PHYSICALLY-BASED SIMULATION OF SOLIDS AND SOLID-FLUID COUPLING

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

Eran Guendelman
June 2006

Abstract

This dissertation presents algorithms for the simulation of solids and fluids, and for two-way coupling between the two. Physically-based simulation has a wide range of applications, and the focus in this dissertation is on creating visually plausible animation for computer graphics and visual effects.

Novel techniques for rigid body simulation are described first. These include a new approach to time integration, merging it with the collision and contact processing algorithms in a fashion that obviates the need for *ad hoc* threshold velocities. In addition, a novel shock propagation algorithm allows for efficient use of the propagation (as opposed to the simultaneous) method for treating contact. Examples are given involving many nonconvex rigid bodies undergoing collision, contact, friction, and stacking. An overview of existing techniques for simulation of thin deforming shells (cloth) is also given.

A basic fluid simulator is described next, including techniques for both smoke and free-surface water, following which a novel method is proposed for coupling infinitesimally thin solids to the fluid. The proposed method works for both rigid and deformable shells. Leaks across the solid are prevented by using robust ray casting and visibility. Incompressibility is properly enforced at the solid-fluid interface so that fluid does not incorrectly compress (and appear to lose mass). Furthermore, computation of a smoother pressure for coupling alleviates some of the stiffness associated with coupling to an incompressible fluid. Coupling to volumetric solids is also described, and a more accurate approach to computing the coupling pressure is suggested. Our framework treats the solids simulator as a black box, allowing any alternative simulator to replace our choice.

Acknowledgements

I would first like to thank my loving parents, Eduardo and Pnina, for all of their support and encouragement. My sisters, Sharon and Yael, have been extremely supportive, too, and I only wish I had been around more over the last ten years to witness as they've matured and found success in their own lives.

I want to thank my adviser, Ron Fedkiw, for taking a "chance" on me as his first Computer Science students (the previous students coming from the SCCM program). Ron has a wealth of knowledge in many areas of science, and I learned a lot from him while pursuing my Ph.D. Ron is incredibly hard working, and passionate about his work. He strives to make each of his students achieve their full potential, and I appreciate all of the work he has put in to get me to where I am today.

It has been a pleasure working alongside everyone in Ron's research group. It is an exceptionally tight-nit and collaborative group, and is full of incredibly smart people. I would particularly like to acknowledge my coauthors, Robert Bridson, Andrew Selle, Frank Losasso, Geoffrey Irving, Rachel Weinstein, and of course Ron. It was especially enjoyable to reunite and collaborate with fellow Waterloo alum Robert Bridson. I would also like to acknowledge the other group members, past and present, for helping to make this such a high-caliber group and for bringing life to the SIGGRAPH deadlines. They are Josh Bao, Douglas Enright, Frederic Gibou, Jeong-Mo Hong, Sergey Koltakov, Ian Mitchell, Neil Molino, Igor Neverov, Duc Nguyen, Avi Robinson-Mosher, Tamar Shinar, Eftychios Sifakis, Robert Strzodka, Jerry Talton, and Joseph Teran. A number of undergraduate and coterminal students helped out during deadlines, and I would particularly like to acknowledge Eilene Hao, Cynthia Lau, Unnur Gretarsdottir, and Jiayi Chong. Everyone mentioned was also involved in developing our C++ library for physically-based modeling (PhysBAM), which incorporates all of our research work including the work described here.

Thanks go out to everyone else in the Stanford graphics lab, and I wish to additionally acknowledge the lab for making available the Buddha model used in one of our simulations.

I would like to thank Leonidas Guibas and Pat Hanrahan for agreeing to join Ron on my reading committee, and Scott Delp and Charles Taylor for joining them during my defense. I appreciate all of the positive feedback they have provided.

I was fortunate to be given the opportunity to consult at Industrial Light + Magic during part of my graduate studies, and I wish to thank everyone there (especially Nick Rasmussen, Ryan Kautzman, and Brice Criswell) for making it such a great experience.

Last, but not least, thank you to all of my friends at Stanford for the fun times, and for putting up with me during those crazy SIGGRAPH deadlines. Thanks, also, to my friends back home and abroad with whom I still manage to keep a connection despite the distance. Finally, to Aviva, thank you for the time we had together.

Contents

List of Figures

Chapter 1

Introduction

Computer simulations of physical phenomena have been in use since the early days of computer technology. Over the years, increasing computational power has allowed for improvements in both model complexity and solution accuracy. At the same time, this has helped bring high quality simulation to the desktop computer and to a wide variety of disciplines. Physically-based simulation started making its way into computer graphics in the late 1980's, and these days it is commonly used for animation and visual effects in the entertainment industry.

This dissertation focuses on simulations of solids and fluids for computer graphics. In targeting computer graphics rather than scientific simulation, we make a number of compromises. First, we place more emphasis on visually *plausible* results rather than on results matching experimental and theoretical data. We also care more about algorithm simplicity (and extensibility) rather than on choosing the highest order accurate method available. Finally, we aim for a general approach rather than a problem-specific approach or one requiring extensive manual intervention to set up. Despite these considerations, we expect that many of the techniques described in this dissertation could be adapted by the engineering community into their more demanding simulation environment. Scientific applications would require further validation and error analysis, and Section 8.6 describes possible future work in this direction.

A computer animator uses physically-based simulation as a tool to create complex animations that would otherwise be hard to create by hand. For example, the animator may use rigid body simulation to visualize a collapsing structure, deformable body simulation to model skin and hair, and fluid simulation to generate water waves and nuclear explosions.

It is worth additionally noting the variety of *scientific* applications in which solid and fluid simulations (often coupled) play a key role. These include:

- Simulation of the human musculoskeletal system
- Aerodynamic simulation for design and testing of aircrafts and parachutes
- Investigation of animal propulsion for flying and swimming
- Simulation of blood flow through the heart and arteries

This dissertation is structured as follows. Chapter 2 describes a novel rigid body simulator that is able to handle many nonconvex bodies undergoing collision, contact, friction, and stacking. Chapter 3 follows with a brief description of the cloth simulator of [17], which we use for some of our two-way coupled examples. Chapter 4 describes a basic fluid simulator for smoke and water which uses a novel node-based discretization. Chapter 5 discusses some of the basic concepts underlying coupled solid-fluid simulations. The subsequent two chapters describe coupling fluids to both thin shells (Chapter 6) and volumetric solids (Chapter 7). A number of key contributions are presented, including a scheme that prevents leaks across thin shells, and novel approaches to reducing fluid mass loss and improving coupling stability. Chapter 8 wraps up with conclusions and some directions for future work.

Much of the content presented in this dissertation is extracted from previous publications by myself and coauthors [51, 52, 83], and only parts I was directly involved with have been included here.

Chapter 2

Rigid Body Simulation

2.1 Introduction

One can differentiate between highly deformable volumetric objects and those where the deformation is either negligible or unimportant, and in the latter case efficiency concerns usually lead to rigid body approximations. [22] notes the weakness of the rigid body approximation to solids and discusses some known flaws in state of the art collision models. Moreover, they discuss some common misconceptions mentioning for example that the coefficient of restitution can be greater than one in frictional collisions. [129] emphasizes the difficulties with nonunique solutions pointing out that it is often impossible to predict which solution occurs in practice since it depends on unavailable details such as material microstructure. He states that one should repeat the calculations with random disturbances to characterize the potential set of solutions. [13] exploited this indeterminacy by adding random texture and structured perturbations to enrich and control the motion respectively. The goal of rigid body simulation then becomes the construction of plausible motion instead of predictive motion. While the physicist strives towards synthesizing a family of solutions that are predictive in the sense that they statistically represent experimental data, the interest in graphics is more likely to focus on obtaining a particularly appealing solution from the set of plausible outcomes, e.g. see [24, 113].

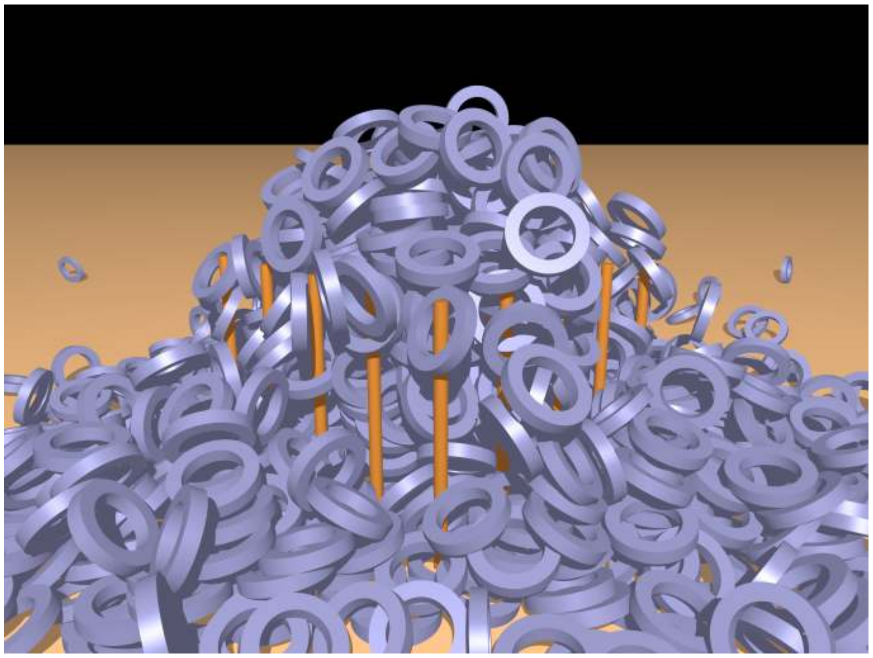

Figure 2.1: **1000 nonconvex rings (with friction) settling after being dropped onto a set of 25 poles. Each ring is made up of 320 triangles, and each pole is made up of 880 triangles.**

With this in mind, we focus on the plausible simulation of nonconvex rigid bodies emphasizing large scale problems with many frictional interactions. Although we start with a triangulated surface representation of the geometry, we also construct a signed distance function defined on a background grid (in the object frame) enabling the use of fast inside/outside tests. The signed distance function also conveniently provides a normal direction at points on and near the surface of the object. We take a closer look at the usual sequence of simulation steps—time integration, collision detection and modeling, contact resolution—and propose a novel approach that more cleanly separates collision from contact by merging both algorithms more tightly with the time integration scheme. This removes the need for *ad hoc* threshold velocities used by many authors to alleviate errors in the contact and collision algorithms, and correctly models difficult frictional effects without requiring microcollision approximations [93, 94] (which also use an *ad hoc* velocity). We also introduce a novel algorithm that increases the efficiency of the propagation method for contact resolution. The efficiency and robustness of our approach is illustrated with a number of simple and complex examples including frictional interactions and large contact groups as is typical in stacking.

The content of this chapter is taken from our publication [51], with some expanded detail (e.g. Section 2.3.1 on creating an implicit surface from a triangulated surface), and a slightly different approach for computing the contact graph (Section 2.8.1).

2.2 Previous Work

[54] considered rigid body collisions by processing the collisions chronologically backing the rigid bodies up to the time of impact. [92] used a timewarp algorithm to back up just the objects that are involved in collisions while still evolving non-colliding objects forward in time. This method works well except when there are a large number of bodies in contact groups, which is the case we are concerned with here. [54] processed collisions with static friction if the result was in the friction cone, and otherwise used kinetic friction. If the approach velocity was smaller than a threshold, the objects were assumed to be in contact and the same equations were applied approximating continuous contact with a series of "instantaneous contacts". [97] instead proposed the use of repulsion forces for contact only using the exact impulse-based treatment for high velocity collisions.

[4] proposed a method for analytically calculating non-colliding contact forces between polygonal objects obtaining an NP-hard quadratic programming problem which was solved using a heuristic approach. He also points out that these ideas could be useful in collision propagation problems such as one billiard ball hitting a number of others (that are lined up) or objects falling in a stack. [5] extended these concepts to curved surfaces. [6] advocated finding either a valid set of contact forces or a valid set of contact impulses stressing that the usual preference of the latter only when the former does not exist may be misplaced. For more details, see [7]. [8] proposed a simpler, faster, and more robust algorithm for solving these types of problems without the use of numerical optimization software. [107] modeled

local surface deformation for contacting "quasi-rigid" bodies represented using point clouds.

[16] discussed the nonlinear differential equations that need to be numerically integrated to analyze the behavior of three-dimensional frictional rigid body impact and pointed out that the problem becomes ill-conditioned at the sticking point. Then they use analysis to enumerate all the possible post-sticking scenarios and discuss the factors that determine a specific result. [94] integrated these same nonlinear differential equations to model both contact and collision proposing a unified model (as did [54]). They use the velocity an object at rest will obtain by falling through the collision envelope (or some threshold in [93]) to identify the contact case and apply a microcollision model that reverses the relative velocity as long as the required impulse lies in the friction cone. This solves the problem of blocks erroneously sliding down inclined planes due to impulse trains that cause them to spend time in a ballistic phase.

Implicitly defined surfaces were used for collision modeling by [136] to create repulsive force fields around objects and [108, 122] who exploited fast inside/outside tests. We use a particular implicit surface approach defining a signed distance function on an underlying grid. Grid-based distance functions have been gaining popularity, see e.g. [39, 56] who used them to treat collision between deformable bodies, and [71] who used them to keep hair from interpenetrating the head. Although [48] pointed out potential difficulties with spurious minima in concave regions, we have not noticed any adverse effects, most likely because we do not use repulsion forces.

[89] used position based physics to simulate the stacking of convex objects and discussed ways of making the simulations appear more physically realistic. [90] considered stacking with standard Newtonian physics using an optimization based method to adjust the predicted positions of the bodies to avoid overlap. One drawback is that the procedure tends to align bodies nonphysically. Quadratic programming is used for contact, collision and the position updates. They consider up to 1000 frictionless spheres, but nonconvex objects can only be considered as unions of convex objects and they indicate that the computational cost scales with the number of convex pieces. Their only nonconvex example considered 50 jacks that were each the

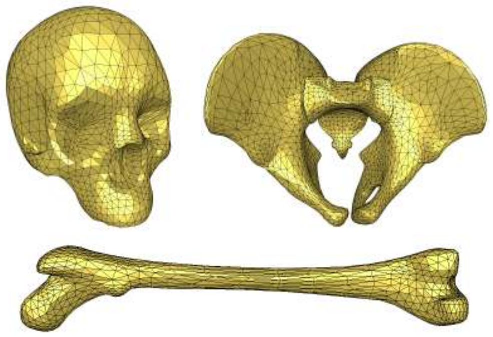

Figure 2.2: **Some nonconvex geometry from our simulations: the cranium, pelvis and femur have 3520, 8680, and 8160 triangles respectively.**

union of 3 boxes. [120] described a freezing technique that identifies when objects can be removed from the simulation, as well as identifying when to add them back. This allows the stacking of 1000 cubes with friction in 1.5 days as opposed to an estimated 45 days for simulating all the cubes. In a more recent paper, [70] proposed a linear time algorithm that used optimization to resolve all frictional contact for each body individually rather than iterating through pairwise contacts. Using this efficient approach they were able to produce fast simulations of 1000 stacking, non-convex rigid bodies, but with only approximate momentum conservation. Large scale simulations of granular materials, consisting of rigid grains made up of spheres, were recently presented by [14].

2.3 Geometric Representation

Since rigid bodies do not deform, they are typically represented with triangulated surfaces, see Figure 2.2. In cases where a rigid body is originally represented volumetrically, surface meshing methods such as marching cubes [81], dual contouring [67], or even the tetrahedral meshing of [96] (keeping only the surface mesh) may be used to obtain a triangulated surface.

Starting with either the density (or mass), algorithms such as [91] can then be used to compute the volume, mass and moments of inertia. For efficiency, we store the object space representation with the center of mass at the origin and the axes aligned with the principal axes of inertia resulting in a diagonal inertia tensor simplifying many calculations, e.g. finding its inverse.

In addition to a triangulated surface, we also store an object space signed distance function for each rigid body. This is stored on either a uniform grid [105] or an octree grid [44] depending on whether speed or memory, respectively, is deemed to be the bottleneck in the subsequent calculations. Discontinuities across octree levels are treated by constraining the fine grid nodes at gradation boundaries to take on the values dictated by interpolating from the coarse level, see e.g. [146]. We use negative values of ϕ inside the rigid body and positive values of ϕ outside so that the normal is defined as $\mathbf{n} = \nabla\phi$. This embedding provides approximations to the normal throughout space as opposed to just on the surface of the object allowing us to accelerate many contact and collision algorithms. Section 2.3.1 describes in more detail the construction of a signed distance function from a triangulated surface.

Using both a triangulated surface and a signed distance function representation has many advantages. For example, one can use the signed distance function to quickly check if a point is inside a rigid body, and if so intersect a ray in the $\mathbf{n} = \nabla\phi$ direction with the triangulated surface to find the surface normal at the closest point. This allows the treatment of very sharp objects with their true surface normals, although signed distance function normals provide a smoother and less costly representation if desired. For more details on collisions involving sharp objects, see [106].

2.3.1 Constructing an Implicit Surface

A variety of techniques exist for computing a signed distance function from a triangulated surface, such as the closest point transform in [86], the level set generation method in [61], the volume construction phase of volumetric model repair in [66], and the method described in [2]. Our approach, while not a novel aspect of our work, is described next for completeness.

The rasterization procedure takes as input a closed triangulated surface with consistently oriented boundary triangles (all triangle normals point *outside*). Additionally, we assume a uniform grid is specified, although this approach can be easily

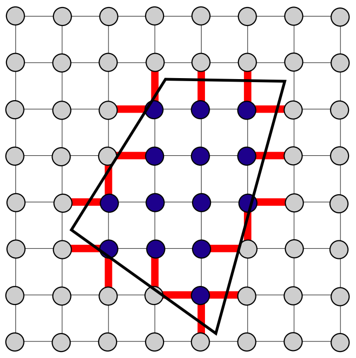

Figure 2.3: **Flood fill is used to find connected regions on the grid, with crossing edges (drawn thickened) denoting flood fill boundaries. In this example, two regions are found, indicated by the different node colors, and it is sufficient to check inside/outside for one representative node from each region.**

adapted to octrees by refining the octree near object boundaries. The signed distance function will be stored on the nodes of this grid, and as mentioned above a negative sign will denote inside, while a positive sign will denote outside. Note that this whole procedure is a one time cost in constructing a rigid body model.

First, all nodes on the grid need to be flagged as being either inside or outside the object. For a given node, this can be determined by casting a ray in an arbitrary direction and marking the node as inside only if that ray intersects the surface and lies on the interior relative to the intersected triangle. As noted in [2], special care must be taken if the ray intersects a vertex or an edge rather than the interior of a triangle. For robustness, we intersect against *thickened* triangles as in [17] (also used in Chapter 6).

Rather than ray casting from each node individually, a more efficient approach first computes connected regions in the grid using a flood fill algorithm. Since all nodes in a connected region have the same sign, it is sufficient to compute the sign of a single representative node for each such region (Figure 2.3). When computing connected regions, the flood fill spreads from a node to its edge-connected neighbors as long as the connecting edge does not intersect the surface. Similar to [66], edges which do intersect the surface are marked as *crossing* edges. One way to find crossing edges efficiently in a pre-processing step is by iterating over all triangles in the mesh, and only checking for intersections between a triangle and grid edges which overlap that triangle's axis-aligned bounding box.

For triangulated surfaces which are not strictly closed, for example containing gaps between some surface triangles, flood fill cannot be reliably used to compute connected regions. Furthermore, the ray intersection approach to determining whether a node is inside or outside is sensitive and error-prone, as the ray may pass through a gap. Of course, an object with an open boundary does not actually have a well-defined "inside". However, we would still like our method to give reasonable results even in the presence of a few small gaps. Our approach to handling these cases is to cast multiple rays from a node, instead of just one, and using a "majority vote" to determine whether the node should be flagged as inside or outside. This alleviates some of the sensitivity of this scheme, since it is less likely for multiple rays to pass through the gaps, and generally gives reasonable results for surfaces that are *mostly* closed. For our particular choice of ray directions, we cast rays in the 6 axis-aligned directions, and flag the node as inside if at least 3 of the 6 rays determine it to be inside. Casting multiple rays from each node would be expensive, but we reduce the number of rays cast by observing that many of the ray intersections are redundant. For example, if there are no crossing edges along the x direction between nodes (i, j, k) and (i', j, k), then casting a ray in the x direction (either positive or negative) from either node will yield the same intersection result.

Having assigned a sign to each node, the next step is to initialize all nodes near the surface (those with opposite sign neighbors) with their shortest distance to the surface. Once this is done, the full signed distance function can be calculated quickly using a marching method [139, 124].

2.4 Interference Detection

[45, 29] found intersections between two implicitly defined surfaces by testing the sample points of one with the inside/outside function of the other. We follow the same strategy using the vertices of the triangulated surface as our sample points.

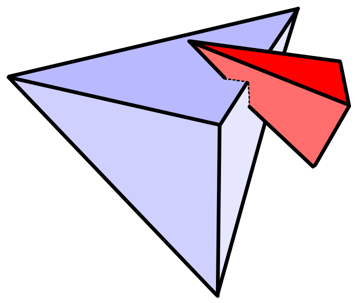

Figure 2.4: **Only looking for vertices inside an implicit surface may be insufficient for bodies with coarse triangulated surfaces, as shown here. Additionally intersecting edges against the implicit surface may be used to correctly detect penetration in such cases.**

This test is not sufficient to detect all collisions, as edge-face collisions are missed when both edge vertices are outside the implicit surface (Figure 2.4). Since the errors are proportional to the edge length, they can be ignored in a well resolved mesh with small triangles. However, when substantial, e.g. when simulating cubes with only 12 triangles, we intersect the triangle edges with the zero isocontour and flag the deepest point on the edge as an interpenetrating sample point. Since we do not consider time dependent collisions, fast moving objects might pass through each other. We alleviate this problem by limiting the size of a time step based on the translational and rotational velocities of the objects and the size of their bounding boxes, although methods exist for treating the entire time swept path as a single implicit surface [121].

A number of accelerations can be used in the interference detection process. For example, the implicit surface inside/outside tests can be accelerated by labeling the voxels that are completely inside and completely outside (this is done for voxels at each level in the octree representation as well) so that ϕ interpolation can be avoided except in cells which contain part of the interface. Labeling the minimum and maximum values of ϕ in each voxel can also be useful. Bounding boxes and spheres are used around each object in order to prune points before doing a full inside/outside test. Moreover, if the bounding volumes are disjoint, no inside/outside tests are needed. For rigid bodies with a large number of triangles, we found an internal box hierarchy with triangles in leaf boxes to be useful especially when doing edge intersection tests. Also, we use a uniform spatial partitioning data structure with local memory storage implemented using a hash table in order to quickly narrow down which rigid bodies might be intersecting. Similar spatial partitioning was used, for example, in [94]. Again, we stress our interest in nonconvex objects and refer the reader to [112, 72] for other algorithms that treat arbitrary nonconvex polyhedral models. For more details on collision detection methods, see e.g. [141, 79, 118].

2.5 Time Integration

The equations for rigid body evolution are

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \qquad \frac{d\mathbf{q}}{dt} = \frac{1}{2}\boldsymbol{\omega}\mathbf{q} \tag{2.1}$$

$$\frac{d\mathbf{v}}{dt} = \mathbf{f}/m, \qquad \frac{d\mathbf{L}}{dt} = \boldsymbol{\tau} \tag{2.2}$$

where $\mathbf{x}$ and $\mathbf{q}$ are the position and orientation (a unit quaternion), $\mathbf{v}$ and $\boldsymbol{\omega}$ are the velocity and angular velocity, $\mathbf{f}$ and $\boldsymbol{\tau}$ are the net force and torque, m is the mass, and $\mathbf{L} = \mathrm{I}\boldsymbol{\omega}$ is the angular momentum with inertia tensor I (all quantities in world space). As mentioned above, we store an object space, diagonal inertia tensor D, and compute the world space inertia tensor using $\mathrm{I} = \mathrm{RDR}^T$ (where R is the orientation matrix corresponding to $\mathbf{q}$). For simplicity we will consider $\mathbf{f} = m\mathbf{g}$ (where $\mathbf{g}$ is the downward pointing gravity vector) and thus $\frac{d\mathbf{v}}{dt} = \mathbf{g}$ throughout the text, but our algorithm is not restricted to this case. While there are a number of highly accurate time integration methods for noninteracting rigid bodies in free flight, see e.g. [19], these algorithms do not retain this accuracy in the presence of contact and collision. Thus, we take a different approach to time integration instead optimizing the treatment of contact and collision. Moreover, we use a simple forward Euler time integration for Equations (2.1) and (2.2).[1]

[1]We use forward Euler for orientation: $\mathbf{q}^{n+1} = \mathbf{q}^n + \frac{1}{2}\Delta t\boldsymbol{\omega}^n\mathbf{q}^n$ (followed by renormalization of $\mathbf{q}^{n+1}$). An alternative approach (see e.g. [144]) is to use $\mathbf{q}^{n+1} = \hat{\mathbf{q}}(\Delta t\boldsymbol{\omega}^n)\mathbf{q}^n$ where $\hat{\mathbf{q}}(\mathbf{w})$ is a unit quaternion representing a rotation by $|\mathbf{w}|$ radians about the axis $\mathbf{w}/|\mathbf{w}|$.

The standard approach is to integrate Equations (2.1) and (2.2) forward in time, and subsequently treat collision and then contact. Generally speaking, collisions require impulses that discontinuously modify the velocity, and contacts are associated with forces and accelerations. However, friction can require the use of impulsive forces in the contact treatment, although the *principle of constraints* requires that the use of impulsive forces be kept to a minimum [85]. [6] suggested that this avoidance of impulsive behavior is neither necessary nor justified and stressed that there are algorithmic advantages to using impulses exclusively. This naturally leads to some blurring between collision and contact handling, and provides a sense of justification to the work of [54] where the same algebraic equations were used for both and the work of [94] who integrated the same nonlinear differential equations for both. However, other authors such as [97, 126, 73] have noted difficulties associated with this blurring and proposed that an impulse based treatment of collisions be separated from a penalty springs approach to contact. They used the magnitude of the relative velocity to differentiate between contact and collision. [94] used the velocity an object at rest will obtain by falling through the collision envelope (or some threshold in [93]) to identify the contact case and applied a microcollision model where the impulse needed to reverse the relative velocity is applied as long as it lies in the friction cone. They showed that this solves the problem of blocks erroneously sliding down inclined planes due to impulse trains that cause them to spend time in a ballistic phase.

A novel aspect of our approach is the clean separation of collision from contact without the need for threshold velocities. We propose the following time sequencing:

- Collision detection and modeling
- Advance the velocities using Equation (2.2)
- Contact resolution
- Advance the positions using Equation (2.1)

The advantages of this time stepping scheme are best realized through an example. Consider a block sitting still on an inclined plane with a large coefficient of restitution, say $\epsilon = 1$, and suppose that friction is large enough that the block should sit still.

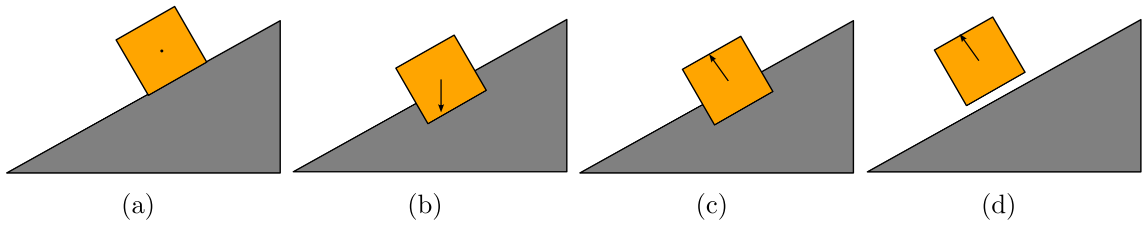

Figure 2.5: **A block with friction resting on an inclined plane incorrectly starting to bounce using the standard time stepping scheme. Figure shows the block (a) initially resting, (b) after position and velocity update, (c) after colliding with the inclined plane, and (d) bouncing during the next position update.**

In a standard time stepping scheme, both position and velocity are updated first, followed by collision and contact resolution (Figure 2.5). During the position and velocity update, the block starts to fall under the effects of gravity. Then in the collision processing stage we detect a low velocity collision between the block and the plane, and since $\epsilon = 1$ the block will change direction and bounce upwards at an angle down the incline. Then in the contact resolution stage, the block and the plane are separating so nothing happens. The block will eventually fall back to the inclined plane, and continue bouncing up and down incorrectly sliding down the inclined plane because of the time it spends in the ballistic phase. This is the same phenomenon that causes objects sitting on the ground to vibrate as they are incorrectly subjected to a number of elastic collisions. Thus, many authors use *ad hoc* threshold velocities in an attempt to prune these cases out of the collision modeling algorithm and instead treat them with a contact model.

Our new time stepping algorithm automatically treats these cases (Figure 2.6). All objects at rest have zero velocities (up to round-off error), so in the collision processing stage we do not get an elastic bounce (up to round-off error). Next, gravity is integrated into the velocity, and then the contact resolution algorithm correctly stops the objects so that they remain still. Thus, nothing happens in the last (position update) step, and we repeat the process. The key to the algorithm is that contact modeling occurs directly after the velocity is updated with gravity. If instead either the collision step or a position update were to follow the velocity

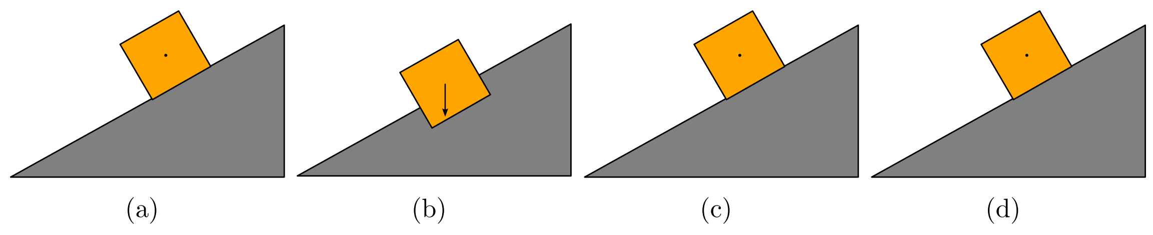

Figure 2.6: **A block with friction resting on an inclined plane correctly simulated using our new time stepping scheme. Figure shows the block (a) initially resting (no collision to process), (b) after velocity update (candidate position drawn), (c) after frictional contact, and (d) after position update.**

update, objects at rest will either incorrectly elastically bounce or move through the floor, respectively. On the other hand, contact processing is the correct algorithm to apply after the velocity update since it resolves forces, and the velocity update is where the forces are included in the dynamics.

One must use care when updating the velocity in between the collision and contact algorithms to ensure that the same exact technique is used to detect contact as was used to detect collision. Otherwise, an object in free flight might not register a collision, have its velocity updated, and then register a contact causing it to incorrectly receive an inelastic (instead of elastic) bounce. We avoid this situation by guaranteeing that the contact detection step registers a negative result whenever the collision detection step does. This is easily accomplished by ensuring that the velocity update has no effect on the contact and collision detection algorithms (discussed in Section 2.6).

We repeated the experiment of a block sliding down an inclined plane from [94] using the methods for collision and contact proposed throughout this chapter and our newly proposed time step sequencing. We used a coefficient of restitution $\epsilon = 1$ in order to accentuate difficulties with erroneous elastic bouncing. Using our new time stepping scheme the decelerating block slides down the inclined plane coming to a stop matching theory, while the standard time stepping scheme performs so poorly that the block bounces down the inclined plane as shown in Figure 2.7. Of course, these poor results are accentuated because we both set $\epsilon = 1$ and do not back up the

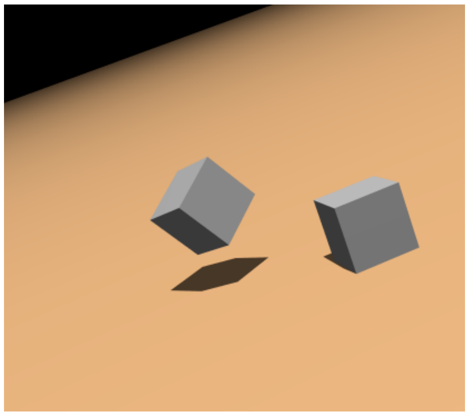

Figure 2.7: **The block and inclined plane test with standard time integration (the block erroneously tumbling) and our new time integration sequencing (the block correctly at rest).**

simulation to the time of collision (which would be impractical and impossible for our large stacking examples). Figure 2.8 shows a comparison between theory and our numerical results. For both the acceleration and deceleration cases, our numerical solution and the theoretical solution lie so closely on top of each other that two distinct lines cannot be seen. Moreover, our results are noticeably better than those depicted in [94] even though we do not use a threshold velocity or their microcollision model.

2.6 Collisions

When there are many interacting bodies, it can be difficult to treat all the collisions especially if they must be resolved in chronological order. Thus instead of rewinding the simulation to process collisions one at a time, we propose a method that simultaneously resolves collisions as did [130, 90]. While this does not give the same result as processing the collisions in chronological order, there is enough uncertainty in the collision modeling that we are already guaranteed to not get *the* exact physically correct answer. Instead we will obtain *a* physically plausible solution, i.e. one of many possible physically correct outcomes which may vary significantly with slight perturbations in initial conditions or the inclusion of unmodeled phenomena such as material microstructure.

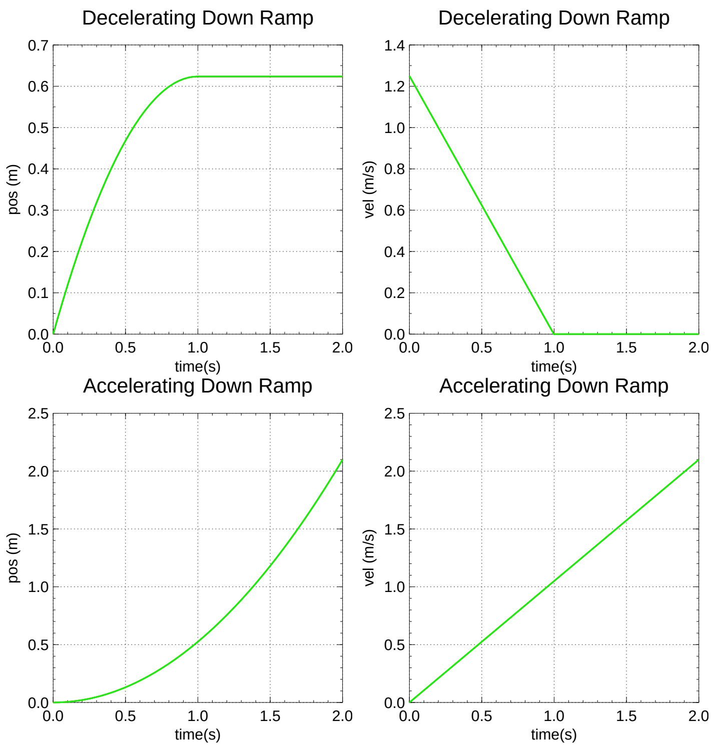

Figure 2.8: **Theoretical and our numerical results for two tests of a block sliding down an inclined plane with friction. As in [94], inclined plane is at $20°$, accelerating block starts at rest with $\mu = 0.5$, and decelerating block starts at $v_0 = 1.25\ m/s$ with $\mu = 0.25$. The curves lie on top of each other in the figures due to the accuracy of our new time sequencing algorithm.**

Collisions are detected by predicting where the objects will move to in the next time step, temporarily moving them there, and checking for interference. The same technique will be used for detecting contacts, and we want the objects to be moved to the same position for both detection algorithms if there are no collisions (as mentioned above). In order to guarantee this, we use the new velocities to predict the positions of the rigid bodies in both steps. Of course, we still use the old velocities to process the collisions and the new velocities to process the contacts. For example, for the collision phase, if an object's current position and velocity are $\mathbf{x}$ and $\mathbf{v}$, we test for interference using the predicted position $\mathbf{x}' = \mathbf{x} + \Delta t(\mathbf{v} + \Delta t\mathbf{g})$, and apply collision impulses to (and using) the current velocity $\mathbf{v}$. During contact processing, we use the

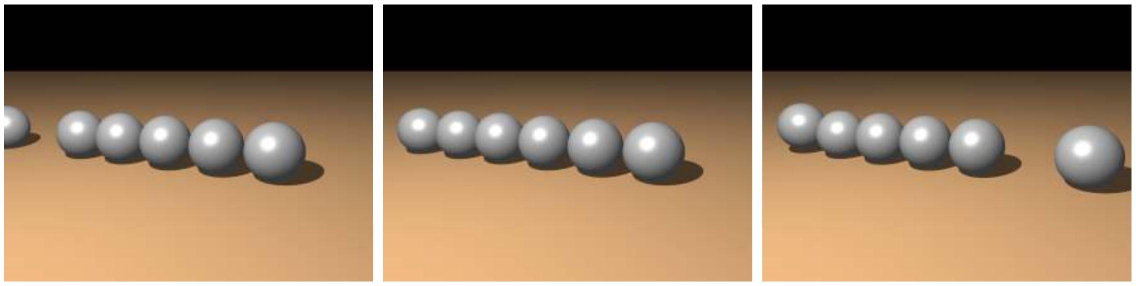

Figure 2.9: **A billiard ball hits one end of a line of billiard balls, and the collision response propagates to the ball on the far right which then starts to roll.**

predicted position $\mathbf{x}' = \mathbf{x} + \Delta t \mathbf{v}'$ and apply impulses to this new velocity $\mathbf{v}'$. Since $\mathbf{v}' = \mathbf{v} + \Delta t \mathbf{g}$ was set in the velocity update step, the candidate positions match and the interference checks are consistent.

The overall structure of the algorithm consists of first moving all rigid bodies to their predicted locations, and then identifying and processing all intersecting pairs. Since collisions change the rigid body's velocity, $\mathbf{v}$, new collisions may occur between pairs of bodies that were not originally identified. Therefore we repeat the entire process a number of times (e.g. five iterations) moving objects to their newly predicted locations and identifying and processing all intersecting pairs. Since pairs are considered one at a time, the order in which this is done needs to be determined. This can be accomplished by initially putting all the rigid bodies into a list, and then considering rigid bodies in the order in which they appear. To reduce the inherent bias in this ordering, we regularly mix up this list by randomly swapping bodies two at a time. This list is used throughout our simulation whenever an algorithm requires an ordering.

For each intersecting pair, we identify all the vertices of each body that are inside the other (and optionally the deepest points on interpenetrating edges as well). Since we do not back up the rigid bodies to the time of collision, we need a method that can deal with nonconvex objects with multiple collision regions and multiple interfering points in each region. We start with the deepest point of interpenetration that has a nonseparating relative velocity as did [97], and use the standard algebraic collision laws (below) to process the collision. Depending on the magnitude of the collision, this may cause the separation of the entire contact region. If we re-evolve the position using the new post-collision velocity, this collision group could be resolved. Whether or not it is resolved, we can once again find the deepest non-separating point and repeat the process until all points are either non-interpenetrating or separating. While this point sampling method is not as accurate as integrating over the collision region as in [56], it is much faster and scales well to large numbers of objects.

We developed an aggressive optimization for the point sampling that in many cases gives similarly plausible results (see e.g. Figure 2.9).[2] As before, one first labels all the non-separating intersecting points and applies a collision to the deepest point. But instead of re-evolving the objects and repeating the expensive collision detection algorithm, we simply keep the objects stationary and use the same list of (initially) interfering points for the entire procedure. After processing the collision, all separating points are removed from the list. Then the remaining deepest nonseparating point is identified and the process is repeated until the list is empty. In this manner, all points in the original list are processed until they are separating *at least once* during the procedure. The idea of lagging collision geometry was also considered by [9] in a slightly different context.

At the collision point, a rigid body's local velocity is $\mathbf{u} = \mathbf{v} + \omega \times \mathbf{r}$ where $\mathbf{r}$ is a radial vector from the center of mass to the collision point. Applying an impulse $\mathbf{j}$ at the collision point changes both linear and angular velocities according to

$$\mathbf{v}' = \mathbf{v} + \mathbf{j}/m \tag{2.3}$$

and

$$\boldsymbol{\omega}' = \boldsymbol{\omega} + \mathrm{I}^{-1}(\mathbf{r} \times \mathbf{j}). \tag{2.4}$$

The new velocity at the point of collision becomes

$$\mathbf{u}' = \mathbf{u} + \mathrm{K}\mathbf{j}, \tag{2.5}$$

[2]In our experience, this optimization appears to be less appropriate for collisions with a very low coefficient of restitution. Greater accuracy is typically desired in these cases, just as it is needed for contact.

where $\mathrm{K} = \delta/m + \mathbf{r}^{*T}\mathrm{I}^{-1}\mathbf{r}^{*}$ with δ the identity matrix and the "$*$" superscript indicating the cross-product matrix.[3] K is the rigid body's inverse effective inertia matrix at the point of collision, playing a role analogous to $1/m$ in Equation (2.3). Bodies which have infinite inertia, including immovable bodies such as the ground as well as bodies whose motion is kinematically prescribed, can be treated by setting $\mathrm{K} = 0$.

Let $\mathbf{u}_{rel} = \mathbf{u}_1 - \mathbf{u}_2$ be the *relative* velocity at the point of collision between body 1 and body 2, and assume we apply an equal and opposite impulse to the two bodies: $+\mathbf{j}$ to body 1 and $-\mathbf{j}$ to body 2. Then subtracting Equation (2.5) for the two bodies, we get

$$\mathbf{u}'_{rel} = \mathbf{u}_{rel} + \mathrm{K}_T\mathbf{j}, \tag{2.6}$$

where $\mathrm{K}_T = \mathrm{K}_1 + \mathrm{K}_2$.

The relative velocity can be decomposed into normal and tangential components, $u_{rel,n} = \mathbf{u}_{rel} \cdot \mathbf{n}$ and $\mathbf{u}_{rel,t} = \mathbf{u}_{rel} - u_{rel,n}\mathbf{n}$ respectively. Each body is assigned a coefficient of restitution, and when two bodies collide we use the minimum between the two coefficients as did [97] to process the collision. Note that, strictly speaking, the coefficient of restitution is a collision property rather than a material property (for example, it may vary by collision speed), so our approach of storing it per body is a simplification. Taking the dot product of Equation (2.6) with $\mathbf{n}$, and assuming $\mathbf{j} = j_n\mathbf{n}$ (a frictionless impulse only acting in the normal direction) and $u'_{rel,n} = -\epsilon u_{rel,n}$ (Newton's law of restitution), we get $-\epsilon u_{rel,n} = u_{rel,n} + \mathbf{n}^T\mathrm{K}_T\mathbf{n}j_n$. We can then solve for j_n,

$$j_n = \frac{-(\epsilon+1)u_{rel,n}}{\mathbf{n}^T\mathrm{K}_T\mathbf{n}}, \tag{2.7}$$

and thus determine the frictionless collision impulse $\mathbf{j}$.

2.7 Static and Kinetic Friction

The collision algorithm above needs to be modified to account for kinetic and static friction. Each body is assigned a coefficient of friction, μ, and we use the maximum

[3]For $\mathbf{r} = (r_x, r_y, r_z)$, $\mathbf{r}^* = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$, so that $\mathbf{r}^*\mathbf{j} = \mathbf{r} \times \mathbf{j}$

of the two possible coefficients when processing a collision as did [97]. Like [54, 97], we first assume that the bodies are stuck at the point of impact due to static friction and solve for the impulse. That is, we set $\mathbf{u}'_{rel,t} = \mathbf{0}$, so that $\mathbf{u}'_{rel} = -\epsilon u_{rel,n}\mathbf{n}$, which allows us to solve Equation (2.6) for the impulse $\mathbf{j}$ by inverting the symmetric positive definite matrix K_T. Then if $\mathbf{j}$ is in the friction cone, i.e. if $|\mathbf{j} - (\mathbf{j}\cdot\mathbf{n})\mathbf{n}| \leq \mu\mathbf{j}\cdot\mathbf{n}$, the point is sticking due to static friction and $\mathbf{j}$ is an acceptable impulse. Otherwise, we apply sliding friction.

Define $\mathbf{t} = \mathbf{u}_{rel,t}/|\mathbf{u}_{rel,t}|$ so that the kinetic friction can be computed with the impulse $\mathbf{j} = j_n\mathbf{n} - \mu j_n\mathbf{t} = j_n(\mathbf{n} - \mu\mathbf{t})$. By a similar derivation to that of Equation (2.7), we can determine the kinetic friction impulse $\mathbf{j}$ by solving

$$j_n = \frac{-(\epsilon+1)u_{rel,n}}{\mathbf{n}^T\mathrm{K}_T(\mathbf{n}-\mu\mathbf{t})}.$$

2.8 Contact

After a few iterations of the collision processing algorithm, the rigid bodies have been elastically bounced around enough to obtain a plausible behavior. So even if collisions are still occurring, we update the velocities of all the rigid bodies and move on to contact resolution. Since the contact modeling algorithm is similar to the collision modeling algorithm except with a zero coefficient of restitution, objects still undergoing collision will be processed with inelastic collisions. This behavior is plausible since objects undergoing many collisions in a single time step will tend to rattle around and quickly lose energy.

The goal of the contact processing algorithm is to resolve the forces between objects. As in collision detection, we detect contacts by predicting where the objects will move to in the next time step disregarding the contact forces, temporarily moving them there, and checking for interference. For example, objects sitting on the ground will fall into the ground under the influence of gravity leading to the flagging of these objects for contact resolution. All interacting pairs are flagged and processed in the order determined by our list. Once again, multiple iterations are needed especially for rigid bodies that sit on top of other rigid bodies. For example, a stack of cubes will

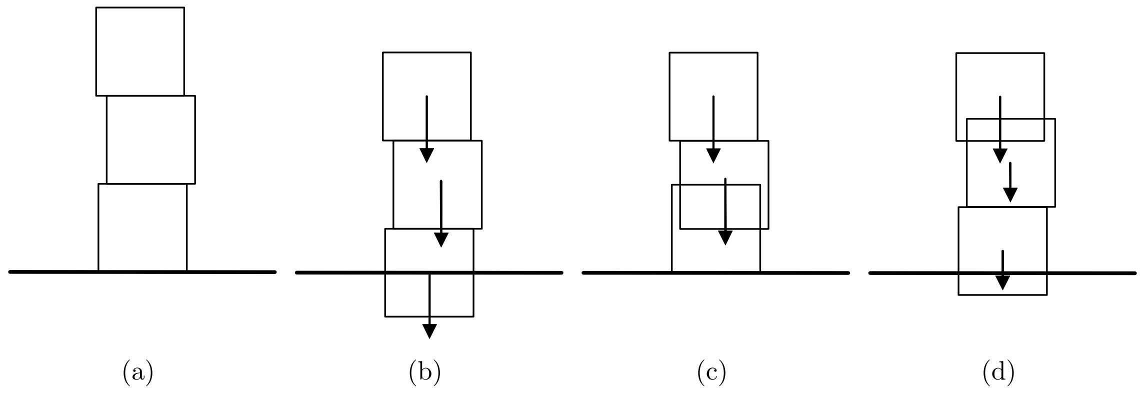

Figure 2.10: **Processing contact in a stack may take many iterations using the propagation model. (a) The initial stack. (b) Candidate positions during contact processing. (c) Contact is resolved between bottom block and ground. (d) Resolving contact between the bottom two blocks pushes the bottom one back into the ground.**

all fall at the same speed under gravity and only the cube on the bottom of the stack will intersect the ground and be flagged for contact resolution (Figure 2.10 (b)). The other cubes experience no interference in this first step. However, after processing the forces on the cube at the bottom of the stack, it will remain stationary and be flagged as interpenetrating with the cube that sits on top of it in the next sweep of the algorithm (Figure 2.10 (c)). This is a propagation model for contact as opposed to the simultaneous solution proposed in [4].

The difficulty with a propagation model is that it can take many iterations to converge. For example, in the next iteration the cube on the ground is stationary and the cube above it is falling due to gravity. If we process an inelastic collision between the two cubes, the result has both cubes falling at half the speed that the top cube was falling (Figure 2.10 (d)). That is, the cube on top does not stop, but only slows down. Even worse, the cube on the ground is now moving again and we have to reprocess the contact with the ground to stop it. In this sense, many iterations are needed since the algorithm does not have a global view of the situation. That is, all the non-interpenetration constraints at contacts can be viewed as one large system of equations, and processing them one at a time is similar to a slow Gauss-Seidel approach to solving this system. Instead, if we simultaneously considered the entire system of equations, one could hope for a more efficient solution, for example by using a better iterative solver. This is the theme in [90] where an optimization based approach is taken. We propose a more light-weight method in Section 2.8.2.

Similar to the collision detection algorithm, for each intersecting pair, we identify all the vertices of each body that are inside the other (and optionally the deepest points on edges as well). Although [4] pointed out that the vertices of the contact region (which lie on the vertices and edges of the original model) need to be considered, we have found our point sampling method to be satisfactory. However, since we have a triangulated surface for each object, we could do this if necessary. As in [54, 94] we use the same equations to process each contact impulse that were used in the collision algorithm, except that we set $\epsilon = 0$. We start with the deepest point of interpenetration that has a non-separating relative velocity, and again use the standard algebraic collision laws. Then a new predicted position can be determined and the process repeated until all points are either non-overlapping or separating. Although the aggressive optimization algorithm that processes all points until they are separating *at least once* could be applied here as well, it is not as attractive for contact as it is for collision since greater accuracy is usually desired for contacts.

For improved accuracy, we propose the following procedure. Rather than applying a fully inelastic impulse of $\epsilon = 0$ at each point of contact, we *gradually* stop the object from approaching. For example, on the first iteration of contact processing we apply impulses using $\epsilon = -.9$, on the next iteration we use $\epsilon = -.8$, and so on until we finally use $\epsilon = 0$ on the last iteration. A negative coefficient of restitution simply indicates that rather than stop or reverse an approaching object, we only slow it down.

In the collision processing algorithm, we used the predicted positions to determine the geometry (e.g. normal) of the collision. Although it would have been better to use the real geometry at the time of collision, the collision time is not readily available and furthermore the accuracy is not required since objects are simply bouncing around. On the other hand, objects should be sitting still in the contact case, and thus more accuracy is required to prevent incorrect rattling around of objects. Moreover, the correct contact geometry is exactly the current position (as opposed to the predicted position), since the contact forces should be applied before the object moved. Thus, we use the current position to process contacts.

2.8.1 Contact graph

Before updating the velocities of the rigid bodies we construct a contact graph similar to [54, 1] with the intention of identifying which bodies or groups of bodies are resting on top of others and determining a useful ordering for contact processing. We perform the following steps:[4]

Compute graph For each rigid body $\mathcal{B}$:

- Compute the candidate position of $\mathcal{B}$ using its new velocity, $\mathbf{v}' = \mathbf{v} + \Delta t\mathbf{g}$, and the candidate positions of the remaining bodies using their old velocities.
- For each body $\mathcal{B}'$ intersected by $\mathcal{B}$ in these candidate positions, add a directed edge $\mathcal{B}' \rightarrow \mathcal{B}$ to indicate that $\mathcal{B}$ is resting on $\mathcal{B}'$.

Remove cycles Find all strongly connected components in the resulting graph and collapse each component into a single node representing the bodies in that component. The resulting graph is *acyclic*.

Assign ordering Use topological sort to assign each node a unique order or "level" for contact processing.

For a stack of cubes, we get a contact graph that points from the ground up one cube at a time to the top of the stack (Figure 2.11 (a)). For difficult problems such as a set of dominoes arranged in a circle on the ground with each one resting on top of the one in front of it, we simply get the ground in one level of the contact graph and *all* the dominoes in a second level. Roughly speaking, objects are grouped into the same level if they have a cyclic dependence on each other (Figure 2.11 (b)).

[4]Note that this differs slightly from the method we described in [51], where one body was moved with gravity while the rest were kept stationary. The method described here handles more general scenarios such as moving contact, e.g. a block resting on the floor of an ascending elevator.

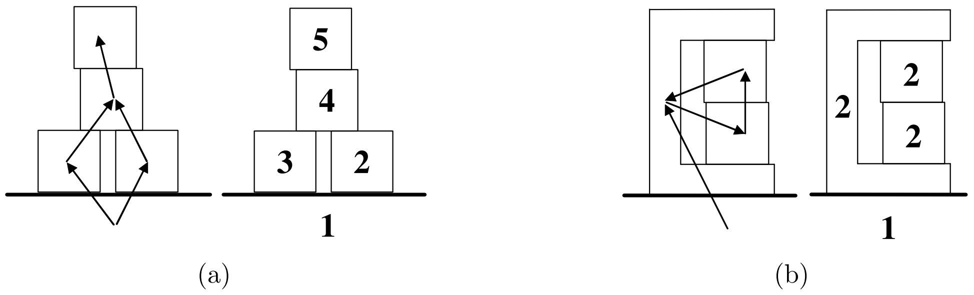

Figure 2.11: **Examples of contact graphs and associated orderings. (a) A basic stack, bodies are ordered from bottom to top, starting with the ground as number 1. (b) Example with a cycle, multiple bodies assigned to level 2.**

The purpose of the contact graph is to suggest an order in which contacts should be processed, and we wish to sweep up and out from the ground and other static (non-simulated) objects in order to increase the efficiency of the contact propagation model. When considering objects in level i, we gather all contacts between objects within level i as well as contacts between objects in this level and ones at lower-numbered levels. With the latter type of contact pairs, the object in level i is, as a result of the way we constructed the contact graph, necessarily "resting on" the lower level object and not the other way around. The contact pairs found for level i are put into a list and treated in any order iterating through this list a number of times before moving on to the next level. Additionally, we sweep along the graph through all levels multiple times for improved accuracy.

2.8.2 Shock propagation

Even with the aid of a contact graph, the propagation model for contact may require many iterations to produce visually appealing results especially in simulations with stacks of rigid bodies. For example, in the cube stack shown in Figure 2.10. To alleviate this effect, we propose a *shock propagation* method that can be applied on the last sweep through the contact graph. After each level is processed in this last sweep, all the objects in that level are assigned infinite mass (their K matrix is set to zero). Here, the benefit of sorting the objects into levels becomes most evident. If

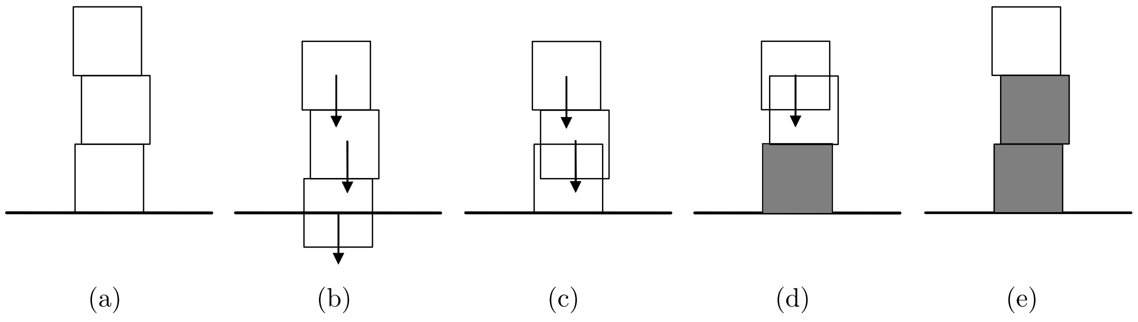

Figure 2.12: **A single sweep of shock propagation correctly resolves contact in this stack. (a) The initial stack. (b) Candidate positions during contact processing. (c) Contact is resolved between bottom block and ground. (d) Bottom block is set to have infinite inertia, and contact is resolved between bottom two blocks (the bottom one does not move). (e) Middle block is set to infinite inertia and contact is resolved between top two blocks.**

an object of infinite mass is later found to be in contact with a higher-level object, its motion is not affected by the impulses applied to resolve contact, and the higher level object will simply have to move out of the way! Once assigned infinite mass, objects retain this mass until the shock propagation phase has completed. As in contact, we iterate a number of times over all contact pairs in each level, but unlike contact we only complete one sweep through all of the levels. Note that when two objects at the same level are in contact, neither has been set to infinite mass yet, so shock propagation in this case is no different than our usual contact processing. However, the potentially slow convergence of the usual contact processing has now been localized to the smaller groups of strongly connected components in the scene.

To see how this algorithm works, consider the stack of objects in Figure 2.12. Starting at the bottom of the stack, each object has its velocity set to zero during contact processing, and its mass subsequently set to be infinite. As we work our way up the stack, the falling objects cannot push the infinite mass objects out of the way so they simply get their velocity set to zero as well. In this fashion, the contact graph allows us to shock the stack to a zero velocity in one sweep. Without a final shock propagation sweep, regular contact iterations would require many iterations to converge to the stationary stack solution (recall Figure 2.10).

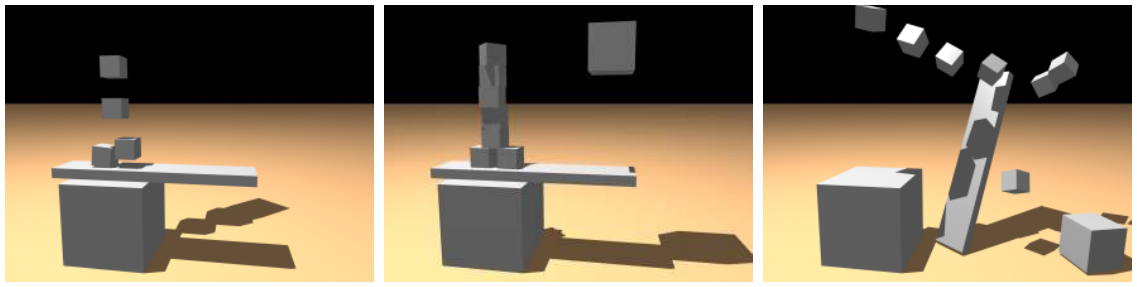

Figure 2.13: **Although the propagation treatment of contact and collision allows the stacking and flipping of boxes as shown in the figure, our shock propagation algorithm makes this both efficient and visually accurate.**

In order to demonstrate why the propagation model for contact is still used for a few iterations before applying shock propagation, we drop a larger box onto a plank supporting a stack of blocks as shown in Figure 2.13. Here the contact graph points up from the ground through all the objects, and when the larger box first comes in contact with the plank, an edge will be added pointing from the plank to the box. If shock propagation was applied immediately the box on the ground and then the plank would have infinite mass. Thus the large falling box would simply see an infinite mass plank and be unable to flip over the stack of boxes. However, contact propagation allows both the plank to push up on the falling box *and* the falling box to push back down. That is, even though "pushing down" increases the number of iterations needed for the contact algorithm to converge, without this objects would not feel the weight of other objects sitting on top of them. Thus, we sweep though our contact graph a number of times in order to get a sense of weight, and then efficiently force the algorithm to converge with a final shock propagation sweep. This allows the stack of boxes to flip over as shown in Figure 2.13 (right).

Figure 2.14 shows another test where a heavy and a light block are both initially at rest on top of a see-saw. When the simulation starts the weight of the heavier block pushes down tilting the see-saw in that direction. Eventually it tilts enough for the heavy block to slide off, and then the see-saw tilts back in the other direction under the weight of the lighter block. Our combination of contact propagation followed by shock propagation correctly and efficiently handles this scenario. On the other hand,

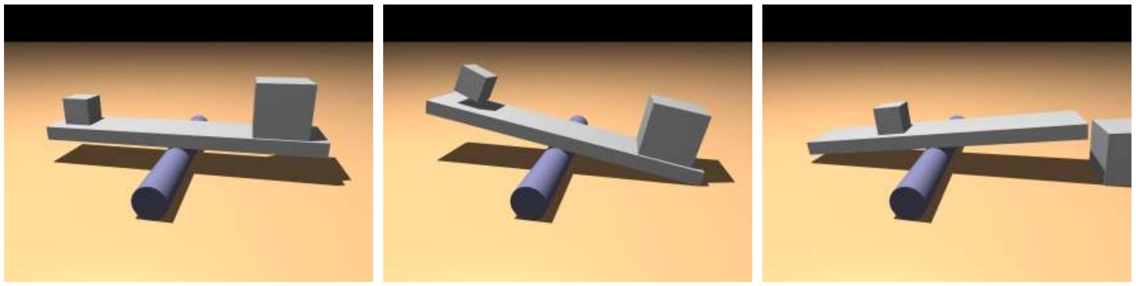

Figure 2.14: **A heavier block on the right tips the see-saw in that direction, and subsequently slides off. Then the smaller block tips the see-saw back in the other direction. The propagation treatment of contact allows the weight of each block to be felt, and our shock propagation method keeps the blocks from interpenetrating without requiring a large number of contact processing iterations.**

if we run shock propagation only (i.e. omitting the contact propagation phase), the see-saw either sits still or tips very slowly since it does not feel the weight of the heavy block.

2.9 Rolling and Spinning Friction

Even when a rigid body has a contact point frozen under the effects of static friction, it still has freedom to both roll and spin. [77, 119] damped these degrees of freedom by adding forces to emulate rolling friction and air drag. Instead, we propose an approach that treats these effects in the same manner as kinetic and static friction. Let μ_r and μ_s designate the coefficients of rolling and spinning friction, and note that these coefficients should depend on the local deformation of the object. This means that they should by scaled by the local curvature with higher values in areas of lower curvature. Thus for a sphere, these values are constant throughout the object.

Both rolling and spinning friction are based on the relative angular velocity, $\boldsymbol{\omega}_{rel}$, with normal and tangential components $\omega_{rel,n} = \boldsymbol{\omega}_{rel} \cdot \mathbf{n}$ and $\boldsymbol{\omega}_{rel,t} = \boldsymbol{\omega}_{rel} - \omega_{rel,n}\mathbf{n}$. The normal component governs spinning and the tangential component governs rolling about $\boldsymbol{\omega}_{rel,t}/|\boldsymbol{\omega}_{rel,t}|$. We modify these by reducing the magnitude of the normal and tangential components by $\mu_s j_n$ and $\mu_r j_n$ respectively. To keep the object from reversing either its spin or roll direction, both of these reductions are limited to zero otherwise preserving the sign. At this point we have a new relative angular velocity $\boldsymbol{\omega}'_{rel}$, and since the objects are sticking the relative velocity at the contact point is $\mathbf{u}'_{rel} = \mathbf{0}$. Next, we construct an impulse to achieve both proposed velocities.

If we apply the impulse at a point, specifying the desired relative velocity determines the impulse $\mathbf{j}$ (through Equation (2.6)) and we are unable to also specify the relative angular velocity. Thus, we assume that the impulse is applied over an area, giving rise to an additional angular impulse contribution of $\mathbf{j}_\tau$, separate from the angular impulse $\mathbf{r} \times \mathbf{j}$ due to $\mathbf{j}$. We still have $\mathbf{v}' = \mathbf{v} + \mathbf{j}/m$, but the change in angular velocity is now $\boldsymbol{\omega}' = \boldsymbol{\omega} + \mathrm{I}^{-1}(\mathbf{j}_\tau + \mathbf{r} \times \mathbf{j})$. We can relate the linear and angular impulses to the resulting linear and angular velocities at the point of collision using the equation

$$\begin{pmatrix} \mathbf{u}' \\ \boldsymbol{\omega}' \end{pmatrix} = \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\omega} \end{pmatrix} + \mathrm{K}^S \begin{pmatrix} \mathbf{j} \\ \mathbf{j}_\tau \end{pmatrix},$$

where

$$\mathrm{K}^S = \begin{bmatrix} \delta/m + \mathbf{r}^{*T}\mathrm{I}^{-1}\mathbf{r}^* & \mathbf{r}^{*T}\mathrm{I}^{-1} \\ \mathrm{I}^{-1}\mathbf{r}^* & \mathrm{I}^{-1} \end{bmatrix} = \begin{bmatrix} \delta & \mathbf{r}^{*T} \\ 0 & \delta \end{bmatrix} \begin{bmatrix} m^{-1}\delta & 0 \\ 0 & \mathrm{I}^{-1} \end{bmatrix} \begin{bmatrix} \delta & 0 \\ \mathbf{r}^* & \delta \end{bmatrix}$$

is the inverse *spatial* inertia matrix for the rigid body at the point of collision. K^S is similar to K in Equation (2.5) but incorporates angular effects. Letting $\mathrm{K}^S_T = \mathrm{K}^S_1 + \mathrm{K}^S_2$, we get the equations for the change in relative velocities:

$$\begin{pmatrix} \mathbf{u}'_{rel} \\ \boldsymbol{\omega}'_{rel} \end{pmatrix} = \begin{pmatrix} \mathbf{u}_{rel} \\ \boldsymbol{\omega}_{rel} \end{pmatrix} + \mathrm{K}^S_T \begin{pmatrix} \mathbf{j} \\ \mathbf{j}_\tau \end{pmatrix}. \quad (2.8)$$

Viewing this as two (vector) equations in two unknowns, $\mathbf{j}$ and $\mathbf{j}_\tau$, we can solve one equation for $\mathbf{j}$ in terms of $\mathbf{j}_\tau$, plug it into the other, and solve for $\mathbf{j}_\tau$. This requires inverting two 3×3 matrices.

It should be noted that these equations also form the basis for the post-stabilization step of the articulated rigid bodies method of [144]. In that paper, the equations are used to compute internal joint impulses which project joint motion to the space of allowed motions (see Section 8.1).

2.10 Pushing Apart Rigid Bodies

After the positions have been updated, interpenetration may still occur due to round-off errors and cyclic dependencies in the contact graph. We experimented with the use of first order physics (similar to [9]) to compute a “first order impulse” which is applied to the objects’ positions and orientations to effect separation (without modifying their velocities). As in shock propagation, we proceeded level by level through the contact graph doing multiple iterations of separation adjustments within each level before assigning infinite masses to each level in preparation for the next. To reduce bias, we *gradually* separated objects within a level, each iteration increasing the fraction of interpenetration that is corrected.

It should be noted that changing position without modifying velocity will in general fail to conserve total angular momentum (with respect to a fixed point). Thus, we plan to investigate further whether alternative strategies exist that reduce interpenetration while still conserving momentum.

2.11 Summary

To summarize, once a time step Δt has been chosen (see Section 2.4), the main steps of the algorithm are:

- Detect interference and apply collision impulses (Section 2.6)
- Compute contact graph (Section 2.8.1)
- Update velocity with gravity and any other external forces (Equation (2.2))
- Detect interference and apply (inelastic) contact impulses (Section 2.8)
- Apply a final sweep of shock propagation (Section 2.8.2)
- Advance bodies to their new positions (Equation (2.1))

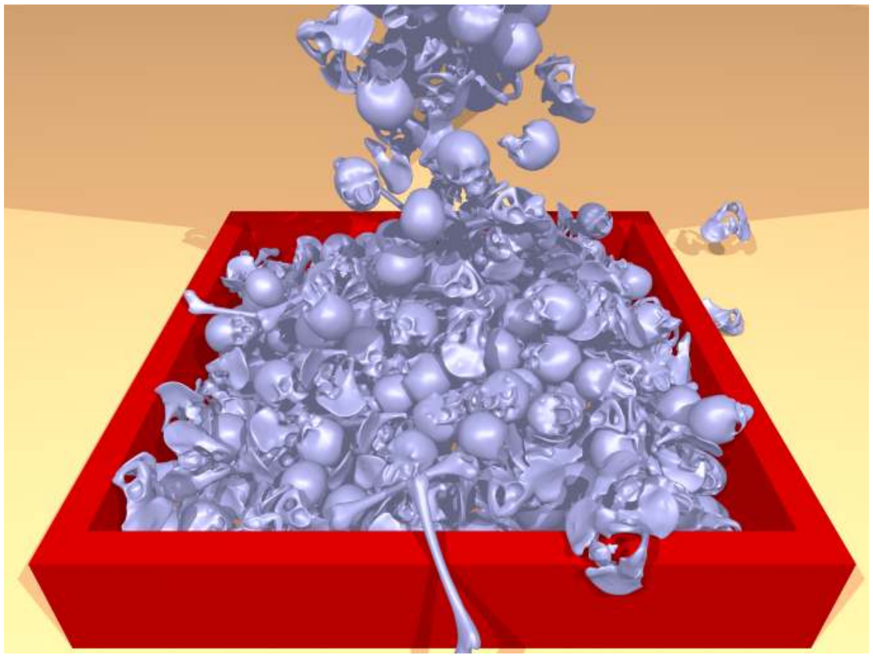

Figure 2.15: **500 nonconvex bones falling into a pile after passing through a funnel. Exact triangle counts are given in Figure 2.2.**

2.12 Results

Besides the basic tests that we discussed throughout the text, we also explored the scalability of our algorithm addressing simulations of large numbers of nonconvex objects with high resolution triangulated surfaces falling into stacks with multiple contact points. While the CPU times for the simple examples were negligible, the simulations depicted in Figures 2.1, 2.15, and 2.16, had a significant computational cost. Dropping 500 and 1000 rings into a stack averaged about 3 minutes and 7 minutes per frame, respectively. Dropping 500 bones through a funnel into a pile averaged about 7 minutes per frame, and we note that this simulation had about 2.8 million triangles total. All examples were run using 5 collision iterations, 10 contact iterations, and a single shock propagation iteration, and all used friction.

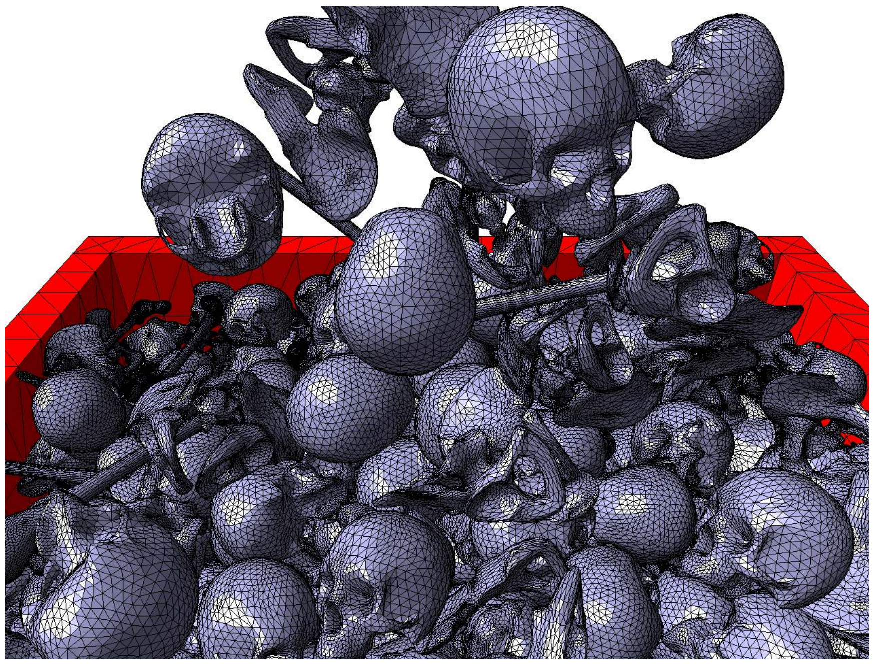

Figure 2.16: **The complexity of our nonconvex objects is emphasized by exposing their underlying tessellation.**

2.13 Conclusions

We proposed a mixed representation of the geometry, combining triangulated surfaces with signed distance functions defined on grids, and illustrated that this approach has a number of advantages. We also proposed a novel time integration scheme that removes the need for *ad hoc* threshold velocities and matches theoretical solutions for blocks sliding and stopping on inclined planes. Finally, we proposed a new shock propagation method that dramatically increases the efficiency and visual accuracy of stacking objects. So far, our rolling and spinning friction model has only produced good results for spheres and we are currently investigating more complex objects.

2.14 Thin Rigid Shells

The simulator described above specified that each rigid body had a dual representation consisting of a triangulated surface and an implicit surface. In fact, since collisions are detected based on nodes of one body penetrating the implicit surface of another body, the algorithm will correctly handle collisions between a body with only a triangulated surface representation and a body with only an implicit representation. In particular, it can correctly simulate a rigid shell interacting with a rigid volumetric solid. This allows us to handle the boat example in Chapter 6 which couples a fully dynamic rigid shell (boat) to water and includes collision between boat and ground (Figure 6.11). The other two rigid shell examples in that chapter, the cup and "Buddha cup", are scripted rather than dynamic.

On the other hand, for simulations of multiple rigid shells, an algorithm for handling collisions between triangulated surfaces must be implemented. Recent work described in [3] applied the geometric collisions framework used for cloth in [17] to handle rigid shell collisions. The basic approach is to process collisions between the two rigid triangulated surfaces by temporarily allowing them to deform and computing cloth-like collision impulses. The computed collision impulses are subsequently applied to the rigid shells in order to update their position. Unlike cloth, nonpenetration is not guaranteed, although it may be reduced by repeating this collision processing for a number of iterations.

Chapter 3

Deformable Body Simulation

3.1 Introduction

For many solids, the rigid body assumption cannot be applied, and fully deformable models must be used. Various techniques exist for physically-based simulation of deformable solids, including masses and springs, the finite element method, and the finite volume method. Masses and springs are easier to implement, typically run faster, and often produce visually plausible results for computer graphics. However, they represent a discrete, heuristic model which is not grounded in continuum mechanics, making it harder to implement arbitrary constitutive models. Both finite element and finite volume methods are based on continuum mechanics, and are thus more popular for scientific applications. For more details on these various approaches we refer the reader to [135, 64] and to the survey paper [100].

This chapter gives a brief overview of cloth simulation, and is included for completeness since cloth is used in some of the examples in Chapter 6. However, we simply used the previously published approach of [17, 18]. Thus we emphasize that this chapter reviews existing work, and does not include any novel contributions.

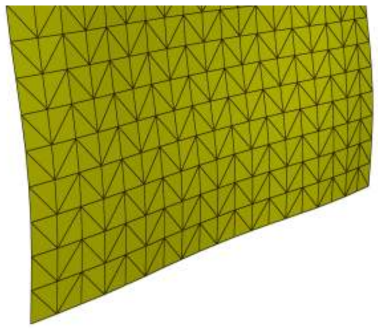

Figure 3.1: **The cloth triangle mesh forms a "herring-bone" pattern of alternating diagonals.**

3.2 Cloth Simulation

We use the basic cloth model of [18] including their bending formulation (see also [50]), and the self-collision algorithm of [17]. A brief overview of the basic components of this simulator is given below, and the reader is referred to the original papers for more details. We note that there are many other interesting strategies for cloth including the dynamics model proposed in [11], the bending model proposed in [25], and the self-interference untangling strategy of [12].

3.2.1 Cloth Dynamics

The cloth mesh is represented by a rectangular grid of particles, with each rectangle split into two triangles in a "herring-bone" pattern that splits neighboring triangles using opposite diagonals (see Figure 3.1). The mass of each particle is computed by adding one third of the mass of all of its incident triangles.

External and internal force contributions are computed for each cloth node. Since collision and contact are handled separately, the only external force considered here is gravity. Internal cloth forces consist of the following three components:

Edge Springs These springs connect particles sharing an edge in the triangle mesh, and are used to model the general "strechiness" of the material. The force includes both an elastic (Hookean) component and a damping component.

Altitude Springs These springs are used to help preserve the shape of the triangle elements, in particular trying to keep them from compressing in-plane. Each triangle has three altitude springs (coinciding with their geometric altitudes), but each time step only the force due to the *shortest* of the three is included in the dynamics.

Bending Elements Bend between adjacent triangles is measured by the dihedral angle, and [18] describes the formulation of forces that resist surface bending while conserving linear and angular momentum and remaining decoupled from in-plane motion.

The strain and strain-rate limiting approach of [17] was also used in order to avoid instabilities and oscillations while allowing large time steps to be taken.

3.2.2 Collision and Contact

In the previous chapter, we were able to create plausible simulation of many rigid bodies interacting. As mentioned in Section 2.10, our algorithm does not guarantee nonpenetration between bodies, and interpenetration may occur in some situations. Interpenetration between volumetric solids, while undesirable, can generally be visually tolerated and potentially corrected as the simulation progresses. This is due to the fact that it is easy to detect when a point is inside a solid, making it possible to continually try to drive that point towards the surface of the penetrated solid. On the other hand, self-intersecting cloth can be very difficult to automatically untangle (see [12]), and ideally self-penetration should be prevented from occurring in the first place.

We use the cloth self-collision algorithm of [17] which combines repulsions and geometric collisions in order to guarantee no self-penetration. Repulsion forces help maintain separation between the cloth elements by pushing apart nodes, edges, and triangles that get too close to one another. Geometric collisions detect cross-overs between moving point-triangle pairs or edge-edge pairs, and collision impulses are used to prevent intersection. The repulsion step is useful because it helps resolve many near-collisions before they occur using a computationally cheaper penalty force approach. In addition, since geometric collisions will catch any intersections not resolved by repulsion alone, the repulsion force stiffness can be kept reasonably low so that the time step is not significantly affected. Through the additional use of adaptive time stepping and rigid impact zones, [17] was able to eliminate *all* cloth self-collision.

3.2.3 Time Step

The basic cloth update loop (taken from [17]) is:

- Compute candidate positions and velocities $(\bar{\mathbf{x}}^{n+1}, \bar{\mathbf{v}}^{n+1})$ using internal cloth dynamics
  - A mixed implicit/explicit time integration scheme is used
  - Cloth nodes with fixed positions (e.g. tacks used to hang cloth) have their velocities manually fixed at zero
- Compute the effective velocity $\bar{\mathbf{v}}^{n+1/2} = \left(\bar{\mathbf{x}}^{n+1} - \mathbf{x}^{n}\right)/\Delta t$
- Apply repulsions and geometric collisions to $\bar{\mathbf{v}}^{n+1/2}$ to get the final velocity $\mathbf{v}^{n+1/2}$
- Compute $\mathbf{x}^{n+1} = \mathbf{x}^{n} + \Delta t \mathbf{v}^{n+1/2}$
- Compute $\mathbf{v}^{n+1}$ from either $\bar{\mathbf{v}}^{n+1}$ (if no repulsions or collisions) or by advancing $\mathbf{v}^{n+1/2}$

For simulations coupling cloth to a fluid, we found it useful to allow the hanging cloth to reach an equilibrium before starting to hit it with fluid. Typically we ran a cloth-only simulation for a number of “preroll” frames before starting the fully coupled simulation. Since the object of preroll is to bring the cloth to an equilibrium, dynamic effects are unimportant, and adding additional damping helps reach steady-state with fewer oscillations. In our case, a simple drag force was added to all cloth nodes for the duration of the preroll.

Chapter 4

Fluid Simulation

4.1 Introduction

Simulations of water, smoke, and fire are becoming increasingly important in computer graphics applications such as feature films, since it is often costly, dangerous, or simply impossible to film the desired interactions between these fluids and their surroundings. In addition, simulation of these phenomena has been pervasive in the engineering and computational physics communities for some time. Thus, while simulations of explosions or tidal waves may be performed for prediction and analysis in the engineering community, they may also be used for visual effects in films.

This chapter introduces a basic fluid simulator which will be augmented for coupling in the chapters that follow. The focus will be on simulation of smoke and water, although it can be extended to support fire simulation as well. In fluid simulation, we are concerned with the time evolution of a set of variables. The main variable of interest is the fluid's velocity field $\mathbf{u}$, and the Navier-Stokes equations governing its evolution are given in Section 4.3. For smoke simulation, we treat the smoke as a collection of tracer particles being carried with the underlying air flow, and we track both its density, ρ_{smoke}, and the temperature of the underlying fluid, T, in order to incorporate buoyancy effects (see Section 4.4.3) and for use in visualization. While gases fill the simulation domain, liquids can take on arbitrary shape and we need to track their moving boundary. We use the particle level set method to track the water-air interface, which makes use of both particles and a signed distance function ϕ as described in Section 4.5.1. Note that for our water simulation, the "air" portion of the computational domain is not simulated, and is modeled as having no effect on the water surface through appropriate boundary conditions (Section 4.5).

The content of this chapter is based on the basic fluid simulation portion of our work published in [52]. For the most part we use previously published techniques [31, 37], although the key contribution of this chapter is our novel node-based approach (Section 4.4.1).

4.2 Previous Work

Simulation of the incompressible three dimensional Navier-Stokes equations was popularized by [43] and later made more efficient by [127] via the use of semi-Lagrangian advection techniques. [37] increased the small scale details with the use of a vorticity confinement term. There is also the recent work of [123] that hybridizes grid-based methods with vortex particle techniques. These equations have also been augmented to model fire [76, 101], particle explosions [38], and even the interior of deformable objects [102]. [117] combined interpolation with two dimensional simulations to create three dimensional nuclear explosions, [138, 34] proposed methods for control, and [128] solved these equations on surfaces creating beautiful imagery.

Previous work addressing liquid simulation includes [69], which solved a linearized form of the three dimensional Navier-Stokes equations for water that interacted with a variable height terrain. [23] solved the two dimensional Navier-Stokes equations using the pressure to define a heightfield. The full three dimensional Navier-Stokes equations were solved in [41, 42] using a particle in cell approach. A hybrid particle and implicit surface approach to simulating water was proposed in [40], which led to the particle level set method of [31]. Additional work includes the modeling of surface tension [32, 59, 27, 82], viscoelastic fluids [49], sand [155], multiphase flow [60, 84], control [87, 88, 125], and adaptive data structures [82, 63]. Level set methods for simulating liquids have enjoyed popularity in recent films including "Terminator 3" [116], "Pirates of the Caribbean" [116], "The Day After Tomorrow" [65], "The Cat in the Hat" [27], and "Scooby Doo 2" [147, 62].

4.3 Equations

As proposed in [37], we ignore viscous effects and use the inviscid form of the Navier-Stokes equations,

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\nabla p + \mathbf{f}, \tag{4.1}$$

where $\mathbf{u} = (u, v, w)$ is the velocity field, $\mathbf{f}$ accounts for acceleration due to external forces, and the fluid density has been absorbed into the pressure p. ∇ is the gradient operator, e.g. $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ in 3D. Equation (4.1) is simply a statement of Newton's second law, "$a = F/m$", for the fluid. In our case, the fluid acceleration "a" equals the *material* (or *total*) time derivative of the velocity, $D\mathbf{u}/Dt$, which expands to precisely the left hand side of Equation (4.1).

Additionally, we assume the fluid is incompressible, which is a reasonable assumption for water and for gases (in an open space) at velocities well below the speed of sound. Incompressibility, together with conservation of mass, give rise to the equation

$$\nabla \cdot \mathbf{u} = 0, \tag{4.2}$$

which is sometimes referred to as the "divergence free condition" for the velocity field. In 3D, $\nabla \cdot \mathbf{u} = \partial u/\partial x + \partial v/\partial y + \partial w/\partial z$.

We solve these equations using the projection method due to Chorin [26], which is commonly used in the computer graphics community. The time derivative in Equation (4.1) is discretized using basic forward Euler: $\frac{\partial \mathbf{u}}{\partial t} \approx \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t}$. The resulting semi-discrete PDE is split into the following fractional steps:

- Compute an intermediate velocity $\mathbf{u}^*$ which incorporates advection and external forces but does not satisfy the incompressibility constraint (Equation (4.2)):

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\left(\mathbf{u}^n \cdot \nabla\right)\mathbf{u}^n + \mathbf{f}. \tag{4.3}$$

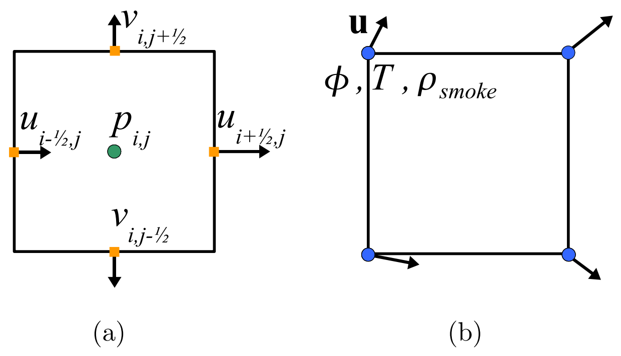

Figure 4.1: **(a) The standard staggered (MAC) grid stores velocities on faces and pressure in the cell center. (b) Our node-based scheme keeps velocity as well as other fluid scalars on the nodes. However, we still use the staggered representation during the projection step.**

- Project $\mathbf{u}^*$ to get a final, divergence free velocity field $\mathbf{u}^{n+1}$:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\nabla p. \tag{4.4}$$

  This requires solving a separate equation for pressure (Section 4.4.4).

Below we describe in more detail how each of these steps is solved. First we discuss our novel node-based approach for fluid simulation.

4.4 Fluid Solver

4.4.1 Node-Based Fluid Solver

We have implemented our algorithm on both uniform and octree grids, although this exposition is primarily geared towards uniform grids with octree grids discussed only when the extension is not obvious. Otherwise, we refer the reader to [82] for more details.

Fluid simulation on uniform grids typically uses a staggered grid configuration, sometimes referred to as a "MAC grid" as it was first used with the marker-and-cell (MAC) technique of [55]. As shown in Figure 4.1 (a), the staggered approach places velocity components on faces and pressure values in the cell centers. The main advantage of this configuration is that it lends itself to a straightforward discretization of the projection equation utilizing compact, second order velocity divergence and pressure gradient operators. The main drawback from our perspective is that storing velocity components on separate faces makes it harder to interpolate velocity values at arbitrary locations. In particular, while it can be done for uniform grids using staggered, component-wise interpolation, it is much more complex in the case of octrees. On octrees, it is most natural to interpolate nodal values, and our desire to maintain consistency between both uniform and octree grids led us to devise a novel node-based fluid solver with the velocity, temperature, smoke density, and level set values all defined on the nodes (Figure 4.1 (b)).

We would still like to solve for the pressure on the standard MAC grid, due to the advantages of this configuration. This could be done by averaging the nodal velocities to the faces, projecting the face velocities to be divergence free, and then averaging the result back to the nodes. However, these extra averaging steps deteriorated the quality of our results so we designed an alternate method. We start by defining the initial velocity field on the faces instead of the nodes. Then at each step, we first average the face values to the nodes before computing the intermediate velocity $\mathbf{u}^*$ on the nodes as will be described below. Then, instead of averaging this intermediate velocity back to the faces, we compute a scaled force on each node as $\Delta\mathbf{u} = \mathbf{u}^* - \mathbf{u}^n$ and average this scaled force back to the faces instead. The scaled force is then used to increment the persistent face velocities to obtain the intermediate velocity on the faces. This alternative method smears out the incremental forces instead of the persistent velocities, and thus leads to excellent results competitive with the standard MAC scheme. To understand the difference, consider that averaging velocities back and forth causes dissipation independent of Δt and even leads to dissipation when $\Delta t = 0$. On the other hand, the smearing of our forces is scaled by Δt and vanishes as Δt approaches 0. Overall, this method allows us to combine a simple and efficient node-based scheme for computing the intermediate velocity with a robust face based scheme for making the intermediate velocity divergence free. The advantage of our strategy over straight back-and-forth averaging is very similar to the advantage of

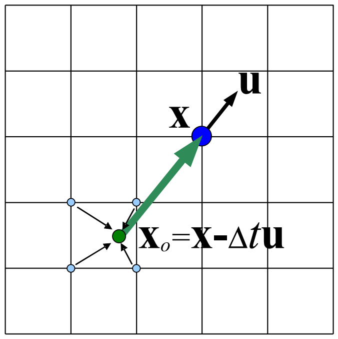

Figure 4.2: **The first order semi-Lagrangian scheme determines the new value at x by tracing back to $\mathbf{x}_o$ and interpolating.**

fluid-implicit-particle (FLIP) over particle-in-cell (PIC) averaging, which was recently discussed in [155].

4.4.2 Semi-Lagrangian Advection

Advection refers to the process by which some quantity is transported from one place to another, in our case carried along with the fluid flow. An advection equation for a scalar φ is of the form $\partial\varphi/\partial t + \mathbf{u}\cdot\nabla\varphi = 0$, or equivalently $D\varphi/Dt = 0$, which can be interpreted as the statement that φ remains unchanged as it is transported along the velocity field $\mathbf{u}$. The quantities to be advected are the smoke density, temperature, the water level set, and even the fluid velocity itself (as one part of computing the intermediate velocity $\mathbf{u}^*$).

We solve these advection equations using the semi-Lagrangian method, introduced by [28] to the atmospheric science community, and popularized in computer graphics by [127]. Its popularity stems from the fact that it is unconditionally stable, allowing for arbitrarily large time steps. The idea behind the method is to trace back along characteristics (flow lines) to determine where the fluid parcel currently at location $\mathbf{x}$ was in the previous time step (at time $t - \Delta t$). With no diffusion or source terms, this quantity is assumed to be transported unchanged, and so assuming the parcel came from $\mathbf{x}_o$, we set $\varphi(\mathbf{x}, t) = \varphi(\mathbf{x}_o, t - \Delta t)$. The first order semi-Lagrangian method approximates the characteristic path by a straight line, using $\mathbf{x}_o = \mathbf{x} - \Delta t\mathbf{u}(\mathbf{x})$, as shown in Figure 4.2. Since $\mathbf{x}_o$ will not, in general, coincide with a grid node, interpolation is used to compute φ at that location.

Note that besides simplifying interpolation on octree grids, our node-based solver also adds the side benefit of requiring the computation of fewer semi-Lagrangian backtraces. This is because typically semi-Lagrangian advection of velocity on staggered grids traces back characteristics from faces, which outnumber nodes by a factor of 3. This will be particularly beneficial when we couple to thin solids, as each semi-Lagrangian backtrace will involve performing an actual intersection test against the solids (see Section 6.3.2).

4.4.3 Computing the Intermediate Velocity

In order to compute the intermediate velocity $\mathbf{u}^*$ in Equation (4.3), we first update the velocity field with the contribution due to the velocity advection term $-(\mathbf{u}^n \cdot \nabla)\,\mathbf{u}^n$. As described above, we use the semi-Lagrangian method to accomplish this. We use $\mathbf{u}^n$ (averaged from faces to nodes as described in Section 4.4.1) to compute the semi-Lagrangian characteristics.

The second term added as part of computing $\mathbf{u}^*$ is the external force $\mathbf{f}$. For water, gravity is simply added to each nodal velocity in the usual manner. For smoke, we add weight and buoyancy terms that depend on the smoke's density and the fluid's temperature: $\mathbf{f} = -\alpha\rho_{smoke}\mathbf{z} + \beta(T - T_a)\mathbf{z}$, where $\mathbf{z}$ is the upward direction, α and β are tunable parameters, and T_a is the ambient temperature.

Vorticity confinement can also be added to amplify small scale detail in the flow. To compute the vorticity confinement force at each grid node, we calculate the curl of the velocity field $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ using the six adjacent nodal velocity vectors. Then we compute the vorticity magnitude. Its gradients, $\nabla|\boldsymbol{\omega}|$, are computed using central differencing of the six neighboring values of the vorticity. These gradients are normalized to obtain the vorticity location vectors, $\mathbf{N}$, which are then used to compute the source term due to vorticity at the node, $\mathbf{f} = \hat{\epsilon}\Delta x(\mathbf{N} \times \boldsymbol{\omega})$ (as in [37]).

4.4.4 Solving for the Pressure

After obtaining face velocities from the nodal ones as described in Section 4.4.1, the intermediate face velocity is made divergence free via

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t \nabla p, \tag{4.5}$$

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

$$\nabla^2 p = \frac{\nabla \cdot \mathbf{u}^*}{\Delta t}. \tag{4.6}$$

The Laplacian operator ∇^2 is simply shorthand for $\nabla \cdot \nabla$, and in 3D $\nabla^2 p = \partial^2 p/\partial x^2 + \partial^2 p/\partial y^2 + \partial^2 p/\partial z^2$. This equation can be derived by taking the divergence of Equation (4.5) and setting $\nabla \cdot \mathbf{u}^{n+1} = 0$.

We solve Equation (4.6) by assembling a symmetric system of linear equations, one for each MAC grid cell (that contains fluid) with the pressure defined at the cell center. Boundary conditions need to be specified for cells along the edges of the computational domain and water cells adjacent to air. In the case of water, we set Dirichlet $p = 1$ boundary conditions in air cells. A Neumann boundary condition implies that the pressure derivative at a cell face is zero ($\partial p/\partial n = 0$, where n is the direction normal to the face), and thus the intermediate velocity is not modified as can be seen in Equation (4.5). By additionally setting $\mathbf{u}^* = \mathbf{u}^{fixed}$ on a face, a fixed velocity can be enforced, and this is used at walls and other solid boundaries.[1]

Conjugate gradient with incomplete Cholesky preconditioning is used to solve the resulting matrix system. We refer the reader to [40] and [82] for more information on the discretization on uniform and octree grids, respectively. Furthermore, a second order accurate pressure boundary condition was introduced in [47] and used for water simulation in [32]. We used this higher order discretization for one of our examples, and the end of Section 6.3.4 highlights the improved results.

[1]An equivalent way to think about it is that in order to achieve the desired velocity $\mathbf{u}^{fixed}$ we need to set the pressure gradient to be equal to the necessary acceleration: $\nabla p = \frac{\mathbf{u}^{fixed} - \mathbf{u}^*}{\Delta t}$.

4.5 Water

For water simulation, we need to keep track of the water-air interface. We use the particle level set method of [31], where the interface is identified with the zero level set of a signed distance function ϕ, and $\phi \leq 0$ denotes water and $\phi > 0$ denotes air. Since we only solve for velocity values in the water, each time step we extrapolate the nodal velocities across the interface into a 3-5 grid cell band to obtain velocity boundary conditions. To do this, we first order all the grid cells in the band based on their values of ϕ, noting that this ordering is only valid after reinitializing ϕ to be a signed distance function (see Section 4.5.1). Then we solve the vector equation $\nabla\phi \cdot \nabla\mathbf{u} = 0$ for the nodes in ϕ increasing order (see [31] for more details).

4.5.1 Level Set Method

[30] showed that the particle level set method relies primarily on the particles for accuracy whereas the role of the level set is to provide connectivity and smoothness. Thus, they showed that high order accurate level set advection could be replaced with a semi-Lagrangian characteristic based scheme without adversely affecting the accuracy.

The level set is maintained to be a signed distance function using the fast marching method (see e.g. [105]). Initially, the nodes adjacent to the water interface are found by checking for sign changes between neighbors in the Cartesian grid directions (or along edges in the octree grid). Then each node in this list is given an initial ϕ value by considering how far it is from the interface in each of the Cartesian grid directions that have a sign change. After initialization, fast marching fills the remaining nodes by marching out from the interface.

4.5.2 Particle Level Set Method

The particle level set method makes use of particles, seeded within a band of the interface, to improve the accuracy of the interface by reducing the volume loss in regions of high curvature resulting from numerical diffusion inherent to the advection scheme. Particles carry a sign denoting which of the two sides of the interface they represent: positive particles represent air, and negative particles represent water. When a particle is close to the interface, it locally adjusts the interface based on its radius – i.e. how far that particle expects the interface to be. It accomplishes this by modifying level set values on nearby grid nodes using a CSG union operation which treats the particle as a small sphere. Since particles exist only in a thin band around the interface and do not need to interact, they can be seeded at higher resolution than the fluid grid to improve accuracy. The particle radii used for level set modification are therefore much smaller than a grid cell, which may result in particles failing to convert a surrounding node to liquid and incorrectly crossing over the the level set zero isocontour. Such particles are flagged as "removed", and no longer play a role in correcting the interface. One approach is to delete removed particles. However, keeping them around is an option which serves two purposes. First, removed particles may return to the correct side of the interface, allowing them to be "reincorporated" and once again help reduce interface capturing errors. Second, removed positive particles may be used to emulate small bubbles in the fluid while removed negative particles may be advanced using ballistic motion in order to emulate water droplets in air. For our simulations we chose to keep removed negative particles, even rendering them as described in Section 6.8, but delete positive particles which become removed.

[30] used a second order Runge-Kutta scheme to advance the particles in the particle level set method. In our case, when advancing the particles to their time $n+1$ positions, we have $\mathbf{u}^n$ but do not yet know the time $n+1$ fluid velocity field, $\mathbf{u}^{n+1}$. Thus, we advance a particle's position using

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{\Delta t}{2}\left(\mathbf{u}^n(\mathbf{x}^n) + \mathbf{u}^n(\mathbf{x}^n + \Delta t \mathbf{u}^n(\mathbf{x}^n))\right), \tag{4.7}$$

which is similar to a second order Runge-Kutta update, but uses a *frozen* velocity field by replacing what should be $\mathbf{u}^{n+1}$ in the second term with $\mathbf{u}^n$. Interpolation is used to evaluate $\mathbf{u}^n$ at arbitrary locations. Periodically, every 10-20 frames, particles are reseeded to get a better representation of the interface.

4.6 Algorithm Summary

Sequentially, the overall algorithm to advance from time n to $n+1$ proceeds as follows.

- Update fluid scalars:

  Smoke: Update smoke density and fluid temperature (Section 4.4.2)

  Water: Update level set and particles (Section 4.5)

- Compute intermediate fluid velocity $\mathbf{u}^*$ (Section 4.4.3)

- Project $\mathbf{u}^*$ to get a divergence-free $\mathbf{u}^{n+1}$ (Section 4.4.4)

This algorithm is unconditionally stable due to our use of the semi-Lagrangian advection scheme. Still, the semi-Lagrangian scheme does suffer from dissipation so in the interest of accuracy we typically try to pick Δt small enough so information does not flow more than a few grid cells in one time step. For coupled simulations, this will be restricted to no more than one grid cell (see Section 6.6).

Chapter 5

Solid-Fluid Coupling Overview

5.1 Introduction

Water and smoke both possess a large number of degrees of freedom and thus produce visually rich motion especially when interacting with solid objects. This makes these media both interesting and popular from a storytelling or entertainment perspective. Often, two-way coupling between fluids and solids is not desirable, since the animator does not want the fluid changing or resisting their artistic development. On the other hand, animators have difficulty when the solid has many degrees of freedom, e.g. cloth, and have resorted to simulation to obtain the desired effects. Moreover, thin or light weight objects need to feel the effect of the fluid, especially heavy fluids such as water, in order to make their animation plausible. Thus, the two-way interaction of thin deformable high degree of freedom solids and heavy liquids with interfaces is highly desirable, and will be examined in Chapter 6. In addition, coupling to volumetric solids will be covered in Chapter 7. In this chapter, some previous work as well as general issues underlying two-way coupled simulations of solids and fluids will be discussed.

5.2 Previous Work

Various authors have used simplified fluid dynamics to blow around solid objects, e.g. [145, 142], and many have used simplified wind models to simulate flags flapping in the wind, e.g. [80]. [23] simulated one-way coupling (separately in both directions) between moving obstacles and heightfield water, while two-way coupling was achieved in the heightfield water model of [104]. [53] used gridless smoothed particle hydrodynamics (SPH) techniques to couple air flows to hair simulation, but since hair is one-dimensional it does not restrict or contain the fluid as shells and volumetric solids do.

SPH models for water were considered in [114, 98], and methods of this type were coupled to deformable solids in [99] using virtual boundary particles. [46] also used inter-particle forces, coupling the MAC fluid particles to the solid particles. [74] coupled an SPH model for water to thin deformable cloth pointing out that particle based fluid methods can be coupled without leaking using robust point face collisions, although their method will leak if the time step is not chosen sufficiently (sometimes severely) small. Of course, this can be alleviated with a more robust point face collision method as in [17].

The drawback of using SPH methods is that it is difficult to obtain the smooth liquid surfaces characteristic of level set methods, and recently [20] proposed a method for the two-way coupling of rigid bodies to level set based fluid simulations. They first rasterize the rigid body velocity onto the grid, and then solve the fluid equations everywhere treating the rasterized rigid body velocities as if they were fluid (this was also done in [40] for modeling slip boundary conditions). Then they modify the velocities in the rigid body region to account for collisions and buoyancy before averaging them to a valid rigid body velocity with a constant translational and rotational component. The authors point out that their method leaks if the objects are too thin (whereas we will consider arbitrarily thin objects in the next chapter), and deformable materials were not considered.

At least as far back as [103] (see also [15] for a review), two-way coupling has been carried out with the fluid's pressure providing forces to the solid, and the solid's velocity providing boundary conditions for the fluid. This commonly used approach was also used in [132, 131] for coupling rigid bodies to liquids and in [151] to couple solids to a compressible fluid. Our coupling strategy is based on this approach.

5.3 Eulerian and Lagrangian Schemes

There are various computational methods for simulating fluids, solids and their coupling. Typically fluids such as water (e.g. [40, 31]) are simulated using Eulerian numerical methods with a fixed mesh that material moves through, whereas solids such as cloth (e.g. [17, 25, 12]) are simulated with a Lagrangian numerical method where the mesh moves with the material.

Using a moving mesh for solids has the advantage that material quantities are automatically transported along with the object's motion, making it easier to track strain history, for example. This is harder in an Eulerian framework because advection schemes are not accurate enough, making fixed-mesh simulation of volumetric solids less desirable, and simulation of thin shells virtually impossible. Furthermore, lack of a body-conforming mesh makes it harder to compute accurate self-collision, rolling, etc. Nevertheless, [149] simulated solids and fluids together on a fixed uniform grid in order to unify treatment of multiple phases of matter. There has also been some interesting recent work with high viscosity and viscoelastic fluids [21, 116, 49], in which the fluid's behavior approaches that of a solid.

Lagrangian, mesh-based simulations of fluids do exist (e.g. [58, 115]), their primary advantage being that it is easier to specify boundary conditions on the fluid since the fluid mesh conforms to the shape of the boundary. However, fluids undergo extreme deformation and (in the case of liquids) topological change due to pinching and merging. As a result, such schemes require expensive, automated remeshing, which makes them less attractive than fixed-mesh approaches. There also exist mesh-free approaches to fluid simulation such as particle-based techniques (e.g. [155, 98, 114]). Unfortunately, particle methods have not yet reached the quality and efficiency of specialized techniques for fluid simulation. High particle densities are typically required to achieve good accuracy for fluids, and it is often difficult to generate a smooth renderable surface from the resulting set of points.

Somewhere in between purely Eulerian and purely Lagrangian simulations lies the arbitrary Lagrangian Eulerian (ALE) method (e.g. [57]). An ALE mesh combines fixed (Eulerian) regions and moving (Lagrangian) regions, including regions whose elements move at intermediate speeds not generally equal to the material speed.

We choose to take the more typical approach of Lagrangian solids (Chapters 2 and 3) and Eulerian fluids (Chapter 4). With separate representations for the solid and the fluid, our coupling algorithm must address how the effect of each will be imposed on the other. Since we use basic uniform or octree grids which do not conform to the boundaries of submerged solids, we will need to take care of dealing with “cut cells” (fluid cells overlapped by a solid), crossed-over nodes, etc.

5.4 Strong and Weak Coupling

In algorithms for solid-fluid coupling, a distinction is often made between “strong” and “weak” coupling (see e.g. [153]). Strong coupling is the term used for simulators which evolve both solid and fluid together as a system that incorporates the effect of each on the other. Alternatively, with weak coupling the solid is updated by fixing the fluid’s current state and advancing the solid without accounting for its immediate effect on the fluid. Subsequently, the fluid is updated given the solid’s new conditions, but ignoring its influence on the solid. [10] did something similar in their “partitioned” approach to coupling rigid and deformable solids, which interleaved existing simulators. Note that the partitioned approach, while typically used for weak coupling, can also be used to implement strong coupling by sub-iterating until solid and fluid states satisfy common boundary conditions to an acceptable tolerance. This represents an iterative solution to the global system.

The main tradeoffs between the two approaches to coupling are efficiency and simplicity. Strong coupling is typically more stable because it ensures the updated solid and fluid states are mutually compatible. However, it is less efficient than weak coupling, because it either requires solving a monolithic system, or requires running sub-iterations in a partitioned scheme.

The coupling approach described in the following chapters achieves weak coupling using a partitioned approach. Key to our algorithm is that the solids simulator will be treated as a black box, allowing it to be easily replaced by any preferred alternative. A novel aspect of our method is the computation of a smoother pressure for coupling, which helps us recover some of the stability lost by choosing to use weak coupling.

5.5 Thin Shells and Volumetric Solids

The description of our coupled simulator will be split into a chapter dealing with thin shells (Chapter 6) and a chapter for volumetric solids (Chapter 7). This distinction is made because coupling a fluid to an infinitesimally thin solid presents a number of unique challenges that do not exist when coupling to volumetric solids. Our focus will be on thin shells which are open surfaces, rather than closed membranes. This exacerbates the problem because while closed membranes can, for example, leverage a ghost fluid approach [36], open shells with no distinct inside or outside require more careful treatment. Of note is the recent work [133] which actually applied a ghost fluid approach to an immersed *open* boundary by defining two distinct sides *locally* to the interface.

One challenge in coupling to thin shells is preventing fluid from leaking across the solid. The key to stopping all leaks will involve modifying our computational stencils to avoid mixing fluid quantities from opposite sides of the solid.

Another big challenges in coupling to thin shells is how to actually accomplish the coupling, given that the solid is infinitesimally thin while the fluid's grid resolution limits its ability to resolve spatial detail. A distinction is often made between "diffuse" and "sharp" methods (see [140] for a nice overview). Diffuse methods give the solid a numerical thickness on the order of a few grid cells and blend its properties onto the fluid grid using a smoothed delta function. For example, the solid's density or internal force might be exerted on the fluid within this thickened region. The immersed boundary method is one example of a diffuse method, and will be discussed further in the following chapter. In contrast, sharp methods incorporate the effect of the solid by modifying the computational stencils near the solid/fluid interface.

Modifications to the stencil may be nontrivial and computationally more expensive, making sharp methods generally more complex than diffuse methods. However, the advantage of sharp interface methods is increased accuracy. In fact, the accuracy of the immersed boundary method is limited to first order for problems with nonsmooth solutions [75]. Our approach treats the solid in a sharp manner as much as possible, performing ray intersections against the solid's Lagrangian mesh to compute one-sided computational stencils. However, in order to enforce the solid's velocity on the fluid we will rasterize it onto the faces of the fluid grid. This is the one part of our algorithm which could arguably be called diffuse, as it distorts the shape of the solid (making first order errors). However, by rasterizing only onto faces we still maintain a compact representation that can support pressure jumps, whereas diffuse methods only obtain smeared pressure profiles.

Chapter 6

Solid-Fluid Coupling for Thin Shells

6.1 Introduction

Very little research has been carried out on algorithms that couple infinitesimally thin Lagrangian-based solids to Eulerian-based fluids, and few computational strategies exist. Moreover, they are mostly focused on single phase fluids, whereas our main interest is fluids with interfaces such as between water and air. Probably the most common strategy for *single phase* fluids is based on the immersed boundary method of [109, 110], and [154] used this method to calculate the motion of a thin flexible filament (a curve) in two spatial dimensions. A good survey of immersed boundary methods is [95]. A thin solid object feels and reacts to fluid forces as molecules collide against it, and the net force on the thin solid comes directly from the pressure differential across it. The immersed boundary method cannot handle this pressure jump and instead forces the pressure to be continuous across the thin solid, and thus (nonexistent) pressure jumps cannot be used to apply forces to the solid. Instead, they simply set the solid velocity to be equal to the velocity of the surrounding fluid, and use ad hoc methods to provide resistance to the fluid motion. For example, [154] smeared out the filament over a number of grid cells converting it into a higher density fluid, and added artificial forces to the right hand side of the Navier-Stokes

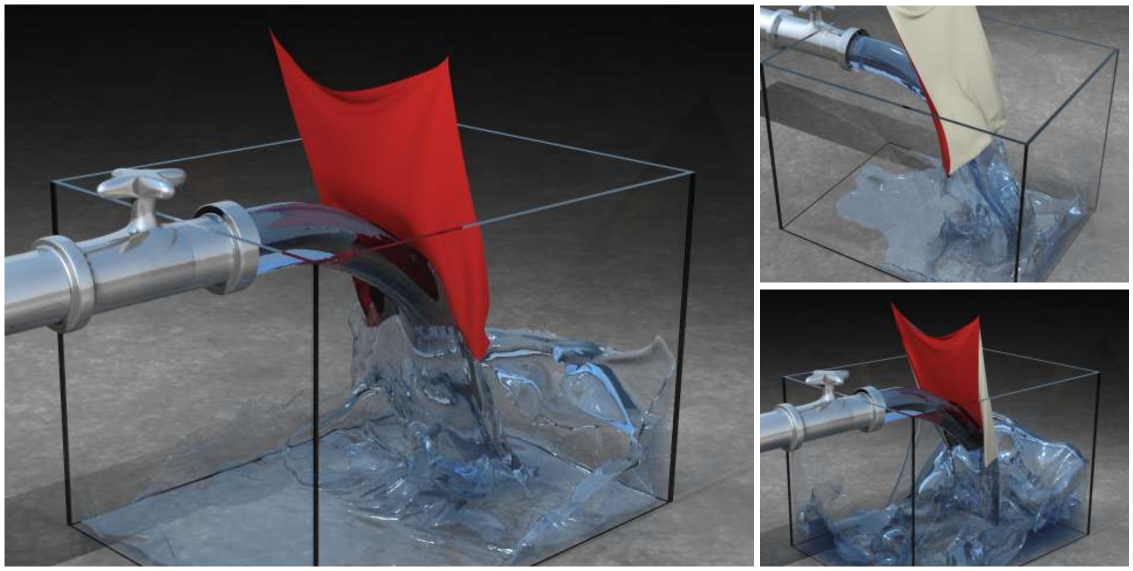

Figure 6.1: **Water and cloth interacting with full two-way coupling (**$256 \times 256 \times 192$ **effective resolution octree, 30K triangles in the cloth).**

equations. Similar to penalty methods for rigid body contact constraints, these forces can only coerce a desired fluid reaction and often require small time steps for stability and accuracy. [33] also simulated a flapping filament, and coupled using an approach more similar to ours (using the solid's velocity and jumps in the fluid's pressure), but still only handled single phase fluids.

[68] pointed out that smeared out pressures profiles, such as those used in the immersed boundary method, can cause parasitic currents when used to make the velocity divergence free (see also [46]). A key to our method is the replacement of penalty forces with analytic constraints on the fluid velocity forcing it to flow as dictated by the velocity of the solid. Heuristically similar to the analytic methods of [7] for solving contact phenomena in rigid bodies, we replace the stiff inaccurate penalty forces of the immersed boundary method with a robust constraint that requires the solution of a linear system of equations greatly reducing the errors. Conveniently, we are already solving a linear system for the pressure, and it is readily modified to include the no flow constraint exactly as opposed to the only approximate enforcement via penalty forces. This is essentially a sharp interface approach similar in spirit to

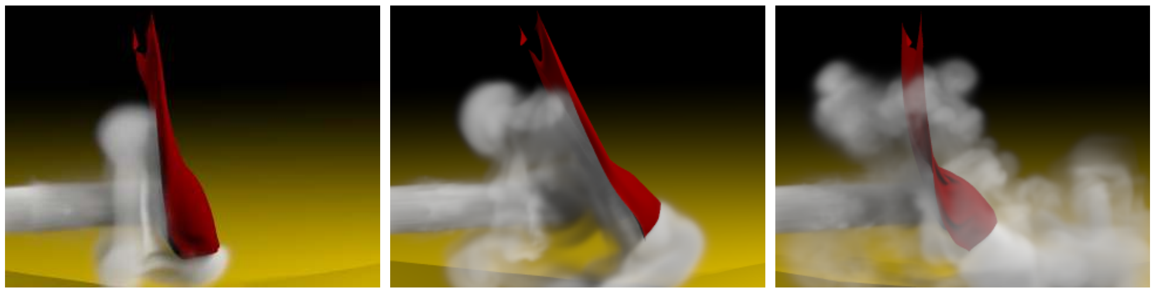

Figure 6.2: **Two-way coupled cloth and smoke (**$210 \times 140 \times 140$ **uniform grid, 30K triangles in the cloth).**

the immersed interface method (see e.g. [78]). However, we note that neither the immersed interface method nor the immersed boundary method has been used to solve solid/fluid coupling problems in the presence of liquid interfaces or thin films as we do here.

Most of the content in this chapter is extracted from our paper on coupling water and smoke to thin deformable and rigid shells [52]. Our main contributions include the use of robust visibility and ray intersections to prevent leaking, enforcing the solid's velocity on the fluid in a manner that reduces fluid mass loss, and the computation of a smoother pressure for coupling to help improve stability.

6.2 Preventing Leaks

Our goal is to completely prevent fluid from leaking across the thin shell which is represented by a moving triangulated surface. Two key elements necessary to achieve this goal are modifying our computational stencils to be one-sided and dealing with crossed-over nodes. These are described next.

6.2.1 One-Sided Stencils

In our fluid simulation, we typically need to interpolate data from the fluid grid. Trilinear interpolation accomplishes this by taking a weighted combination of the values at the 8 corner nodes of the surrounding grid cell. However, for use in our coupled

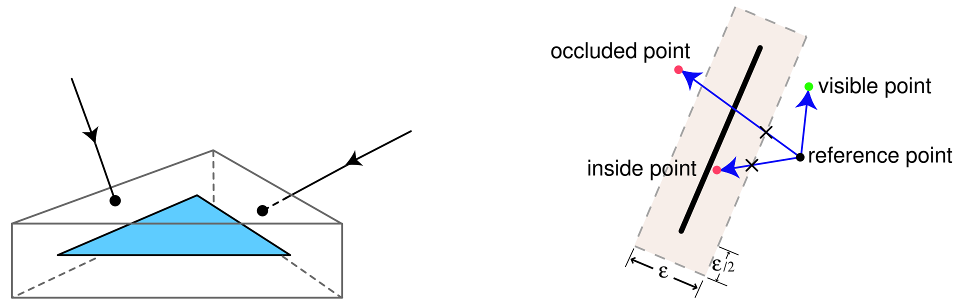

Figure 6.3: **(Left) We intersect with a triangle wedge that is formed by extending the edges and face in normal directions by $\varepsilon/2$. (Right) Given a reference point, another point can be classified as *visible*, *inside* a triangle wedge, or *occluded* by the result of a ray cast.**

simulator, we must be careful to exclude from our interpolation stencil nodes which lie on the opposite side of a solid's surface, as they represent data which is decoupled from the local fluid. Our approach is to use visibility and occlusion to determine which point combinations lie on the same side of the surface and may be used for interpolation (as well as for finite differences and other computations). This is accomplished via robust ray casting against thickened triangle wedges as in [17] (see also Figure 6.3 (left)). From the perspective of any reference point in space, the world is broken up into three regions: *visible* points, *occluded* points, and points *inside* a triangle wedge of the object. This partitioning is accomplished by casting a ray from the reference point to a point in question as shown in Figure 6.3 (right). Intersection tests are prone to roundoff errors, leading to a fuzzy notion of where exactly the object's surface lies. We pick the thickness ε so that the thickened wedge encloses this fuzzy region, ensuring that points outside of the thickened wedge can be determined to be visible or occluded in a robust manner depending on the result of the ray intersection test. The visibility of points labeled *inside* cannot be robustly determined, and in order to prevent any leaks we rule out these points, in addition to all *occluded* points, when constructing stencils for interpolation, differentiation, etc. Note that reference points which are themselves inside the thickened wedge are considered *invalid* (see the next section), and will not be incorporated into computational stencils of their neighbors since they are not visible to any other point.

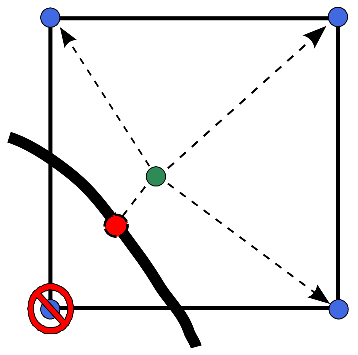

Figure 6.4: **In 2D, one-sided interpolation involves casting rays to the 4 surrounding nodes. In this case, the bottom-left node is occluded by the thin solid, and a replacement value will be used in its place in the interpolation formula.**

Our procedure for one-sided trilinear interpolation first tests the visibility of the 8 surrounding nodes as described above. Values at nodes which are not visible cannot be used, so instead we determine replacement values which will be used in their place during interpolation (see Figure 6.4). These replacement values are not stored on the grid, they are simply temporary ghost values computed on the fly. In particular, we use an ambient temperature value as a replacement value for temperature, and zero for smoke density. For velocity we use the local object velocity at the location where the visibility ray (first) intersects the object. Our technique for computing a replacement ghost value for ϕ will be described in Section 6.4.1. Note that our approach is flexible in how these are computed, so improved replacement values may be easily incorporated into our simulator.[1]

The main point here is that without a global notion of inside and outside for the solid, we are forced to determine whether points are on the same side on the fly, and compute replacement values as necessary. This is in contrast to the more typical ghost fluid approach of filling ghost values in advance. It is worth noting that recently [84] computed ghost values for multiphase flow on the fly in order to simplify their treatment of jump conditions.

[1]For example, a more careful consideration of normal and tangential velocity at the solid-fluid interface, similar to our approach for volumetric solids in Section 7.3.1, would allow us to model full-slip boundary conditions.

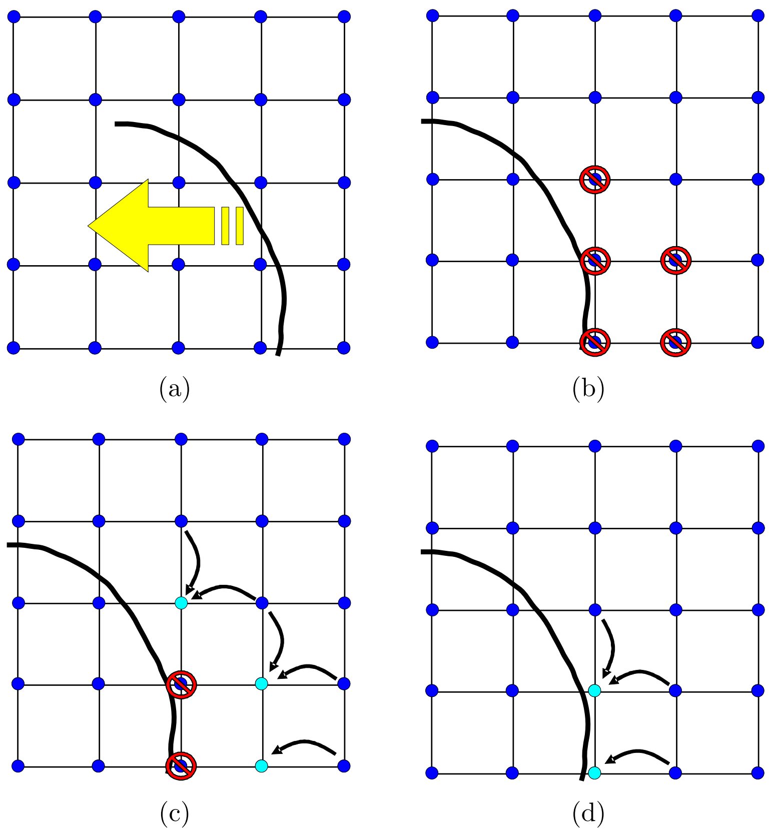

Figure 6.5: **Nodes crossed over by the solid between (a) and (b) are marked as invalid, and are revalidated by iteratively averaging from valid neighbors in (c) and (d).**

6.2.2 Invalid Nodes and Revalidation

When a thin solid moves, a point originally on one side of the object surface may be swept over by the surface and end up on the other side of it (see Figure 6.5 (a)-(b)). In this case, the values located at that point are invalidated for all subsequent interpolation, since they represent information from the other side of the object. Detecting such points is crucial to preventing leaks and is accomplished on a per-triangle basis. Each time step, we move the triangle nodes with linear trajectories, and consider a point *invalid* if it intersects the triangle itself (unthickened) during the time step. Checking this amounts to solving a cubic equation as in [17]. For robustness, we additionally consider a point *invalid* if it is either inside the triangle wedge at the beginning or at the end of a time step. Any point that does not start or stop *inside* a triangle wedge will robustly register a collision with an interior triangle if it crosses from one side of the object to the other. In the case of octrees, refinement leads to new point values that are also marked as *invalid*. Coarsening only involves the removal of nodes, and thus nothing special need be done.

We provide valid values for *invalid* nodes using a Gauss-Jacobi iterative scheme to propagate information. Each iteration, every *invalid* node is assigned the average of its valid *visible* one ring neighbor values and marked *valid* (see Figure 6.5 (c)-(d)). This technique of averaging uncovered points is similar to the blending methods used by others, see e.g. [15]. Complicated object geometry or folding may produce nodes that are still *invalid* after all iterations are complete. These nodes have no *valid visible* neighbors, and thus we again iterate in a Gauss-Jacobi fashion except this time using specially chosen values when visibility rays intersect an object. For example, we use the object velocity, a zero density, an ambient or object temperature, the positive distance to the object, etc.

6.2.3 Acceleration Structures

Since ray intersections are a key part of our leak prevention strategy, we employ a number of acceleration structures for efficiency. A standard axis-aligned box hierarchy (see e.g. [17]) is used for the triangulated surface accelerating intersection tests, etc. Moreover, for each triangle, a slightly enlarged bounding box is used to label all the voxels from the fluid simulation that are in close proximity to the surface, thus we can subsequently determine in constant time whether a grid location is sufficiently far from the surface so that we can avoid having to use the more expensive one-sided stencil.

6.3 Modified Fluid Simulation

We now describe the modifications necessary in order to incorporate the effect of the solid into our fluid solver described in Chapter 4.

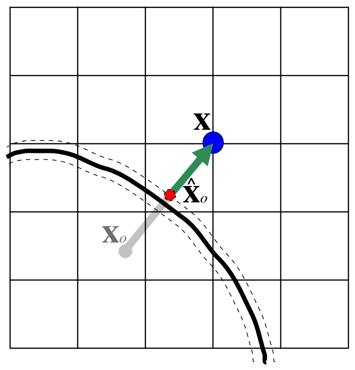

Figure 6.6: **The semi-Lagrangian ray from x to $\mathbf{x}_o$ is clipped against the thickened solid, and the intersection point $\hat{\mathbf{x}}_o$ is used in place of $\mathbf{x}_o$ as the base interpolation point.**

6.3.1 Node-Based Fluid Solver

A few modifications are required in our mixed node/face based scheme described in Section 4.4.1. First, when averaging from the faces to the nodes, rays are traced to the appropriate four faces (for a specific velocity component) and all *visible* faces are used to compute an average velocity on the node. If no faces are *visible*, we average the local object velocity from the locations where the visibility rays intersected the object. Then when averaging the scaled forces back to the faces, the appropriate rays are traced and all *visible* nodes are used to compute the average scaled force on a face. If no nodes are *visible*, we replace the face velocity with the average obtained using the object velocities determined by the visibility rays, i.e. scaled forces are not used in this case.

6.3.2 Semi-Lagrangian Advection

Our semi-Lagrangian advection procedure (Section 4.4.2) needs to be modified to be one-sided in order to prevent leaks. For each node, we trace a "semi-Lagrangian ray" from $\mathbf{x}$ to $\mathbf{x}_o = \mathbf{x} - \Delta t\mathbf{u}$, intersecting it with a triangle wedge that is double the usual size (i.e. using $\varepsilon' = 2\varepsilon$ in Figure 6.3 (left)). If an intersection is found, the point of intersection, $\hat{\mathbf{x}}_o$, will be subsequently used in place of $\mathbf{x}_o$ as the base interpolation point (see Figure 6.6). Doubling the thickness ensures that this clipped point remains *visible* to the point we are updating, $\mathbf{x}$, and that the subsequent 8 interpolation rays we send out from $\hat{\mathbf{x}}_o$ can accurately predict visibility[2] for the interpolation stencil. If any of these 8 secondary rays intersect the object, we use replacement values for that term in the trilinear interpolation formula (as in Section 6.2.1).

Since each time step we advect all of the nodal quantities along the same characteristics, we can avoid performing redundant computation by caching the results of the ray intersections performed during semi-Lagrangian advection, and reusing these for all advected quantities. For example, for a given node we can cache whether or not the semi-Lagrangian ray intersects the solid, and can also cache visibility information for one-sided interpolation at $\mathbf{x}_o$ (or $\hat{\mathbf{x}}_o$ if the semi-Lagrangian ray was clipped). Since replacement values are handled differently for the different quantities, these will be computed during the individual advections. Note that having all quantities share the same semi-Lagrangian rays is another benefit of having everything on the nodes in our fluids solver.

6.3.3 Computing the Intermediate Velocity

After advecting the velocity field using one-sided semi-Lagrangian, the object is moved to its new location and we label all nodes crossed over by the moving object as *invalid*. Finally, these nodes are given *valid* values as described in Section 6.2.2.

For vorticity confinement, when computing the curl of the velocity field we use the solid velocity in place of the fluid velocity for any of the six adjacent nodes which are not *visible*. The gradients of the vorticity magnitude are computed using central differencing of the six neighboring vorticity values, with each replaced by the vorticity of the center node if it is not *visible*.

6.3.4 Solving for the Pressure

The key mechanism by which the solid will "push on" the fluid is by enforcing the solid's velocity as Neumann boundary conditions at the solid-fluid interface. We found that thin films of water can quickly compress and lose mass against thin solid objects

[2]Through the two-ray path: from $\mathbf{x}$ to $\hat{\mathbf{x}}_o$ to the fluid node.

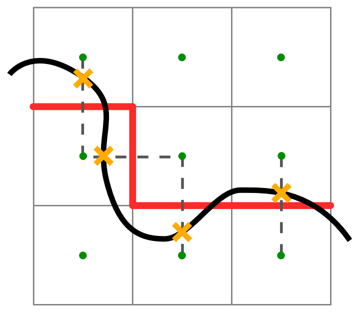

Figure 6.7: **Neumann boundary conditions (denoted by thickened faces) are enforced at a cell face if the ray between two adjacent cell centers (where pressures are defined) intersects an object.**

if one is not careful in how these boundary conditions are handled. In fact, correctly handling the boundary conditions is of utmost importance for mass conservation in general, as discrepancies between the fluid and object velocity cause fluid to flow into or out of an object losing or gaining mass respectively. Our method for handling this is one of the key observations and contributions of our work. First, we note that $\mathbf{u}^{n+1}$, the velocity we compute for the fluid during the divergence free projection, will be used in the *next* time step (i.e. going from time $n+1$ to $n+2$), and thus we need to make this commensurate with what the solid will do in the *next* time step. In order to do this, we calculate the size of the next fluid time step, evolve the solid object to its time $n+2$ state allowing the solid to take as many substeps as it needs to remain accurate and stable, calculate an *effective velocity* for each node in the solid by dividing its positional change by the size of the next fluid time step, and finally rewind the solid back to its time $n+1$ state (saving its $n+2$ state for later). Now the *effective velocity* represents exactly what the solid will do between time $n+1$ and $n+2$, and we use Neumann boundary conditions to force the fluid to move in exactly this manner allowing for excellent resolution of thin films of water colliding against cloth and thin shells. We cast rays from a cell center to the six neighboring cell centers to see if an object cuts through the line segment connecting the pressures as shown in Figure 6.7. And if so, we set a Neumann boundary condition at the cell face and set the constrained velocity there equal to the appropriate component of the *effective velocity* of the object. Then the divergence is computed in the standard

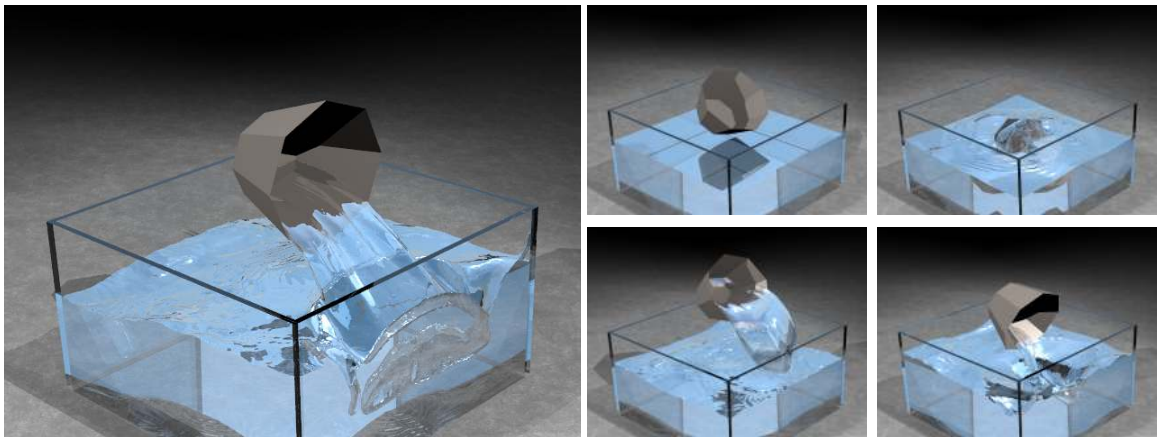

Figure 6.8: **A thin rigid kinematically controlled cup is filled with water, and then poured out ($160 \times 192 \times 160$ effective resolution octree).**

fashion, Equation (4.6) is solved, and the results are used in Equation (4.5).

A rather common difficulty with simulating highly deformable thin objects such as cloth in a fluid flow is that the cloth folds over on itself and pockets of fluid get separated from the flow. These are simple to identify by performing a flood fill algorithm over the fluid cell centers using the Neumann and Dirichlet conditions as the fill boundaries. If any region is surrounded entirely by Neumann boundary conditions, then the coefficient matrix assembled using Equation (4.6) has a null space corresponding to the vector of all 1's and is not invertible. However, there is a version of the conjugate gradient algorithm that can be applied to this matrix, if we first enforce the compatibility condition [111]. This is enforced independently in each region that has a null space using the area and velocity of the faces on the boundary to calculate the net flow per unit area into or out of the region. Then for each boundary face, we use this and the face area to obtain new temporary velocities that enforce no net flow across the region boundary. Finally, we solve for the pressure and make this region divergence free.

For water simulation, [32] describes a second order accurate discretization for enforcing the pressure boundary conditions at the water-air interface, which can significantly improve the smoothness of the water surface and reduce non-physical surface bumps. This technique was easily adapted to our coupled simulator. It requires

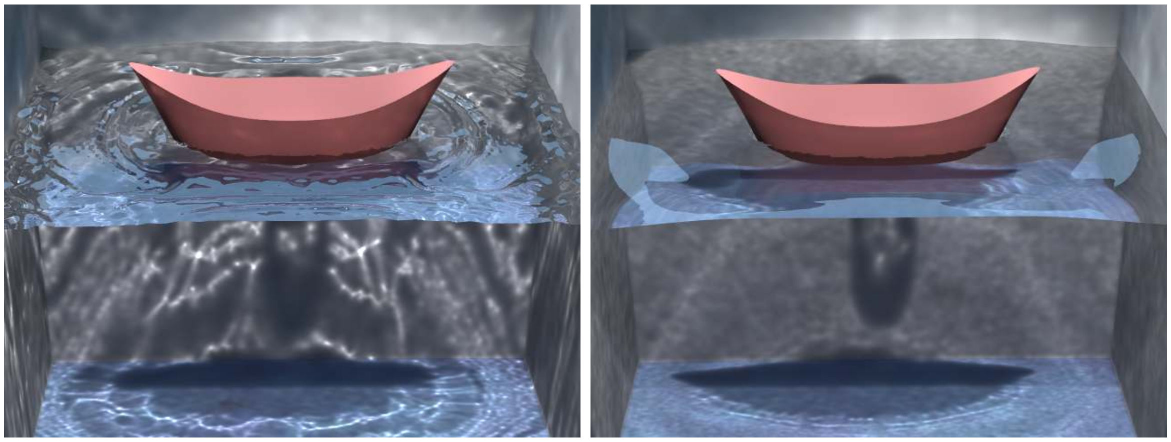

Figure 6.9: **(Left) First order errors in boundary pressure result in a bumpy surface. (Right) Using a second order accurate discretization results in a significantly smoother surface.**

cell-centered ϕ values, and these were computed using one-sided averaging from the neighboring nodal ϕ values. Using these cell-centered ϕ, a more accurate estimate for the location of the interface can be computed, and the second order accurate discretization is achieved by effectively enforcing the Dirichlet pressure boundary condition $p = 1$ at that interface location rather than at the center of the air cell. The dramatic difference in water surface as a result of using this technique is highlighted in Figure 6.9.

6.4 Water

For extrapolation of nodal velocities across the water-air interface, we use the basic technique described in Section 4.5, but in order to prevent velocities from leaking across objects, during the extrapolation procedure we rule out neighbor values that are not *visible*. It is possible that some points have no *visible* neighbors, and we temporarily label these points *invalid*. After extrapolation is complete, all *invalid* points are given *valid* values as explained in Section 6.2.2.

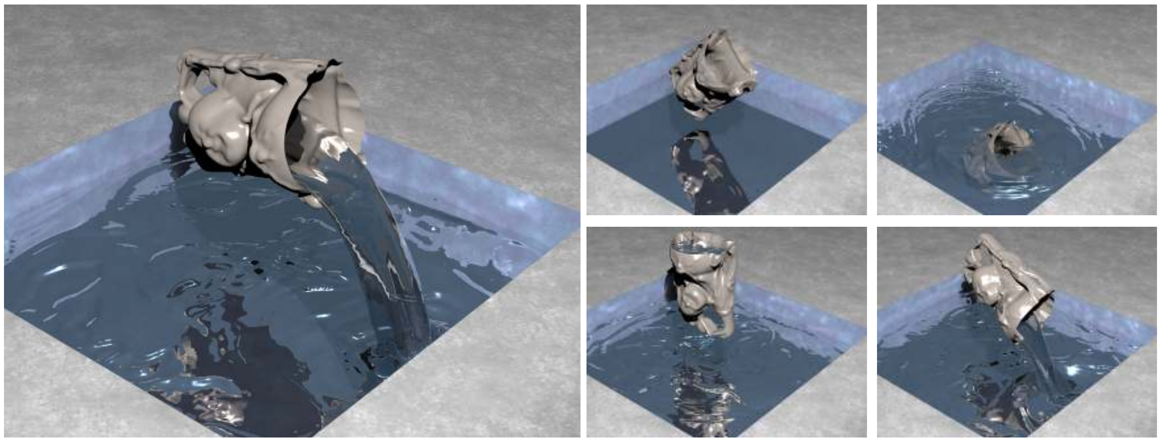

Figure 6.10: **A rigid kinematically controlled "Buddha cup" dipped, filled and poured out** (192^3 **effective resolution octree, 60K triangles in the rigid cup).**

6.4.1 Level Set Method

The level set is defined at the grid nodes, and thus we trace the same semi-Lagrangian rays as for velocity advection. When gathering the 8 values for trilinear interpolation, we replace nodes that are not *visible* with values averaged from a subset of the other 7 grid nodes of the cell whenever possible. In particular, we try to find a replacement value using the *nearest* neighbors on the same side of the solid as the interpolation point, which achieves something similar to constant extrapolation of the level set across the solid. Each point that needs a replacement value first looks to see if any of its three 1-ring neighbors (its edge-connected neighbors within the cell in question) are *visible* to the base interpolation point. If so, they are averaged to obtain a new value for the node in question. Otherwise, we check and average the three 2-ring neighbors (connected by a two edge path), or if that fails we consider the single 3-ring neighbor (diagonally opposite the node). If the process fails, there are no *visible* nodes in the cell and the point in question cannot be updated. We mark this node's ϕ value *invalid* and fix it in a postprocess (see Section 6.2.2).

For redistancing using the fast marching method, typically a list of interface nodes is found by looking for adjacent nodes with opposite signs in ϕ. We modify this list for the presence of thin solids by adding to this list any water node ($\phi \leq 0$) that has a neighbor that is not *visible*, and subtracting from this list any air node ($\phi > 0$) that does not contain a *visible* water node. These last two adjustments ensure that an interface exists up against the solid object, and that water does not have influence across the thin triangulated surface. Typically, each node in this list is given an initial ϕ value by considering how far it is from the interface in each of the directions that have a sign change. However, for air nodes we ignore directions where the neighbor is not *visible*, and for water nodes we use the minimum between the distance to the solid and $|\phi|$ in directions that are not *visible*. This last adjustment prevents water from incorrectly sticking to object. After initialization, we employ the fast marching method to march out from the interface in the usual fashion, ruling out neighbors that are not *visible* when updating a given point (similar to extrapolation of velocity values).

6.4.2 Particle Level Set Method

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

The particle velocity is determined by one-sided interpolation from the fluid grid. For negative particles that are closer than their collision distance to the object, we clamp the normal component of their velocity to be at least that of the object so that they do not get any closer to it. [30] showed that second order accurate particle evolution was quite important, especially when using the semi-Lagrangian method

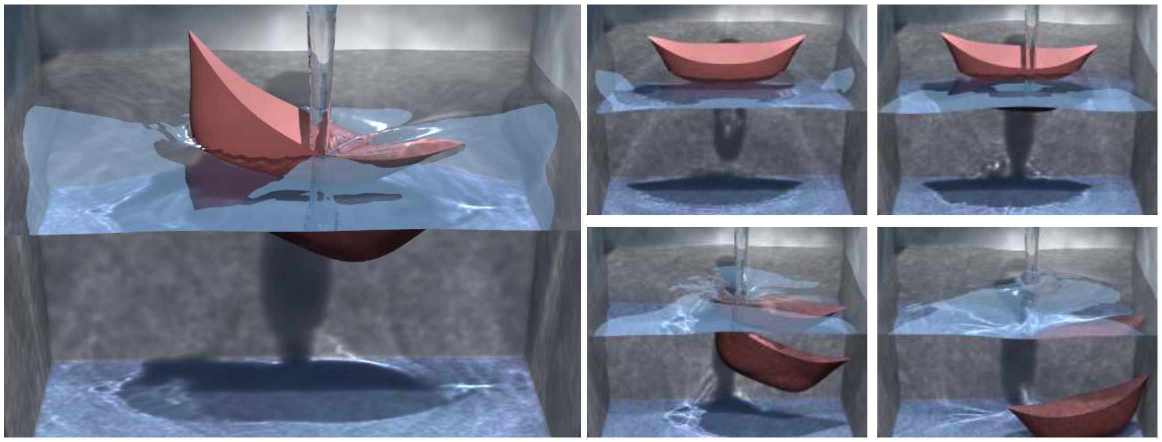

Figure 6.11: **A full dynamic simulation of a rigid body shell two-way coupled with water. The boat is heavier than the water, but retains buoyancy due to Archimedes' principle (effectively replacing displaced water with the air in its hull). Filling its hull with water causes it to sink, until it dynamically collides with the ground ($148 \times 148 \times 111$ uniform grid, 2.5K triangles in the dynamically simulated rigid boat).**

for level set advection. To achieve this, we first evolve the particles forward in time robustly colliding against the (stationary) object. Then we interpolate a new velocity at this location and average it with the original velocity to get a second order accurate velocity (as per Equation (4.7)), before moving the particle back to its original location. For negative particles, we clamp the normal component of this averaged velocity if they are either within their collision distance to the object or they collided with the object when they were originally evolved.

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

After updating both the level set and the particles, we modify the level set values using the particles. This is done in a one-sided fashion, only modifying a node using particles *visible* to it. The final step in the particle level set method is to adjust the radius of the particles based on the values of ϕ, and possibly delete particles that have escaped too far from the interface. This is accomplished by evaluating ϕ at the center of the particle in the same fashion as is done at the base of a semi-Lagrangian ray during level set advection. Particle reseeding (every 10-20 frames) is performed by initially disregarding the object altogether (for efficiency). Then, as a postprocess, we evaluate ϕ at the center of the particle and delete particles with the wrong sign.

6.5 Cloth and Thin Shell Simulation

A significant feature of our approach is that one can use their favorite simulation technology for the solid object independent of the fluid solver and the solid/fluid two-way coupling algorithm. All that is required is the ability to apply external forces to the solid, and to get back the positions of the nodes of the triangulated surface from which we can calculate velocities assuming linear motion between fluid time steps. When velocities within a triangle are required, we interpolate using barycentric coordinates. For rigid body simulation, we use the method described in Chapter 2, although our examples require no technology beyond that in [54, 97]. For cloth, we use the basic cloth model from [17, 18] as outlined in Chapter 3.

6.5.1 Coupling to the Fluid

The traditional method for coupling fluids and solids is to use the solid to prescribe velocity boundary conditions on the fluid, and the fluid to provide force boundary conditions on the solid [15]. And as mentioned in Section 6.3.4, it is important to coerce the fluid to move with the velocity of the solid so that thin films and sheets can be supported with little to no mass loss. For compressible fluids, one typically uses

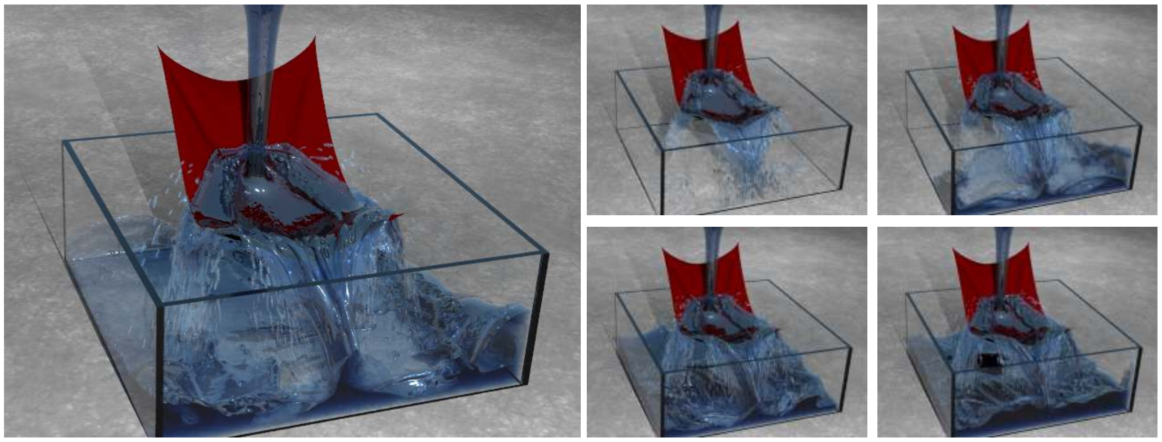

Figure 6.12: **Two-way coupled water flows over cloth (suspended at four corners), demonstrating that thin objects can support a sheet of water without leaks** (200^3 **effective resolution octree grid, 30K triangles in the cloth).**

the pressure to provide a force on the surface of the solid, but for incompressible flow the pressure can be both stiff and noisy as compared to the velocity field, as discussed in [35]. This is because the pressure computed in the projection step essentially acts as a Lagrange multiplier enforcing incompressibility as a hard constraint. At the same time, the projection step is used to enforce the solid velocities on the fluid, required for better mass conservation. Trying to enforce these two hard constraints simultaneously leads to a stiff pressure.

Our approach is to compute a separate, smoother pressure p_c more suitable for coupling by relaxing one of these two hard constraints – namely, by not enforcing the solid velocity on the fluid. To do this, we temporarily treat the solid as a fluid of different density. We emphasize that the following steps are done as a side computation, only used to compute p_c, and not affecting the actual fluid velocity or incompressible pressure p.

As in Figure 6.7, we "rasterize" the solid onto the grid faces by casting rays from each cell center to the 6 neighboring cell centers to see if an object cuts through the line segment connecting the pressures. If so, we copy the local solid object velocity to the corresponding cell face in the usual fashion. We have some choice as to which solid velocity to use: we could use its instantaneous time n velocity, or its *effective velocity* between time n and $n+1$ (as is done for the projection step). For the examples in this chapter we chose the former approach, using the current instantaneous velocity of the solid. However, in Section 7.6.1 a better choice for the solid's velocity will be described, producing more accurate buoyancy effects.

In addition to setting the solid's velocity on the grid, we also need to rasterize its density in order to make the coupling pressure reflect the correct relative density between the fluid and the solid. In our discretization, density lives on the cell faces, just like velocity. In order to compute the effective density that should be set on the face, we multiply the area of the face by the surface density of the solid to calculate a mass, m, for the cell face. Then we divide this mass by the local control volume which amounts to half of each neighboring cell (divided up appropriately among the faces for octrees). For a uniform grid $V = \triangle x \triangle y \triangle z$. Finally, the density of the cell face is set to m/V. All other cell faces have their density set to the fluid density. Since this is now a variable density flow, we need to solve a variable coefficient Poisson equation as in [101] where the analogue to Equation (4.6) is

$$\nabla \cdot \left(\frac{\nabla p_c}{\rho_{mix}} \right) = \frac{\nabla \cdot \mathbf{u}^*_{mix}}{\Delta t}. \tag{6.1}$$

Note that $\mathbf{u}^*_{mix}$ is set to the fluid velocity $\mathbf{u}^*$ on the fluid faces and the solid velocity (as computed above) on the solid faces. Similarly, ρ_{mix} is set to the mixed fluid and solid densities.

Note that we do not use p_c to modify the fluid velocity, nor do we attempt to move the solid with the fluid velocity as for example in [20] and [154]. We only compute p_c in order to determine an external force to apply to the solid. The key advantage is that we do not have to figure out how to model solid response due to buoyancy, collisions, elasticity, etc. on the fluid dynamics grid and can instead simulate the solid with any black box method including finite elements, masses and springs, one's favorite collision algorithms, implicit or semi-implicit time integration, etc. As an added bonus, we are not hindered by the resolution of the fluid dynamics grid when evolving the solid, and a coarse grid merely provides a smeared out fluid force while still allowing all other solid dynamics and motion to be modeled with independently high resolution.

The force on the solid will be based on pressure jumps across the solid, so we first calculate the pressure difference across each cell face that was considered to be part of the solid. Since interpolation is more straightforward on nodes, the next step is to transfer these pressure jump components from faces to nodes. We do this by assigning the nodal value to be the average of the up to four face values (for a given coordinate axis) that may be adjacent to it. We then spread these nodal differences to a band of grid points near the solid using the same Gauss-Jacobi averaging used to validate *invalid* nodes. This entire process is carried out independently for the pressure differences in each of the three coordinate directions, and results in vector pressure jumps stored on the nodes.

The force on a triangle is computed by first interpolating the nodal pressure jumps to the barycenter of the triangle, multiplying by the triangle's area (to get force from pressure), and finally projecting onto the negative normal direction for that triangle (negative because the fluid pushes towards *lower* pressure regions). That is, $\mathbf{F} = -A\left([\mathbf{p}_c] \cdot \mathbf{n}\right)\mathbf{n}$, where $[\mathbf{p}_c]$ is the vector pressure jump and A is the triangle area. In the case of a rigid body, the force and torque are accumulated by accounting for all the triangles. In the case of cloth, one third of the net force on a triangle is distributed to each node. For objects with coarse triangles, multiple sample points on each triangle may be used for more accurate computation and application of pressure jumps.

Since Neumann boundary conditions are enforced across the thin triangulated surface when solving Equation (4.6), the computed pressure may have large jumps and contain noise. In contrast, Neumann conditions are not enforced when solving Equation (6.1), providing for a much more stable coupling mechanism and smoother pressure values. However, the key is that our algorithm consists of *both* of these important pressure solves: Equation (6.1) is used to compute a smoother pressure for coupling in order to make the fluid affect the solid, and Equation (4.6) is used to compute an incompressible pressure which enforces the solid's velocity on the fluid. [20] only performed a single pressure solve, treating the solid as a fluid similar to our solve for p_c, but without additionally enforcing the solid's velocity on the fluid in a separate projection step. It then takes the continuous velocity field computed for the combined solid/fluid domain and constrains the grid cells within the solid to have a velocity consistent with rigid body motion. This makes the velocity field discontinuous at the surface of the solid, allowing fluid to flow directly into the solid and disappear.

6.6 Algorithm Summary

The overall coupling algorithm to advance from time n to $n+1$ proceeds as shown below (c.f. Section 4.6). Note that at the beginning of this time step we already know the solid's $n+1$ state as it was already computed during the previous time step. Knowing the solid's state one step ahead of the fluid is necessary because in order to compute $\mathbf{u}^{n+1}$ at the end of the time step we need to know the solid's *effective velocity* between $n+1$ and $n+2$. However, by saving the solid's state before advancing it, we maintain at our disposal both its start and end states for this time step, and can make sure that computations are done with the solid in the appropriate configuration. For example, when checking for crossed-over nodes, we intersect grid nodes against the solid's swept motion between its n and $n+1$ states.

- Update fluid scalars:

  Smoke: Update smoke density and fluid temperature in a one-sided manner (Section 6.3.2)

  Water: Update level set and particles in a one-sided manner (Section 6.4)

- Compute intermediate fluid velocity $\mathbf{u}^*$ using knowledge of the solid's position at both time n and $n+1$ (Section 6.3.3)

- Move solid to its time $n+1$ state

- Compute coupling pressure p_c (Section 6.5.1)

- Advance solid to time $n+2$ state, applying forces based on p_c (Section 6.5.1)

- Move solid back to its time $n+1$ state (saving its time $n+2$ state for later)
- Project $\mathbf{u}^*$ to get a divergence-free $\mathbf{u}^{n+1}$ while enforcing the solid's *effective velocity* ($n+1$ to $n+2$) at the solid-fluid interface (Section 6.3.4)

For our coupled simulator we pick a time step Δt such that information travels at most a fraction α of a grid cell. We commonly use values such as $\alpha = 0.9$ or $\alpha = 0.5$ in our simulations. We also make sure Δt is small enough so that moving solids do not cross more than that fraction of a grid cell in a time step, though this is typically already satisfied by our first restriction since the fluid moves at the same speed as the solid at their common interface.

6.7 Rendering

Rendering smoke and water against thin objects such as cloth poses complications as standard techniques interpolate density and level set values with stencils that intersect with thin bodies. Thus, we augmented our basic ray tracer with robust intersections and interpolations. That is, we used the same visibility based schemes used for simulation as described in Section 6.2.1. This effectively removes visual artifacts caused by smoke and water showing through to the other side of objects, and air pockets showing through to the smoke and water side.

Additionally, we experimented with rendering the removed negative particles generated by the particle level set method in order to add visually interesting splash and spray to our simulation of water flowing over suspended cloth. Spray particles were also rendered in [40]. Recall that the removed negative particles are negative (water) particles which have escaped into the air domain. By evolving them using simple ballistic motion we can consider them to represent water droplets. To make them "appear" like water droplets in motion, we assigned each removed negative particle an ellipsoidal shape by stretching a sphere along the direction parallel to the particle's velocity. In particular, we assume the base sphere has radius $\hat{r} = sr_{particle}$, s a scale factor multiplying the radius, $r_{particle}$, assigned to the particle in the particle level set method. Then the stretched ellipsoid is set to have major axis radius of $r_1 = \hat{r} + \frac{1}{2}\Delta t\ v$

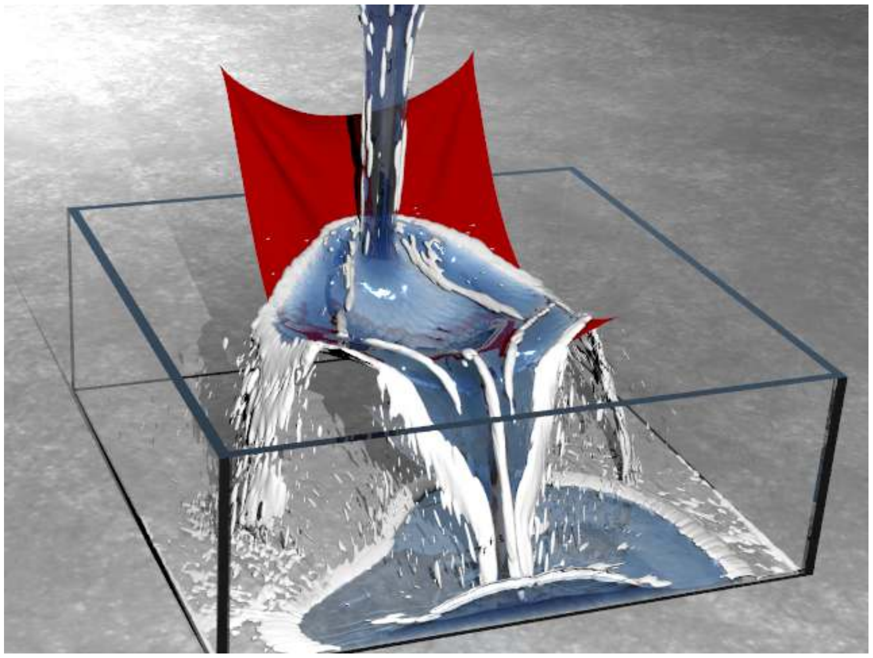

Figure 6.13: **Illustration of the removed negative particles rendered as an opaque Lambertian surface (after proper blending). These particles have crossed over to the wrong side of the level set surface, but are too finely detailed to be properly represented by the fluid simulation grid.**

and the other two axes are set to $r_2 = r_3 = \sqrt{\hat{r}^3/r_1}$ which preserves the volume of the sphere. For our example we used $s = 12$.

One approach to rendering these ellipsoidal droplets could be to directly ray trace them using analytic intersection routines. Instead we implicitly represent the droplets using a level set, leveraging our existing code for ray tracing a level set volume. Since the removed particles are too small to be represented on the grid (otherwise the particles would be surrounded by water nodes and they would not be "removed"), the grid around each particle must be further refined. This is most suitably done using octrees, since they allow adaptive refinement.[3] Note that all of this is a post-process, and since we are not simulating on these grids we can typically get away with rendering on highly refined grids.

[3]Of course, a simulation originally performed on a uniform grid can transfer its data to an octree grid for rendering if particle rendering is desired.

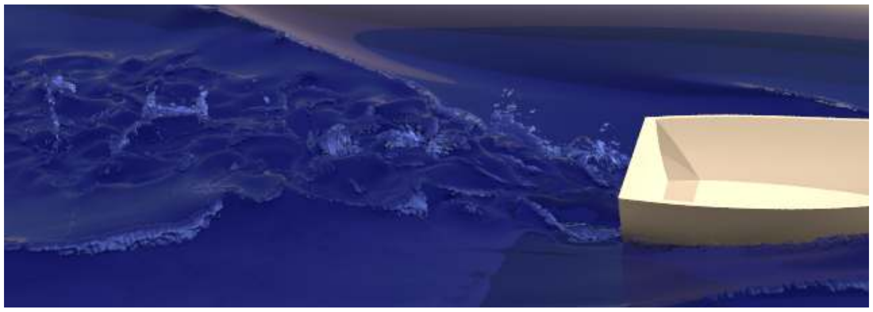

Figure 6.14: **Our removed particle rendering technique was recently used to add detail to a boat wake in [63].**

The water droplets are rasterized on the refined octree, treating them as ellipsoidal "blobbies" in order to create smoother blending between particles. The main level set (representing the simulated water) is also resampled onto the same octree. The two level sets are then merged by taking the *min* of their signed distance values and redistancing, resulting in a single level set volume for rendering. By keeping track of which portions of the level set came from the bulk water and which from the droplets it is possible to shade the two differently. For example, the particles were rendered opaque white in Figure 6.13 and slightly lighter than the bulk water in Figure 6.12. As can be seen in these figures, the blending between the droplet shapes and the bulk water is still not as smooth as we would like, so particle rendering is one area for future work. More recently we used this technique in [63] to render splash and spray in the wake of a boat (Figure 6.14).

6.8 Examples

We were able to simulate computational grids with effective resolutions as large as $256 \times 256 \times 192$ for the fluid and as many as 90k triangles for the rigid and deforming bodies using a 3 GHz Pentium 4. The computational cost ranged from 5 to 20 minutes per frame, and thus the longest examples took a couple of days. The most expensive computations in our algorithm were the many ray intersection tests associated with the particles, since there tended to be many particles and they needed intersection tests for one-sided interpolation (ϕ and $\mathbf{u}$) as well as for handling collisions against the moving solids.

Figure 6.2 depicts a smoke stream flowing toward a suspended cloth curtain, and the two-way coupling generates interesting wrinkles and folds as well as smoke motion. Figure 6.8 depicts a kinematically controlled cup dipped, raised, and poured, demonstrating that our technique can model liquid behavior on both sides of a triangulated surface independently. Figure 6.10 shows similar behavior for more complex geometry. Full two-way coupling can be used with rigid shells as well, and Figure 6.11 shows a fully dynamic simulation of a boat floating until a stream of water sinks it. Figure 6.12 shows a stream of water flowing over a piece of cloth demonstrating full two-way coupling of cloth and water. Note specifically that the cloth supports the water (without leaks) and produces highly detailed thin water sheets flowing off the sides. For this example we augmented our ray tracer to additionally render the removed negative particles as described in Section 6.7. Lastly, Figure 6.1 depicts a stream of water flowing onto a cloth curtain causing it to deform.

6.9 Conclusions

We have presented a new computational algorithm for the coupling of incompressible flows to thin objects represented by moving triangulated surfaces. Examples were presented to demonstrate that this algorithm works well for one phase fluids such as smoke and for fluids with interfaces such as water. Moreover, it works with both rigid and deformable triangulated surfaces. Most importantly, our method prevents the leaking of material across the triangulated surface, while accurately enforcing the incompressibility condition in a one-sided fashion allowing for the interaction of thin films of water with highly deformable thin objects such as cloth.

Chapter 7

Solid-Fluid Coupling for Volumetric Solids

7.1 Introduction

Our strategy for coupling to volumetric bodies is partly based on that presented in the preceding chapter for coupling to thin shells. In particular, it employs two different pressure solves to achieve two-way coupling, as for thin shells, but does not require one-sided computational stencils. It is general enough to handle both rigid and deformable volumetric solids, although the example presented in Section 7.7 will demonstrate coupling to rigid solids only. Recall that our approach allows one to use one's favorite (and previously implemented) methods for simulating the solid independent of the coupling. This is in contrast to [20] which, in addition to being restricted to rigid bodies, requires special treatment of the solid. The main philosophical difference between the two approaches is that we compute the effect of the fluid on the solid via a force per vertex that can be added to *any* solid simulation technique, while [20] allows the fluid to completely determine the solid's velocity overwriting all internal dynamics (such as elastic deformation). In recent work, [152] applied the distributed Lagrange multiplier approach, which forms the basis of the rigid fluid method of [20], to coupling with *deformable* volumetric solids.

A number of components of the algorithm we describe below are previously published techniques. One *novel* aspect of our algorithm is adapting our thin shells coupling strategy to volumetric solids, allowing us to compute a smoother pressure for coupling while still enforcing the solid's velocity on the fluid (Section 7.6). The other main contribution is a more accurate computation for buoyancy (Section 7.6.1). The content of this chapter is based on part of our work on melting and burning solids [83], which incorporated two-way coupling for one of its examples.

7.2 Rasterizing Solids onto the Fluid Grid

For thin solids, we had to resort to one-sided computational stencils which involved performing ray intersections with the solid. Since volumetric solids have a distinct inside and outside, we can instead rasterize them onto the fluid grid in order to facilitate interaction with the fluid. This is a diffuse rather than sharp approach, but the benefit is in the reduced computational expense as compared to performing many intersection tests. In Section 8.4, we consider a possible sharp approach for volumetric solids.

As was done in [61, 116], we create a level set representation of the solids to be stored on the fluid grid. Creating this representation can be accomplished by resampling a solid's existing signed distance function onto the fluid grid, or by following the rasterization procedure outlined in Section 2.3.1 for solids with no existing level set representation. Having a level set for the solids defined on the fluid grid allows us to quickly determine whether a given fluid node is inside or outside any solid. In fact, multiple solids can all be rasterized onto a single signed distance function, essentially performing a union operation on their individual level sets. This will make inside/outside queries constant time regardless of the number of bodies present. A body identification number can additionally be stored on the fluid grid for those cases where knowledge of the specific body present at that location is needed. The level set for the solids, ϕ_{solid}, is negative inside and positive outside, so its gradient gives us an outward-pointing normal, $\mathbf{n}_{solid}$.

7.3 Ghost Values for Boundary Conditions

Since one-sided computational stencils will not be used for volumetric solids, values at grid nodes which lie inside a solid may be used during fluid computation, and therefore need to be filled with ghost fluid values. For thin shells, ghost values were computed on the fly, for example replacing values of non-visible nodes during one-sided interpolation and advection. For volumetric solids, we can fill nodes inside the solid ($\phi_{solid} \leq 0$) with ghost values in advance. Given a time step restriction limiting how many grid cells the semi-Lagrangian characteristic ray may span (see Section 6.6), only nodes within a fixed band inside the solid will ever participate in fluid computation, reducing the number of nodes that must be filled.

Similar to thin shells, there are a number of different quantities we may need to compute ghost values for, and we may compute them in different ways in order to enforce desired boundary conditions. For smoke simulation, ghost values for smoke density and temperature need to be filled inside the solid. As for thin shells, we can simply use zero density, and ambient or object temperature. Ghost values for velocity and the level set in the case of water are more complex and will be discussed next.

7.3.1 Velocity Ghost Values

We use the constrained velocity extrapolation approach of [61] (see also [116]) to fill velocity ghost values for nodes inside the solid. The computation of velocity values inside the solid is motivated by the boundary condition we are trying to model at the solid-fluid interface. At the interface, a physical kinematic boundary conditions dictates that the velocities of the fluid and the solid match in the normal direction. On the other hand, enforcing this condition may hinder a liquid from separating from the solid, and an approach to help improve separation is to instead enforce $(\mathbf{u}_{fluid} - \mathbf{u}_{solid}) \cdot \mathbf{n}_{solid} \geq 0$. That is, the fluid velocity at the interface is only prevented from penetrating the moving solid in the normal direction. In the tangential direction, we can model a no-slip condition in which the tangential velocity components are equal (as would occur with a viscous fluid), or a full-slip condition in which the tangential components are decoupled. As in [61], a friction coefficient $0 \leq \mu \leq 1$

may be used to create a partial slip condition (interpolating between the no-slip and full-slip boundary conditions).

In practice, we implement these boundary conditions in the interior of the solid, using ghost values, rather than at the solid-fluid interface. In order to construct these ghost values we need both solid and fluid velocities at an interior solid node. The solid velocity can be determined by querying the solid occupying that space for its local velocity. To compute a "fluid" velocity inside the solid, we would like to copy the velocity from the nearest fluid node, and one way to approximately achieve this is by extrapolating surrounding fluid velocities into the solid. We can extrapolate using the same technique used in Section 4.5.1 to extrapolate velocity across the water-air interface, but using the signed distance function $-\phi_{solid}$ (negated because we want to extrapolate *into* the solid). Then the ghost value at a node is computed by combining the solid velocity $\mathbf{u}_{solid}$ and the extrapolated fluid velocity $\tilde{\mathbf{u}}_{fluid}$ using

$$\mathbf{u}_{ghost} = \max(\tilde{u}_{fluid,n}, u_{solid,n})\mathbf{n}_{solid} + ((1-\mu)\tilde{\mathbf{u}}_{fluid,t} + \mu\mathbf{u}_{solid,t}) \,, \tag{7.1}$$

where the additional subscripts n and t denote normal and tangential components, respectively.

7.3.2 Level Set Ghost Values

For water, ghost values for its level set ϕ need to be computed for nodes interior to the solid. We follow the technique described in [116], whereby ϕ is extrapolated from surrounding fluid nodes into the solid. Additionally, we need to take care to reduce mass gain due to pulling water out of the solid. Following [116], nodes at which the extrapolated level set value satisfies $\tilde{\phi} < 0$ and at which the extrapolated fluid velocity additionally satisfies a separation criteria ($\tilde{u}_{fluid,n} - u_{solid,n} > .1|\tilde{\mathbf{u}}_{fluid} - \mathbf{u}_{solid}|$) are instead set to a positive value $|\phi_{solid}|$.

This solution is still not completely satisfactory because while it helps water separate by preventing $\tilde{\phi} < 0$ from getting pulled out of the solid, it still allows $\tilde{\phi} > 0$ to be pulled out which may create air pockets underwater. One possible improvement is to disallow a separating fluid velocity unless there are neighboring air nodes. Conceptually, this allows water to separate from an object only if the formed cavity can be filled by nearby air. We have had some success with this neighboring air condition, but it seems as though it might be easier to control separation conditions at the solid-fluid interface rather than by filling ghost values inside the solid. For this reason it may be worth clipping semi-Lagrangian rays against the solid, and incorporating a one-sided approach for volumetric solids similar to that used for thin shells. This will be discussed as possible future work in Section 8.4.

7.3.3 Freshly Cleared Nodes

Similar to crossed-over nodes in thin shells coupling, when volumetric solids move, some nodes previously inside the solid end up outside. We need to ensure these "freshly cleared nodes" have reasonable values for the subsequent fluid simulation. For thin shells, these nodes obtained valid values through averaging from valid neighbors (see Section 6.2.2). In the case of volumetric solids, our approach is to simply allow freshly cleared nodes to retain the ghost values they were filled with. We have found that these values give reasonable results in our simulations.

7.4 Particle Level Set Method

Handling the interaction between the solids and the particles of the particle level set method is easier for volumetric solids than for thin solids. For thin shells, particles require ray intersections for one-sided interpolation as well as to prevent them from crossing over the solids during particle advection. With volumetric bodies we are lucky because the ghost values defined in the solid can be used in place of performing one-sided interpolation, and colliding the particles against the solid is made easy by using ϕ_{solid}. Using this signed distance function allows us to easily determine when particles are within collision distance of the solid, and allow us to push particles out of the solid along the normal direction $\mathbf{n}_{solid}$. In particular, we can push them out to their precomputed target collision distance to help uniformly stagger them in a band around the solid's interface. This is similar to the approach in [116].

7.5 Solving for the Pressure

One of the key contributions of our thin shells coupling approach was reducing mass loss at the solid-fluid interface by enforcing the *effective velocity* of the solid on the fluid. We do the same for volumetric solids, enforcing this velocity during the pressure projection step (as in Section 6.3.4).

For volumetric solids, we need to set Neumann boundary conditions on grid faces along the boundary of the solid. There are various ways this can be accomplished. For example, one approach is setting Neumann conditions on *all* faces inside the solid, since setting interior solid faces to Neumann in addition to boundary faces does not affect the pressure solution in the fluid region. Determining which faces are inside could be done by performing an inside/outside test at the location of the face center, or by checking the sign of ϕ_{solid} averaged from the four corner nodes of a face. An alternative approach, which only sets boundary faces to Neumann, is computing cell-centered ϕ_{solid} and marking faces whose two adjacent cell centers have opposite signs. In any case, these various approaches all give boundaries that are to within $O(\Delta x)$ of each other. The *effective* solid velocities to be enforced on the fluid are also set on these Neumann faces. The rest of the pressure projection step proceeds just as for thin shells.

7.6 Coupling to the Fluid

As for thin shells, we determine fluid forces on the solid by computing a smoother coupling pressure p_c, temporarily treating the solid as a fluid. We set the solid's density and velocity on all faces inside the solid domain, and solve the variable density Poisson equation (Equation (6.1)) without enforcing Neumann boundary conditions. Since the solid's density is already a volumetric density, we can simply set that density on the grid face without having to compute masses and control volumes as was done for thin shells.

For thin shells, it was the pressure *jump* across the solid that was producing a net force on the solid. In the case of volumetric solids, we instead apply the pressure itself around the boundary of the solid. In order to do this, a pressure value needs to be computed at the barycenter of each surface triangle. Solving Equation (6.1) determines p_c everywhere in the combined fluid and solid domain,[1] and we have found the simulation gave reasonable results if we simply interpolated the pressure values directly from the grid without differentiating between solid and fluid cells. Alternatively, one could extrapolate pressure from fluid cells into the solid in order to ensure only pressure values associated with fluid are used in the interpolation. Additionally, it should be noted that pressure values would need to be transferred from cell centers to the nodes in order to enable interpolation on octrees.

Force is computed from the interpolated pressure using $\mathbf{F} = -Ap_c\mathbf{n}$. For rigid bodies, these forces and associated torques are accumulated around the boundary to produce a net force and torque on the center of mass. For deformable solids, the force is equally distributed to the nodes of the surface triangle. As for thin shells, coarse surface triangles can be supersampled to compute a more accurate force distribution.

7.6.1 Improved Coupling Force

As mentioned in Section 6.5.1, when solving Equation (6.1) for p_c, we set $\mathbf{u}^*_{mix}$ to $\mathbf{u}^*$ in the fluid region and to the solid's velocity in the solid region. Options for which solid velocity to use included its instantaneous velocity and its *effective velocity*, but in [83] we proposed a better choice which improves the overall accuracy of the fluid to solid coupling forces. Consider a stationary, neutrally buoyant object submerged in a still fluid, where it should remain at rest. Since the advection term is $\mathbf{0}$, the fluid's intermediate velocity is simply $\mathbf{u}^* = \Delta t\mathbf{g}$. In order for Equation (6.1) to give the correct (hydrostatic) coupling pressure, $\mathbf{u}^*_{mix}$ inside the solid should be identical to

[1]However, pressure values are not computed in cells just outside the simulation domain, where either Dirichlet or Neumann boundary conditions are enforced. In particular, while Dirichlet cells have a fixed pressure set, exterior cells adjacent to Neumann boundary faces will not have a pressure explicitly computed. One option is to set an exterior Neumann cell to have the same pressure as its interior neighbor, consistent with the Neumann condition, $\partial p/\partial n = 0$, at their joint face. A more accurate alternative is mentioned in [83].

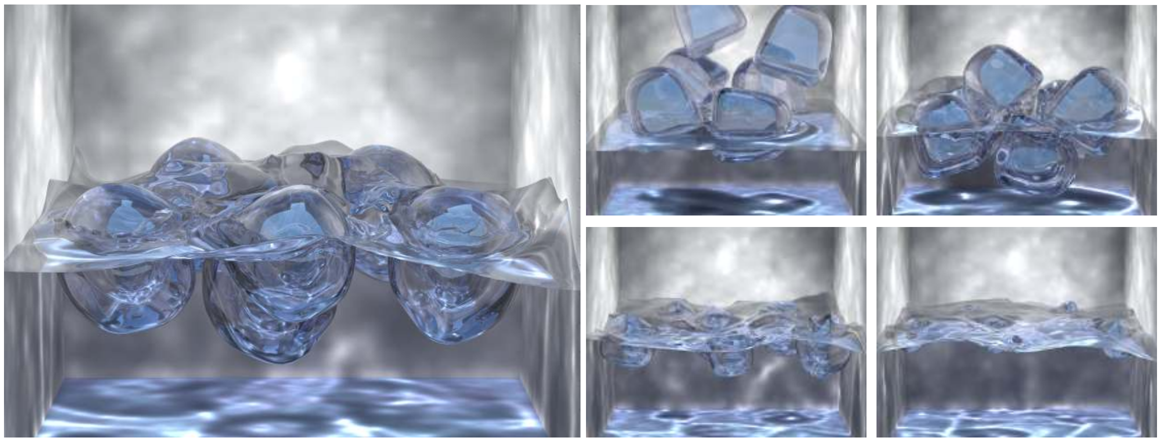

Figure 7.1: **Rigid ice cubes floating and melting in water with full two-way force coupling (100^3 grid, 600K total surface triangles).**

the fluid's $\mathbf{u}^*$. If $\mathbf{u}^*_{mix} = \mathbf{0}$ was used inside the solid region (which is both the solid's instantaneous and *effective velocity*), an incorrect coupling pressure would arise due to the incompatibility between the solid and fluid velocity causing the solid to incorrectly accelerate. This illustrates that the velocity used in the solid region should be more in line with the intermediate $\mathbf{u}^*$ velocity computed for the fluid, which accounts for all forces except those due to the pressure. That is, we need to compute the velocity the solid would have in the absence of fluid forces. Our approach computes an estimate for this without having to tentatively advance the solid. Starting with the *effective velocity* of the solid which represents the change in the solids position from time n to time $n + 1$, we subtract out the fluid force applied to the solid in the *last* iteration. Using this for $\mathbf{u}^*_{mix}$ in the solid region gives us an approximation to the solid velocity which incorporates all forces except fluid pressure. In particular, it correctly handles the neutrally buoyant example.

7.7 Example

In [83], we presented an algorithm for melting and burning solids. A particular example of ice cubes melting in water made use of the two-way coupling technique described in this chapter (Figure 7.1). Since the erosion of the ice cubes occurred in a discrete manner, with one erosion step per simulation step, between erosion steps the ice cubes could be treated as simple rigid bodies and simulated in the manner described in Chapter 2. During an erosion step the shape and mass properties of each ice cube were recomputed, and the solids were re-rasterized onto the fluid grid. This simulation was run on a uniform grid, and more accurate buoyancy was observed when computing $\mathbf{u}^*_{mix}$ in the solid region using the technique proposed in the previous section as opposed to using either the solid's instantaneous or *effective velocity*.

7.8 Conclusions

This chapter described a method for adapting the fluid simulator of Chapter 4 for two-way coupling with volumetric solids, using the two pressure solve approach developed for thin shells in Chapter 6. One-sided computational stencils and ray intersections, used for coupling with thin shells, were replaced by solid rasterization and interior ghost fluid values. Coupling the solid to the fluid was achieved by integrating pressure, rather than pressure jumps, along the solid's surface. Accuracy of the computed coupling forces was further improved by a more careful treatment of solid velocities used in the variable density Poisson equation. An example of floating, melting ice cubes was presented which demonstrates this algorithm for volumetric rigid bodies. Although applying this algorithm to deformable solids is left for future work, we believe this approach is general enough to handle this case.

Chapter 8

Conclusions and Future Work

This dissertation presented novel algorithms for rigid body simulation as well as for simulations coupling thin and volumetric solids to a fluid. The focus was on applications for computer graphics, and the techniques described were aimed at producing high-resolution, visually detailed simulations, rather than trying to achieve real-time results. Applications such as virtual surgery and video games demand real-time simulations, and while our algorithms can run in real-time for sufficiently simple examples, this was not our aim, and faster results could probably be achieved through additional simplifications which emphasize speed over accuracy.

In addition to working on algorithms for real-time simulation, there are various avenues for future work stemming from the research described in this dissertation. We mention some examples next, including extensions which we have been recently pursuing.

8.1 Articulated Rigid Bodies

One natural extension to our work on rigid bodies is to add joints and articulation constraints to multi-body systems. This was pursued by colleagues in [144], where an impulse-based, generalized coordinate approach to articulated bodies was described. Joint constraints are enforced using a combination of pre- and post-stabilization steps. Post-stabilization occurs after the velocity update, and ensures the new velocity is consistent with the allowable joint motions. Equations similar to those presented for rolling and spinning friction (e.g. Equation (2.8)) are used to determine the joint impulses necessary to enforce constraint-satisfying velocities. Pre-stabilization occurs before the position update, and ensures the updated configuration of the rigid bodies will exactly satisfy the joint constraints. More recently, in [143], I helped augment these articulated rigid bodies with joint and muscle control, solving for the actuation impulses needed to smoothly follow desired joint trajectories. Future work along these lines would include improving the muscle model by including the underlying activation dynamics (as in [137]), and modeling volumetric muscles (as in [135]) rather than line-segment muscles. It would also be interesting to couple an actuated, articulated body, to a fluid, with the aim of eventually simulating a human swimming through water (see [150] for a simplified model).

8.2 Parallel Computation

Fluid simulation often takes hours, and sometimes days, to run. This slow turnaround is undesirable in a visual effects pipeline where simulations often need to be tweaked and re-run many times until a certain look is achieved. One technique that allows for more detailed simulations at lower runtimes is parallel computation. We recently explored this in [63], where we used the message passing interface (MPI) to parallelize a hybrid 2D/3D approach to water simulation. The adaptive grid structure used in that work helped reduce the communication load between processors, but MPI helps speed up even uniform grid simulations, as shown by colleagues in [84]. Of course, other than just speeding up simulations, parallelizing the fluid code opens the door to grid resolutions that would not normally fit in a single machine's memory. Future work could include improving the grid partitioning for better load balancing, and also parallelizing our solids code to enjoy similar speedups for large scale solids simulation.

8.3 The MAC Discretization, Revisited

The fluid simulator presented in this dissertation made use of a novel mixed node/face technique which performed advection and interpolation on nodes (Section 4.4.1). This was needed primarily to facilitate interpolation on octrees, and was done in a way that avoids highly dissipative back-and-forth averaging. However, this technique still exhibits slightly more dissipation than the regular staggered (MAC) scheme, and in addition requires extra care when determining face and cell-centered values from values stored on the nodes. For example, a cell-centered ϕ value is required in order to determine whether a cell contains air or water, and maintaining ϕ on the nodes makes it harder to consistently determine the corresponding cell value, especially in the presence of immersed solids. Storing ϕ in the cell center is preferred for this reason. Recently, we have explored going back to using a MAC discretization, leveraging the fact that the octree grids are fully refined near the water-air and solid-fluid interfaces. In these maximally refined regions, interpolation on the octree behaves precisely as for uniform grids, so there is no problem performing MAC interpolation. It is only away from these regions that more complex, and potentially less accurate, octree interpolation strategies need to be used to handle non-nodal data. Adapting the coupling algorithms described in this dissertation to a MAC discretization requires changing some details, such as the location of visibility rays for one-sided interpolation and advection, but does not otherwise change our main framework or contributions.

8.4 Sharp Treatment for Volumetric Solids

As discussed in Chapter 7, boundary conditions at the interface between a fluid and a volumetric solid can be implemented by filling nodes inside the solid with ghost fluid values. This is in contrast to thin solids which use one-sided stencils computed by intersecting rays against the solid, and using replacement ghost values for occluded nodes. Although computationally more expensive, the one-sided approach does have some advantages over filling ghost values. First, it is sharp interface approach that resolves surface details of the solid independent of grid resolution. Second, clipping semi-Lagrangian rays against the solid is more consistent with the fact that the advecting fluid does not originate inside the solid. This obviates the need for special "separation conditions" to prevent inappropriately pulling air or water out of the solid (Section 7.3). Finally, using the same treatment for both thin and volumetric solids is attractive from a software engineering perspective. Since using the sharp approach exclusively might be undesirably slow, it could be beneficial to allow both approaches to be used interchangeably for different parts of the algorithm. This would essentially give explicit control over the tradeoff between the slower, geometrically accurate approach, and the faster, diffuse approach. It should be noted that computing replacement ghost values on the fly rather than in advance also makes it easier to incorporate jump conditions in multiphase simulations, and this was the approach taken in [84].

8.5 Improving Accuracy of Rasterized Density

Another item for future work is investigating more accurate techniques for computing the rasterized thin shell density used in the variable density Poisson equation (Section 6.5.1). Currently, the density is set to m/V on any face whose dual edge crosses the solid. However, m is calculated by multiplying the area of the face by the solid's density rather than by exactly computing the solid mass inside the control volume corresponding to that face. That is, we assume the amount of solid surface area enclosed by the control volume equals the surface area of the face. This is not the case in general, and we would like to investigate methods for improving the accuracy of this computation.

8.6 Validation Studies

In addition to computer graphics, the techniques described in this dissertation could be used for a variety of scientific applications as well. Here we suggest a number of validation studies that could be explored to get a better sense of the physical accuracy of our algorithms. This would help the scientific community evaluate our simulation results.

The rigid body simulation technique described in Chapter 2 was validated against an analytic solution of a block sliding down an inclined plane with friction (Figure 2.8). One possible experiment that can at least qualitatively test a larger scale simulation would be to try to reproduce the behavior of granular material. As in [14], each rigid body can be composed of a few rigidly connected spheres, and a cylindrical "silo" can be filled with these "grains". The pressure exerted on the silo should reach some maximum value independent of height. If this physical phenomenon is faithfully reproduced then it would give us additional confidence in our simulation results.

There are many interesting experiments that could be used to test our solid-fluid coupling algorithms. [154, 33] simulated and performed experiments with a flapping flexible filament in soap film, and observed interesting results relating the mass and length of the filament to its tendency for sustained flapping. It would be interesting to try to reproduce these results using our thin shells coupling approach.

[102] simulated a soft object by filling a closed membrane with a fluid. They actually used a compressible fluid for the simulation in order to improve the stability of their coupling. It would be interesting to try this with an incompressible fluid, and see whether our weak coupling approach can handle such a tightly coupled example.

One experiment to qualitatively test our volumetric coupling approach would be to drop two heavier spheres into a fluid and see whether they exhibit the common phenomenon of "drafting, kissing, and tumbling" (as in [20]).

Finally, an analytic solution due to Womersley exists for pulsatile *viscous* flow through a rigid or elastic cylinder [148], and this is commonly used as a simple 1D model for cardiovascular simulation [134]. If viscosity is added to our simulator, it would be instructive to try such a simulation and compare against these analytic results.

8.7 Additional Future Work

By rendering removed water particles we were able to add some interesting splash and spray effects to our water (Section 6.7). However, there is definitely room for improvement in shaping, blending, and rendering of these particles. In addition, removed particles currently only undergo simple ballistic motion. In cases where there are many particles, we could use a particle-based fluid approach (such as SPH) to give these ejected particles more realistic dynamics.

For our examples coupling cloth and water, it would be interesting to model the absorption of water by the cloth, changing both its appearance and simulation properties. Also interesting, but not currently modeled, is the adhesion of water to the cloth.

Finally, we would like to apply our volumetric coupling approach to more examples, in particular ones coupling to *deformable* volumetric solids.

Bibliography

[1] G. Baciu and S. K. Wong. The impulse graph: a new dynamic structure for global collisions. *Comput. Graph. Forum (Eurographics Proc.)*, 19(3):229–238, 2000.

[2] J. A. Baerentzen and H. Aanaees. Generating signed distance fields from triangle meshes. Technical Report 21, IMM, Technical University of Denmark, 2002.

[3] Z. Bao. *Fracturing Deformable and Rigid Materials.* PhD thesis, Stanford University, June 2006.

[4] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Comput. Graph. (Proc. SIGGRAPH 89)*, 23(3):223–232, 1989.

[5] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Comput. Graph. (Proc. SIGGRAPH 90)*, 24(4):19–28, 1990.

[6] D. Baraff. Coping with friction for non-penetrating rigid body simulation. *Comput. Graph. (Proc. SIGGRAPH 91)*, 25(4):31–40, 1991.

[7] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.

[8] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. SIGGRAPH 94*, pages 23–34, 1994.

[9] D. Baraff. Interactive simulation of solid rigid bodies. *IEEE Comput. Graph. and Appl.*, 15(3):63–75, 1995.

[10] D. Baraff and A. Witkin. Partitioned dynamics. Technical Report CMU-RI-TR-97-33, Robotics Institute, Carnegie Mellon University, 1997.

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

[12] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):862–870, 2003.

[13] R. Barzel, J. F. Hughes, and D. N. Wood. Plausible motion simulation for computer graphics animation. In *Comput. Anim. and Sim. '96*, Proc. Eurographics Wrkshp., pages 183–197, 1996.

[14] N. Bell, Y. Yu, and P. J. Mucha. Particle-based simulation of granular materials. In *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 77–86, 2005.

[15] D. Benson. Computational methods in Lagrangian and Eulerian hydrocodes. *Comput. Meth. in Appl. Mech. and Eng.*, 99:235–394, 1992.

[16] V. Bhatt and J. Koechling. Three-dimensional frictional rigid-body impact. *ASME J. Appl. Mech.*, 62(4):893–898, 1995.

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

[18] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 28–36, 2003.

[19] S. R. Buss. Accurate and efficient simulation of rigid-body rotations. *J. Comput. Phys.*, 164(2):377–406, 2000.

[20] M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):377–384, 2004.

[21] M. Carlson, P. J. Mucha, R. B. Van Horn, and G. Turk. Melting and flowing. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 167–174, 2002.

[22] A. Chatterjee and A. Ruina. A new algebraic rigid body collision law based on impulse space considerations. *ASME J. Appl. Mech.*, 65(4):939–951, 1998.

[23] J. X. Chen and N. V. Lobo. Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations. *Graphical Models and Image Processing*, 57(2):107–116, 1995.

[24] S. Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proc. SIGGRAPH 2000*, pages 219–228, 2000.

[25] K.-J. Choi and H.-S. Ko. Stable but responsive cloth. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):604–611, 2002.

[26] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comput.*, 22:745–762, 1968.

[27] J. M. Cohen and M. J. Molemaker. Practical simulation of surface tension flows. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[28] R. Courant, E. Isaacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Comm. on Pure and Appl. Math.*, 5:243–255, 1952.

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

[30] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures*, 83:479–490, 2005.

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

[32] D. Enright, D. Nguyen, F. Gibou, and R. Fedkiw. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proc. 4th ASME-JSME Joint Fluids Eng. Conf.*, pages 1–6. ASME, 2003.

[33] D. J. J. Farnell, T. David, and D. C. Barton. Numerical simulations of a filament in a flowing soap film. *Int. J. Num. Meth. Fluids*, 44(3):313–330, 2004.

[34] R. Fattal and D. Lischinski. Target-driven smoke animation. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):441–448, 2004.

[35] R. Fedkiw. Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method. *J. Comput. Phys.*, 175(1):200–224, 2002.

[36] 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(2):457–492, 1999.

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

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

[39] S. Fisher and M. C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Comput. Anim. and Sim. '01*, Proc. Eurographics Wrkshp., pages 99–111, 2001.

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

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

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

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

[44] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. SIGGRAPH 2000*, pages 249–254, 2000.

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

[46] O. Génevaux, A. Habibi, and J.-M. Dischler. Simulating fluid-solid interaction. In *Graphics Interface*, pages 31–38. A K Peters, 2003.

[47] F. Gibou, R. Fedkiw, L.-T. Cheng, and M. Kang. A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *J. Comput. Phys.*, 176(1):205–227, 2002.

[48] S. F. F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proc. IEEE Symp. on Vol. Vis.*, pages 23–30, 1998.

[49] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien. A method for animating viscoelastic fluids. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):463–468, 2004.

[50] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schröder. Discrete shells. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 62–67, 2003.

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

[52] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):973–981, 2005.

[53] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Comput. Graph. Forum (Eurographics Proc.)*, 20(3):329–338, 2001.

[54] J. K. Hahn. Realistic animation of rigid bodies. *Comput. Graph. (Proc. SIGGRAPH 88)*, 22(4):299–308, 1988.

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

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

[57] C. W. Hirt, A. A. Amsden, and J. L. Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *J. Comput. Phys.*, 14(3):227–253, 1974.

[58] C. W. Hirt, J. L. Cook, and T. D. Butler. A Lagrangian method for calculating the dynamics of an incompressible fluid with free surface. *J. Comput. Phys.*, 5(1):103–124, 1970.

[59] J.-M. Hong and C.-H. Kim. Animation of bubbles in liquid. *Comput. Graph. Forum (Eurographics Proc.)*, 22(3):253–262, 2003.

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

[61] B. Houston, C. Bond, and M. Wiebe. A unified approach for modeling complex occlusions in fluid simulations. In *SIGGRAPH 2003 Sketches & Applications*. ACM Press, 2003.

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

[63] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph. (SIGGRAPH Proc.) (in press)*, 2006.

[64] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 131–140, 2004.

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

[66] T. Ju. Robust repair of polygonal models. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):888–895, 2004.

[67] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of Hermite data. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):339–346, 2002.

[68] M. Kang, R. P. Fedkiw, and X.-D. Liu. A boundary condition capturing method for multiphase incompressible flow. *J. Sci. Comput.*, 15(3):323–360, 2000.

[69] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. *Comput. Graph. (Proc. SIGGRAPH 90)*, 24(4):49–57, 1990.

[70] D. M. Kaufman, T. Edmunds, and D. K. Pai. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):946–956, 2005.

[71] T.-Y. Kim and U. Neumann. Interactive multiresolution hair modeling and editing. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21(3):620–629, 2002.

[72] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 23–31, 2002.

[73] E. Kokkevis, D. Metaxas, and N. I. Badler. User-controlled physics-based animation for articulated figures. In *Proc. Comput. Anim. '96*, pages 16–26, 1996.

[74] N. Kondoh, A. Kunimatsu, and S. Sasagawa. Creating animations of fluids and cloth with moving characters. In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[75] M.-C. Lai and C. S. Peskin. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *J. Comput. Phys.*, 160(2):705–719, 2000.

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

[77] A. D. Lewis and R. M. Murray. Variational principles in constrained systems: Theory and experiment. *Int. J. Nonlinear Mech.*, 30(6):793–815, 1995.

[78] Z. Li and M.-C. Lai. The immersed interface method for the Navier-Stokes equations with singular forces. *J. Comput. Phys.*, 171(2):822–842, 2001.

[79] M. C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proc. of IMA Conf. on Math. of Surfaces*, pages 37–56, 1998.

[80] L. Ling, M. Damodaran, and R. K. L. Gay. Aerodynamic force models for animating cloth motion in air flow. *The Vis. Comput.*, 12(2):84–104, 1996.

[81] W. E. Lorensen and H. E. Cline. Marching cubes: A high-resolution 3D surface construction algorithm. *Comput. Graph. (Proc. SIGGRAPH 87)*, 21(4):163–169, 1987.

[82] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):457–462, 2004.

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

[84] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw. Multiple interacting liquids. *ACM Trans. Graph. (SIGGRAPH Proc.) (in press)*, 2006.

[85] M. Mason and Y. Wang. On the inconsistency of rigid-body frictional planar mechanics. In *IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 524–528, April 1988.

[86] S. Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations.* PhD thesis, California Institute of Technology, April 2003.

[87] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23(3):449–456, 2004.

[88] V. Mihalef, D. Metaxas, and M. Sussman. Animation and control of breaking waves. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 315–324, 2004.

[89] V. J. Milenkovic. Position-based physics: Simulating the motion of many highly interacting spheres and polyhedra. In *Proc. SIGGRAPH 96*, pages 129–136, 1996.

[90] V. J. Milenkovic and H. Schmidl. Optimization-based animation. In *Proc. SIGGRAPH 2001*, pages 37–46, 2001.

[91] B. Mirtich. Fast and accurate computation of polyhedral mass properties. *J. Graph. Tools*, 1(2):31–50, 1996.

[92] B. Mirtich. Timewarp rigid body simulation. In *Proc. SIGGRAPH 2000*, pages 193–200, 2000.

[93] B. Mirtich and J. Canny. Impulse-based dynamic simulation. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Alg. Found. of Robotics*, pages 407–418. A. K. Peters, Boston, MA, 1995.

[94] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proc. of 1995 Symp. on Int. 3D Graph.*, pages 181–188, 217, 1995.

[95] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.

[96] 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.

[97] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Comput. Graph. (Proc. SIGGRAPH 88)*, 22(4):289–298, 1988.

[98] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 154–159, 2003.

[99] 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, 2004.

[100] A. Nealan, M. Müller, R. Keiser, E. Boxermann, and M. Carlson. Physically based deformable models in computer graphics. In Y. Chrysanthou and M. Magnor, editors, *STAR Proc. of Eurographics 2005*, pages 71–94, Sept. 2005.

[101] D. Nguyen, R. Fedkiw, and H. Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 29(3):721–728, 2002.

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

[103] W. Noh. *CEL: A time-dependent, two-space-dimensional, coupled Eulerian-Lagrange code*, pages 117–179. Academic Press, New York, 1964.

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

[105] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer-Verlag, 2002. New York, NY.

[106] A. Pandolfi, C. Kane, J. E. Marsden, and M. Ortiz. Time-discretized variational formulation of non-smooth frictional contact. *Int. J. Num. Meth. Eng.*, 53:1801–1829, 2002.

[107] M. Pauly, D. K. Pai, and L. J. Guibas. Quasi-rigid objects in contact. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 109–119, 2004.

[108] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. *Comput. Graph. (Proc. SIGGRAPH 89)*, 23(3):215–222, 1989.

[109] C. S. Peskin. Flow patterns around heart valves: A numerical method. *J. Comput. Phys.*, 10:252–271, 1972.

[110] C. S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.

[111] R. Peyret and T. D. Taylor. *Computational methods for fluid flow.* Springer-Verlag, 1983. New York.

[112] M. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between solid models. In *Proc. ACM Symp. Solid Model. and Appl.*, pages 293–304, 1995.

[113] J. Popović, S. M. Seitz, M. Erdmann, Z. Popović, and A. Witkin. Interactive manipulation of rigid body simulations. *Proc. SIGGRAPH 2000*, pages 209–217, 2000.

[114] S. Premoze, T. Tasdizen, J. Bigler, A. Lefohn, and R. T. Whitaker. Particle-based simulation of fluids. *Comput. Graph. Forum (Eurographics Proc.)*, 22(3):401–410, 2003.

[115] R. Radovitzky and M. Ortiz. Lagrangian finite element analysis of Newtonian fluid flows. *Int. J. Num. Meth. Eng.*, 43(4):607–619, 1998.

[116] 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.

[117] N. Rasmussen, D. Q. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):703–707, 2003.

[118] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Comput. Graph. Forum (Eurographics Proc.)*, 21(3):279–288, 2002.

[119] J. Sauer and E. Schömer. A constraint-based approach to rigid body dynamics for virtual reality applications. In *Proc. ACM Symp. on Virt. Reality Soft. and Tech.*, pages 153–162, 1998.

[120] H. Schmidl. *Optimization-based animation.* PhD thesis, University of Miami, May 2002.

[121] W. J. Schroeder, W. E. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. In *Proc. of Vis.*, pages 40–55. IEEE Comput. Society Press, 1994.

[122] S. Sclaroff and A. Pentland. Generalized implicit functions for computer graphics. *Comput. Graph. (Proc. SIGGRAPH 91)*, 25(4):247–250, 1991.

[123] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):910–914, 2005.

[124] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci.*, 93(4):1591–1595, 1996.

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

[126] K. Sims. Evolving virtual creatures. In *Proc. SIGGRAPH 94*, pages 15–22, 1994.

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

[128] J. Stam. Flows on surfaces of arbitrary topology. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):724–731, 2003.

[129] D. E. Stewart. Rigid-body dynamics with friction and impact. *SIAM Review*, 42(1):3–39, 2000.

[130] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with Coulomb friction. In *IEEE Int. Conf. on Robotics and Automation*, pages 162–169, 2000.

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

[132] T. Takahashi, H. Ueki, A. Kunimatsu, and H. Fujii. The simulation of fluid-rigid body interaction. In *SIGGRAPH 2002 Sketches & Applications*. ACM Press, 2002.

[133] D. Tam, R. Radovitzky, and R. Samtaney. An algorithm for modeling the interaction of a flexible rod with a two-dimensional high-speed flow. *Int. J. Num. Meth. Eng. (in press)*, 2005.

[134] C. A. Taylor. *Blood Flow*, volume 3 of *Encyclopedia of Computational Mechanics*, chapter 16, pages 527–543. John Wiley & Sons, 2004.

[135] J. Teran, S. Blemker, V. Ng-Thow-Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 68–74, 2003.

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

[137] D. G. Thelen, F. C. Anderson, and S. L. Delp. Generating dynamic simulations of movement using computed muscle control. *J. Biomech.*, 36(3):321–328, 2003.

[138] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulations. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):716–723, 2003.

[139] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control*, 40:1528–1538, 1995.

[140] H. S. Udaykumar, R. Mittal, P. Rampunggoon, and A. Khanna. A sharp interface Cartesian grid method for simulating flows with complex moving boundaries. *J. Comput. Phys.*, 174(1):345–380, 2001.

[141] R. Webb and M. Gigante. Using dynamic bounding volume hierarchies to improve efficiency of rigid body simulations. In *CGI Proc. 1992*, pages 825–841, 1992.

[142] X. Wei, Y. Zhao, Z. Fan, W. Li, S. Yoakum-Stover, and A. Kaufman. Blowing in the wind. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 75–85, 2003.

[143] R. Weinstein, E. Guendelman, and R. Fedkiw. Impulse-based PD control for joints and muscles. In *SIGGRAPH 2006 Sketches & Applications (to appear)*. ACM Press, 2006.

[144] 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.

[145] J. Wejchert and D. Haumann. Animation aerodynamics. *Comput. Graph. (Proc. SIGGRAPH 91)*, 25(4):19–22, 1991.

[146] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Vis. Comput.*, 15(2):100–111, 1999.

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

[148] J. R. Womersley. Oscillatory motion of a viscous liquid in a thin-walled elastic tube – I: The linear approximation for long waves. *Philos. Mag.*, 46:199–221, 1955.

[149] T. Yabe, K. Takizawa, F. Xiao, and A. Ikebata. Universal solver CIP for all phases of matter. In *Int. Conf. on Sci. Comput. and Partial Differential Equations On the Occasion of Stanley Osher's 60th birthday*, 2002.

[150] P.-F. Yang, J. Laszlo, and K. Singh. Layered dynamic control for interactive character swimming. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 39–47, 2004.

[151] G. D. Yngve, J. F. O'Brien, and J. K. Hodgins. Animating explosions. In *Proc. SIGGRAPH 2000*, pages 29–36, 2000.

[152] Z. Yu. A DLM/FD method for fluid/flexible-body interactions. *J. Comput. Phys.*, 207(1):1–27, 2005.

[153] Q. Zhang and T. Hisada. Studies of the strong coupling and weak coupling methods in FSI analysis. *Int. J. Num. Meth. Eng.*, 60(12):2013–2029, 2004.

[154] L. Zhu and C. S. Peskin. Simulation of a flapping flexible filament in a flowing soap film by the immersed boundary method. *J. Comput. Phys.*, 179(2):452–468, 2002.

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