tension, and the air is fully simulated. The letters then rise through the water, bursting into flames as they break the surface.

4.8 Conclusions and Future Work

Notably, the new technique does exacerbate the limitations of the original particle level set method with regards to volume loss by facilitating increased scene complexity. This is evident in the Rayleigh-Taylor simulation depicted in Figure 4.5. In that example, the majority of the mass loss occurs away from triple points, in regions where our method is identical to the original particle level set method (see section 4.2.1). Of course, this can be addressed by increasing the particle count for the particle level set method or using an adaptive octree or run length encoded type level set simulation, at the expense of increased code complexity and/or CPU time.

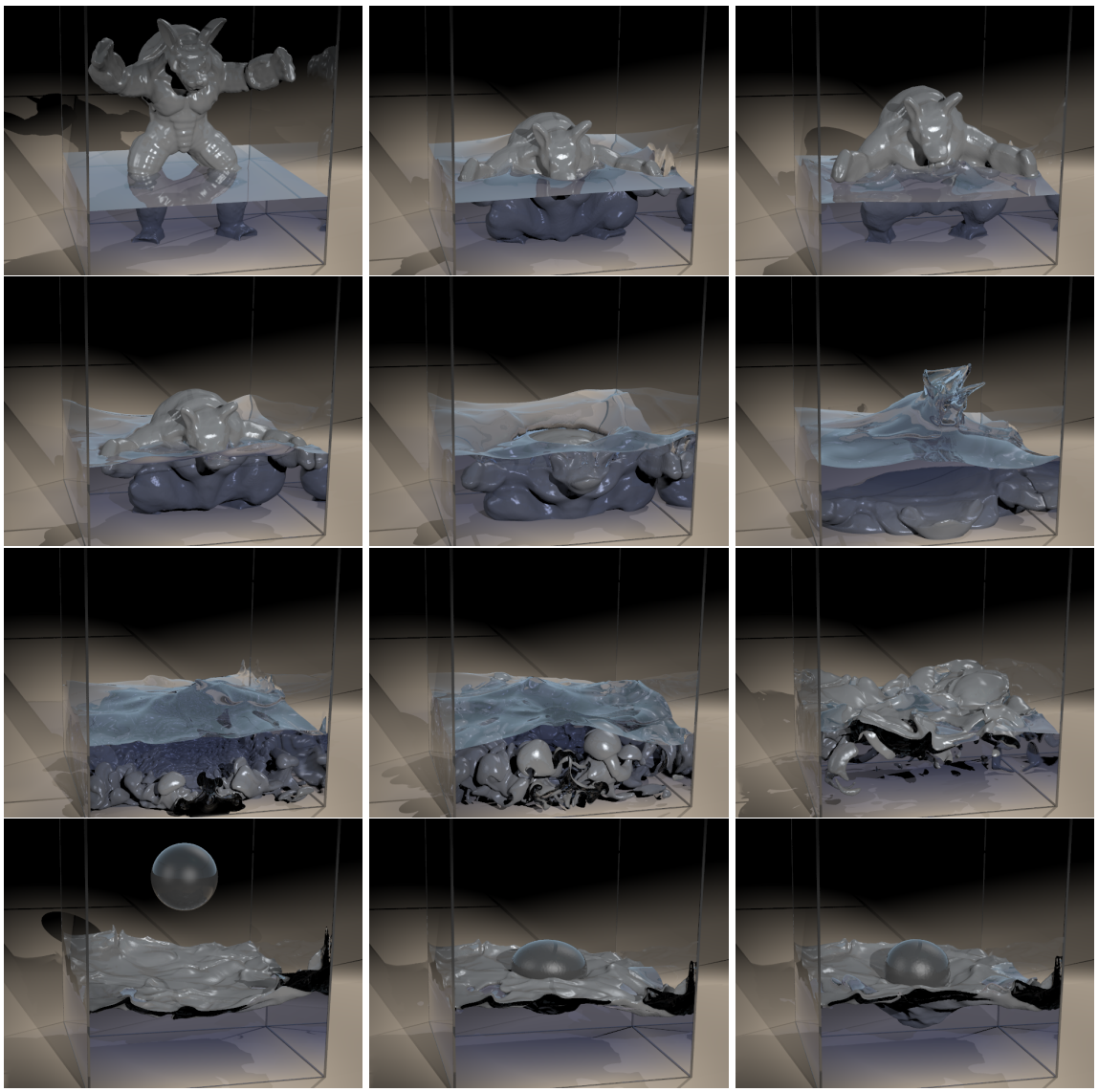


Figure 4.12: An armadillo that starts out viscoelastic, becomes viscous and more dense than the water, then inviscid and lighter than the water, and finally viscoelastic again before another viscoelastic liquid is dropped onto it ($250 \times 275 \times 250$ grid, 4 phases).

Chapter 5

Two-Way Coupling of Solids and Fluids

We present a novel solid/fluid coupling scheme (see [69]), which is derived from the physical principle that momentum should be conserved at the solid/fluid boundary subject to the constraint that the relative velocities are zero. In addition to conserving momentum and enforcing the no-slip boundary condition, our approach utilizes the standard Cartesian Eulerian grid for the fluid and Lagrangian mesh for the solid and results in a sparse, symmetric linear system. The method is sufficiently general to treat rigid, deformable, volumetric and thin solids coupled to multiphase incompressible flow.

5.1 Introduction

State-of-the-art solvers typically use Eulerian methods for fluids and Lagrangian methods for solids, but it has proven difficult to couple these disparate simulation methods together. Thus, many researchers have taken a fully Lagrangian approach [6,34,46,59,60,83,91], e.g. using particle-based methods for both the fluid and the solid ([46, 59]). Alternatively, one could use Eulerian methods for both the fluid and the solid, treating solids as high viscosity or viscoelastic Eulerian fluids [13,29,52,68]. See also [20] for an interesting approach that applies an Eulerian contact model based on implicitly integrated repulsion forces to various

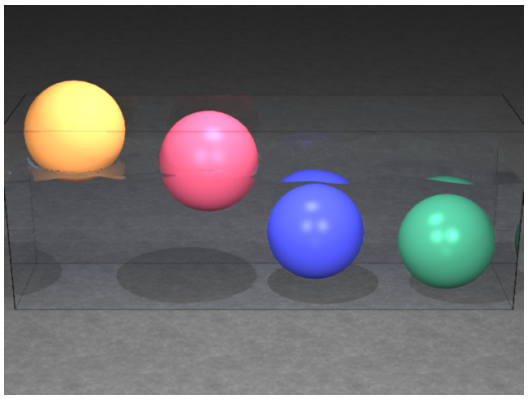


Figure 5.1: We demonstrate that our method handles buoyancy correctly by releasing rigid spheres of varying density in a pool of water ($200 \times 75 \times 50$ fluid grid).

simulation objects including Smoothed Particle Hydrodynamics fluid with thin deformable shells and an Eulerian fluid with volumetric rigid and deformable solids.

Coupling an Eulerian fluid to a Lagrangian solid is typically accomplished using the solid velocity as a boundary condition for the fluid while integrating the pressure force on the surface of the solid, e.g. [14, 27, 33, 51, 90]. These approaches treat coupling in either an explicit or semi-implicit fashion and thus stability and accuracy issues remain. For example, [14, 33, 51] solve a Poisson equation with the solid density rasterized onto the fluid grid alleviating some stability issues, but this rasterization does not account for the rigidity of objects, internal elastic forces, etc. Afterwards, [14] projects the velocity field pertaining to the rigid body to enforce rigidity, but this creates a discontinuous velocity which allows fluid to leak into and out of the solid. [33, 51] instead only use the Poisson equation to calculate forces on the solid, and afterwards project the fluid to be consistent with the resulting solid velocity in order to prevent leaking – however, this makes the method less stable.

[7, 15, 47] consider fully implicit stable two-way coupled interactions between solids and fluids. [7, 47] are limited to rigid bodies, and although they add only a rank 6 update per body to their fluid Poisson matrix, the resulting number of elements scales like a four spatial dimensional problem instead of three. [15] addresses deformable objects, but obtains a non-symmetric discretization. These methods do not address rigid shells or cloth.

The straightforward method of using Neumann boundary conditions on the fluid and integrating the pressure force on the boundary of the solid as used in [33, 51] and the Eulerian version of [15] does not conserve momentum. Thus even though [15] proposes a fully implicit coupling, the Eulerian version of their method can still be unstable and yield non-physical behavior. The key to fixing this is to properly and conservatively account for all pressure forces that transmit momentum between fluid and solid, as our formulation does while yielding a symmetrically coupled system. Unlike their Cartesian counterparts, Arbitrary Lagrangian-Eulerian (ALE) meshes can more readily be made to conserve momentum as long as the solid/fluid boundary faces are treated in the same conservative fashion as internal fluid/fluid faces. However, ALE methods require moving meshes that can lead to poor aspect ratios requiring frequent remeshing for interfaces subject to large deformations, and thus we prefer an Eulerian approach. Moreover, it would be difficult to create an ALE fluid mesh that conforms to a deforming piece of cloth.

5.2 Motivation

In resolving solid/solid contact and interpenetration in Chapter 3, we first formulated the velocity constraint for the interaction and then solved for an impulse that would conservatively enforce the constraint. Similarly, for a fluid/fluid or fluid/solid contact, a given interface boundary condition determines the velocity constraint, and we introduce an unknown impulse that conservatively enforces this constraint. A simple example is discussed here.

Consider a two-dimensional example of an axis-aligned interface, and a particular dual cell (a staggered cell between two pressure samples in the standard MAC grid discretization) which is half filled with each of two distinct materials with densities ρ_1 and ρ_2 (Figure 5.2). This simple case degenerates into a one-dimensional problem. Assume that the materials are initially in contact. Given the constraint that the materials remain in contact, moving together, we can compute the pressure, $p_{i+1/2}$, at the interface between the materials at

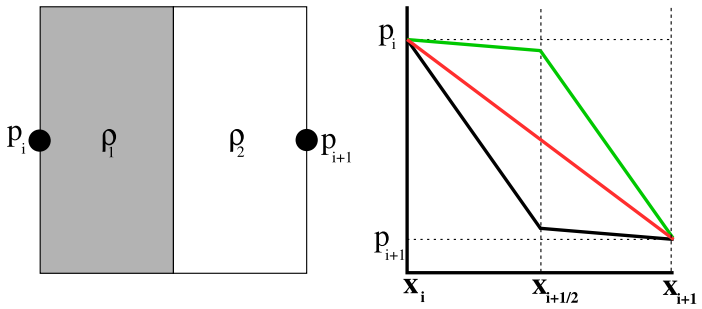


Figure 5.2: (Left) A two-dimensional drawing of a dual cell containing two materials of different densities. (Right) A graph of the pressure profile on a cross-section through the cell connecting p_i to p_{i+1}, i.e. $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$.

$x_{i+1/2}$. Conservation of momentum gives

$$\frac{Du_1}{Dt} = \frac{p_i - p_{i+1/2}}{\rho_1 \Delta x/2}, \quad \frac{Du_2}{Dt} = \frac{p_{i+1/2} - p_{i+1}}{\rho_2 \Delta x/2}. \tag{5.1}$$

The constraint that the interface remain in contact implies that $Du_1/Dt = Du_2/Dt$, and thus

$$p_{i+1/2} = \frac{\rho_2 p_i + \rho_1 p_{i+1}}{\rho_1 + \rho_2}. \tag{5.2}$$

This interface pressure can be regarded as determining the constraint force enforcing the contact constraint between the two materials.

When $\rho_1 = \rho_2$, the pressure profile is linear as illustrated by the red line in Figure 5.2. When ρ_1 is larger (smaller) than ρ_2, the pressure profile instead looks like the black (green) line. While the case considered here exhibits a monotonic, continuous pressure profile, in general, the pressure profile need not be monotonic in the cell and can even have discontinuous jumps, e.g., in the case of surface tension forces or thin shells.

In solving conservation of momentum for the two materials in the above example, we introduced an additional unknown, $p_{i+1/2}$, the pressure at the solid/fluid interface, and an additional equation given by the velocity constraint. Similarly, we augment our solid/fluid system for conservation of momentum with a set of unknown impulses and a corresponding

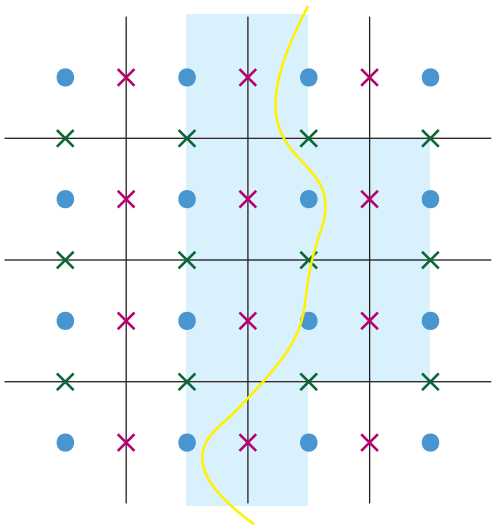


Figure 5.3: The MAC grid used in the Eulerian fluid simulation is depicted, with blue circles representing pressure samples and red and green x's representing the x and y components of the velocity, respectively. The yellow line represents the boundary of the Lagrangian solid. The mixed x-component dual cells, i.e. those containing both solid and fluid, are highlighted in light blue.

set of velocity constraint equations to close the system. These impulses are the momentum exchanged at the solid/fluid interface in order to satisfy the velocity constraint at the interface.

5.3 Solid/Fluid Interaction

The use of the Eulerian grid for the fluid and the Lagrangian mesh for the solid necessitates the definition of operators for transferring physical quantities between these disparate meshes. We define the physical quantities associated with solid/fluid interaction to be collocated with the fluid velocities. Thus we require operators for transferring quantities between the fluid velocity locations and the solid's nodal degrees of freedom.

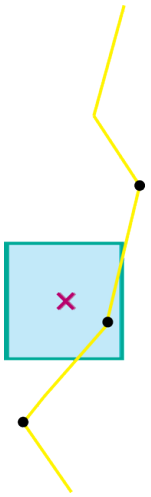
In the MAC grid discretization, x, y, and z velocity components are sampled in dual cells at the corresponding faces of the MAC grid cells (see Figure 5.3). In coupling the solid and

fluid, we deal with each dimension independently. We distinguish between dual cells that are fully fluid, fully solid, or *mixed*, as illustrated in Figure 5.3. The solid/fluid interaction occurs in the mixed dual cells.

5.3.1 Velocity Interpolation

In order to discretize the velocity constraint at the solid/fluid interface, we define a velocity interpolation operator, W_{DC} which maps solid nodal velocities to a dual cell velocity, u_{DC}, as

$$u_{DC} = W_{DC}\mathbf{V}, \tag{5.3}$$



where $\mathbf{V}$ is a column vector containing all the solid velocities, and W_{DC} is a row vector whose entries sum to one. For example, in the dual cell depicted to the right, the three velocity samples at the solid nodes are interpolated to the center of the cell to determine u_{DC}. In our implementation, the weights were chosen according to the relative areas of the solid faces in the dual cell.

5.3.2 Force Distribution

In order to apply a force, $\mathbf{f}_{DC}$, to the Lagrangian solid at the mixed dual cell, we need to find an equivalent set of forces, $\mathbf{F}$, on the solid's degrees of freedom. Consider an arbitrary virtual displacement δX of the solid nodes. By equation (5.3), the corresponding virtual displacement at the dual cell is $\delta X_{DC} = W_{DC}\delta X$. Equating the virtual work done by $\mathbf{F}$ and $\mathbf{f}_{DC}$ for the virtual displacement δX, we get

$$\mathbf{F} \cdot \delta X = \mathbf{f}_{DC} \cdot \delta X_{DC} = \mathbf{f}_{DC} \cdot W_{DC}\delta X,$$

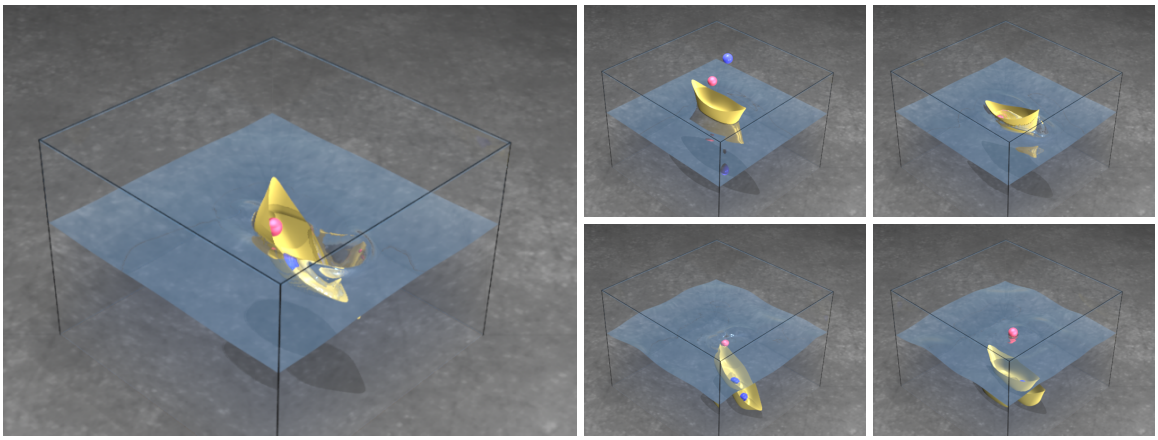


Figure 5.4: A light sphere and then a heavy sphere are dropped into a thin shell rigid boat floating in a pool of water ($160 \times 120 \times 160$ fluid grid). The light sphere barely rocks the boat, while the heavy sphere sinks it, and the light sphere bobs back to the surface.

from which it follows that

$$\mathbf{F} = W_{DC}^T \mathbf{f}_{DC}. \tag{5.4}$$

Thus whenever a force or impulse is applied at the dual cell, equation (5.4) gives the expression for the equivalent force or impulse on the solid nodes. See [49] for a discussion of virtual work and equivalent forces.

5.4 Conservation of Momentum

5.4.1 Fluid Momentum

Consider an x-component dual cell indexed by $i+1/2, j, k$. If the dual cell was fully fluid, the momentum update equation (in control volume form) for that dual cell would be

$$m_i u_{i+1/2}^{n+1} - m_i u_{i+1/2}^{\star} + \Delta t(A_{i+1} p_{i+1} - A_i p_i) = 0,$$

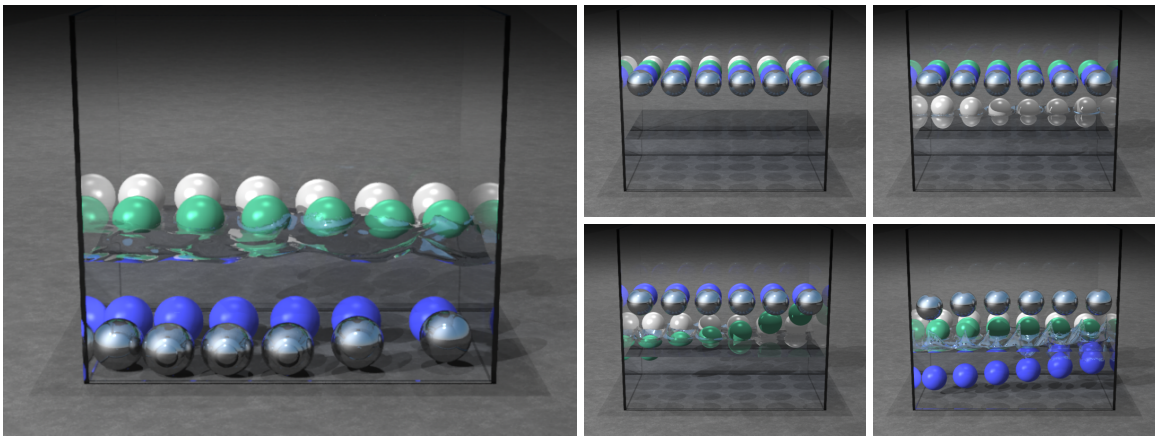


Figure 5.5: 24 rigid spheres of varying density are dropped into a pool of water, demonstrating scalability to many bodies ($225 \times 300 \times 150$ fluid grid).

where m_i is the mass of the fluid in the dual cell, $u^{\star}_{i+1/2}$ is the intermediate velocity (see Section 4.3), A_i is the area of the dual cell face at i, and the subscripts for the y and z dimensions have been omitted for brevity. Now assume that the dual cell is mixed, containing both fluid and solid. The update equation for the fluid in that dual cell is then given by

$$m_i u^{n+1}_{i+1/2} - m_i u^{\star}_{i+1/2} + \Delta t(A_{i+1}p_{i+1} - A_i p_i) + I_{i+1/2} = 0, \tag{5.5}$$

where $I_{i+1/2}$ is an impulse representing the net momentum exchanged between the fluid and solid. Note that A_i and A_{i+1} may be between 0 or the full face area, $\Delta y \Delta z$, depending on how much of the face is occupied by the fluid. In our implementation, we approximate A_i as

$$A_i = \begin{cases} A, & \text{for } p_i \text{ in the fluid} \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

where A is a constant giving the area of a face of the uniform grid. Let V be the constant giving the volume of a dual cell, so that $V = \Delta x A$. We then write equation (5.5) as

$$m_i u^{n+1}_{i+1/2} - m_i u^{\star}_{i+1/2} + \Delta t V G_{i+1/2} p + I_{i+1/2} = 0. \tag{5.7}$$

where $G_{i+1/2}$ denotes a gradient operator taking into account the criteria in (5.6).

Next, we write down the vector form of equation (5.7) for all the fluid velocities as

$$M_F\mathbf{u}^{n+1} - M_F\mathbf{u}^\star + \Delta t V\mathbf{G}p + \mathbf{I} = 0 \tag{5.8}$$

where M_F is the diagonal mass matrix for the fluid, $\mathbf{u}$ is the vector of all the fluid dual cell velocities, $\mathbf{G}$ is the gradient operator for all the dual cells, and $\mathbf{I}$ is the vector containing all the momentum quantities exchanged between the solid and fluid in the mixed dual cells (or zero for fully fluid dual cells).

5.4.2 Solid Momentum

Ignoring the solid/fluid interaction, the solid momentum equation is given by

$$M_S\mathbf{V}^{n+1} = M_S\mathbf{V}^\star + \Delta t D\mathbf{V}^{n+1},$$

where M_S is the mass matrix of the solid, $\mathbf{V}^\star$ incorporates explicitly integrated forces, and D is the symmetric coefficient matrix for the implicitly integrated damping forces.

Now consider a mixed dual cell, again at $i+1/2, j, k$. From equation (5.7), the fluid loses a momentum quantity $I_{i+1/2}$, and thus the solid gains the equivalent momentum quantity, given by equation (5.4), $W^T_{i+1/2}I_{i+1/2}$. We define W to be the matrix whose non-zero rows are the row vectors W_{DC} of equation (5.3). Taking into account the solid/fluid interaction through the mixed dual cells, the solid momentum equation is then given by

$$M_S\mathbf{V}^{n+1} = M_S\mathbf{V}^\star + \Delta t D\mathbf{V}^{n+1} + W^T\mathbf{I}. \tag{5.9}$$

Solving for $\mathbf{I}$ in equation (5.8) and substituting into equation (5.9), we get

$$M_S\mathbf{V}^{n+1} = M_S\mathbf{V}^\star + \Delta t D\mathbf{V}^{n+1} - W^T(M_F\mathbf{u}^{n+1} - M_F\mathbf{u}^\star + \Delta t V\mathbf{G}p).$$

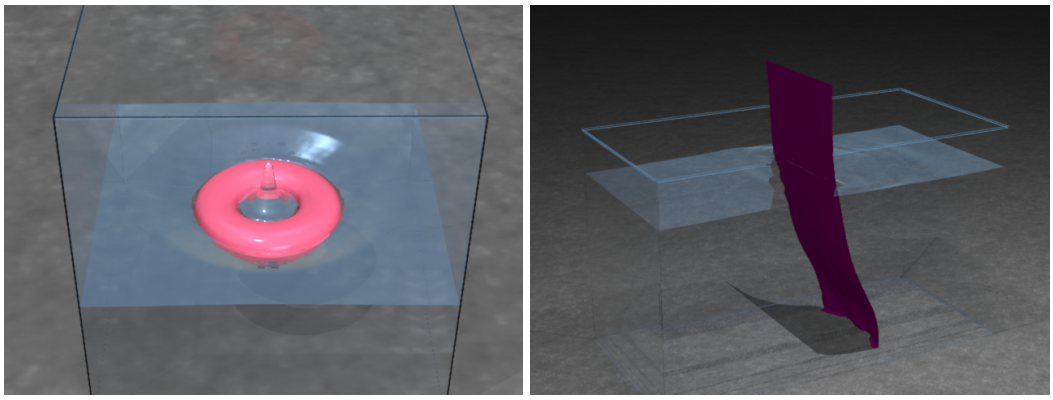


Figure 5.6: Our method supports coupling to both volumetric and thin deformable solids. Left: a soft torus is dropped into a pool of water, showing two-way coupling with a deformable body ($100 \times 100 \times 100$ fluid grid, 44k tetrahedra). Right: sheet of cloth is pulled out of a tank of water ($140 \times 140 \times 70$ fluid grid, 2.5k triangles).

Rearranging and using the fact that $W^T M_F \mathbf{u}^{n+1} = W^T M_F W \mathbf{V}^{n+1}$, we have

$$(M_S + W^T M_F W)\mathbf{V}^{n+1} + W^T \Delta t V \mathbf{G} p = M_S \mathbf{V}^\star + \Delta t D \mathbf{V}^{n+1} + W^T M_F \mathbf{u}^\star.$$

We regard $M_S + W^T M_F W$ as the combined lumped mass of the solid and fluid in the mixed dual cells, and denote it by $\tilde{M}_S$. Moving all the unknowns to the left hand side, we get

$$(\tilde{M}_S - \Delta t D)\mathbf{V}^{n+1} + W^T \Delta t V \mathbf{G} p = M_S \mathbf{V}^\star + W^T M_F \mathbf{u}^\star. \tag{5.10}$$

as the final form for the solid momentum equation.

Combined Solid/Fluid Mass

In [69], equation (5.10) was derived by considering the lumping of the fluid mass and momentum onto the solid separately, rather than substituting the value $\mathbf{I}$ from equation (5.8). In particular, $\tilde{M}_S$ was determined by lumping the fluid mass m_{DC} in each mixed dual cell onto the solid nodes using $W_{DC} m_{DC}$, resulting in a diagonal matrix rather than the symmetric expression above. All of the examples discussed below were run with the diagonal $\tilde{M}_S$.

Solid Time Integration

The time integration for the solid can be explicit, semi-implicit, or fully implicit. For fully explicit time integration of the solid, such as in the rigid body approach of [47] or [7], $D = 0$ and all forces are accounted for in $\mathbf{V}^\star$. This is also true for deformable objects if both elastic and damping forces are handled explicitly. In the semi-implicit approach of [10] the elastic forces are handled explicitly but the damping forces are handled implicitly, and D is a symmetric negative definite matrix. Our formulation also allows for a fully implicit approach, as in [4], where D is a linearization of both the elastic and damping forces (notably, in the [4] approach D is also symmetric, and thus we will assume here that it is symmetric). Note when using the fully implicit scheme that D represents the linearization for one Newton-Raphson iteration, and thus repeated solves are required if more iterations are necessary.

5.5 Linear System

The system of linear equations for the fluid is typically derived by enforcing incompressibility via

$$\nabla \cdot \mathbf{u}^{n+1} = 0 \tag{5.11}$$

where in our formulation the components u_{DC} of $\mathbf{u}$ are given by

$$u_{DC}^{n+1} = \begin{cases} u_{DC}^\star - \Delta t G_{DC} p/\rho & \text{all fluid dual cell} \\ W_{DC}\mathbf{V}^{n+1} & \text{mixed solid/fluid dual cell} \end{cases} \tag{5.12}$$

This is identical to a voxelized solid formulation, except that in the latter $W_{DC}\mathbf{V}^{n+1}$ is replaced with the known solid velocity. For convenience, we use a scaled pressure of $\hat{p} = p\Delta t$ and for symmetry we scale the divergence by V. Using Equations (5.10), (5.11)

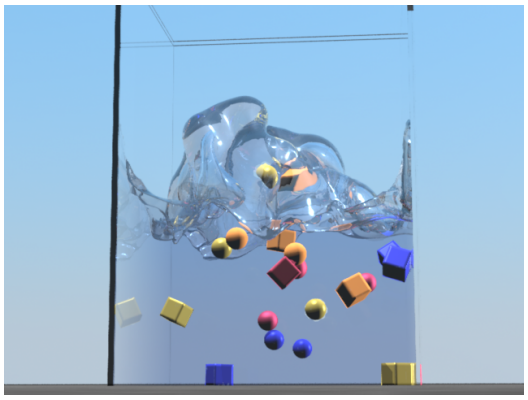


Figure 5.7: Many rigid bodies circulate in a turbulent fountain, demonstrating scalability and dynamic interactions between rigid bodies (i.e. collisions) ($100 \times 150 \times 100$ fluid grid).

and (5.12) we write our linear system as

$$\begin{pmatrix} V\mathbf{G}^T \frac{1}{\rho}\mathbf{G} & -V\mathbf{G}^T W \\ -W^T V\mathbf{G} & -\tilde{M}_S + \Delta t D \end{pmatrix} \begin{pmatrix} \hat{p} \\ \mathbf{V}^{n+1} \end{pmatrix} = \begin{pmatrix} V\mathbf{G}^T \mathbf{u}^\star \\ -M_S \mathbf{V}^\star - W^T M_F \mathbf{u}^\star \end{pmatrix} \tag{5.13}$$

where $\mathbf{G}$ is the gradient operator and $\mathbf{G}^T$ is minus one times the divergence operator.

5.5.1 Solving the Linear System

Independently, the solid and fluid systems can be made at least symmetric positive semidefinite, but the coupled system (5.13) is symmetric indefinite. However, it can still be solved quite efficiently with MINRES [65], since it is symmetric. We use an Incomplete Cholesky preconditioner for the pressure rows of our system, and a block diagonal mass-inverse preconditioner for the solid velocity rows. See [69] for further discussion of the solution of related indefinite systems.

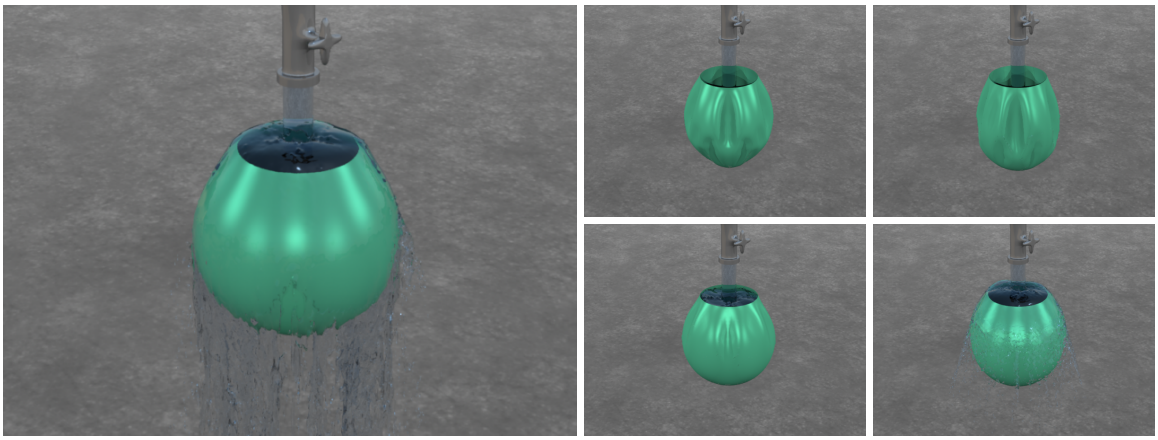


Figure 5.8: A stream of water pours into an elastic cloth bag suspended by its rim (100 $\times$ 375 $\times$ 100 fluid grid, 1k triangles). The bag deforms under the impact of the water and then recovers, filling and expanding until the water overflows and runs down its sides.

5.6 Time Integration

Our two-way coupled time integration scheme hybridizes fluid evolution with a Newmark method for solid integration [10]. Typically Newmark iteration requires one to solve a linear system for the solid velocities twice per time step, and these solves are replaced with Equation (5.13). The first coupled solve is done with the positions frozen at time t^n and thus all fluid forces are used except convection (i.e. except $\mathbf{u} \cdot \nabla \mathbf{u}$). The second solve is used to update the momentum, and there convection *is* applied. Our method proceeds as follows:

F1: Use all non-pressure based and non-advection based fluid forces (i.e. external forces and viscosity) to advance the fluid velocity to time $t^{n+1/2}$,

$$\mathbf{u}^{n+1/2} = \mathbf{u}^n + (\Delta t/2)(\mathbf{f} + \nu \Delta \mathbf{u}^{n+1/2}). \tag{5.14}$$

S1: Integrate all explicit solid forces to time $t^{n+1/2}$, $\mathbf{V}^n \rightarrow V^{n+1/2\star}$. Solve Equation (5.13) for the coupled system to obtain $\mathbf{V}^{n+1/2}$. The resulting $\mathbf{V}^{n+1/2}$ is then used to update the solid positions, for collisions, etc., just as in a standard Newmark algorithm (e.g. as described in Chapter 3). This step follows the standard position update procedures for

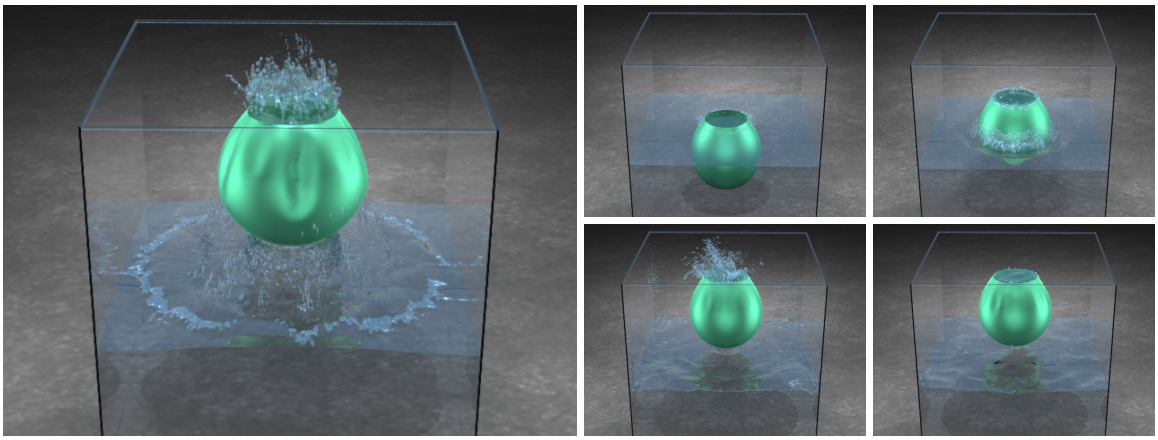


Figure 5.9: An elastic cloth bag is submerged in and then quickly pulled from a pool of water, carrying fluid with it ($140 \times 210 \times 140$ fluid grid, 1k triangles). The bag bounces as it is raised, expanding and contracting, which causes water to splash out. The bag eventually settles.

deformable bodies and rigid bodies, except that the velocity solve incorporates the fluid pressure. After the solve, the fluid pressure and the results of F1 are discarded.

F2: Following [33], we prevent leaking by forcing the fluid to move with the solid effective velocity, calculated as the change in the position of the solid during S1. A standard fluid Poisson equation is solved using the solid effective velocity mapped onto the Eulerian grid by W as Neumann boundary conditions to project $\mathbf{u}^n$. The resulting projected velocity is our leak-proof advection velocity $\mathbf{u}_{ADV}$.

F3: We calculate the intermediate fluid velocity via

$$\frac{(\mathbf{u}^\star - \mathbf{u}^n)}{\Delta t} + \mathbf{u}_{ADV} \cdot \nabla \mathbf{u}^n = \mathbf{f} + \nu \Delta \mathbf{u}^\star. \tag{5.15}$$

Note that the advection velocity $\mathbf{u}_{ADV}$ is used to formulate the rays in the typical semi-Lagrangian scheme [80], but the advected quantity is the actual fluid velocity $\mathbf{u}^n$. $\mathbf{u}_{ADV}$ is also used to advect all other fluid scalar quantities to time t^{n+1}. Since this advection velocity exactly conforms to the effective velocity of the solid, it prevents leaking.

S2: Integrate all explicit solid forces to time t^{n+1}, $\mathbf{V}^n \rightarrow \mathbf{V}^\star$. Solve Equation (5.13) to find

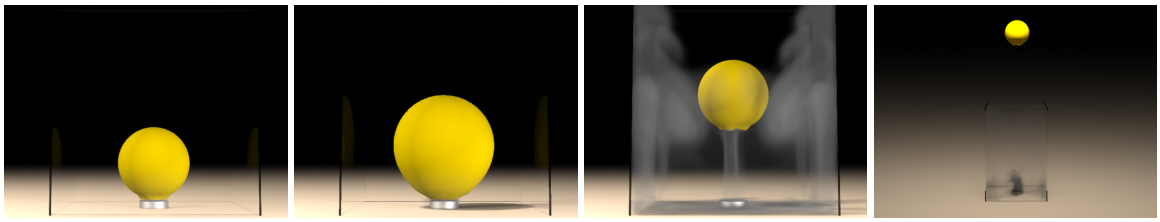


Figure 5.10: A balloon is filled by a jet of fast-moving smoke, reaching a state of strong tension ($100 \times 150 \times 100$ fluid grid, 1k triangles). The balloon is then released and expels smoke at a very high speed, accelerating upwards, and passes through the edge of the domain without any discontinuities. The high tension and fast velocities highlight the stability of our fully coupled method.

$\mathbf{V}^{n+1}$, and otherwise carry out the standard algorithms for deformable and rigid bodies (see Chapter 3). Note that this time the fluid pressure is not discarded.

F4: Using the fluid pressure from S2, project the intermediate fluid velocity $\mathbf{u}^\star$ to be divergence free, $\mathbf{u}^\star \rightarrow \mathbf{u}^{n+1}$.

In summary, there are three implicit solves, two for the coupled system (using MINRES) and one for the fluid (using conjugate gradients). This is the typical situation even in interleaved computations as Newmark time integration requires two conjugate gradient solves and fluids require one. The additional cost here is that the two conjugate gradient solves for the solid have the added coupling to the fluid in the linear system and now require MINRES due to indefiniteness.

5.7 Examples

We demonstrate coupling of volumetric and thin shell rigid and deformable bodies to both water and smoke. Our examples were simulated in parallel on four to sixteen processors across a number of four processor Opteron machines and took anywhere from a few minutes per frame to several hours in some extremely high velocity cases. The solid was simulated on a single processor and the fluid was split up across the remaining processors.

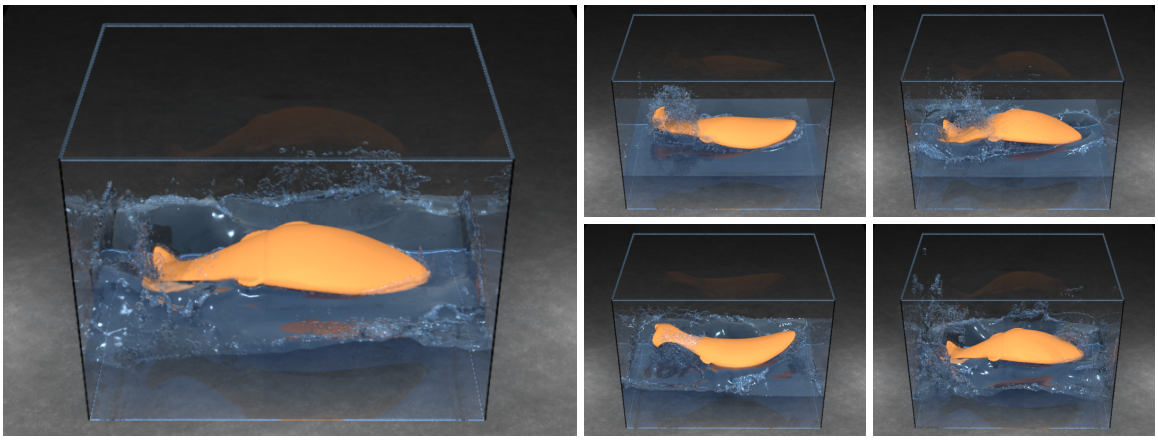


Figure 5.11: Water is two-way coupled to a fish with an embedded proportional derivative controlled articulated skeleton and a deformable exterior ($192 \times 216 \times 144$ fluid grid, 42k tetrahedra).

Figures 5.1 and 5.5 demonstrate coupling to rigid bodies with varying density ratios. Figure 5.6 (left) shows a volumetric deformable object coupled to water. Figures 5.6 (right), 5.8 and 5.9 show cloth coupled to water. Figure 5.10 shows stability under high tension and high velocities and coupling of cloth to smoke. Figures 5.5 and 5.7 show scalability to large numbers of objects. Figure 5.4 shows interaction between volumetric and thin shell rigid bodies and water. Figure 5.11 offers an example of the potential for combining our method with other state-of-the-art simulation techniques.

Bibliography

[1] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):617–625, 2005.

[2] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *Proc. of ACM SIGGRAPH 1996*, pages 137–146, 1996.

[3] D. Baraff and A. Witkin. Partitioned dynamics. Technical report, Carnegie Mellon University, 1997.

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

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

[6] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 26(3), 2007.

[7] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 26(3):100, 2007.

[8] J. Bender. Impulse-based dynamic simulation in linear time. *Computer Animation and Virtual Worlds*, 18:225–233, 2007.

[9] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, 2002.

[10] 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.

[11] 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.

[12] 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.

[13] M. Carlson, P. Mucha, R. Van Horn, and G. Turk. Melting and flowing. In *ACM Trans. Graph. (SIGGRAPH Proc.)*, volume 21, pages 167–174, 2002.

[14] 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.

[15] N. Chentanez, T. G. Goktekin, B. Feldman, and J. O'Brien. Simultaneous coupling of fluids and deformable bodies. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 325–333, 2006.

[16] B. Criswell, K. Derlich, and D. Hatch. Davy jones' beard: rigid tentacle simulation. In *SIGGRAPH 2006 Sketches*, page 117, 2006.

[17] 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.

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

[19] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *ACM Trans. Graph. (SIGGRAPH Proc.)*, pages 251–260, 2001.

[20] François Faure, Jérémie Allard, and Matthieu Nesme. Eulerian contact for versatile collision processing. Technical report, INRIA, 2007. http://hal.inria.fr/inria-00149706.

[21] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152:457–492, 1999.

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

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

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

[25] N. Galoppo, M. Otaduy, S. Tekin, M. Gross, and M. C. Lin. Soft articulated characters with fast contact handling. *Comput. Graph. Forum (Proc. Eurographics)*, 26(3):243–253, 2007.

[26] M.-P. Gascuel. An implicit formulation for precise contact modeling between flexible solids. In *Proc. SIGGRAPH 93*, pages 313–320, 1993.

[27] O. Génevaux, A. Habibi, and J.-M. Dischler. Simulating fluid-solid interaction. In *Graph. Interface*, pages 31–38, June 2003.

[28] M. Gissler, M. Becker, and M. Teschner. Local constraint methods for deformable objects. In *Proc. of the 3rd Workshop in VR Interactions and Physical Simulation (VRIPHYS)*, pages 1–8, 2006.

[29] 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.

[30] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient simulation of inextensible cloth. *ACM Trans. Graph.*, 26(3):49, 2007.

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

[32] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):871–878, 2003.

[33] 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.

[34] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Comput. Graph. Forum*, 20(3), 2001.

[35] J. Hodgins, W. Wooten, D. Brogan, and J. O'Brien. Animating human athletics. In *Proc. of SIGGRAPH '95*, pages 71–78, 1995.

[36] J.-M. Hong. *Visual Simulation of Fluids with Discontinuous State Variables*. PhD thesis, Korea University, June 2005.

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

[38] T. J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover, 2000.

[39] I. Ihm, B. Kang, and D. Cha. Animation of reactive gaseous fluids through chemical kinetics. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 203–212, 2004.

[40] G. Irving. *Methods for the physically-based simulation of solids and fluids*. PhD thesis, Stanford University, June 2007.

[41] 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.

[42] 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.

[43] J. Jansson and J.S.M. Vergeest. Combining deformable and rigid body mechanics simulation. *The Vis. Comput. J.*, 2003.

[44] C. A. Felippa K. C. Park and J. A. DeRuntz. Stabilization of staggered solution procedures for fluid-structure interaction analysis. In Belytshcko and Geers, editors, *Computational Methods for Fluid-Structure Interaction Problems*, volume 26, pages 94–124. ASME Applied Mechanics Symposia Series, AMD, 1977.

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

[46] R. Keiser, B. Adams, D. Gasser, P. Bazzi, P. Dutré, and M. Gross. A unified Lagrangian approach to solid-fluid animation. In *Eurographics Symp. on Point-Based Graph.*, 2005.

[47] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. In *ACM Trans. Graph. (SIGGRAPH Proc.)*, volume 25, pages 820–825, August 2006.

[48] A. Lamorlette and N. Foster. Structural modeling of flames for a production environment. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):729–735, 2002.

[49] Cornelius Lanczos. *The Variational Principles of Mechanics*. Dover, 1970.

[50] J. Lenoir and S. Fonteneau. Mixing deformable and rigid-body mechanics simulation. In *Comput. Graph. Int.*, pages 327–334, june 16-19 2004.

[51] F. Losasso, G. Irving, E. Guendelman, and R. Fedkiw. Melting and burning solids into liquids and gases. *IEEE Trans. on Vis. and Comput. Graph.*, 12(3):343–352, 2006.

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

[53] R. Loubère and E. J. Caramana. The force/work differencing of exceptional points in the discrete, compatible formulation of Lagrangian hydrodynamics. *J. Comput. Phys.*, 216:1–18, 2006.

[54] B. Merriman, J. Bence, and S. Osher. Motion of multiple junctions: A level set approach. *J. Comput. Phys.*, 112:334–363, 1994.

[55] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23:385–392, 2004.

[56] 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.

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

[58] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):471–478, 2005.

[59] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 141–151, 2004.

[60] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross. Interaction of fluids with deformable solids. *J. Comput. Anim. and Virt. Worlds*, 15(3–4):159–171, July 2004.

[61] M. Müller, M. Teschner, and M. Gross. Physically-based simulation of objects represented by surface meshes. In *Proc. Comput. Graph. Int.*, pages 156–165, June 2004.

[62] D. Nguyen, R. Fedkiw, and H. Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:721–728, 2002.

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

[64] J. F. O'Brien, V. B. Zordan, and J. K. Hodgins. Combining active and passive simulations for secondary motion. *IEEE Comput. Graph. Appl.*, 20, 2000.

[65] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[66] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross, and L. Guibas. Meshless animation of fracturing solids. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):957–964, 2005.

[67] S. Piperno and C. Farhat. Partitioned procedures for the transient solution of coupled aeroelastic problems - part ii: Energy transfer analysis and three-dimensional applications. *Computer Methods in Applied Mechanics and Engineering*, 190:3147–3170, 2001.

[68] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 193–202, 2004.

[69] A. Robinson-Mosher, T. Shinar, J. Gretarsson, J. Su, and R. Fedkiw. Two-way coupling of fluids to rigid and deformable solids and shells, 2008.

[70] S. Ruuth. A diffusion-generated approach to multiphase motion. *J. Comput. Phys.*, 145:166–192, 1998.

[71] C. Schroeder. PhD thesis, Stanford University, June 2011.

[72] A. Selle, J. Su, G. Irving, and R. Fedkiw. Highly detailed folds and wrinkles for cloth simulation. *IEEE Trans. on Vis. and Comput. Graph. (In Press)*, 2007.

[73] T. Shinar, C. Schroeder, and R. Fedkiw. Two-way coupling of rigid and deformable bodies. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 2008.

[74] E. Sifakis. *Algorithmic aspects of the simulation and control of computer generated human anatomy models*. PhD thesis, Stanford University, June 2007.

[75] E. Sifakis, K. Der, and R. Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim. (in press)*, 2007.

[76] E. Sifakis, I. Neverov, and R. Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3), 2005.

[77] E. Sifakis, A. Selle, A. Robinson-Mosher, and R. Fedkiw. Simulating speech with a physics-based facial muscle model. *ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 261–270, 2006.

[78] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw. Hybrid simulation of deformable solids. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 81–90, 2007.

[79] K. Smith, F. Solis, and D. Chopp. A projection method for motion of triple junctions by level sets. *Interfaces and Free Boundaries*, 4(3):263–276, 2002.

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

[81] D. Steinemann, M. A. Otaduy, and M. Gross. Fast arbitrary splitting of deforming objects. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 63–72, 2006.

[82] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. Robust quasistatic finite elements and flesh simulation. *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 181–190, 2005.

[83] D. Terzopoulos, J. Platt, and K. Fleischer. Heating and melting deformable models (from goop to glop). In *Graph. Interface*, pages 219–226, 1989.

[84] L. Vese and T. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *Int. J. of Comput. Vision*, 50(3):271–293, 2002.

[85] R. Weinstein. *Simulation and Control of Articulated Rigid Bodies*. PhD thesis, Stanford University, June 2007.

[86] R. Weinstein, E. Guendelman, and R. Fedkiw. Impulse-based control of joints and muscles. *IEEE Trans. on Vis. and Comput. Graph.*, 14(1):37–46, 2008.

[87] R. Weinstein, J. Teran, and R. Fedkiw. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Trans. on Vis. and Comput. Graph.*, 12(3):365–374, 2006.

[88] M. Wicke, D. Steinemann, and M. Gross. Efficient animation of point-sampled thin shells. In *Proc. of Eurographics*, volume 24, 2005.

[89] A. Witkin and M. Kass. Spacetime constraints. In *Comput. Graph. (Proc. SIGGRAPH '88)*, volume 22, pages 159–168, 1988.

[90] G. Yngve, J. O'Brien, and J. Hodgins. Animating explosions. In *Proc. of ACM SIGGRAPH 2000*, pages 29–36, 2000.

[91] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 26(3):99, 2007.

[92] H.-K. Zhao, T. Chan, B. Merriman, and S. Osher. A variational level set approach to multiphase motion. *J. Comput. Phys.*, 127:179–195, 1996.