COMPUTATIONAL ASPECTS OF DYNAMIC SURFACES

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

Robert Edward Bridson
July 2003

Abstract

This thesis is concerned with various computational aspects of surface dynamics. By this I mean algorithms and data structures to support simulations of the physics taking place on or about two-dimensional manifolds. This includes internal dynamics (e.g. elastic forces) and external dynamics (e.g. collisions).

The core of the thesis is simulating cloth motion, including the internal elastic dynamics (chapter 1), the external dynamics of contact and collision (chapter 2), and post-processing of the data for rendering (chapter 3). The contact and collision algorithm, based on the notion of a collision resolution pipeline where candidate trajectories are passed through several stages designed to resolve different classes of collisions, was subsequently adapted to rigid body simulation (chapter 4).

Surface dynamics are also potentially the most important part of simulations of deformable objects, driving the deformation of the interior. Chapter 5 presents a tetrahedral mesh generation algorithm designed to support large surface deformations. Adaptivity is also discussed.

Throughout the thesis implicit surfaces defined by level sets are used to represent geometry. Level sets are used in many other areas, especially computational fluid dynamics for simulations of the surface between distinct regions of flow (e.g. material boundaries). Their chief drawback is storage expense. Chapter 6 presents a data structure for efficiently storing level sets without needing to change algorithms designed for regular grids. An example application, an interactive sculpting tool, is presented to illustrate the structure.

Acknowledgements

I would first and foremost like to thank my wife Rowena and my son Jonathan for their tremendous love, encouragement, and help throughout my degree. My parents John and Mary and my sister Jessica gave me wonderful support and occasional baby-sitting, and I also thank my father for crystallographic insights and help deciding on the BCC lattice of chapter 5. I also would like to thank my in-laws Amira and Harvey for more support and help with Jonathan.

My advisor Ronald Fedkiw was brilliant as a teacher, mentor, and research collaborator, and provided incredible support (and thanks for the trip to Kauai to finish writing our first cloth paper!)

The Stanford Graduate Fellowship program paid for most of my time at Stanford.

I owe a lot to my coauthors: Ronald Fedkiw, Eran Guendelman, Neil Molino, Joseph Teran, Sebastian Marino, and John Anderson. Much of the material in this thesis comes from the following papers with them (though parts of the papers with which I wasn't directly involved have been removed from this thesis):

- R. Bridson, R. Fedkiw, and J. Anderson, *Robust Treatment of Collisions, Contact and Friction for Cloth Animation* [27]
- R. Bridson, S. Marino, and R. Fedkiw, *Simulation of clothing with folds and wrinkles* [28]
- E. Guendelman, R. Bridson, and R. Fedkiw, *Simulation of Non-Convex Rigid Bodies with Stacking* [76]
- N. Molino, R. Bridson, J. Teran and R. Fedkiw, *A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra* [119]

I also want to thank my reading committee, Ron, Leonidas Guibas, and Marc Levoy, for making this process very smooth, and also Bernard Roth and Margot Gerritsen who joined them on my defense committee.

Wei-Pai Tang, my advisor for my masters thesis, and the rest of the SciCom group at Waterloo really gave me a great start in the field. Esmond Ng at LBL also was great to me in that first summer in California, and I hope to catch up soon with all the ideas we generated back then.

The code that was used in this thesis is mostly integrated with the PhysBAM physics-based modeling library, which is currently co-developed by Zhaosheng Bao, myself, Doug Enright, Ron Fedkiw, Frederic Gibou, Eran Guendelman, Sergey Koltakov, Frank Losasso, Sebastian Marino, Ian Mitchell, Neil Molino, Igor Neverov, Duc Nguyen, Nick Rasmussen, Eftychios Sifakis, Joseph Teran, and Rachel Weinstein (in alphabetical order). Some of the rendering was performed using Henrik Wann Jensen's *dali* ray-tracer, and some with a customized version of *lrt* by Matt Pharr, Greg Humphreys and Pat Hanrahan.

The dragon and Buddha models were originally from the Stanford Computer Graphics Laboratory, but were then processed by Hong-Kai Zhao, Ron Fedkiw and Stanley Osher in [190] to produce the level sets used in this thesis.

My office mates, Doug, Ian, Eran, and Rachel, and other friends, Neil, Joey, Igor, the rest of Ron's group, Sep, and Maureen made grad student life a lot of fun. Outside of school, the 58–62 courtyard and other friends in Escondido Village have been a great community.

And to all the other people I haven't mentioned but probably should have—thanks too!

Contents

Abstract **v**

Acknowledgements **vii**

1 Internal Cloth Dynamics **1**

1.1 Previous Work 1
1.2 A Mass-Spring Model 3
1.3 Strain Limiting 5
1.4 Bending Elements 7
1.5 Time Integration 13

2 Contact and Collision for Cloth **15**

2.1 Collision with Objects 17
2.2 A Collision Resolution Pipeline 19
2.3 Time Integration 21
2.4 Proximity and Colliding Trajectory Detection 23
2.5 Interpolating Within the Cloth 24
2.6 Repulsion Forces 25
2.7 Friction 27
2.8 Geometric Collisions 29
2.9 Rigid Impact Zones 30

3 Post-Processing Cloth **33**

3.1 Related Work 33
3.2 Collision-Aware Subdivision 34

4 Contact and Collision for Rigid Bodies **37**

4.1 Introduction 38
4.2 Previous Work 39

4.3 Geometric Representation 40
4.4 Interference Detection 41
4.5 Time Integration 42
4.6 Collisions 45
4.7 Static and Kinetic Friction 48
4.8 Contact 48
4.8.1 Contact graph 50
4.8.2 Shock propagation 51
4.9 Rolling and Spinning Friction 53
4.10 Results 54
4.11 Conclusions and Future Work 54

5 Mesh Generation **57**
5.1 Related Work 58
5.2 The BCC Background Mesh 61
5.3 Red-Green Refinement 62
5.4 Selecting a Topologically Robust Sub-Mesh 65
5.5 Compressing the Surface Mesh to the Level Set 67
5.6 Results 69
5.7 Generalizations 71

6 A Sparse Level Set Structure **75**
6.1 Sparse Block Grids 76
6.2 Example Application: Sculpting 78

Bibliography **82**

List of Tables

5.1 The vectors used for pattern search in optimizing the internal nodes of the mesh. . . 69
5.2 Quality measures for the example meshes. The aspect ratio of a tetrahedron is defined as the longest edge over the shortest altitude. The max/min edge length ratio indicates the degree of adaptivity. 71

List of Figures

1.1 A common model for cloth internal dynamics is a rectangular grid of springs. 4

1.2 A bending element with dihedral angle $\pi - \theta$. 8

1.3 Nonzero rest angles allow pre-sculpted folds, as in this “cloth” Buddha collapsing under its own weight. 10

1.4 The “cloth” Buddha with significantly stronger bending springs that dominate all other forces, illustrating that our formulation is well behaved in the stiff limit, robust across a wide range of materials. 12

2.1 Initially, a curtain is draped over a ball on the ground. The ball moves, flipping the curtain over on top of itself producing stable folds and wrinkles using our static friction model. Another ball enters the scene and pushes through the complicated structure of folds before slipping underneath unraveling the curtain. 16

2.2 A tablecloth draped over table legs with no tabletop is dragged to the ground by a descending sphere. 20

2.3 The friction between a rotating sphere and a piece of cloth draped over it creates a complex structure of wrinkles and folds. 28

4.1 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. 38

4.2 Some nonconvex geometry from our simulations: the cranium, pelvis and femur have 3520, 8680 and 8160 triangles respectively. 41

4.3 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). 45

4.4 Theoretical and our numerical results for two tests of a block sliding down an inclined plane with friction. The curves lie on top of each other in the figures due to the accuracy of our new time sequencing algorithm. 46

4.5 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. 48

4.6 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. 51
4.7 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. 51
4.8 500 nonconvex bones falling into a pile after passing through a funnel. Exact triangle counts are given in figure 4.2. 53
4.9 The complexity of our nonconvex objects is emphasized by exposing their underlying tessellation. 55

5.1 Tetrahedral mesh of a cranium (100K elements) with cutaway views showing element regularity and gradation to very coarse tetrahedra in the interior. 59
5.2 A portion of the BCC lattice with a tetrahedron highlighted. The two black cubic grids are interlaced together with the blue edges. 62
5.3 The standard red refinement (depicted in red, far left) produces eight children that reside on a BCC lattice one half the size. Three classes of green refinement are allowed (depicted in green). 62
5.4 Tetrahedral mesh of a sphere (18K elements). The cutaway view illustrates that the interior mesh can be fairly coarse even if high resolution is desired on the boundary. 64
5.5 Tetrahedral mesh of a torus (8.5K elements). The principal curvatures were used to increase the level of resolution in the inner ring. 64
5.6 The initial candidate mesh for the cranium (a subset of the tetrahedra of the refined background mesh) before the compression phase deforms the surface to match the level set. 67
5.7 Tetrahedral mesh of a model dragon (500K elements). The final compression stage can be carried out with masses and springs, finite elements, or an optimization based method. 70
5.8 Tetrahedral mesh of a model Buddha (900K elements), adaptively refined to resolve features on multiple scales. 72
5.9 Our method is easily extended to create a triangle mesh of a manifold with boundary. The peeled away regions of the dragon were modeled with a second level set. 73

6.1 A screenshot from the interactive level set sculpting application (at 240^3 resolution). 79
6.2 A sculpted sparse level set, composited into a real photograph. 81

Chapter 1

Internal Cloth Dynamics

The simulation of cloth in a visually plausible and attractive manner is becoming an essential part of the special effects industry, and has spurred much research effort in the computer graphics community. This chapter discusses the modeling of the internal dynamics—that is the stretching, shearing, and bending as opposed to collision and contact interactions—of fabric and related materials.

Cloth has a fairly complex structure, spanning several length scales from the nanostructure of polymers to the microstructure of fibers to the mesostructure of the weaving pattern, but for the purposes of animation it is typically adequate to abstract this down to a homogeneous continuum model. Indeed, one of the challenges on the near horizon will be to tackle simulations of the small scale structures in cloth; I believe the collision resolution algorithms presented in later chapters may play a useful role in this endeavor.

1.1 Previous Work

Essentially all of the literature to date deals with cloth as a two-dimensional continuum. Though some works such as [26] claim a "particle" representation that differs from the standard notion of continua, they are more naturally seen simply as a particular discretization of continua.

In the textile mechanics community, finite element methods were used with parameters from measured data in an attempt to predict the static drape of cloth in simple situations[105, 40, 62, 36, 6, 55]. Plate or shell elements are the preferred type to use to handle bending accurately. Several works found some agreement with experiments, but only for simple quantities like the area of the projected shadow of a tablecloth, or straightforward bending tests. It remains unclear whether the considerable overhead of such complex methods is justified for computer graphics; simpler and faster approaches allow much finer resolution meshes that can resolve more details in more realistic, interesting situations.

In graphics, among the first works is [183]. A technique for modeling static drape was presented there that heuristically combined physically-motivated procedures to produce elegant images. Later work that also strove directly toward the ultimate shape of the cloth not necessarily by physical simulation was [98], which turned to singularity theory to analyze folds and wrinkles in cloth.

Terzopoulos et al.[166] introduced rigorous simulation of elastic deformation to the computer graphics community, discussing surfaces as well as volumes and curves, with finite difference discretization and implicit time integration. The next year [165, 164] extended the models to include viscoelasticity and plasticity along with a simple model of fracture. Later [4] extended this work, using a finite difference discretization of two-dimensional elasticity that handled inhomogeneity and advanced viscoelasticity models. Aerodynamic forces (generated by a vortex method not tied to a background grid) were added to the Terzopoulos model in [103, 104].

As an alternative model in graphics, [176] developed a force model for unstructured triangle meshes based on continuum mechanics, essentially approximating a finite volume calculation for the in-plane forces and basing the bending forces on a discrete approximation of curvature.

A different approach was to use a grid of particles connected by nonlinear springs with carefully modeled force/strain curves. Breen et al.[26] matched the curves to measured data, then calculated equilibrium positions of cloth draped over objects. This was accelerated by a multigrid algorithm in [127]. The work of [54] elaborated the model to calculate full dynamics for animation, not just static poses.

Even simpler was the use of simple linear springs in mass-spring networks, with various improvements. For example, [139] introduced a strain limiting procedure to improve the quality and speed of explicit time integration. To simplify the involved calculations of their earlier model[176], Volino et al. attempted to include more realistic resistance to area change (i.e. a better apparent Poisson's ratio) in a simple mass-spring model[178]. Van Gelder[174] determined a method for selecting spring stiffnesses in mass-spring networks to better approximate elastic continua. In recent work, [71] generalized linear edge springs to triangle springs and bending springs by way of proposed energy functions.

The early work of Terzopoulos used implicit time integration with a simple direct matrix solver. This is not efficient for larger models, so authors turned to explicit time integration scheme such as symplectic Euler (often confusing it with forward Euler), Runge-Kutta methods, etc. Unfortunately, as we will show later, these fully explicit methods are also very inefficient when damping forces are considered, even with strain limiting as in [139]. Baraff and Witkin returned to implicit methods in [17], demonstrating that a conjugate gradient iterative solver could be very efficient for large cloth models. Their force model is more sophisticated than masses and springs, based on a continuum description of potential energies. More recently, [5] improved the theoretical basis (and speed of convergence) of their implicit solver. The chief problem with Baraff and Witkin's work, however, was that small-scale features such as wrinkles are artificially dissipated by the fully implicit time integration; Choi and Ko[39] carefully adjusted the simple bending model of papers such as [139] to mitigate this, based on a heuristic buckling analysis. Parks and Forsyth[133] explored the use of α-methods from computational mechanics to control the artificial dissipation in time integration without changing the forces.

Speed was the overriding concern in [51], which made approximations to the implicit integration of [17] to make the matrix that needs to be inverted constant from one time step to the next, precomputed its inverse, and added ideas from [139] to a mass-spring model to achieve interactive cloth animation. Kang et al.[91] further simplified and accelerated this with a simple sparse approximation to the precomputed matrix inverse. To add back detail missing from the coarse meshes used in this work, [92] added sinusoidal wrinkles to a cubic spline interpolation of the mass-points whenever springs were compressed. [179] tried to take a middle ground in efficiency/accuracy between [17] and [51]. This was brought into a later study[180] that compared several implicit and explicit methods in simple test cases of cloth motion.

Adaptivity has been considered in several papers. Thingvold and Cohen[168] modeled cloth forces with springs and hinges between the control points of a B-spline surface, with refinement (of the tensor-product mesh) possible during a simulation. Adaptive refinement was later advanced in [87] who established how to do it for a rectangular mass-spring network without being restricted to tensor-product meshes. More recently, [72] demonstrated an easy yet rigorous approach to adaptive finite element simulations, including shell models for surfaces such as aluminum cans and pillows.

Several useful surveys of cloth in computer graphics have been published, beginning with [109] in 1991. In 1996, [126] gave a fairly comprehensive survey of methods, summarizing and categorizing seventeen earlier techniques. The book by House and Breen[84] collects several important articles on cloth animation and serves as a very useful overview of cloth, touching on other aspects such as control and rendering that are not discussed here.

1.2 A Mass-Spring Model

In this section I will discuss an approach to modeling the stretch and shear forces that resist in-plane deformation of a surface.

Although clearly more sophisticated methods, such as the finite element models discussed in [105, 40, 62, 36, 6, 55, 72], are necessary for some scientific applications it appears that fairly simple mass-spring models are adequate for visual plausibility. In addition to speed their simplicity also allows easy and intuitive modifications, as in [139] upon which I will expand.

The simplest mass-spring model is based on a rectangular grid of equal mass nodes[1]. “Stretch” springs connect adjacent nodes along the grid lines, and “shear” springs connect adjacent nodes along the diagonals, as illustrated in figure 1.1. All in-plane deformations are resisted to some

[1]It is arguable that nodes on the edges should have less mass as they represent less material, but in practice this doesn’t seem to add any extra realism to simulation. Given that most cloth is sewn with a hem around the edges, it is reasonable to have slightly heavier edges anyhow.

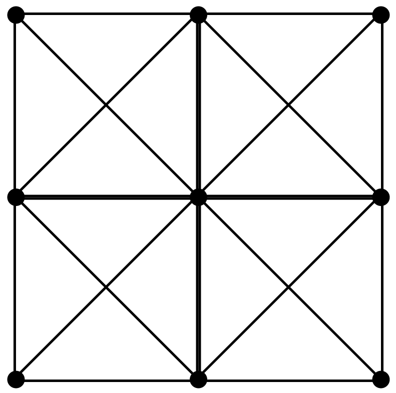

Figure 1.1: A common model for cloth internal dynamics is a rectangular grid of springs.

extent by this model, which has been used in works such as [139, 39]. Claims are made in [26] that (assuming the grid is aligned with the weave/weft directions of the cloth) this allows easy modeling of the actual anisotropy of woven cloth. However, it is not intuitive how to pick stiffness parameters for the springs since they are not orthogonal.

This grid is not enough: a triangle mesh is required for our computation of bending forces (though a simpler technique just adds additional springs between cloth nodes separated by two edges in the grid) and for collision handling. The standard approach is to split each square of the grid with the same diagonal. However, this does introduce bias into the mesh; a better approach is to alternate the diagonal used, leading to a "herring bone" pattern without an obvious bias (though each interior cloth node still has degree equal to six).

While the grid-based approach is by far the simplest for applications where a rectangular piece of cloth is desired (e.g. flags, curtains, sheets, some simple garments, etc.), it is not clear how to extend it to more complicated geometry. Unstructured meshes are needed in this case. Again, finite element or finite volume methods are an obvious choice, but mass-spring methods can be devised as well.

Starting with a triangle mesh, one can put a spring on each mesh edge. The two issues are how to assign masses to the nodes and stiffnesses to the springs.

The standard method for distributing the mass of the cloth to the nodes of its unstructured mesh is to compute the mass of each triangle by multiplying the cloth planar density with the triangle area, then distribute the mass of each triangle to its three corner nodes. The simplest approach is just to give each node a third of the mass of each incident triangle. A more complicated alternative is to form the Voronoi cell of each node in the mesh (clipped by the boundary) and integrate the density over these to get the nodal masses. For some regular geometries the two result in exactly the same answers, and in general, for well-shaped triangles (as one would want for numerical simulation), the two do not differ by much. Given the simple nature of the rest of the model, there is no pressing reason to prefer the more complex Voronoi calculation.

The question of reasonable choices for the spring stiffnesses, given some Young's modulus E that one wants to imbue the cloth with, was tackled by [174]. In that work, the spring stiffness was scaled by the sum of the areas of the incident triangles divided by the length of the edge[2]. In [119] we provide a similar, but much shorter argument for scaling of this nature. The frequency of a spring scales as $\sqrt{k/ml_o}$ so the sound speed scales as $l_o\sqrt{k/ml_o} = \sqrt{kl_o/m}$. We simply require the sound speed to be a mesh-independent material property as it should be, implying that k must scale as m/l_o. This result is consistent with [174], since the mass and the area both scale identically with their ratio equal to the density.

Lying somewhere between finite element models rigorously based in continuum mechanics and heuristic mass-spring models are the work in [17] and [71] where forces are derived from variational derivatives of discrete geometric "energies" such as the sum of squared differences of triangle areas from their rest-state areas. I do not pursue this approach here, but note that it may be useful to derive connections between mass-spring models and finite elements going through this geometric route, for better understanding of the discrete models.

1.3 Strain Limiting

One of the interesting behaviors of cloth is its non-linear response to tension and compression. Roughly speaking, the resistance to stretching is fairly weak in the initial phase of extension where threads in the cloth are merely flattening out, but the resistance climbs significantly as the threads themselves are extended. This can be modeled directly in the springs as in [26]. An alternative approach has its origins in [139].

Provot's algorithm in [139] was motivated primarily by efficiency. An explicit time integrator is limited to step sizes that are proportional to one over the square root of the material stiffness. For fast simulation times, it is thus desirable to decrease the stiffness. Of course, this results in cloth that looks too stretchy. To correct for this Provot modified the mesh after each time step, looping through the segments looking for springs that had deformed by more than 10% from their rest lengths, and moving the endpoints inwards (an equal amount in the absence of constraints) to reduce the strain[3] to that 10% limit. This iterative procedure was empirically observed to converge quickly, resulting in visually pleasing cloth animation.

In [27, 28] we elaborated on this idea. The first change is to make the adjustments to the velocities of mesh nodes instead of just altering positions. After calculating new velocities for the mesh based on the standard spring model accelerations, we loop through the springs looking for those

[2]The actual formulas in [174] actually have an edge length squared; we incorporate that extra edge length directly into the force calculation, $F = k\frac{l-l_o}{l_o}$, so that we can deal with the dimensionless strain $(l - l_o)/l_o$ of the spring.

[3]Note that the paper mistakenly refers to "deformation rate" when deformation, or more precisely, strain, is implied.

whose predicted strain at the end of the time step is too high. We apply a momentum-conserving impulse to the endpoints of each such spring, altering the velocities so that the predicted strain will be in the valid range (exactly clamping it in fact). This is done sequentially, i.e. the velocities are updated immediately after each impulse and thus can affect the impulses that may be applied to other incident springs. Similar results to [139] are obtained but with conservation of momentum and a set of velocities that is consistent with the position updates. It was also observed that this procedure can be interpreted loosely as a Gauss-Seidel iteration for solving the nonlinear equations of an implicit time integrator using a more realistic biphasic model of cloth forces. That is, it can be expected (and is observed in practice) that *more* realistic cloth motion is achieved with this process than by simply increasing the stiffness of the springs, because the real fabric only gently resists small amounts of stretching (when the threads are just straightening out). Small scale wrinkling and other subtle details become possible with biphasic behavior, though they would be strongly resisted in a stiff linear model.

The other significant change in our papers was to allow for different strain limits in extension and compression. In particular, for a generic cloth model we restrict extension to the usual 10% but restrict compression to 0%, disallowing it completely. This at first seems counter-intuitive, since cloth doesn't have a strong resistance to compression as it does to extension beyond a limit. In fact, what we are modeling is that the *ratio* between the resistance to in-plane compression and out-of-plane buckling is high: if one attempts to compress cloth, it invariably buckles into folds and wrinkles instead of compressing. By applying impulses to stop the in-plane compression, we introduce more bending out-of-plane, forcing any slight deviation from planarity (even at the level of round-off error) into a buckling fold. This improves the realism of cloth simulation significantly, allowing the correct qualitative behavior to be captured even on very coarse grids. (A lengthy discussion of cloth buckling may be found in [39], where bending resistance during compression was reduced to mitigate the artificial damping of small scale features present in standard implicit integration methods for cloth. This achieved a similar ratio between in-plane compression resistance and out-of-plane bending resistance, though the authors did not realize this was the fundamental issue.)

Another extension to the strain limiting idea was to also limit strain rate. A rule of thumb in computational mechanics is to avoid deformations of more than 10% during a single time step, see e.g. [31], and also [17] for an example in computer graphics cloth simulation. The natural way of doing this is to monitor the strain rate, and adaptively reduce the size of the time step to ensure this limit. Thus if a deformation is predicted to be happening too fast to be accurately resolved at the current time step size, the simulation slows down and makes sure it is resolved.

However there are situations with high strain rates where accuracy may already have been lost and it is pointless to be restricted to punishingly small time steps after the fact. For example, high velocity collisions arise in computer graphics that to accurately model would require far smaller time steps and far finer meshes than are practical or necessary for visual purposes. Using appropriate large time steps instead introduces sharp discontinuities in the velocity field of the mesh, resulting in high strain rates. Other pragmatic examples include problems due to hand-animated characters moving with non-physically high acceleration (and dragging physically simulated cloth with them), and of course bugs in the software. For these situations it is still imperative to do something to reduce the amount of strain per time step.

An attractive technique we proposed in [27] is to iteratively apply damping impulses that reduce the strain rate (or specifically the strain per time step) of springs to a reasonable limit, exactly the same way as we apply impulses to reduce strain. This stably and efficiently damps out problematic areas of the cloth in these situations, eliminating bad oscillations, containing the problem (so it doesn't adversely affect other parts of the cloth through internal dynamics or collisions), and allowing continued large time steps. The added robustness may be a big gain in production environments.

Finally, an interesting generalization of the two procedures is naturally suggested by using the words strain and strain rate. Instead of applying this to simple linear springs, one can apply it to more interesting elastic models. In [119] we used exactly this approach in limiting the strain of tetrahedral finite elements (measured by the deformation of altitudes) to make element collapse impossible.

1.4 Bending Elements

The forces that resist the bending of a surface are more problematic than the in-plane forces. A classic approach to modeling bending problems in continuum mechanics are plates and shells (i.e. plates with non-flat rest positions). However, use of finite element plates and shells necessitates going to higher order basis functions, introduces a large range of numerical difficulties, complicates collision handling, etc. While this is appropriate for accurate simulation of relatively well understood small deformations of materials such as steel, there is no compelling reason to use such a complex force model given the crude state of cloth modeling. Thus I present a simpler more direct model.

Another popular approach in computer graphics has been to augment the rectangular grid discussed above with springs connecting nodes that are two grid locations apart (e.g. [26, 139, 39]). These resist bending of the cloth (in addition to stretching). However, such a model isn't easily generalized to unstructured meshes, and moreover, cannot properly model curved rest positions without introducing "popping". That is, if a curved rest position is used to set the rest lengths of the springs, the mirror image of the cloth (curved in the opposite direction) is also in stable equilibrium, so nodes may flutter back and forth between the two stable points. The model I present here does not suffer from this defect.

In order to handle unstructured triangle meshes and get finer, more robust control over bending than in vertex-centric models, we posit as our basic bending element two triangles sharing an edge.

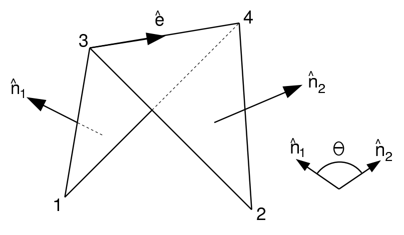

Figure 1.2: A bending element with dihedral angle $\pi - \theta$.

Our bending elements will be based on the dihedral angle and its rate of change, as in [17]. We label the element as in figure 1.2, with vertex positions x_i and velocities v_i, $i = 1, \ldots, 4$, and angle θ between the normals n_1 and n_2.

The vector of velocities $v = (v_1, v_2, v_3, v_4)$ and the vector of bending forces $F = (F_1, F_2, F_3, F_4)$ live in a 12 dimensional linear space. One can select a basis for this space identifying twelve distinct "modes" of motion. For bending it is natural to select for the first eleven modes the three rigid body translations, the three (instantaneous) rigid body rotations, the two in-plane motions of vertex 1, the two in-plane motions of vertex 2, and the one in-line stretching of edge 3–4. None of these change the dihedral angle, and thus should not participate in bending force calculations. This leaves the twelfth mode, the bending mode, which is the unique mode orthogonal to the other eleven up to an arbitrary scaling factor. This mode changes the dihedral angle but does not cause any in-plane deformation or rigid body motion. Let us call it $u = (u_1, u_2, u_3, u_4)$. From the condition of orthogonality to the in-plane motions of vertices 1 and 2, we find that u_1 is parallel to $\hat{n}_1$ and u_2 is parallel to $\hat{n}_2$. From the condition of orthogonality to the in-axis stretching of edge 3–4, we see that $u_3 - u_4$ must be in the span of $\hat{n}_1$ and $\hat{n}_2$. Orthogonality to the rigid body translations implies that the sum $u_1 + u_2 + u_3 + u_4$ is zero, and hence $u_3 + u_4$ is also in the span of $\hat{n}_1$ and $\hat{n}_2$, thus u_3 and u_4 are each in this span. Finally, after making u orthogonal to rigid rotations (which we can conveniently choose to be about the axes $\hat{n}_1$, $\hat{n}_2$ and $\hat{e}$) we end up with

$$u_1 = |E| \frac{N_1}{|N_1|^2} \qquad u_2 = |E| \frac{N_2}{|N_2|^2}$$

$$u_3 = \frac{(x_1 - x_4) \cdot E}{|E|} \frac{N_1}{|N_1|^2} + \frac{(x_2 - x_4) \cdot E}{|E|} \frac{N_2}{|N_2|^2}$$

$$u_4 = -\frac{(x_1 - x_3) \cdot E}{|E|} \frac{N_1}{|N_1|^2} - \frac{(x_2 - x_3) \cdot E}{|E|} \frac{N_2}{|N_2|^2}$$

up to an arbitrary scaling factor, where $N_1 = (x_1 - x_3) \times (x_1 - x_4)$ and $N_2 = (x_2 - x_4) \times (x_2 - x_3)$ are the area weighted normals and $E = x_4 - x_3$ is the common edge. Thus u_1 and u_2 are inversely proportional to their distance from the common edge, and u_3 and u_4 are a linear combination of u_1 and u_2 based on the barycentric coordinates of x_1 and x_2 with respect to the common edge. The bending elastic and damping forces *must* be proportional to this mode. One immediate observation is that orthogonality to rigid body modes implies these forces conserve linear and angular momentum. In fact, every bending model based on two triangles that does *not* use exactly these force directions will violate either the fundamental conservation laws or will influence in-plane (i.e. non-bending) deformations[4].

For simplicity we choose the magnitude of elastic force so that

$$F_i^e = k^e \frac{|E|^2}{|N_1| + |N_2|} \sin(\theta/2)\, u_i,$$

for $i = 1, \ldots, 4$. The elastic bending stiffness k^e is a mesh-independent material property, the middle factor scales this according to the anisotropy of the mesh (so the look of the cloth doesn't change significantly with remeshing[5]), and the sine factor measures how far from flat the cloth is. This is the simplest quantity to compute that smoothly and monotonically increases from a minimum when sharply folded at $\theta = -\pi$, is zero when flat at $\theta = 0$, and rises to a maximum at the other sharply folded state $\theta = \pi$. We use the formula $\sin(\theta/2) = \pm\sqrt{(1 - \hat{n}_1 \cdot \hat{n}_2)/2}$ where the sign is chosen to match the sign of $\sin\theta$, which is just $\hat{n}_1 \times \hat{n}_2 \cdot \hat{e}$. Naturally more complex nonlinear models, e.g. including powers of θ for increased resistance at sharper angles, are possible if a more accurate model is known. But we stress that these factors must multiply all of the forces so that the force directions and proportionalities do not change.

In many cases an artist desires that particular folds should consistently appear in a character's clothing to define their look. Even the best tailoring may not do this when the character is in motion, and one tool in cloth simulation that can overcome this is sculpting folds directly into the garment. We can straightforwardly model this with non-zero rest angles. Other manifolds such as skin, skin-tight synthetic suits, molded rubber, etc. also naturally have non-flat resting positions. To account for these we use

$$F_i^e = k^e \frac{|E|^2}{|N_1| + |N_2|} \left(\sin(\theta/2) - \sin(\theta_0/2)\right) u_i,$$

for $i = 1, \ldots, 4$ where θ_0 is the rest angle. Nonzero rest angles are also useful in other areas: e.g. [71] derived essentially the same bending model as a lightweight alternative to shells for simulating stiff curved materials such as hats and creased paper, [34] used nonzero rest angles to model curly hair styles, and [124] designed "angular springs" to aid in volume preservation of muscle tissue. Figures 1.3 and 1.4 show examples of nonzero rest angles with widely different bending stiffnesses.

[4]Some may argue that in reality, in-plane and bending deformations are subtly coupled, but the exact nature of this coupling varies between materials and is not understood for even the simplest fabrics, thus it is wisest to avoid adding arbitrary and artificial coupling.

[5]Thanks go to David Baraff for helping us arrive at the correct force scaling which gives convergence in the mesh refinement limit.

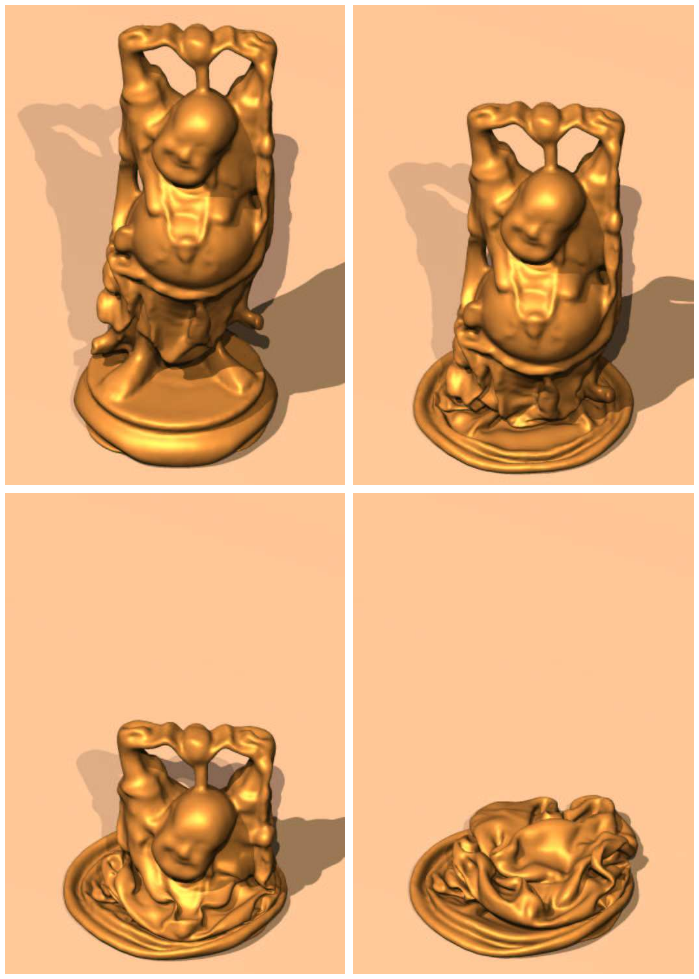

Figure 1.3: Nonzero rest angles allow pre-sculpted folds, as in this “cloth” Buddha collapsing under its own weight.

The damping component of the bending forces depends on the rate of change of the dihedral angle and acts to slow it down. Taking the time derivative of $\cos\theta = \hat{n}_1 \cdot \hat{n}_2$ we find after simplification that

$$d\theta/dt = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3 + u_4 \cdot v_4.$$

Parenthetically, this is essentially an alternative derivation of the bending mode u, i.e. it is the gradient (the steepest ascent direction) of the dihedral angle. Then the damping bending forces are

$$F_i^d = -k^d |E| (d\theta/dt) u_i,$$

where we again include a mesh scaling factor $|E|$ so that k^d is a material property and may be kept constant as the mesh is refined, possibly anisotropically. This can be rewritten as $F^d = -k^d |E| (u \otimes u) v$, showing that the damping forces are linear in the velocities and have a symmetric negative semi-definite Jacobian matrix that allows efficient linear solves (with conjugate gradients, for example) for implicitly integrating the velocities. Again, while a more complex non-linear function of $d\theta/dt$ may be used for the force magnitude (any such function will preserve symmetry), a model that uses a combination of the velocities other than $\sum_i u_i \cdot v_i$ or generates forces in different directions is not correct: such a model is damping motion other than bending.

We also note that in all these formulas, for extra efficiency the factors such as $|E|$, $|N_1|$, etc. can be kept constant from their rest pose values, and thus precomputed. For small amounts of in-plane stretching, the difference is not large, and it is not clear which approach is better (again, since the real physics are poorly understood and the models are crude).

An alternative approach to bending that some authors have taken (e.g. [166, 17, 71]) is to define bending forces as the variational derivative of a potential that is approximately proportional to the integral of mean curvature squared. In particular, [17, 71] did this by looking at the dihedral angles between adjacent triangles. The formula in the first paper isn't quite correct (i.e. it does not converge to the correct integral when the mesh is refined toward some smooth limit geometry), but should be:

$$E_{bend} = k^e \sum \frac{|E|^2}{|N_1| + |N_2|} \theta^2$$

where the sum is over all interior edges of the mesh, and E, N_1, N_2, and θ are defined as above. This formula can easily be verified for the simple case of regular meshes wrapped around cylinders. An alternative derivation of this formula can be found in [71]. Note that for small angles $\theta \approx \sin(\theta)$ and that as a mesh is refined toward a smooth limit surface the angles between adjacent triangle normals do approach zero. Also, since $u_i = \partial\theta/\partial x_i$ as mentioned above, taking the variational derivative of E_{bend} does result in elastic forces very close to the ones we propose above. That being said, the direct "modal analysis" approach is in some ways preferable: it is simpler to modify the resistance (just directly replace $sin(\theta/2)$ with the desired function of θ) and it is simpler to understand how modifications will change the dynamics. In addition, the authors of [17] found that the variational

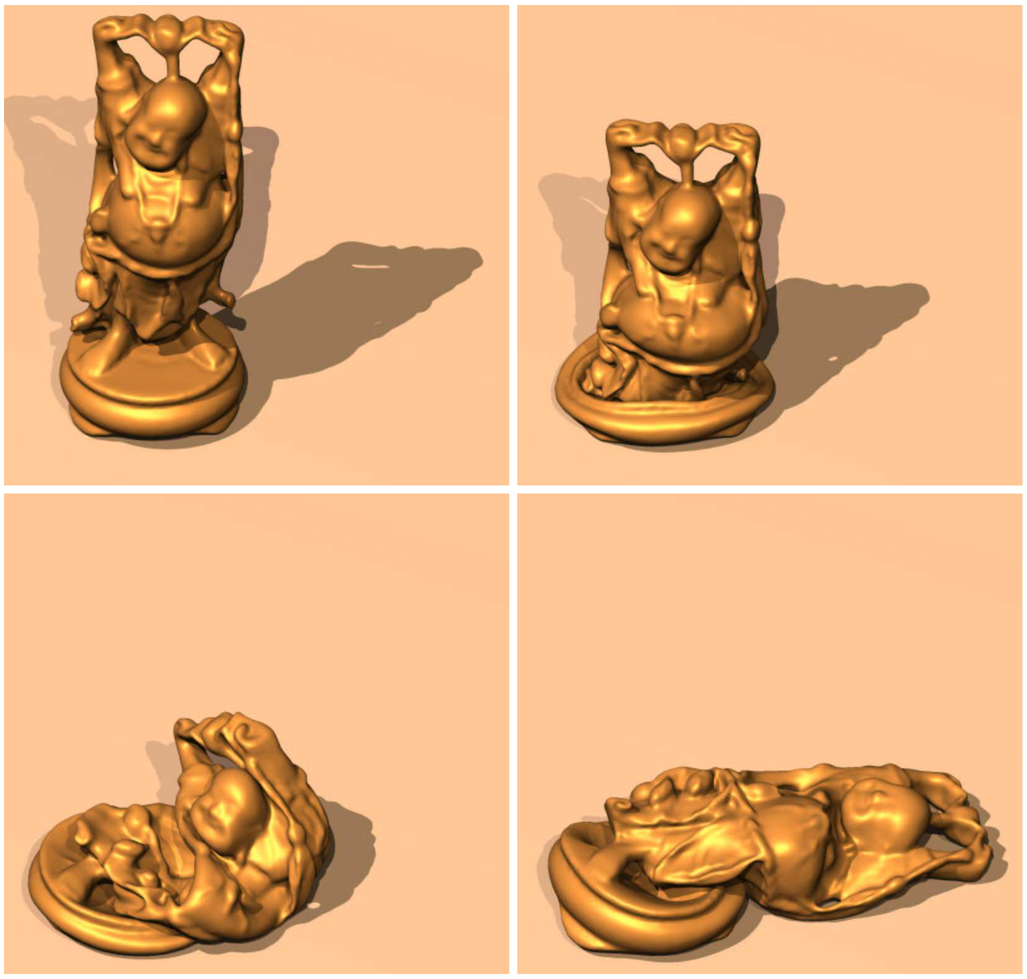

Figure 1.4: The “cloth” Buddha with significantly stronger bending springs that dominate all other forces, illustrating that our formulation is well behaved in the stiff limit, robust across a wide range of materials.

derivatives are tricky enough that approximations (assuming some terms are constant) were used, and the authors of [71] resorted to automatic differentiation to evaluate forces.

1.5 Time Integration

At this point, the thesis has presented the details of force computations and a strain (-rate) limiting post-processing method. In order to actually simulate the motion of a piece of cloth, however, a scheme for integrating Newton's equation of motion $F = ma$ forward in time is required.

In [28] we argue that a mixed explicit/implicit scheme is highly effective for cloth simulation. The motivation is to combine the flexibility and simplicity of explicit methods (such as Runge-Kutta) with the efficiency of implicit schemes (such as backward Euler). In particular, we go explicit on the elastic forces (those that are independent of velocity) and implicit on the damping forces (the velocity-dependent forces). The algorithm is presented below, with x denoting positions, v velocities, and a accelerations (from Newton's law $F = ma$):

- $v^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, v^n)$ (explicit update)
- Modify $v^{n+1/2}$ to get $\tilde{v}^{n+1/2}$ to limit strain, etc.
- $x^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$ (explicit update)
- $v^{n+1} = v^{n+1/2} + \frac{\Delta t}{2} a(t^{n+1}, x^{n+1}, v^{n+1})$ (implicit solve)
- Modify v^{n+1} in place to limit strain, etc.

This corresponds to the central Newmark scheme (see e.g. [86]). Essentially we are combining the explicit second order accurate leapfrog scheme for the position update with an implicit second order accurate trapezoidal rule for the velocity update (decomposed here into a half step with forward Euler and a half step with backward Euler; substitute the definition of $v^{n+1/2}$ into the implicit step to see the standard form of trapezoidal rule). The combination is a second order accurate method that is stable for $\Delta t < O(\sqrt{\rho/k^e}\Delta x)$ where ρ is the density, k^e is the material stiffness, and Δx is the smallest edge length of the mesh, completely independent of how large the damping forces are. This is a critical point, since a fully explicit scheme like Runge-Kutta has a much more stringent quadratic time step restriction dependent on the damping forces, $\Delta t < O(\rho/k^d \Delta x^2)$ where k^d is the material damping parameter. The extra factor of Δx means an order of magnitude more steps are required for such a scheme. On the other hand, using much larger time steps with an implicit scheme as in [17] introduces significant artificial damping, which only recently was mitigated by the carefully tailored bending model in [39]: we do not need this fix, but instead are free to use the more sophisticated bending model presented above.

An additional advantage of only going implicit on velocities is that while the forces in cloth simulation are typically nonlinear and depend on positions in a complicated manner, the damping forces are often linear in the velocities with a symmetric Jacobian. This means we can use a fast conjugate gradient solver without need for simplifying approximations such as those made in [17].

The use of an explicit update for position allows us to modify the velocities very simply to enforce a wide variety of constraints, and further allows the strain limiting procedure discussed above. We are careful, however, to avoid performing the strain limiting procedure in the middle of a trapezoidal rule (i.e. we do the backward Euler half-step starting from the *unmodified* results of the forward Euler half-step), to avoid disturbing accuracy and stability.

We note that for highly damped materials that while the above algorithm is stable it is not "monotone". That is, spurious oscillations may be introduced, because the first explicit half step of the velocity (following which the position is updated) overshoots, and then is damped back down to stability by the implicit half step.[6] This effect can be minimized by simply switching the order of the velocity steps, using the fact that backward Euler is unconditionally monotone:

- $v^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, v^{n+1/2})$ (implicit solve)
- Modify $v^{n+1/2}$ in place to limit strain, etc.
- $x^{n+1} = x^n + \Delta t v^{n+1/2}$ (explicit update)
- $v^{n+1} = v^{n+1/2} + \frac{\Delta t}{2} a(t^{n+1}, x^{n+1}, v^{n+1/2})$ (explicit update)

Note that the trapezoidal rule is used between $v^{n-1/2}$ and $v^{n+1/2}$, not v^n and v^{n+1} as before, thus we only get second order accuracy and stability independent of damping forces if the time step Δt remains constant between time steps. If a variable time step is desired, as is usually the case, we no longer have the trapezoidal rule and destroy both accuracy and stability. A better approach that is preferable in the highly damped situation is to update the position with the implicit half step, but then rewind and use the trapezoidal rule as in our first method:

- $\tilde{v}^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, v^{n+1/2})$ (implicit solve)
- Modify $\tilde{v}^{n+1/2}$ in place to limit strain, etc.
- $x^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$ (explicit update)
- $v^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, v^n)$ (explicit update)
- $v^{n+1} = v^{n+1/2} + \frac{\Delta t}{2} a(t^{n+1}, x^{n+1}, v^{n+1})$ (implicit solve)
- Modify v^{n+1} in place to limit strain, etc.

This allows variable step sizes with second order accuracy and monotone behavior at the small expense of an additional implicit solve. To summarize: we use the first method by default, but switch to this last method for highly damped materials.

[6]Even fully implicit methods suffer from these oscillations when second order or higher accuracy is required, e.g. the BDF2 method used in [39]. Essentially all implicit methods with higher order than backward Euler consist of an explicit partial step or partial extrapolation, which will overshoot if the explicit time step restriction is not obeyed, followed by a backward Euler step for the "rest" of the time step, which damps any stability problems.

Chapter 2

Contact and Collision for Cloth

The main bottleneck in cloth simulation is typically the contact/collision resolution. This can be divided into object collision (cloth versus other objects) and self-collision (cloth versus itself).

Object collision is in many respects the simpler problem, at least for adequately smooth and thick objects. Small penetrations of the cloth with the object are usually acceptable during simulation since they can easily be prevented from going too deep, and can simply be fixed on a frame-by-frame basis in parallel during rendering.

For self-collision there are two problems that must be tackled. One is simply the size of the problem: since cloth is so flexible and since every element of the mesh is on the surface, huge numbers of contacts or collisions are expected in common folding situations. Naive algorithms such as resolving collisions in chronological order with an event queue will simply grind to a halt. The other problem is low tolerance: since the cloth thickness is not modeled in the geometry, any unresolved contact or collision results in cloth self-intersecting and poking out the other side. Such self-intersections tend to get caught, tangling the cloth wholly unphysically and destroying plausibility. Figure 2.1 illustrates the potential complexity of self-collision, in a simulation with the algorithm presented below. Briefly put, this algorithm combines fast and accurate (but not robust) repulsion forces with robust (but slower and less accurate) geometric methods, exploiting the complementary strengths of both approaches.

Most citations to previous work in collisions will be made in context. However, as a general overview of the main sources for this research, see Terzopoulos et al.[166] who used penalty methods for deformable bodies, Provot[140] who took an alternate and more robust geometric approach for cloth, Baraff[9, 10, 11, 12, 13] who carried out a detailed study of numerical methods for rigid body motion with contact and friction in a series of papers, and Mirtich and Canny[118] who showed that one could use just impulses even for resting contact in a micro-collision framework.

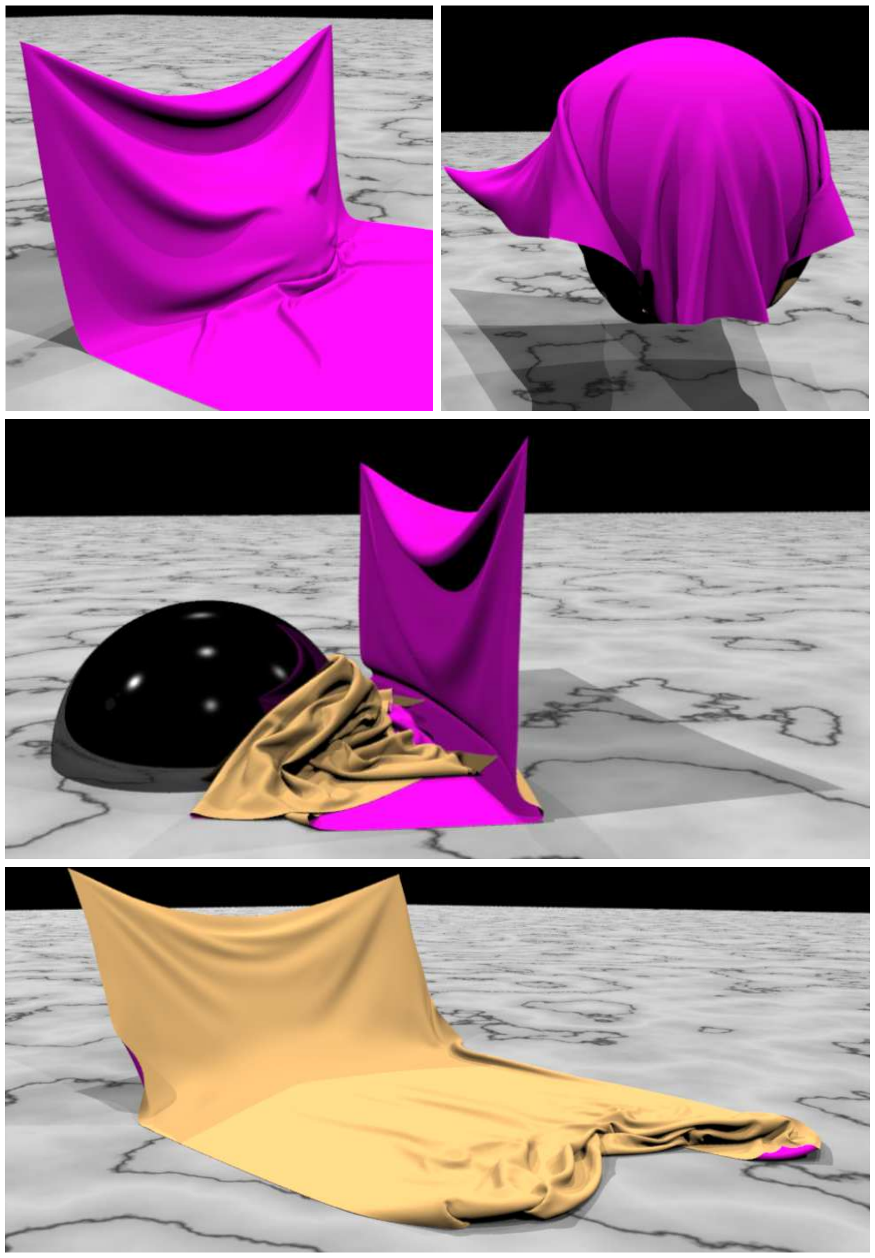

Figure 2.1: Initially, a curtain is draped over a ball on the ground. The ball moves, flipping the curtain over on top of itself producing stable folds and wrinkles using our static friction model. Another ball enters the scene and pushes through the complicated structure of folds before slipping underneath unraveling the curtain.

2.1 Collision with Objects

Our first assumption is that the other objects in the simulation are sufficiently massive relative to the cloth that the cloth will not affect their motion. This is often the case in computer animation produced in layers, where skeletons are animated first, flesh and skin are added on top driven by the skeleton, and clothes come last.

We further restrict attention to thick and smooth objects (relative to the sampling density of the cloth mesh). These are very well represented by level sets (see e.g. [131]), that is, as the implicit zero isocontour of a signed distance function discretely sampled on a grid. While the signed distance functions for simple geometry such as planes and spheres are straightforward analytic expressions, in general one approximates signed distance with a numerical method. The fast marching method[171, 151] is an efficient $O(N \log N)$ first-order accurate method (where $N = n^3$ is the number of points in the three-dimensional grid). An alternative that runs in linear time is the fast sweeping method[170]. Signed distance functions can also be directly sculpted or modeled [136, 122, 44] or obtained from range images [43, 185, 45]. When collision objects are required at times in between precomputed level sets, it is simple to interpolate the necessary information as was done for motion blur in [57]. Other methods exist as well, for example rasterizing the time swept surface [149].

We also make use of object velocities in our method (see below). For deformable bodies, we map the local velocity of the object to the points interior to the object, and then use a velocity extrapolation method [2] to extend the velocities to the grid points outside the object as well (see also [57] where this method was used to define water velocities in the air region). Note that this does not need to be done for rigid bodies since their pointwise velocities are intrinsically defined throughout space by their translational and rotational components.

Essentially our algorithm detects any cloth nodes that penetrate collision objects and applies impulses to push them to the surface. Although [53] pointed out that merely checking the points of one object inside the other is not sufficient to catch all collisions—more complex tests may be needed especially when the sample points do not adequately resolve the object—we make the assumption that we do in fact have adequate resolution.

Consider a collision between a point on our cloth p moving with velocity v_p and the level set collision object. At the spatial location of p, we use the level set value ϕ, the outward pointing normal $N = \nabla\phi$, and the velocity of the level set collision object v at that point. We compute the distance p must travel along N to resolve the future interference of the point with the object. Thus we anticipate collisions, similar to the suggestion in [67]. During a time step Δt, we predict that our point will move to the

$$\phi^{new} = \phi + \triangle t(v_p - v) \cdot N$$

isocontour where the second term accounts for the relative motion between the point and the collision object in the normal direction. If ϕ^{new} is positive, the point will end up outside the level set and nothing needs to be done, but otherwise we need to process a collision. First, we calculate the normal and tangential velocities for both the point and the level set, e.g.:

$$\begin{aligned} v_N &= v \cdot N \\ v_T &= v - v_N N. \end{aligned}$$

We update both the normal and tangential components to obtain $v_p^{new} = v_{p,N}^{new} N + v_{p,T}^{new}$ following the collision. The new normal velocity is calculated as

$$v_{p,N}^{new} = v_{p,N} - \phi^{new}/\Delta t$$

which predicts the point to lie exactly on the zero isocontour after time Δt. Friction is incorporated into the relative tangential velocity update, $v_{T,rel} = v_{p,T} - v_T$ using the formula established below in section 2.7,

$$v_{T,rel}^{new} = \max\left(0, 1 - \mu \frac{v_{p,N}^{new} - v_{p,N}}{|v_{T,rel}|}\right) v_{T,rel}$$

where μ is the friction coefficient. This formula includes both the static and kinetic friction cases, and in particular, exactly enforces zero relative tangential velocity for static friction, which is critical in stably maintaining wrinkles on surfaces (methods that allow small slippage tend to let wrinkles dissipate, degrading realism). The final post-collision velocity is

$$v_{p,T}^{new} = v_T + v_{T,rel}^{new}.$$

The examples in figures 2.1 and 2.3 demonstrate this algorithm.

Thin but smooth objects cannot be handled in this fashion, and should be dealt with like cloth self-collision, discussed in the rest of this chapter. With just this point sampling approach, cloth nodes can erroneously pop through to the other side of the object. With the self-collision algorithm discussed below appropriately adapted to include static objects, we can guarantee that this won't happen. The example shown in figure 2.2 demonstrates this.

However, this still isn't sufficient for *sharp* objects. Even with the robust algorithm adapted from our self collision procedure, problems are apparent: cloth snags unnaturally on sharp points, far beyond the natural snagging one sees in real-life. By ensuring that the triangle mesh cannot intersect the object, but otherwise not constraining their interaction, one quickly ends up with the sharp point next to a node of the cloth mesh. In order for an incident triangle in the cloth to slide past the sharp point, sufficient torque must be applied at the point of contact to *rotate* the triangle—but the point of contact naturally ends up at nearly the least efficient spot for leverage. Thus immense force is required, far more than should be in reality. Visually it appears as if the cloth has snagged or if the friction coefficient is unrealistically high.

This problem hasn't been solved yet, but I will make suggestions for possible solutions. The obvious idea to resolve the problem is to relax the rigidity of the triangles and instead use higher order elements that can curve such as subdivision surfaces—however, this complicates robust collision detection considerably. Using curved elements might only reduce the problem, anyhow, as there is still fundamentally a mismatch between the length scale of the cloth mesh and the length scale of the sharp point. The simulated cloth simply cannot reliably represent folds with that high curvature, and so will strongly resist attempts to get those folds. A useful intuition that will reappear in chapter 5 is to think of the discretized cloth as a real continuum with a set of artificial constraints that reduce its possible configurations to a finite dimensional space. In this case, it may be that the forces trying to drag the cloth around the sharp point will end up fighting these artificial, infinitely strong constraints more than the actual physical elastic resistance of the cloth, because they operate on different length scales.

A more complex approach is to use an Arbitrary Lagrangian-Eulerian (ALE) method[80], decoupling the rigid triangle geometry from the cloth material to some extent. The cloth material could then slide *through* the mesh, around the sharp point, without needing to rotate triangles. In addition to additional complexity (and the need for more advanced continuum models than masses and springs) this may transfer the problem from snagging to mesh tangling, however.

A better idea would be to try to regularize the sharp point in some sense. It may not be possible to actually round off the tip (as was actually done in figure 2.2). An alternate approach is to distribute the repulsion forces or collision impulses to an area of the cloth around the tip, not just to the triangle in contact, virtually rounding off the tip in a sense. Torque will no longer be applied at a single point.

The rest of this chapter concentrates on self-collision.

2.2 A Collision Resolution Pipeline

For the more difficult problem of self-collision, I propose combining existing ideas (namely, repulsion forces, collision impulses, and rigid impact zones) in a *collision resolution pipeline*. After determining a candidate new state for the cloth based on internal dynamics, we process the implied trajectory through a series of stages in a pipeline, each stage detecting and handling some of the collisions.

This notion of a pipeline allows for efficient and robust simulation with *plausible* results. The first stages in the pipeline deal with simple collisions: they are inexpensive to detect and handle, and in the absence of complications the collision response is very accurate. The later stages deal with the remaining complex collisions, with algorithms that are more expensive and may give less accurate (but stable) results.

We expect high efficiency since most collisions are simple, and thus are handled inexpensively at the start of the pipeline. The later stages may be more expensive per collision, but will only need to

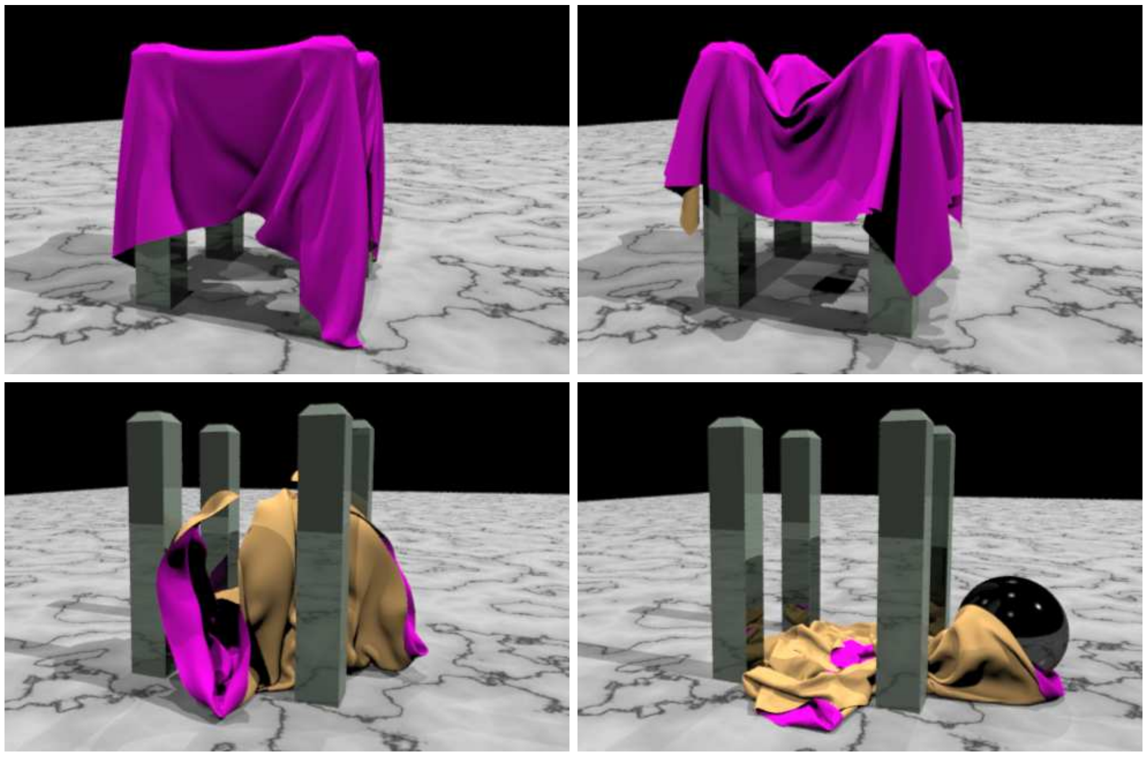

Figure 2.2: A tablecloth draped over table legs with no tabletop is dragged to the ground by a descending sphere.

deal with the few isolated problems that remain. In fact, typically the later stages do nothing more than quickly verify that the earlier stages handled everything. Furthermore, even when complicated collision scenarios arise that need to be dealt with in the later stages, once they have been resolved in one time step (and their colliding motions interrupted) they become simple and later time steps can resolve them in the first stages of the pipeline.

This pipeline also promotes robustness, since the later stages of the pipeline can afford to be very careful about not missing collisions, and can tolerate larger errors in collision resolution (e.g. overly damping relative motion) in the interests of robustness.

The final argument for the pipeline framework is that it nicely corresponds to an audience's sense of plausibility. Humans are relatively good at detecting errors in physics for simple situations (where we employ accurate collision response at the start of the pipeline), but in complex situations where it may be impossible to see much of the detailed interaction and where the collision problem is probably ill-posed or even chaotic to begin with, audiences may accept far less accurate simulations as long as there are no gross violations of the laws of physics.

The same general framework works in other situations apart from cloth self-collision. For example, [76] uses it for rigid body simulation; I will discuss this in a later chapter.

One can interpret the algorithm of [140] as a two-stage pipeline: pairwise impulse dynamics followed by rigid "impact zones". Our later paper [27] added the missing first stage that makes it efficient and accurate: repulsion forces. The rest of this chapter details this complete pipeline.

2.3 Time Integration

We cleanly separate the time evolution of the internal cloth dynamics (and the environment around the cloth) from the collision processing algorithm. This allows us to easily integrate our collision, contact and friction processing algorithms with a pre-existing cloth dynamics engine. Starting at time $t = 0$ with cloth positions x^0 and velocities v^0, the algorithm is as follows. For $n = 0, 1, 2, \dots$

- Select a collision time step size Δt and set $t^{n+1} = t^n + \Delta t$
- Advance to candidate positions $\bar{x}^{n+1}$ and velocities $\bar{v}^{n+1}$ at time t^{n+1} with internal dynamics
- Compute the average velocity $\bar{v}^{n+1/2} = (\bar{x}^{n+1} - x^n)/\Delta t$
- Check x^n for proximity (section 2.4), then apply repulsion impulses (section 2.6) and friction (section 2.7) to the average velocity to get $\tilde{v}^{n+1/2}$
- Check linear trajectories from x^n with $\tilde{v}^{n+1/2}$ for collisions (section 2.4), resolving them with a final mid-step velocity $v^{n+1/2}$ (sections 2.8 and 2.9)
- Compute the final positions $x^{n+1} = x^n + \Delta t v^{n+1/2}$
- If there were no repulsions or collisions, set $v^{n+1} = \bar{v}^{n+1}$
- Otherwise, advance $v^{n+1/2}$ to v^{n+1} and limit the strain rate (section 1.3).

When repulsions or collisions appear, our method for determining the final velocities is similar to the one used for the internal dynamics in the previous chapter. (Note, however, that any reasonable algorithm could be used for internal dynamics in the first line of the algorithm, e.g. one large implicit time step as in [17] or many small explicit Runge-Kutta steps).

The algorithm is stable for any collision time step Δt, thus Δt can be chosen adaptively in a straightforward manner. For example, we choose a minimum Δt as the time step of the cloth dynamics evolution and a maximum Δt on the order of one frame, and start with the maximum. We halve the time step when an actual collision occurs, i.e. the repulsion forces aren't adequate, and try the time step over again only doing the full collision processing at the minimum Δt. Whenever we get three successful time steps in a row we double Δt again. Adaptive time stepping was also addressed in [17]. Recent experiments indicate that this probably isn't necessary, and fixing Δt to allow eight steps per frame is adequate for most animation.

When there are repulsions or collisions, we need to update the velocity from $v^{n+1/2}$ to v^{n+1}. For central differencing we need to solve the implicit equation

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

In many cloth models, such as ours, the damping forces (hence accelerations) are linear in the velocities giving the linear system

$$(I - \tfrac{\Delta t}{2}\tfrac{\partial a}{\partial v})v^{n+1} = v^{n+1/2} + \tfrac{\Delta t}{2} a_e(t^{n+1}, x^{n+1}).$$

Here $\partial a/\partial v$ is the Jacobian matrix of accelerations with respect to velocities, and $a_e(t,x)$ is the elastic component of acceleration, i.e. everything apart from damping. In any reasonable cloth model this matrix will be symmetric positive definite (or can be safely made so, see [17]) after multiplying both sides by the mass matrix, i.e. converting velocities into momenta and accelerations into forces. Then the conjugate gradient algorithm can be used to solve for v^{n+1}. For nonlinear damping Newton's method can be used requiring similar linear solves.

It was observed in [51] that implicit updates such as this last velocity step can be interpreted as a smoothing filter. The additional strain rate limiting from section 1.3 further smooths out any excessive jumps (as a nonlinear filter). This is especially useful here, since collision processing may introduce discontinuities in the cloth's velocity field which left untouched may cause excessive strain rates, leading to visual artifacts and additional unwanted collisions in subsequent time steps that slow the simulation down. Another benefit of the final velocity update step is that when one node collides and impulsively changes its velocity so that it is out of sync with surrounding nodes, the velocity update puts the nodes back in sync reducing the relative velocity of surrounding nodes, even though they have not yet been involved in a collision. This again reduces unnecessary stress on the cloth and also increases the likelihood that repulsion forces will stop the surrounding nodes before they actually collide, reducing the number of collisions that have to be handled.

2.4 Proximity and Colliding Trajectory Detection

To accelerate the detection of proximities for repulsions and of intersections for collisions, we use an axis-aligned bounding box hierarchy. For more details on hierarchical methods and bounding volume hierarchies, see [77, 182, 18, 70, 102]. Further pruning of unnecessary tests between adjacent patches in low curvature regions of cloth is possible, at least for static proximity tests, see [177, 176]; we have not used this optimization.

Our box hierarchy is built bottom-up once at the beginning of the simulation using the topology of the cloth mesh. In one sweep we greedily pair off adjacent triangles to get parent nodes, then in a sweep in the opposite direction pair off these to get the next level up, and so on alternating sweep directions until we are left with one root node. At each time step we calculate the extents of the axis-aligned boxes for the repulsion calculation (section 2.6) by taking a box around each triangle enlarged by the thickness of the cloth (e.g. 1mm), and then taking the union of the extents in each axis direction as we move up the hierarchy. We also recalculate the hierarchy from the leaves up for each iteration of the collision algorithm (section 2.8), taking a box around each triangle *and* its candidate position at the end of the step, since we have to cover the path that the triangle takes during the time step. We further enlarge the leaf boxes by some rounding error tolerance (e.g. 10^{-6}m). In both cases, we get a set of candidates for interference by intersecting the box to be tested with the root of the tree; only if it overlaps do we recursively check its children, proceeding until we reach the leaves of the tree. The leaves we reach indicate on which triangles the actual geometry tests need to be performed. These tests break down into checking points against triangular faces and edges against other edges (naturally we don't compare a point against the triangle that contains it, or an edge against an edge that shares an endpoint).

In what follows we use the shorthand x_{ij} to mean $x_i - x_j$. To check if a point x_4 is closer than some thickness h to a triangle $x_1x_2x_3$ with normal $\hat{n}$ we first check if the point is close to the plane containing the triangle: $|x_{43} \cdot \hat{n}| < h$. If so, we project the point onto the plane and compute the barycentric coordinates w_1, w_2, w_3 with respect to the triangle:

$$\begin{bmatrix} x_{13} \cdot x_{13} & x_{13} \cdot x_{23} \\ x_{13} \cdot x_{23} & x_{23} \cdot x_{23} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} x_{13} \cdot x_{43} \\ x_{23} \cdot x_{43} \end{bmatrix}$$

$$w_1 + w_2 + w_3 = 1.$$

These are the normal equations for the least-squares problem of finding the point $w_1x_1+w_2x_2+w_3x_3$ (in the plane) closest to x_4. If the barycentric coordinates are all within the interval $[-\delta, 1+\delta]$ where δ is h divided by a characteristic length of the triangle, the point is close.

To check if an edge x_1x_2 is close to another edge x_3x_4 we find the pair of points, one on each edge, that are closest and check their distance. The search for the two closest points begins by checking for the degenerate case of parallel edges, i.e. if $|x_{21} \times x_{43}|$ is smaller than a round-off tolerance. If so, it reduces to a simple one-dimensional problem. Otherwise, we find the points on the infinite lines that are closest via the normal equations:

$$\begin{bmatrix} x_{21}\cdot x_{21} & -x_{21}\cdot x_{43} \\ -x_{21}\cdot x_{43} & x_{43}\cdot x_{43} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x_{21}\cdot x_{31} \\ -x_{43}\cdot x_{31} \end{bmatrix}.$$

If these points are on the finite edges we are done, otherwise we clamp them to the endpoints. The point that moved the most during clamping is guaranteed to be one part of the answer, and the second point is found by projecting the first onto the second infinite line and clamping to the finite edge. The direction pointing from one point to the other is saved as the "normal" vector. We also save their relative positions along the edges, i.e. the weights $0 \le a, b \le 1$ so that the points are $x_1 + ax_{21}$ and $x_3 + bx_{43}$.

To detect a collision between a moving point and a moving triangle, or between two moving edges, we first find the times at which the four points are coplanar assuming they move with constant velocity over the collision time step as in [120, 140, 52]. The latter two showed this reduces to finding the roots of a cubic equation,

$$(x_{21} + tv_{21}) \times (x_{31} + tv_{31}) \cdot (x_{41} + tv_{41}) = 0.$$

Any roots outside of the interval $[0, \Delta t]$ (slightly enlarged by a small round-off error tolerance, say 10^{-6}) are discarded. The remaining roots are checked in increasing order with proximity tests at time t. If the elements are closer than a small rounding error tolerance (10^{-6}m for our simulations, which is 1000 times smaller than the cloth thickness), we register a collision. We likewise check for proximity at the end of the time step, $t = \Delta t$, in case rounding errors hide a collision at the boundary between two time steps. While earlier works neglected rounding error, our approach guarantees (if collisions are resolved) that the cloth is separated by at least the rounding error tolerance at every time step and never self-intersects during time steps.

2.5 Interpolating Within the Cloth

We often need to deal with two points from the cloth, computing their relative velocity or applying an impulse to them. However, we cannot directly look up or alter the velocities of such points, and instead must deal with the corners of the triangle or endpoints of the edges.

To compute the velocity of a point interior to a triangle or edge we use linear interpolation, which is exact for affine deformations of the element (i.e. translation, rotation, and affine shearing and scaling). In particular, if a point interior to a triangle $x_1x_2x_3$ has barycentric coordinates w_1, w_2, w_3 (calculated during proximity or collision detection) its interpolated velocity is $w_1v_1 + w_2v_2 + w_3v_3$, and similarly if a point interior to an edge x_1x_2 is the fraction a along the edge then its interpolated velocity is $(1-a)v_1 + av_2$. Note that we are finding the velocity of a specific piece of material involved in a contact or collision, i.e. the weights w_i or a are fixed so their time derivatives do not appear.

If an impulse of magnitude I in direction $\hat{n}$ needs to be applied to two points in the cloth (i.e. $I\hat{n}$ to the first and $-I\hat{n}$ to the second), we instead apply impulses to the triangle corners or edge endpoints, weighted by the barycentric coordinates, so that the desired change in relative (interpolated) velocity for the two points is achieved. For the point-triangle case, where an interior point of triangle $x_1x_2x_3$ with barycentric coordinates w_1, w_2, w_3 is interacting with point x_4, we compute adjusted impulses

$$\tilde{I} = \tfrac{2I}{1+w_1^2+w_2^2+w_3^2}$$

$$\begin{aligned} v_i^{new} &= v_i + w_i(\tilde{I}/m)\hat{n} \qquad i = 1, 2, 3 \\ v_4^{new} &= v_4 - (\tilde{I}/m)\hat{n} \end{aligned}$$

assuming all nodes have mass m. For the edge-edge case where a point with relative position a along the edge x_1x_2 interacts with a point with relative position b along the edge x_3x_4, the adjusted impulses are

$$\tilde{I} = \tfrac{2I}{a^2+(1-a)^2+b^2+(1-b)^2}$$

$$\begin{aligned} v_1^{new} &= v_1 + (1-a)(\tilde{I}/m)\hat{n} & v_2^{new} &= v_2 + a(\tilde{I}/m)\hat{n} \\ v_3^{new} &= v_3 - (1-b)(\tilde{I}/m)\hat{n} & v_4^{new} &= v_4 - b(\tilde{I}/m)\hat{n}. \end{aligned}$$

Weighting the impulses in this way introduces appropriate torques for off-center interactions as well as giving continuity across triangle boundaries, and converges to the expected formulas when the interior points approach mesh nodes.

2.6 Repulsion Forces

Repulsion forces, also known as penalty forces, are a popular technique for collision resolution due to their simplicity and apparent speed. However, if a simulation relies exclusively on them, robustness may demand highly stiff forces that necessitate small time steps—and thus slow simulation times. This may not always be an issue, for example [129, 128] were already limited to very small time steps by the extremely stiff internal dynamics of the materials they simulated fracturing, but for cloth it is a problem. In our pipeline we have a robust geometric algorithm as a back-up, and thus are free to use weaker repulsion forces that don't slow down the simulation. Furthermore, we can actually exploit them as a modeling tool. By varying the spatial extent of the repulsion force field we can model the thickness of the cloth, and by varying the stiffness we can model the compressibility.

On the flip side, geometrically resolving the tens of thousands of collisions that can readily occur in folding and contact situations would be prohibitively expensive. This is why repulsion forces are mandatory. They significantly reduce the number of actual collisions usually eliminating them altogether, making our collision pipeline efficient.

Our cloth is given a realistic thickness, e.g. 1mm, and repulsion forces are only applied between pieces of cloth that have this close proximity. As discussed in section 2.4, we use an axis-aligned bounding box hierarchy to make this proximity detection efficient. Proximity is determined for both point-triangle pairs and edge-edge pairs. If a pair is close enough, then two kinds of repulsion forces are applied. The first is based on an inelastic collision, and the second is a spring based force.

Baraff and Witkin[15, 16] discussed collision modeling for deformable bodies pointing out that when a discretized rod collides with a wall, the endpoint of the rod should come impulsively to rest, i.e. the endpoint is subject to a completely inelastic collision impulse. Subsequently the rod stores energy in compression and then expands, separating from the wall. The loading and unloading of the elastic rod models a completely elastic collision. They pointed out that inelasticity can only be introduced by adding damping forces internal to the rod that dissipate energy due to the collision. Other authors have used completely inelastic collisions as well, such as [51] and [32] (who used completely inelastic collisions to remove the “kicks” generated by the repulsion springs of [99]). We take a similar approach, modeling all cloth-cloth collisions and cloth-object collisions using an identically zero restitution coefficient in Poisson’s hypothesis. The energy stored in deformations of our mass-spring model when one of the nodes abruptly comes to rest against an object in its path is released as the cloth restores itself, causing it to bounce. In fact, most real-world cloth-cloth and cloth-object collisions are fairly inelastic, so even with a zero restitution coefficient one should take care to monitor the energy stored within the cloth. We do this intrinsically by limiting the strain rate as discussed in section 1.3.

Since our repulsion forces are meant to dramatically reduce the number of subsequent collisions, we incorporate a completely inelastic collision into our repulsion force. Suppose two points in the cloth, one inside a triangle and one a node of the mesh or both interior to mesh edges, are close and have relative velocity v_N in the normal direction which is less than zero, i.e. they are approaching each other. (See section 2.5 above for details on interpolating velocities interior to triangles and edges, and section 2.4 for details on the normal direction used in point-triangle and edge-edge interactions.) To stop the imminent collision we apply an inelastic impulse of magnitude $I_c = mv_N/2$ in the normal direction. (See section 2.5 for how we distribute the impulse to the mesh nodes involved.)

Since our cloth model includes a realistic cloth thickness, we would like to ensure that pieces of the cloth are well separated at a distance on the order of this cloth thickness. This helps cloth to slide over itself (and objects) without the excessive snagging caused by the discretization errors resulting from the representation of smooth surfaces with discrete triangles. When pieces of cloth are too close together, there is a compression of cloth fibers, and a second repulsion force is applied to model this compression. The repulsion force is proportional to the overlap beyond the cloth thickness h (e.g. 1mm). However, since our robust collision handling algorithm (see section 2.8) will stop every intersection, our spring repulsion force is limited to a maximum when the objects touch, thus avoiding problems with stiffness. Furthermore, we limit our repulsion force so that objects are never propelled outside this overlap region in a single time step. This not only helps to reduce stiffness, but allows cloth in contact to stay close enough together to feel repulsion forces in subsequent time steps. This is important since the repulsion force magnitude is used to model friction, and thus friction forces are also felt in future time steps producing stable folds and wrinkles that add to the visual realism. Many other authors have used spring based repulsion forces, see e.g. [90, 110], but their methods suffered from undue stiffness since an arbitrary amount of interpenetration was allowed to occur. Again, this is alleviated in our model by the robust geometric collision algorithm that stops interpenetration resulting in a bound on the magnitude of the spring based repulsion force.

The spring based repulsion force is modeled with a spring of stiffness k. As a simple heuristic, we found that matching the stiffness of the stretch springs in the cloth gave good results—though of course, this can be tuned to give a particular look of compressibility, e.g. to distinguish leather from fluffy towels. The overlap is

$$d = h - (x_4 - w_1 x_1 - w_2 x_2 - w_3 x_3) \cdot \hat{n}$$

giving a spring force of kd in the direction $\hat{n}$. Multiplying by Δt gives the impulse. As discussed above, we limit this so that the relative velocity change will reduce the overlap by at most some fixed fraction (e.g. $.1h$) in one Δt time step. If the normal component of relative velocity $v_N \geq .1d/\Delta t$ already we apply no repulsion, otherwise we compute the impulse magnitude

$$I_r = -\min\left(\Delta t k d, m\left(\frac{.1d}{\Delta t} - v_N\right)\right)$$

and distribute it to the mesh nodes as explained in section 2.5.

2.7 Friction

We use Coulomb's model for friction, both static and kinetic, with a single friction parameter μ. The repulsion force F_R from section 2.6 is the negative of the normal force F_N pressing the cloth together. Therefore a friction force, in the direction of the pre-friction relative tangential velocity v_T^{pre} but opposite to it, has magnitude at most μF_N. This integrates to an impulse of magnitude $\mu F_N \Delta t$ in the same direction, and thus a change in the relative tangential velocity of at most $\mu F_N \Delta t / m$ where m is the mass (assumed equal for all particles involved). If this is larger than $|v_T^{pre}|$, then either the cloth was slipping and stopped due to kinetic friction, or was stuck and shouldn't be allowed

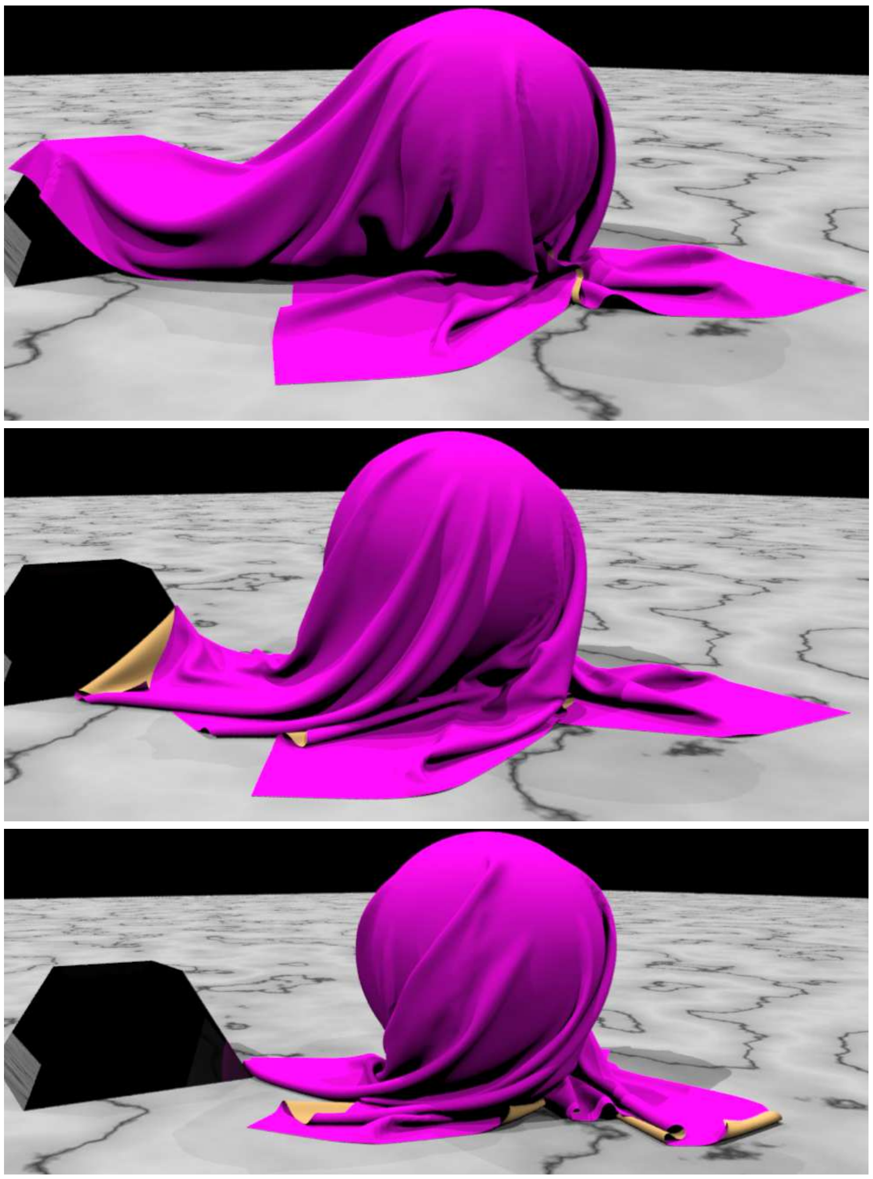

Figure 2.3: The friction between a rotating sphere and a piece of cloth draped over it creates a complex structure of wrinkles and folds.

to start slipping due to static friction. Either way the new relative tangential velocity should be zero. If not, we can simply subtract this off the magnitude of the relative tangential velocity to account for kinetic friction slowing down the slipping. This calculation can be simplified by noting that $F_N \Delta t/m$ is just Δv_N, the change in relative velocity in the normal direction, which can be directly calculated in the repulsion algorithm. Then the decrease in the magnitude of the relative tangential velocity is $\min(\mu \Delta v_N, |v_T^{pre}|)$, i.e. our final relative tangential velocity is

$$v_T = \max\left(1 - \mu \frac{\Delta v_N}{|v_T^{pre}|}, 0\right) v_T^{pre}$$

We apply impulses to achieve this for both point-face proximities and edge-edge proximities. Figure 2.3 illustrates the use of this formula for object collisions as well (as mentioned in section 2.1).

2.8 Geometric Collisions

Repulsion forces alone cannot ensure that no interpenetrations will occur since positions are only checked at discrete moments in time. For robust collision handling, the path of the cloth between time steps must be considered as discussed in section 2.4.

Some authors back up simulations in time to treat collisions in chronological order, e.g. [77]. When a single time step may have thousands of collisions and contacts, as is characteristic of highly deformable bodies like cloth, this would be too expensive. The problem was partially addressed for rigid bodies by [116] who processed the rigid bodies in parallel using 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 small number of contact groups which unfortunately is the case for cloth as the entire piece of cloth has every node in contact with every other node through the mass-spring network.

Instead of rewinding the simulation to process one collision at a time, we resolve them all simultaneously with an iterative procedure as did [176, 140, 113]. We also take this approach later for rigid bodies in [76] (chapter 4). This does not give the same result as processing the collisions in chronological order. However, there is enough uncertainty in the collision modeling and error in the cloth discretization 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 interactions between fuzzy strands of cloth. More details on sampling plausible solutions according to probability distributions reflecting a number of factors can be found in [38] who addressed the related problem of multiple colliding rigid bodies.

As discussed in section 2.4, our geometric collision processing algorithm is activated either when a collision actually occurs or when geometry (points and faces or edges and edges) is in (too) close proximity at the end of a time step. Thus, we need to account for both approaching and separating objects when a "collision" is registered. If the geometry is approaching, we apply a completely inelastic repulsion impulse. Otherwise, if the geometry is already separating (as may happen at the end of the time step, i.e. a close call rather than a true collision), we apply a spring based repulsion force. See section 2.6 for more details on both of these.

While processing all the collisions that occurred during a time step, we may have inadvertently caused new secondary collisions to occur. In order to keep the cloth interference free, we must analyze the newly computed post-collision path of the cloth for possible collisions and process these as well. This means that the bounding box hierarchy needs to be adjusted to account for the new post-collision velocities. Then secondary collisions can be detected and corrected, etc., and the process continues until a final interference free state can be computed. Since relatively large bounding boxes that contain the moving triangles need to be recomputed for every iteration, and a cubic equation has to be solved for every possible collision, this may be expensive. Luckily, repulsion forces tend to catch and treat almost all collisions making the iteration scheme here practical to apply even for high velocity cloth with many nodes and a high degree of folding and contact. However, there are still situations where many iterations are required, so after a few iterations we switch to a failsafe method which quickly eliminates all collisions. We use the method proposed by [140], but not followed through in the literature, possibly because the formulas proposed in [140] *do not* give true rigid body motion. We give corrected versions below.

2.9 Rigid Impact Zones

Provot [140] proposed collecting the nodes involved in multiple collisions into "impact zones" which are treated as rigid bodies. This is justified by observing that when cloth bunches together friction will prevent most relative motion. After a few iterations of applying local impulses as outlined above, we switch to the last stage in the pipeline, merging lists of colliding nodes representing impact zones. Initially, each node in its own list. Then, when a point-face or edge-edge collision occurs, the lists containing the four involved nodes are merged together into one larger impact zone. The impact zones are grown until the cloth is collision free. The velocity of the nodes in the impact zone is derived from a rigid body motion that preserves linear and angular momentum. The formula for angular velocity given in [140] is flawed, so we present a corrected version here.

To find the rigid body motion we first compute the initial center of mass of the impact zone and its average velocity

$$x_{CM} = \frac{\sum_i m x_i^n}{\sum_i m}, \qquad v_{CM} = \frac{\sum_i m v_i^{n+1/2}}{\sum_i m}$$

then the angular momentum of the nodes with respect to their center of mass

$$L = \sum_i m(x_i^n - x_{CM}) \times (v_i^{n+1/2} - v_{CM})$$

and the 3×3 inertia tensor of the current configuration of nodes (using δ to represent the identity tensor)

$$I = \sum_i m \left(|x_i^n - x_{CM}|^2 \delta - (x_i^n - x_{CM}) \otimes (x_i^n - x_{CM}) \right).$$

The rigid body angular velocity that would preserve angular momentum is $\omega = I^{-1}L$, so the new instantaneous velocity of node i is

$$v_{CM} + \omega \times (x_i - x_{CM})$$

Applied on its own, this impact zone method has a tendency to freeze the cloth into nonphysical clumps. However, a combination of our repulsion forces and the initial collision impulses tend to keep these impact zones small, isolated and infrequent. Moreover, once formed, they are short-lived as the repulsion forces tend to quickly separate the offending nodes.

The key element of the impact zone idea is that self collision is impossible during rigid body motion. However, this can be generalized: self collision is also impossible for general affine motions. (Of course, motion under any nondegenerate continuous velocity field cannot cause self collision, but will not in general preserve the piecewise linear geometry of the triangle mesh; in approximating the motion by just moving the nodes of the mesh, self collision may be induced.) In fact, since we use a forward Euler update and do not exactly move the cloth nodes in their rigid body motion calculated above, we actually already introduce an error that is equivalent to some dilation proportional to the rotation—but again, to underscore the point, self intersection is still impossible during this step.

However, we could take advantage of this property of general affine transformations. A possible refinement of the impact zone idea that wouldn't damp out quite as much kinetic energy is to project the motion of the impact zone onto the space of affine transformations instead of just rigid body transformations. This might be important in difficult simulations where bad freezing artifacts become apparent with standard rigid impact zones.

To be precise about this projection, let $u^1, u^2, \ldots, u^k$ be certain modes of allowed motion (u_i^k is the vector in $\mathbf{R}^3$ showing the motion of node i in mode k). For affine motions, $k = 12$. Let U be the $3n \times k$ matrix with these modes as columns. If the original nodal velocities are similarly put into a column vector v, we want a new set of velocities v^{new} in the span of U, i.e. $v^{new} = U\alpha$ for some vector α of k coefficients.

The change in motion will be achieved through some set of impulses $p = M(v^{new} - v)$ where M is the mass matrix (a diagonal $3n \times 3n$ matrix with m_i appearing three times along the diagonal for node i). Similar to the modal analysis of bending presented earlier, we want to make sure that the impulses do not perturb the motion of the nodes in the selected modes U, and instead entirely work in the modal space orthogonal to U. This will guarantee conservation of linear and angular momentum (as U contains the translations and rotations respectively). This condition is simply $U^T p = 0$, or written out, $U^T M(U\alpha - v) = 0$. The solution is

$$\alpha = (U^T MU)^{-1} U^T Mv$$

whence the new velocities are given by

$$v^{new} = U(U^T MU)^{-1} U^T Mv$$

It can be easily verified that kinetic energy cannot increase with this operation. This is in fact the calculation that is performed above using just the translations and rotations as the modes, but taking advantage of orthogonality in the translations to reduce the matrix to be inverted to the 3×3 inertia tensor for just the rotations.

Chapter 3

Post-Processing Cloth

For visual purposes, surprisingly good cloth simulation results can be achieved on relatively coarse meshes, i.e. ones on which the fine scale wrinkles and folds are just barely resolved. The key issue is that these details must be qualitatively captured for a realistic look, but their motion doesn't have to be quantitatively very accurate to fool the eye. However, simply rendering the coarse meshes is unacceptable: the jagged edges of the triangle mesh show up very clearly in the fine scale features. For visually pleasing animations a smoother surface is desired. In this chapter I will describe a fast and simple collision-aware subdivision algorithm I developed in [27] that generates such a surface without introducing artifacts.

3.1 Related Work

Some authors have directly simulated smooth surfaces instead of simple triangle meshes. For example, [168] used dynamic B-splines which allowed them to interactively refine the mesh in regions of interest associating the control points with their dynamic simulation mesh nodes. They derived a number of rules for when, where and how to refine, even detecting when mesh refinement would cause intersection and then either stopping refinement or backing up the simulation in time to avoid intersection. In [49] the convex hull properties of a subdivision surface model of cloth were exploited to accelerate collision detection. Grinspun et al.[73] rigorously modeled thin manifolds with subdivision surfaces detecting collisions with their derived bounds on surface normals and refining the mesh as required to resolve them.

The fast simulation of [92] partly recovered details lost when using coarse meshes by modulating a cubic spline interpolant (in post-processing) with sinusoids wherever springs were compressed. This added interesting wrinkles and helped to preserve cloth area, enforcing buckling folds instead of actual compression.

Another motivation for our post-processing is found in [85] who addressed cloth collisions with volumetric objects. They ensured that cloth nodes remained outside the objects making the collision-handling algorithm faster and simpler, but allowed edge and face collisions with the objects. These were handled in a post-processing step before rendering where they added nonactive points and adjusted their positions to eliminate intersections. Additional processing attempted to preserve area, though we have not found this to be an issue worth addressing.

In [14] Baraff suggested "first-order physics" to post-process rigid body positions to eliminate interpenetrations, that is adjusting positions without altering velocities etc. or otherwise disturbing the simulation. Our paper [28] made use of a similar strategy since during the simulation we only push cloth nodes to the surface of level set collision objects without actually guaranteeing they do in fact get to the surface, and definitely not doing anything about cloth triangles possible intersecting objects. (However, as mentioned in the previous chapter, in the special case of dealing with thin objects we instead use an object collision algorithm based on our self-collision algorithm which robustly guarantees cloth triangles will not intersect objects.)

We identified another potential problem related to post-processing cloth-object collisions in [28], namely that often simulations generate attractive wrinkling that slightly intersects objects. Pushing the cloth geometry to the smooth level set surfaces eliminates these wrinkles, significantly reducing the realism of the animation. This was fixed by monotonically mapping penetrating nodes to a narrow band outside the level set instead, preserving relative depths.

Once object collisions with the cloth mesh have been satisfactorily dealt with, the problem still remains of how to pass a smooth limit surface through the simulated nodes.

3.2 Collision-Aware Subdivision

Our post-processing scheme takes the existing intersection-free simulation data and produces a finer, more detailed and smoother approximation to the manifold. Each frame is processed independently after simulation, so refinement need not be considered at all in the simulation code and in particular doesn't need to be done for all the intermediate time steps. This of course permits simple parallelization.

The algorithm proceeds as follows. If we allowed intersection with objects in the simulation, we begin by adjusting the cloth positions in the given frame to eliminate them, iterating back and forth with an adjustment to eliminate cloth-cloth intersections that those adjustments may have caused. We use the repulsions and collision impulses discussed earlier. When this is finished, our original mesh is intersection-free even accounting for rounding error. We then subdivide the mesh putting a new node at the midpoint of each edge. Since the original mesh was intersection-free and the subdivided mesh lies exactly within the original mesh, the subdivided mesh is guaranteed to be intersection-free as well.

Next we use the modified Loop subdivision scheme [107] to find smoother positions for all the nodes of the subdivided mesh. Unfortunately, moving to these smoothed positions may create intersections despite the convex hull property. However, we can view the vector from a node's original position to its smoothed position as a pseudo-velocity and apply our collision detection algorithms to determine when the intersection would occur. We stop the nodes at that point (or just before that point to avoid difficulties with rounding error) as if they had inelastically collided. We of course need to check again to see if these adjustments to the smoothed positions caused new intersections. Typically only a few iterations are required to eliminate all intersections especially since the convex-hull property of the subdivision means intersections are unlikely in the first place. A solution is guaranteed to exist, since the new nodes can simply be left in the positions in which they were created. Once we have a smoothed but intersection-free subdivided mesh, we can subdivide again continuing until the desired resolution is reached.

Since the cloth is originally separated by a finite distance, but each step of subdivision smoothing moves the nodes exponentially less and less, we very quickly find no more adjustments need to be made. In particular, we halve the minimum separation distance at each subdivision step, while the distances to the smoothed positions in the Loop scheme tend to decrease to a quarter. Thus in short order no more intersections are possible and regular subdivision can be used (if such resolution is needed).

This post-processing technique is not a cure for simulations that were too coarse. If the mesh was under-resolved, visual "popping" artifacts may be apparent since each frame is processed independently: there may not be enough temporal coherence in the coarse mesh. However, in all the simulations we have run, this has not been the case.

Chapter 4

Contact and Collision for Rigid Bodies

The ideas presented in previous chapters on cloth simulation can be generalized to other domains. While rigid body simulation is not the focus of this thesis, it is instructive to see how the idea of a collision resolution pipeline and a post-processing treatment of friction can be carried over to rigid bodies. This chapter presents a self-contained preprint of our paper [76] that followed this approach.

Repulsion forces turned out to not be as useful in this context, since they make large stacks of objects somewhat "squishy", allowing continued gradual slipping instead of a sharp termination of motion, and thus they were eliminated from the collision resolution pipeline. While the rigid impact zone idea appears to be useful for free-fall simulations (without a fixed ground), for standard situations it results in unrealistic frozen stacks that should in reality topple, thus a different final stage for the pipeline was chosen: our contact graph/shock propagation algorithm still allows global contact information transfer, but now in one direction only, from the ground up. The treatment of friction needed to be extended to handle angular motion, and rolling and spinning friction on top of sliding friction. Finding a good way to model rolling and spinning friction is still very much an open problem, but the current approach gives acceptable results for spheres at least. Another important new contribution is an improved time integration for rigid bodies that cleanly separates collision from contact and makes *ad hoc* velocity thresholds unnecessary in an impulse-based (i.e. micro-collision) algorithm. Finally, a dual explicit/implicit representation of geometry (polygonal mesh plus level set) allows very efficient calculations even with complicated nonconvex objects.

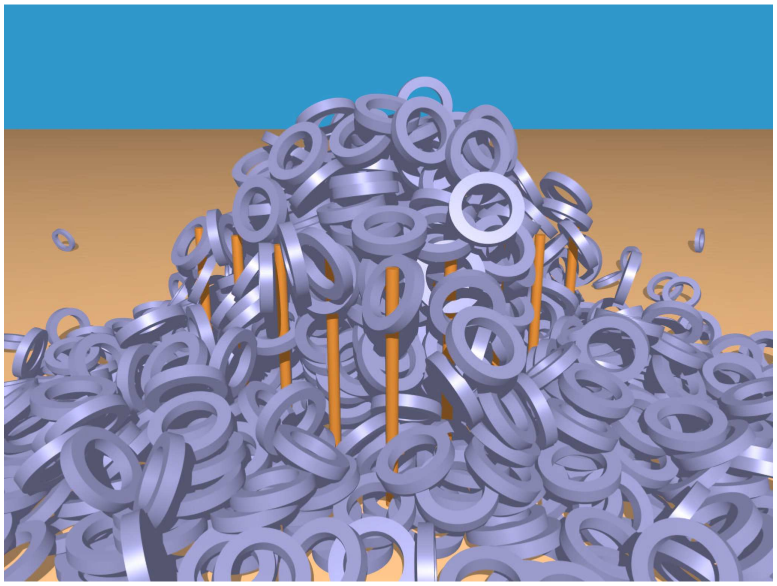

Figure 4.1: 1000 nonconvex rings (with friction) settling after being dropped onto a set of 25 poles. Each ring is made up of 320 triangles, and each pole is made up of 880 triangles.

4.1 Introduction

Dynamic volumetric objects are pervasive in everyday life, and the ability to numerically simulate their behavior is important to a number of industries and applications including feature films, computer games and the automobile industry. 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. Although there are certain areas such as articulated figures, see e.g. [97], where the rigid body approximation is regularly used to remove the need for more expensive deformable models, there are other areas such as brittle fracture, see e.g. [129, 121], where the rigid body approximation could be useful (especially for small fragments) and has not yet been aggressively pursued.

[35] 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. [156] 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. [19] 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 toward 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 [38, 138].

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 [117, 118] (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.

4.2 Previous Work

[77] considered rigid body collisions by processing the collisions chronologically backing the rigid bodies up to the time of impact. [116] 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 in this paper. [77] 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". [120] instead proposed the use of repulsion forces for contact only using the exact impulse-based treatment for high velocity collisions.

[9] 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. [10] extended these concepts to curved surfaces. [11] 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 [12]. [13] proposed a simpler,

faster and more robust algorithm for solving these types of problems without the use of numerical optimization software.

[22] 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. [118] integrated these same nonlinear differential equations to model both contact and collision proposing a unified model (as did [77]). They use the velocity an object at rest will obtain by falling through the collision envelope (or some threshold in [117]) 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 [166] to create repulsive force fields around objects and [135, 150] 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. [58, 79] who used them to treat collision between deformable bodies, and [94] who used them to keep hair from interpenetrating the head. Although [67] 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.

[112] used position based physics to simulate the stacking of convex objects and discussed ways of making the simulations appear more physically realistic. [113] 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 union of 3 boxes. More recently, [147] developed 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 their 45 day estimate for simulating all the cubes.

4.3 Geometric Representation

Since rigid bodies do not deform, they are typically represented with triangulated surfaces. Starting with either the density (or mass), algorithms such as [115] 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.

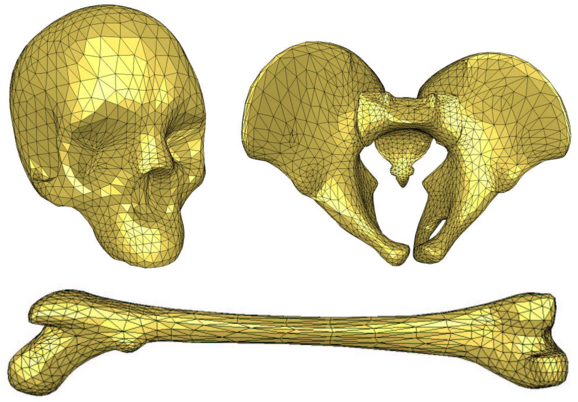

Figure 4.2: Some nonconvex geometry from our simulations: the cranium, pelvis and femur have 3520, 8680 and 8160 triangles respectively.

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 [131] or an octree grid [60] 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. [184]. We use negative values of ϕ inside the rigid body and positive values of ϕ outside so that the normal is defined as $N = \nabla\phi$. 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. A signed distance function can be calculated quickly using a marching method [171, 151] after initializing grid points near the surface with appropriate small negative and positive values. This is a one time cost in constructing a rigid body model and is currently used in several systems, see e.g. [44, 122].

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 $N = \nabla\phi$ 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 [132].

4.4 Interference Detection

[65, 50] 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. [53] pointed out that this test is not sufficient to detect all collisions, as edge-face collisions are missed when both edge vertices are outside the implicit surface. 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 [149].

A number of accelerations can be used in the interference detection process. For example, the 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 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 in, for example, [118, 189]. Again, we stress our interest in nonconvex objects and refer the reader to [137, 95] for other algorithms that treat arbitrary nonconvex polyhedral models. For more details on collision detection methods, see e.g. [182, 18, 70, 102, 143].

4.5 Time Integration

The equations for rigid body evolution are

$$x_t = v, \quad q_t = \tfrac{1}{2}\omega q \tag{4.1}$$

$$v_t = F/m, \quad L_t = \tau \tag{4.2}$$

where x and q are the position and orientation (a unit quaternion), v and ω are the velocity and angular velocity, F is the net force, m is the mass, $L = I\omega$ is the angular momentum with inertia tensor $I = RDR^T$ (R is the orientation matrix and D is the diagonal inertia tensor in object space), and τ is the net torque. For simplicity we will consider $F = mg$ and thus $v_t = g$ 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. [30], 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 4.1 and 4.2.

The standard approach is to integrate equations 4.1 and 4.2 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. [11] 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 [77] where the same algebraic equations were used for both and the work of [118] who integrated the same nonlinear differential equations for both. However, other authors such as [120, 155, 97] 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. [118] used the velocity an object at rest will obtain by falling through the collision envelope (or some threshold in [117]) 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 4.2.
- Contact resolution.
- Advance the positions using equation 4.1.

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. In a standard time stepping scheme, both position and velocity are updated first, followed by collision and contact resolution. 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. 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 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 4.6).

We repeated the experiment of a block sliding down an inclined plane from [118] using the methods for collision and contact proposed throughout this paper 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 4.3. Of course, these poor results are accentuated because we both set $\epsilon = 1$ and do not back up the simulation to the time of collision (which would be impractical and impossible for our large stacking examples). Figure 4.4 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 [118] even though we do not use a threshold velocity or their microcollision model.

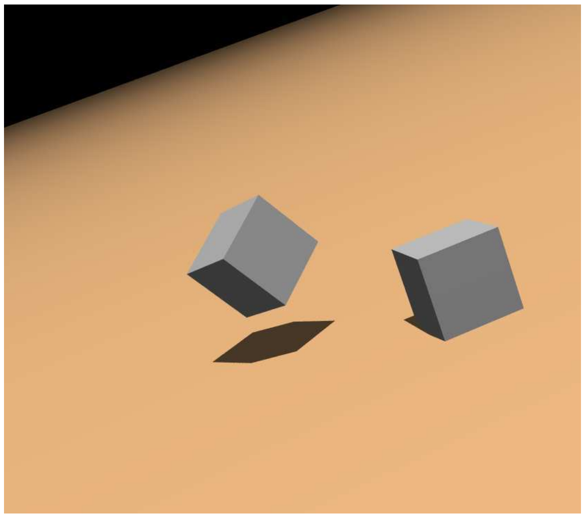

Figure 4.3: The block and inclined plane test with standard time integration (the block erroneously tumbling) and our new time integration sequencing (the block correctly at rest).

4.6 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 [157, 113]. 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.

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 x and v, we test for interference using the predicted position $x' = x + \Delta t(v + \Delta t g)$, and apply collision impulses to (and using) the current velocity v. During contact processing, we use the predicted position $x' = x + \Delta t v'$ and apply impulses to this new velocity v'. Since $v' = v + \Delta t g$ was set in the velocity update step, the candidate positions match and the interference checks are consistent.

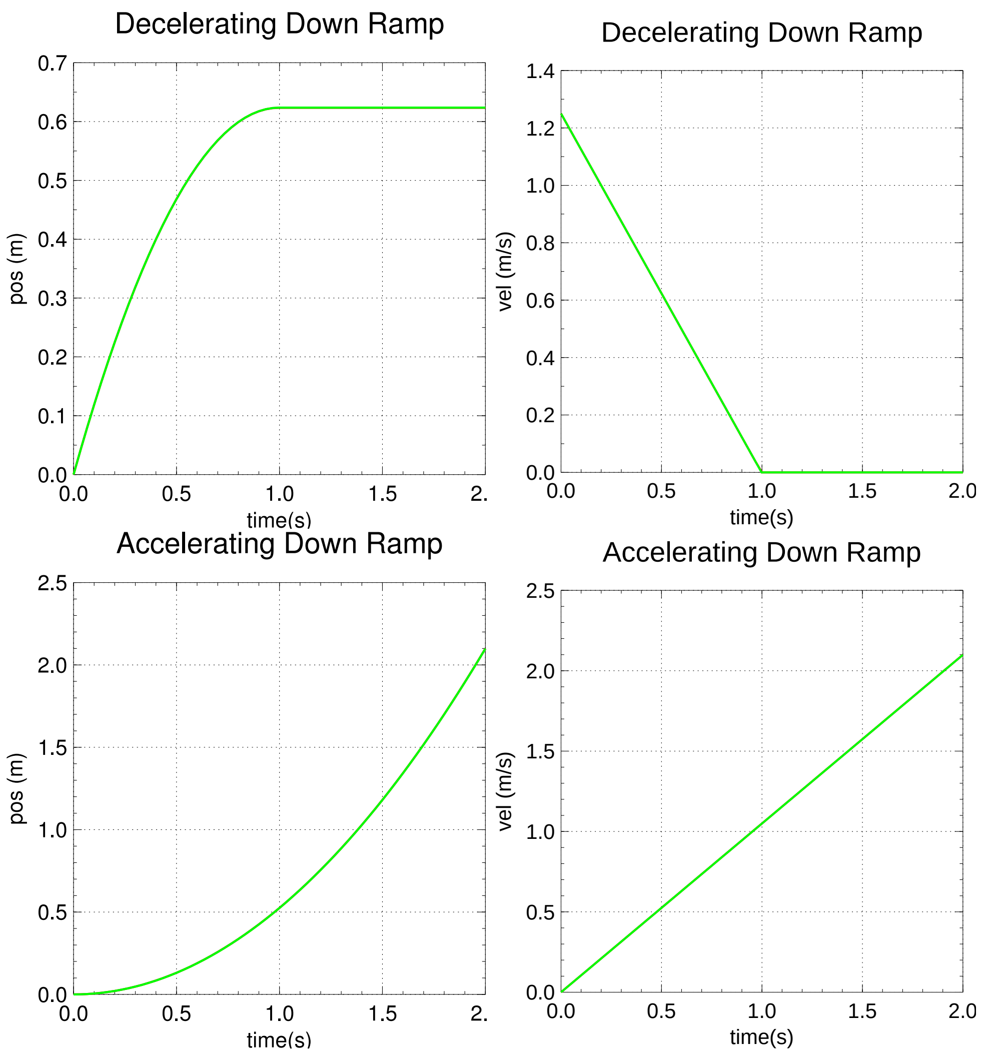

Figure 4.4: Theoretical and our numerical results for two tests of a block sliding down an inclined plane with friction. The curves lie on top of each other in the figures due to the accuracy of our new time sequencing algorithm.

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, 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 [120], 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 [79], it is much faster and scales well to large numbers of objects.

We developed an aggressive optimization for the point sampling that gives similarly plausible results. 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 [14] in a slightly different context.

Each body is assigned a coefficient of restitution, and when two bodies collide we use the minimum between the two coefficients as did [120] to process the collision. Suppose the relative velocity at the collision point was originally u_{rel} with (scalar) normal and (vector) tangential components, $u_{rel,n} = u_{rel} \cdot N$ and $u_{rel,t} = u_{rel} - u_{rel,n}N$ respectively. Then we apply equal and opposite impulses j to each body to obtain $v' = v \pm j/m$ and $\omega' = \omega \pm I^{-1}(r \times j)$ where r points from their respective centers of mass to the collision location. The new velocities at the point of collision will

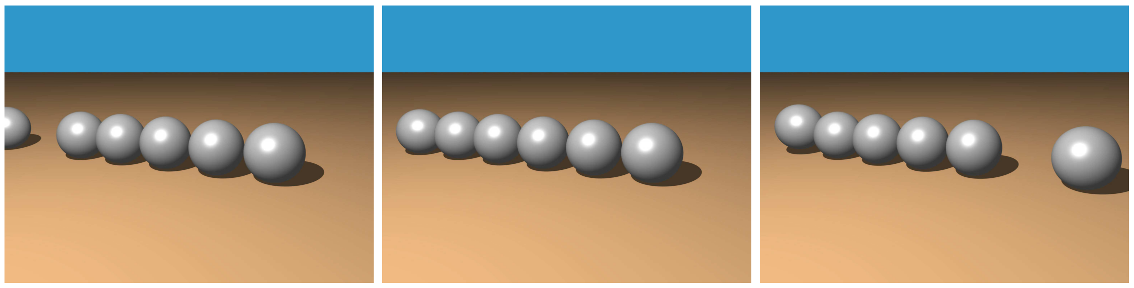

Figure 4.5: A billiard ball hits one end of a line of billiard balls, and the collision response propagates to the ball on the far right which then starts to roll.

be $u' = u \pm Kj$ where $K = \delta/m + r^{*T} I^{-1} r^*$ with δ the identity matrix and the "$*$" superscript indicating the cross-product matrix. Finally, $u'_{rel,n} = u_{rel,n} + N^T K_T N j_n$ where K_T is the sum of the individual K's and $j = j_n N$ is our frictionless impulse. So given a final relative normal velocity $u'_{rel,n} = -\epsilon u_{rel,n}$, we can find the impulse j. Immovable static objects like the ground plane can be treated by setting $K = 0$ and not updating their velocities.

4.7 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, and we use the maximum of the two possible coefficients when processing a collision as did [120]. Like [77, 120], 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 $u'_{rel,t} = 0$ so that $u'_{rel} = -\epsilon u_{rel,n} N$ allows us to solve $u'_{rel} = u_{rel} + K_T j$ for the impulse j by inverting the symmetric positive definite matrix K_T. Then if j is in the friction cone, i.e. if $|j - (j \cdot N)N| \leq \mu j \cdot N$, the point is sticking due to static friction and j is an acceptable impulse. Otherwise, we apply sliding friction.

Define $T = u_{rel,t}/|u_{rel,t}|$ so that the kinetic friction can be computed with the impulse $j = j_n N - \mu j_n T$. Then take the dot product of $u'_{rel} = u_{rel} + K_T j$ with N to obtain $u'_{rel,n} = u_{rel,n} + N^T K_T j$ or $-\epsilon u_{rel,n} = u_{rel,n} + N^T K_T j$. Plugging in the definition of j we can solve to find $j_n = -(\epsilon + 1)u_{rel,n}/(N^T K_T (N - \mu T))$ from which the kinetic friction impulse j is determined.

4.8 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 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. 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. This is a propagation model for contact as opposed to the simultaneous solution proposed in [9].

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. 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 [113] where an optimization based approach is taken. We propose a more light-weight method in section 4.8.2.

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 [9] 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 [77, 118] 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.

4.8.1 Contact graph

At the beginning of the contact resolution stage we construct a contact graph similar to [77, 7] with the intention of identifying which bodies or groups of bodies are resting on top of others. We individually allow each object to fall under the influence of gravity (keeping the others stationary) for a characteristic time $\triangle\tau$ (on the order of a time step), and record all resulting interferences adding a directed edge pointing toward the falling object from the other object. Then we apply a simple topological sort algorithm that uses two depth first searches to collapse strongly connected components resulting in a directed acyclic graph. 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. 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.

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

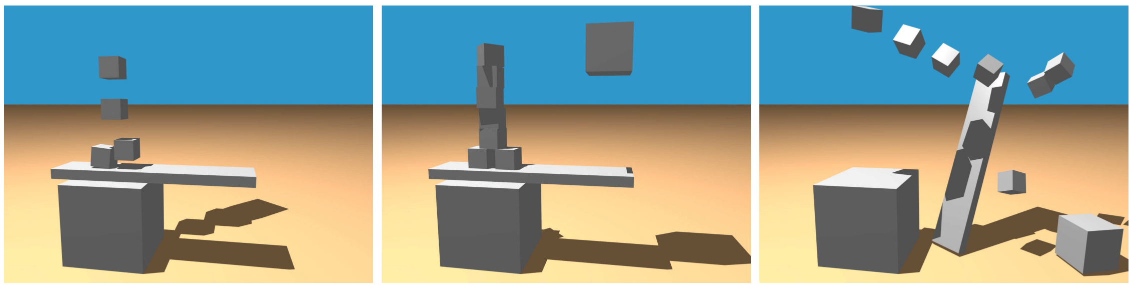

Figure 4.6: Although the propagation treatment of contact and collision allows the stacking and flipping of boxes as shown in the figure, our shock propagation algorithm makes this both efficient and visually accurate.

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.

4.8.2 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 4.6 (center), the cubes will start sinking into each other if not enough iterations are used. 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 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

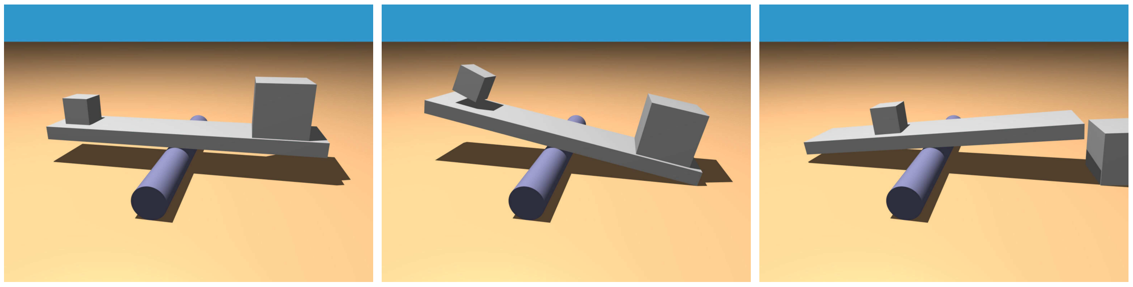

Figure 4.7: A heavier block on the right tips the see-saw in that direction, and subsequently slides off. Then the smaller block tips the see-saw back in the other direction. The propagation treatment of contact allows the weight of each block to be felt, and our shock propagation method keeps the blocks from interpenetrating without requiring a large number of contact processing iterations.

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 4.6 (center). Starting at the bottom of the stack, each object has its velocity set to zero 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.

In order to demonstrate why the propagation model for contact is used for a few iterations before applying shock propagation we drop a larger box onto the plank as shown in figure 4.6 (center). 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 as shown in figure 4.6 (center). 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 4.6 (right).

Figure 4.7 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, 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.

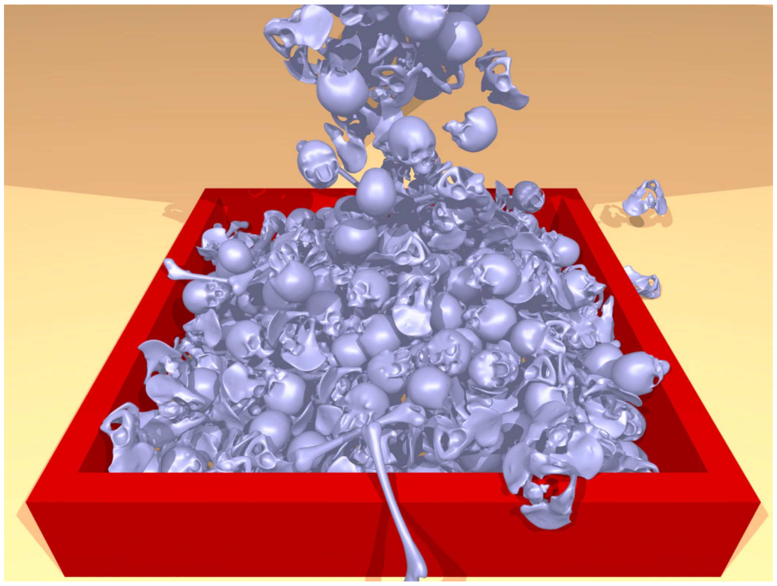
Figure 4.8: 500 nonconvex bones falling into a pile after passing through a funnel. Exact triangle counts are given in figure 4.2.

4.9 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. [100, 146] 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 ω_{rel} with normal and tangential components $\omega_{rel,n} = \omega_{rel} \cdot N$ and $\omega_{rel,t} = \omega_{rel} - \omega_{rel,n} N$. The normal component governs spinning and the tangential component governs rolling about $T = \omega_{rel,t}/|\omega_{rel,t}|$. 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 ω'_{rel}, and since the objects are sticking the relative velocity at the contact point is $u'_{rel} = 0$. Next, we construct an impulse to achieve both proposed velocities.

If we apply the impulse at a point, specifying the relative velocity determines the impulse j and we are unable to also specify the relative angular velocity. Thus, we assume that the impulse is applied over an area instead. We still have $v' = v \pm j/m$ for the center of mass velocity, but the angular velocity is treated differently. First we explicitly write out the change in angular momentum (about our fixed world origin) due to an angular impulse j_τ as $x \times mv' + I\omega' = x \times mv + I\omega \pm j_\tau$ which can be rearranged to give $\omega' = \omega \pm I^{-1}(j_\tau - x \times j)$. When j_τ is equal to the cross product of the point of contact and j, this reduces to $\omega' = \omega \pm I^{-1}(r \times j)$ as above, but here we consider the more general case in order to get control over the angular velocity. The new velocities at the point of collision will be $u' = u \pm (K_1 j + K_2 j_\tau)$ where $K_1 = \delta/m + r^* I^{-1} x^*$ and $K_2 = -r^* I^{-1}$. Then $u'_{rel} = u_{rel} + K_{1,T} j + K_{2,T} j_\tau$ where the $K_{i,T}$'s are the sum of the individual K_i's. Similar manipulation gives $\omega'_{rel} = \omega_{rel} + K_{3,T} j + K_{4,T} j_\tau$, where $K_3 = -I^{-1} x^*$ and $K_4 = I^{-1}$. This is two equations in two unknowns, j and j_τ, so we can solve one equation for j, plug it into the other, and solve for j_τ. This requires inverting two 3×3 matrices.

4.10 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 4.1, 4.8 and 4.9 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.

4.11 Conclusions and Future Work

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.

After the positions have been updated, interpenetration may still occur due to round-off errors and in-level contact between objects. We experimented with the use of first order physics (similar to [14]) to compute a "first order impulse" to apply 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

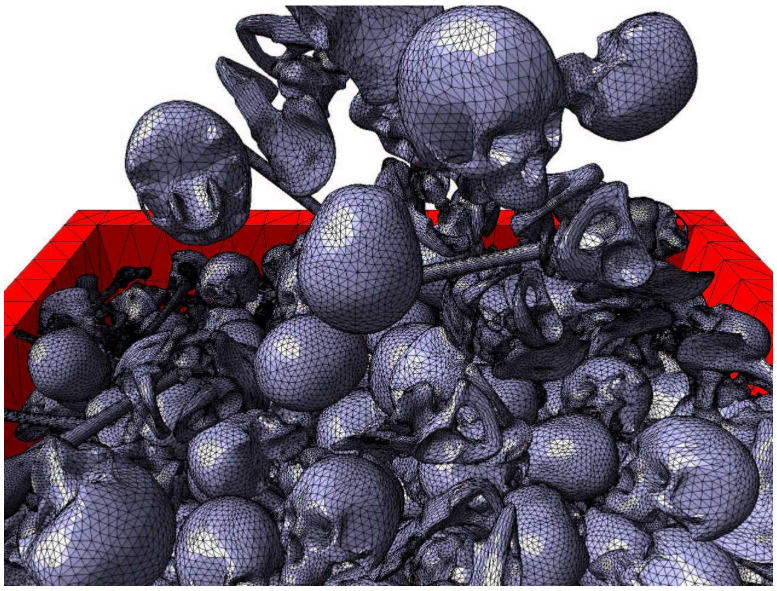

Figure 4.9: The complexity of our nonconvex objects is emphasized by exposing their underlying tessellation.

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. We plan to investigate this further in future work.

Chapter 5

Mesh Generation

This chapter is concerned with the generation of tetrahedral meshes, and specifically how to deal with problems that arise at or near the surface when such volumetric meshes are deformed. The algorithm for creating meshes presented here (from our paper [119]) dynamically deforms the surface of a semi-structured candidate mesh to match specified geometry, with the interior of the mesh dragged along automatically. Thus by necessity we discovered and fixed specific problems that occur in large surface deformations, and thus argue that our technique is perhaps the best technique for generating meshes for simulations of large deformations. Other meshing algorithms make no guarantees about the suitability of their results for such simulations, and may produce initial meshes with good quality measures but with hidden deficiencies that cause poor accuracy or even element collapse in subsequent simulation.

Tetrahedral meshes are used in a number of areas including fracture [129, 128], haptics [48], solid modeling [44], surgical simulations [63], and the modeling of biological tissue including the brain [72], the knee [79] and fatty tissue [88]. We emphasize and concentrate on their use in simulations of highly deformable bodies in graphics, biomechanics, virtual surgery, etc. The most developed area of mesh generation, in contrast, is for static or barely deforming meshes used for fluid dynamics, heat flow, small deformation solid mechanics, electromagnetism, etc. The requirements in these domains are quite different. For example an optimal mesh for a fluid flow simulation should include anisotropically compressed elements in boundary layers, e.g. [64, 106], but stretched elements in deformable bodies tend to be ill-conditioned, bad for both accuracy and efficiency.

The objects that we want to mesh, mostly of biological origins, are also different from the geometry typically considered in the meshing literature. There, CAD packages provide exact geometry made of polygonal faces and/or spline surfaces with sharply defined edges and corners, which must be matched by the mesh. On the other hand, objects such as muscles, organs, bones, etc. are smooth and their geometry is not known exactly precisely (and maybe cannot be defined precisely in some sense, since it is constantly changing[78]): the mesh just needs to approximate this geometry reasonably.

We use level sets as the representation for input geometry, that is a function ϕ sampled on a regular grid giving signed distance to the surface. Motivated by crystallography, we use a body-centered cubic (BCC) background mesh (see e.g. [29]), adaptively refined with a red-green strategy, from which we carefully select a topologically-robust subset of tetrahedra that roughly match the level set. This candidate mesh is then "compressed" to match the level set boundary as closely as possible while maintaining element quality. See figure 5.1 for an example of this semi-structured approach.

Additional advantages of our method over existing three-dimensional meshing algorithms include the relative simplicity of implementation—it is a fully robust method that doesn't require lots of special cases, careful handling of round-off error or the use of exact arithmetic, complex acceleration structures, etc.—and the fact that it generates semi-structured meshes. The underlying grid structure can be an advantage when constructing preconditioners for iterative solvers, writing multigrid algorithms, saving storage and increasing performance by reducing pointer indirection with implicit indexing, etc.

5.1 Related Work

Delaunay methods have not been as successful in three spatial dimensions as in two, since they admit flat sliver tetrahedra of negligible volume. Shewchuk[153] provides a nice overview of these methods, including a discussion of why some of the theoretical results are not reassuring in practice, but discusses how the worst slivers can often be removed. Sliver removal is also discussed in [37], but the theorem presented there gives a "miserably tiny" estimate of quality.

Advancing front methods start with a boundary discretization and march a "front" inward forming new elements attached to the existing ones, see e.g. [148]. While they conform well to the boundary, they have difficulty when fronts merge, which unfortunately can occur very near the important boundary in regions of high curvature, see e.g. [64, 106]. In [141], the authors start with a face-centered cubic (FCC) lattice defined on an octree and use an advancing front approach to march inward from a surface mesh, constructing a Delaunay mesh with the predetermined nodes of the FCC lattice. Their motivation for choosing FCC over BCC structure was a slight improvement in a specific finite element error bound; we believe that the more regular connectivity of the BCC lattice is preferable for deformation, since those bounds will be changed significantly then anyhow (especially since they discard the original FCC tetrahedra when reconnecting the deformed nodes)

Shimada and Gossard[154] packed spheres (or ellipsoids [188]) into the domain with mutual attraction and repulsion forces, and generated tetrahedra using the sphere centers as sample points via either a Delaunay or advancing front method. However, *ad hoc* addition and deletion of spheres is required in a search for a steady state, and both local minima and "popping" can be problematic.

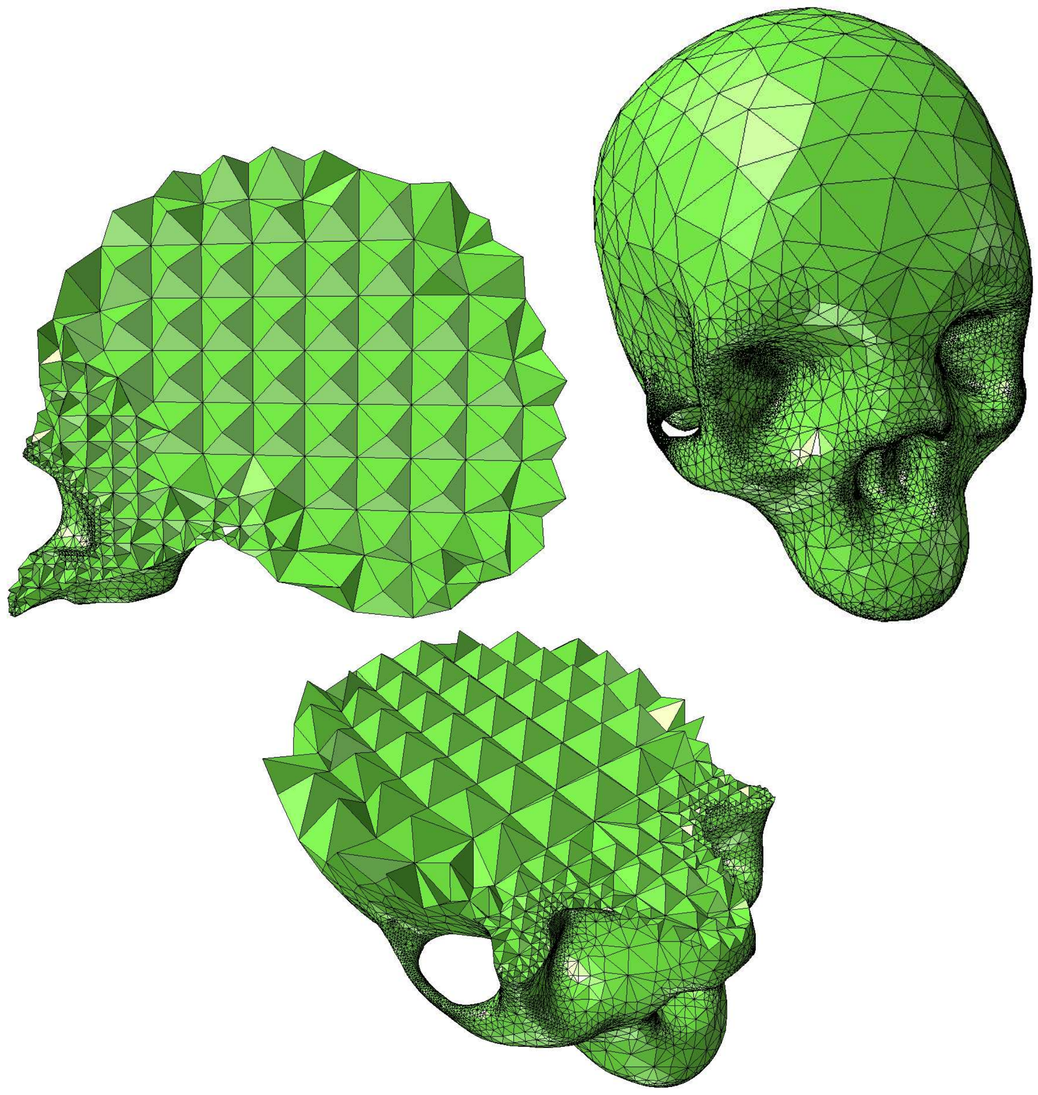

Figure 5.1: Tetrahedral mesh of a cranium (100K elements) with cutaway views showing element regularity and gradation to very coarse tetrahedra in the interior.

This led [101] to propose the removal of the dynamics from the packing process, instead marching in from the boundary removing spherical "bites" of volume one at a time, which gets back to the problems of marching front algorithms.

Cutler et al.[44] start with either a uniform grid or a graded octree mesh and subdivide each cell into five tetrahedra. Tetrahedra that cross an internal or external interface are simply split, possibly forming poorly shaped elements for simulation. Neighboring tetrahedra are also split to avoid T-junctions. The usual tetrahedral flips and edge collapses are carried out in an attempt to repair the ill-conditioned mesh.

Adaptive three-dimensional meshes have been problematic. Debunne et al.[48] carried out finite element simulations on a hierarchy of tetrahedral meshes switching to the finer meshes only locally where more detail is needed. They state that one motivation for choosing this strategy is the difficulty in preserving mesh quality when subdividing tetrahedral meshes. Grinspun et al.[72] proposed a basis refinement viewpoint, partly as a way to avoid having to resolve T-junctions in adaptive three-dimensional meshes. Our red-green method alleviates both of these concerns: it provides high quality elements (in fact, when our BCC mesh is subdivided, we get exactly the BCC mesh with half the edge length) and is straightforward to implement.

Our compression phase moves the nodes on the boundary of our candidate mesh to the implicit surface providing boundary conformity. Related work includes [23] which attracted particles to implicit surfaces for visualization, [172, 173] which used particles and repulsion forces to sample and retriangulate surfaces, [47, 42] which used the same idea to triangulate implicit surfaces, [162] which modeled surfaces with oriented particles, and [186] which used particle repulsion to sample (and control) implicit surfaces.

This wrapping of our boundary around the level set is related to snakes [93] or GDM's [114] which have been used to triangulate isosurfaces, see e.g. [145]. [125] started with a marching cubes mesh and moved vertices to the centroid of their neighbors before projecting them onto the zero level set in the neighboring triangles' average normal direction. [74] improved this method using internodal springs instead of projection to the centroid, incremental projection to the zero isocontour, adaptive subdivision, edge collapse and edge swapping. [96] used related ideas to wrap a mesh with subdivision connectivity around an arbitrary one (other interesting earlier work includes [82, 81]), but had difficulty projecting nodes in one step, emphasizing the need for slower evolution. Similar techniques were used by [20, 83] to wrap subdivision surfaces around implicit surfaces. To improve robustness, [187] adaptively sampled the implicit surface on the faces of triangles, redistributed the isovalues times the triangle normals to the vertices to obtain forces, and replaced the spring forces with a modified Laplacian smoothing restricted to the tangential direction. [130] advocated moving the triangle centroids to the zero isocontour instead of the nodes, and matching the triangle normals with the implicit surface normals.

Although we derive motivation from this work, we note that our problem is significantly more difficult since these authors move their mesh in a direction normal to the surface, which is orthogonal to their measure of mesh quality (shapes of triangles tangent to the surface). When we move our mesh normal to the surface, it directly conflicts with the quality of the surface tetrahedra. In two dimensions, [68] did move the mesh in a direction that opposed the element quality. They started with a uniform Cartesian grid bisected into triangles, threw out elements that intersected or were outside the domain, and moved nodes to the boundary in the direction of the gradient of the level set function using traditional smoothing, edge-swapping, insertion and deletion techniques. In three dimensions, [47] evolved a volumetric mass spring system in order to align it with (but not compress it to) the zero isocontour, but the measure of mesh quality was again perpendicular to the evolution direction since the goal was to triangulate the zero isocontour. Later, however, [175] proposed the basic idea behind our algorithm, pushing in a direction conflicting with mesh quality. They deformed a uniform-resolution Freudenthal lattice to obtain tetrahedralizations using a mass spring model, but were restricted to simple geometries mostly because they did not deal with the topological problems for deformation that we identify and fix here.

5.2 The BCC Background Mesh

Our algorithm begins with a background tetrahedral mesh tiling space. Unlike in two dimensions, it is impossible to tile three-dimensional space with equilateral simplices. Of the several methods for tiling space (see e.g. [29, 33]) the body-centered cubic (BCC) lattice structure is one of the most regular. This structure is fundamental in nature, forming the basis of crystals such as iron and lithium[29]. It consists of two interlaced cubic grids, the nodes of one located at the centers of the cells of the other, with additional edges between a node and its eight closest neighbors in the other grid. Each tetrahedron is formed from two adjacent nodes of one grid and two of the nearest adjacent nodes in the other grid (see figure 5.2).

Among the desirable properties of this mesh, one can list:

- all tetrahedra are congruent to each other
- all faces of all tetrahedra are congruent to each other
- all edges are of two lengths, only differing by a factor of $\sqrt{3/4} \approx 0.866$
- the faces are isosceles, with two angles $\approx 54.7°$ and one $\approx 70.5°$
- the dihedral angles are $60°$ and $90°$
- it is Delaunay
- the node locations are optimal for sampling in the sense that their Fourier transform is a (FCC) close-packing of spheres[167]
- the maximum degree (and only degree) of a node is just 14

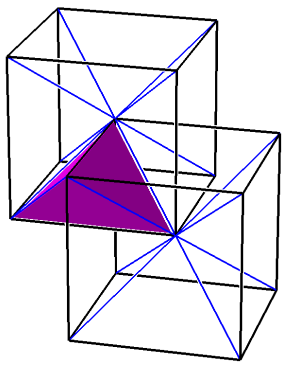

Figure 5.2: A portion of the BCC lattice with a tetrahedron highlighted. The two black cubic grids are interlaced together with the blue edges.

- it is homogeneous and isotropic in the sense that the mesh is invariant under translations between two nodes and rotations of multiples of 90° around the grid axes

For simulation these properties give rise to accurate force calculations and well-conditioned, highly-structured matrices to be solved, and avoid error artifacts such as preferred directions (anisotropy) or "checkerboarding" (alternate grid nodes behaving differently).

In addition, as discussed in the next section, the tetrahedra of the BCC lattice can be regularly refined, unlike most meshes. If each tetrahedron is split 1:8 into eight smaller tetrahedra the correct way, the resulting mesh is a BCC mesh of twice the resolution. In fact, if 1:8 subdivision is used on an arbitrary tetrahedral mesh, in the limit almost every part of the mesh will have the connectivity of the BCC mesh (but not necessarily its regular geometry).

5.3 Red-Green Refinement

Many applications do not require and cannot afford (due to computation time and memory restrictions) a uniformly high resolution mesh. For example, many graphical simulations can tolerate low accuracy in the unseen interior of a body, and many phenomena such as contact and fracture

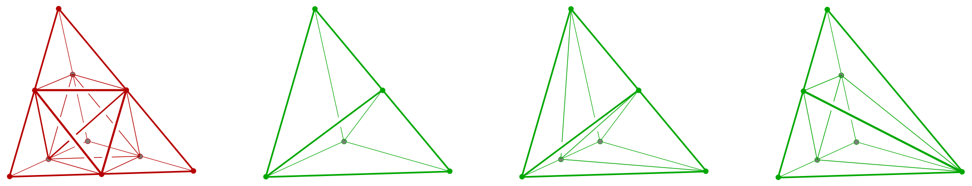

Figure 5.3: The standard red refinement (depicted in red, far left) produces eight children that reside on a BCC lattice one half the size. Three classes of green refinement are allowed (depicted in green).

show highly concentrated stress patterns, often near high surface curvature, outside of which larger tetrahedra are acceptable. Thus, we require the ability to generate adaptive meshes. A significant advantage of the BCC mesh is that it is easily refined in a regular way, initially or during the calculation. Each regular BCC tetrahedron can be refined into eight tetrahedra, shown in red in figure 5.3, with a 1:8 refinement. When the shortest of the three possible choices for the edge internal to the tetrahedron is taken, the newly formed tetrahedra are *exactly* the BCC tetrahedra that result from a mesh with cells one half the size, guaranteeing element quality.

As the BCC lattice is built from cubic grids, one natural approach to adaptivity is to build its analog based on an octree. We implemented this by adding body centers to the octree leaves, after ensuring the octree was graded with no face- or edge-adjacent cells differing by more than one level. The resulting BCC lattices at different scales were then patched together with special case tetrahedra. For more on octrees in mesh generation, see e.g. [152, 141].

However, we found that red-green refinement (see e.g. [21, 75, 46]) is more economical, simpler to implement, more flexible, and gives higher quality elements. The initial BCC lattice tetrahedra are labeled red, as are any children obtained with 1:8 subdivision shown in red in figure 5.3. Performing a red refinement on a tetrahedron creates inconsistent T-junctions at the newly-created edge midpoints where neighboring tetrahedra are not refined to the same level. To eliminate these, the red tetrahedra with T-junctions are irregularly refined into fewer than eight children after introducing some of their midpoints. These children are labeled green. They are of lower quality than the red tetrahedra, and thus we never refine them further: to do so would ultimately create slivers. When higher resolution is desired in a region occupied by a green tetrahedron, the entire family of green tetrahedra is removed from its red parent, and the red parent is refined regularly to obtain eight red children that can then undergo subsequent refinement.

A red tetrahedron that needs a green refinement can have between one and five midpoints on its edges (in the case of all six midpoints we use red refinement). We reduce the possibilities for green refinement to those shown in figure 5.3, adding extra edge midpoints if necessary. Note that adding these extra midpoints may in fact result in red refinement. This restriction (where all triangles are either bisected or quadrisected) smooths the gradation further and guarantees higher quality green tetrahedra, as shown in figure 5.4. While there can of course be a cascading effect as the extra midpoints may induce more red or green refinements, it is a small price to pay for the superior mesh quality and seems to be a minor issue in practice.

A significant advantage of the red-green framework is the possibility for refinement during simulation based on *a posteriori* error estimates, with superior quality guarantees derived from the BCC lattice instead of an arbitrary initial mesh. The lower quality green tetrahedra can be replaced by finer red tetrahedra which admit further refinement, if we maintain the red-green structure from the initial BCC mesh.

Any criteria may be used to drive refinement. In this exploratory work we investigated using

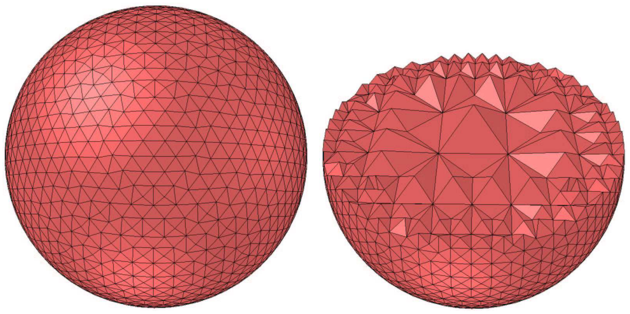

Figure 5.4: Tetrahedral mesh of a sphere (18K elements). The cutaway view illustrates that the interior mesh can be fairly coarse even if high resolution is desired on the boundary.

surface curvature: whenever a tetrahedron that intersects the level set surface is of a length scale much larger than the radius of curvature of the level set, we refine. The principal curvatures of a level set are easy to evaluate at any point in space with finite differences[3]. Note that the mean curvature (the average of the principal curvatures and the simplest quantity to compute) is not sufficient, as it may be close to zero near saddle points of the surface even when the principal curvatures are very large but of opposite sign. Figure 5.5 illustrates how the sum of the absolute values of the principal curvatures may be used to drive *a priori* refinement, with the inner ring of the torus refined more even though it consists of saddle points with low mean curvature.

Many of our examples used noisy data, for which a second derivative quantity such as curvature may be unreliable. We used the power of the level set framework to smooth the rough geometry with motion by mean curvature (see e.g. [131]), but remaining spurious small-scale features may still baffle a curvature-driven refinement scheme. Thus in these cases we instead used an error metric similar to [60, 136], where we compare the linear interpolation of the level set function ϕ from the

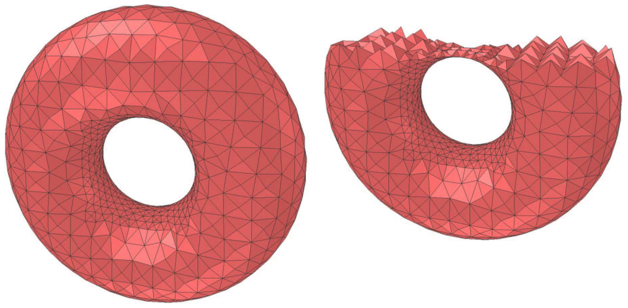

Figure 5.5: Tetrahedral mesh of a torus (8.5K elements). The principal curvatures were used to increase the level of resolution in the inner ring.

vertices of a tetrahedron that intersects the surface to the actual values on the grid, and refine only when there is a large discrepancy.

5.4 Selecting a Topologically Robust Sub-Mesh

In the previous sections we described the BCC lattice and a red-green refinement strategy. This section explains how to use these to generate a rough candidate mesh for the object (as a subset of a background mesh) that is ready for the final "compression" stage which will match the geometry of the level set without changing the connectivity of the mesh.

The first step is to cover an appropriately sized bounding box of the object with a coarse BCC mesh. Then we use a conservative discard process to remove tetrahedra that are guaranteed to lie completely outside of the zero isocontour: tetrahedra with four positive ϕ values all larger than the maximum edge length are removed, since if ϕ is signed distance to the surface, such a tetrahedron must be completely outside the object.

In the next step, the remaining tetrahedra are refined according to any user defined criteria, such as *a posteriori* error estimates, indicator variables or geometric properties. The previous section discussed some of the simple geometric refinement schemes we investigated. We limit refinement to a user-specified number of levels.

From the adaptively refined lattice we will select a subset of tetrahedra that closely matches the object. However, there are specific topological requirements necessary to ensure a valid mesh that behaves well under deformation:

- the boundary of the subset must be a manifold
- no tetrahedron may have all four nodes on the boundary
- no interior edge may connect two boundary nodes.

The first requirement is clear—if the boundary mesh of the tetrahedra is topologically not a manifold, it cannot possibly be matched to the level set manifold. The other two requirements deal with subsequent deformation.

A useful intuition for understanding discretization in continuum mechanics is that instead of simply approximating a PDE with numerical calculations, we actually are doing experiments with a *real* continuum that happens to be subject to a number of constraints which reduce the allowed deformations to a finite dimensional subspace. These constraints make the material stiffer. This is well known in finite element analysis, leading to rules of thumb such as the use of lower order accurate quadrature formulas that actually lead to smaller errors since they underestimate, lowering the effective stiffness, and thus partially cancel out the artificial extra stiffness due to discretization[160]. A more extreme example is the phenomenon of locking, where if care is not taken in the finite element discretization of incompressible materials, the constraints may interact to artificially eliminate *all* deformations.

The topology, i.e. connectivity, of the mesh defines the character of the constraints implied by discretization. If either of the latter two conditions above were violated in the mesh, there are serious consequences for the possible surface deformations of the mesh. If a tetrahedron has all four nodes on the boundary, then it is impossible (without crushing or inverting the tetrahedron) to flatten or indent that part of the surface. If an interior edge of the mesh connects two boundary nodes, then it is similarly impossible to flatten or indent the surface between those nodes (without crushing or inverting the tetrahedra that lie in between). Thus any mesh generation algorithm that does not take care of these cases may produce meshes that simply cannot support interesting surface deformations.[1]

To satisfy the three conditions on the subset we choose a set of "enveloped" nodes from the background mesh and select all tetrahedra incident on these. This guarantees we will satisfy the second condition, since every tetrahedron will be incident on at least one of these interior nodes. The enveloped nodes are chosen to be those where $\phi < 0$ (i.e. nodes that are inside the object) that have all their incident edges at least 25% inside the zero isocontour as determined by linear interpolation of ϕ along the edge. In this manner, we expect the isocontour to cut roughly halfway through the surface layer of selected tetrahedra, giving us a good approximation to the object when we begin compression.

In reasonably convex regions, i.e. regions where the geometry is adequately resolved by the nodal samples, this strategy also tends to satisfy the other two requirements. However, we need additional processing to guarantee them. In a first pass, all edges incident on non-manifold nodes and all bad interior edges are bisected, and the red-green structure is updated to further refine the nearby tetrahedra. If any edges were bisected, we recalculate the set of enveloped nodes. In subsequent passes, we add to the enveloped set any non-manifold node and one node from every bad interior edge (the node with the smaller ϕ value), stopping once all problems have been resolved. Optionally we may also add any nodes with surface degree equal to three to the set of enveloped nodes, since if these nodes were to remain on the surface the final surface mesh would typically contain obtuse angles over 120°. Typically at most two passes are required, after which we have a candidate mesh of high quality elements that approximates the level set object fairly well, and that has connectivity suitable for large deformation simulations. See figure 5.6 for example. All that remains is to better match the surface of the mesh to the level set without degrading element quality too much.

[1]There is in fact one additional degeneracy that may occur in theory, an interior triangle that connects three boundary nodes (which would prohibit some extraordinarily severe deformations). Fortunately this is very rare in practice, and would indicate that the mesh is woefully under-resolved. Larger problems with simple numerical accuracy, demanding refinement thus eliminating the degeneracy, would be apparent before the topological constraint comes into effect. Therefore we do not explicitly check for this case.

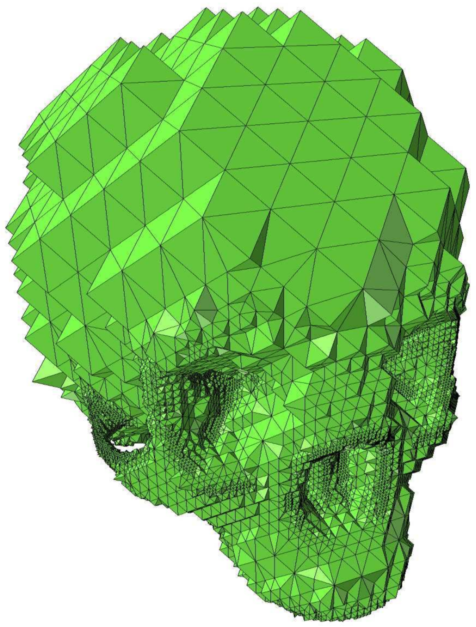

Figure 5.6: The initial candidate mesh for the cranium (a subset of the tetrahedra of the refined background mesh) before the compression phase deforms the surface to match the level set.

5.5 Compressing the Surface Mesh to the Level Set

In [119] we experimented with three techniques for compressing the surface of the candidate mesh to the level set, adjusting interior nodes to maintain quality elements. The first two methods were based in physics, actually simulating the evolution of the mesh as if it were an elastic object subject to boundary forces which ensure conformity to the level set. I will briefly describe these before concentrating on currently the most useful third technique based on direct optimization of element quality.

In the physics-based methods we apply boundary forces or set boundary velocity constraints, in the normal direction, proportional to the value of $-\phi$. That is, the further from the zero isocontour a surface node is, the more it is pushed toward the level set, coming to stable equilibrium on the level set. There are no forces or constraints in the tangential direction, allowing more freedom to maintain high quality elements. It is important to compute the normal direction based on the mesh (e.g. as an average of triangle normals), not on the level set: this way the motion tends to smooth out jagged parts of the mesh, not fold them over onto the level set collapsing elements in the process.

The two physics-based methods differ in how internal forces in the mesh are calculated. These forces resist distortion of elements, so the equilibrium state maintains high quality tetrahedra. The first method uses a mass-spring network containing both edge springs and altitude springs. The use of springs in mesh smoothing dates back to at least [69] as a generalization of Laplacian smoothing (iteratively moving nodes to the centroid of their neighbors). They are a substantial improvement over Laplacian smoothing, since they can repel concentrations of nodes away from each other as pointed out in [25]. The addition of altitude springs greatly improves element quality, since these more directly resist element collapse, as discussed in [41].

The second, more promising, physics-based method uses a full finite element discretization of the equations of elasticity to compute internal forces. Though a little more complicated to implement (possibly explaining why finite elements apparently haven't been used before in mesh generation), they are more robust and give rise to higher quality meshes. They are a more natural way of describing and resisting three-dimensional distortions of elements, and allow for interesting generalizations. For example, one idea for future work is to use the actual constitutive model of the material to be simulated in the mesh generation phase, or perhaps some sort of adjoint or dual to it, in an effort to get a mesh optimally suited for the simulation.

Currently the most useful technique (in terms of speed and mesh quality) is a direct optimization of element quality. Freitag and Ollivier-Gooch[59] demonstrated that optimizing node positions in a smoothing sweep, i.e. considering one node at a time and placing it at a location that maximizes the quality of incident elements, is far superior to Laplacian smoothing in three spatial dimensions. We combine this optimization sweeping with boundary constraints by first moving boundary nodes in the incident triangles' average normal direction by an amount proportional to the local signed distance value. Then the optimization is constrained to only move boundary nodes in the tangential direction. It is important to move boundary nodes gradually over several sweeps just as with physical simulation, since otherwise the optimization gets stuck in local extrema. We also found it helpful to order the nodes in the sweep with the boundary nodes first, their interior neighbors next, and so on into the interior. Then we sweep in the reverse order and repeat. This efficiently transfers information from the boundary compression to the rest of the mesh. Typically we do five sweeps of moving the boundary nodes 1/3 of the signed distance in the mesh normal direction, then finish off with five to ten sweeps moving boundary nodes the full signed distance to ensure a tight boundary fit. To speed up the sweeps, we do not bother moving nodes that are incident on tetrahedra of sufficiently high quality relative to the worst tetrahedron currently in the mesh. In the initial sweeps we end up only optimizing roughly 10% of the nodes, and in the final sweeps we optimize 30%-50% of the nodes.

While more efficient gradient methods may be used for the nodal optimization, we found a simple pattern search (see e.g. [169]) to be attractive for its robustness, simplicity of implementation, and flexibility in easily accommodating any quality metric. For interior nodes we used seven fairly well spread-out directions in the pattern search (see table 5.1). We implemented the normal direction constraint on boundary nodes simply by choosing five equispaced pattern directions orthogonal to the average mesh normal at the node. The initial step size of the pattern search was 0.05 times the minimum distance to the opposite triangle in any tetrahedron incident on the node (so we would be sure that we wouldn't waste time on steps that crush elements). After four "strikes" (searches at a

x	y	z
1,	0,	0
0.2188,	0.9758,	0
0.2188,	-0.5972,	-0.7717
-0.0541,	0.2364,	0.9701
-0.9017,	-0.3457,	0.2596
0.2186,	-0.8664,	0.4489
-0.5882,	0.3562,	-0.7260

Table 5.1: The vectors used for pattern search in optimizing the internal nodes of the mesh.

given step size that yielded no improvement in quality, causing the step size to be halved) we give up on optimizing the node. For interior nodes we used as a quality metric the minimum of

$$\frac{a}{L} + \frac{1}{4}\cos(\theta_M)$$

over the incident tetrahedra, where a is the minimum altitude length, L is the maximum edge length, and θ_M is the maximum angle between face normals. This is a simple combination of the inverse aspect ratio and the minimum dihedral angle: the larger it is, the better. For surface nodes we added to this quality metric a measure of the quality of the incident boundary triangles, the minimum of

$$\frac{a_t}{L_t} + \frac{1}{\psi_M}$$

where a_t is the minimum triangle altitude, L_t is the maximum triangle edge, and ψ_M is the maximum triangle angle. We found that including the extra terms beyond the tetrahedron aspect ratios helped guide the optimization out of local minima and actually resulted in better aspect ratios.

5.6 Results

Meshes generated using this algorithm have been successfully used to simulate highly deformable volumetric objects with a variety of constitutive models, e.g. the muscles in [163], as well as two dimensional shells [28] (the "cloth" Buddha earlier in this thesis was a surface mesh taken from an adaptive tetrahedral mesh with additional smoothing). Included here are three example meshes, figures 5.1 (the cranium), 5.7 (the dragon), and 5.8 (the Buddha) in order of increasing complexity.

We track a number of quality measures including the maximum aspect ratio (defined as the tetrahedron's maximum edge length divided by its minimum altitude), minimum dihedral angle, and maximum dihedral angle. The worst aspect ratios of our candidate mesh start at about 3.5 regardless of the degree of adaptivity, emphasizing the desirability of our combined red-green adaptive BCC approach. This number is reached by the green tetrahedra (the red tetrahedra have aspect ratios of $\sqrt{2}$). In the more complicated models, the worst aspect ratio in the mesh tends to increase to

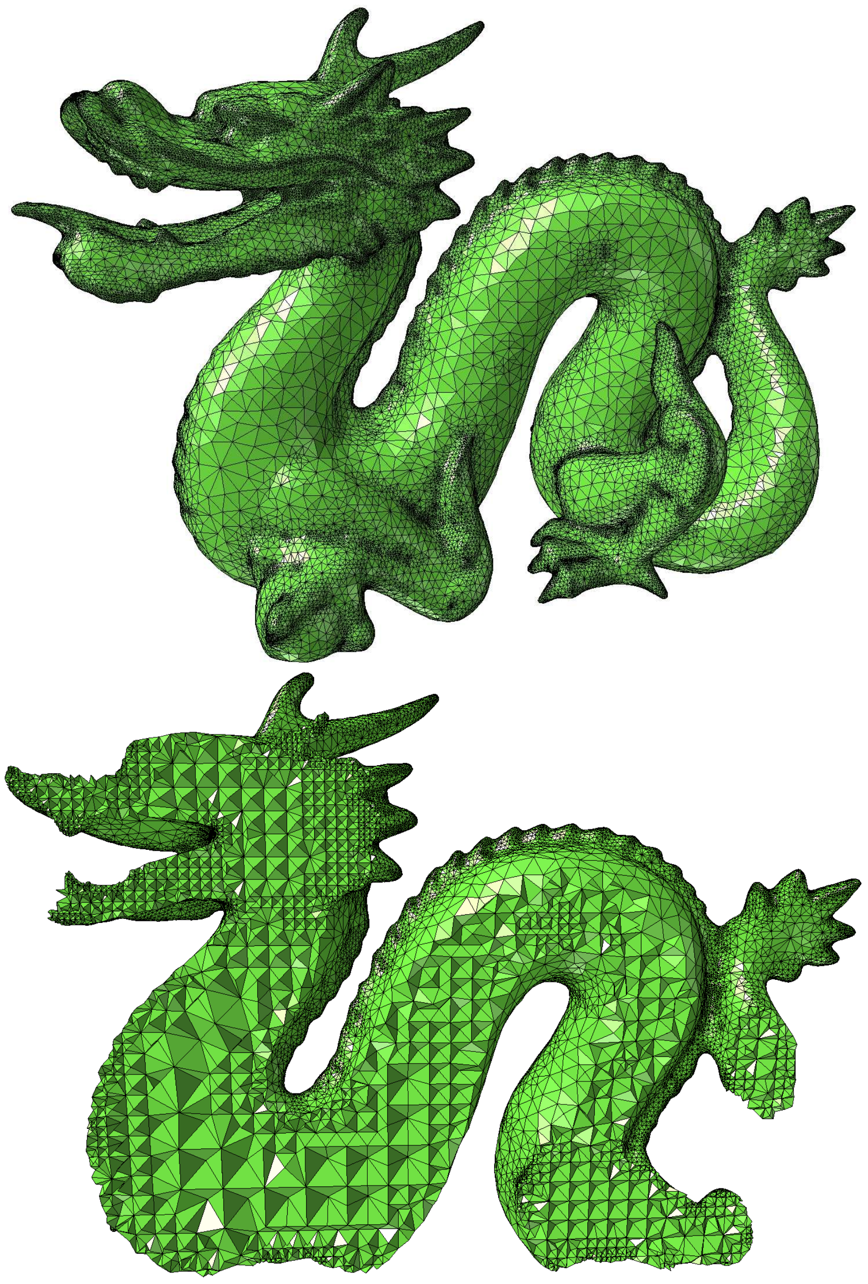

Figure 5.7: Tetrahedral mesh of a model dragon (500K elements). The final compression stage can be carried out with masses and springs, finite elements, or an optimization based method.

Optimization results			
Example	Cranium	Dragon	Buddha
Max. aspect ratio	4.5	5.3	5.9
Avg. aspect ratio	2.3	2.3	2.3
Min. dihedral	18°	16°	16°
Max. dihedral	145°	150°	150°
Max./min. edge	94	94	100

Finite element results			
Example	Cranium	Dragon	Buddha
Max. aspect ratio	6.5	7.6	8.1
Avg. aspect ratio	2.1	2.2	2.3
Min. dihedral	17°	13°	13°
Max. dihedral	147°	154°	156°
Max./min. edge	94	94	100

Table 5.2: Quality measures for the example meshes. The aspect ratio of a tetrahedron is defined as the longest edge over the shortest altitude. The max/min edge length ratio indicates the degree of adaptivity.

around 6–8 for the physics based compression methods and to around 5–6 for the optimization based compression. Dihedral angles typically range between 16° and 150° for optimization, but slightly worse for physics-based methods.

Finite element compression takes slightly longer (ranging from a few minutes to a few hours on the largest meshes) than the mass-spring method, but produces a slightly higher quality mesh. While mass-spring networks have a long tradition in mesh generation, the finite element approach offers greater flexibility and robustness that we anticipate will allow better three-dimensional mesh generation in the future. However, the best method is currently the optimization based compression, faster roughly by a factor of ten and producing higher quality meshes. Quality metrics are given in table 5.2, along with an indication of the degree of adaptivity: the smallest elements in the meshes were roughly 100 times smaller than the largest elements. Note that for optimization, the aspect ratios are all below 6, i.e. less than twice the initial values from the red-green hierarchy.

5.7 Generalizations

The general theme of our algorithm—tile ambient space as regularly as possible, select a subset of elements that are nicely connected and roughly conform to the object, then deform them to match the boundary—is applicable in any dimension and on general manifolds. Figure 5.9 shows the result of triangulating a two-dimensional manifold with boundary. We began with a surface mesh of the dragon actually created as the boundary of a tetrahedral mesh from our method (with additional edge-swapping and smoothing), and a level set indicating areas of the surface to trim away. We kept a subset of the surface triangles inside the trimming level set and compressed to the trimming

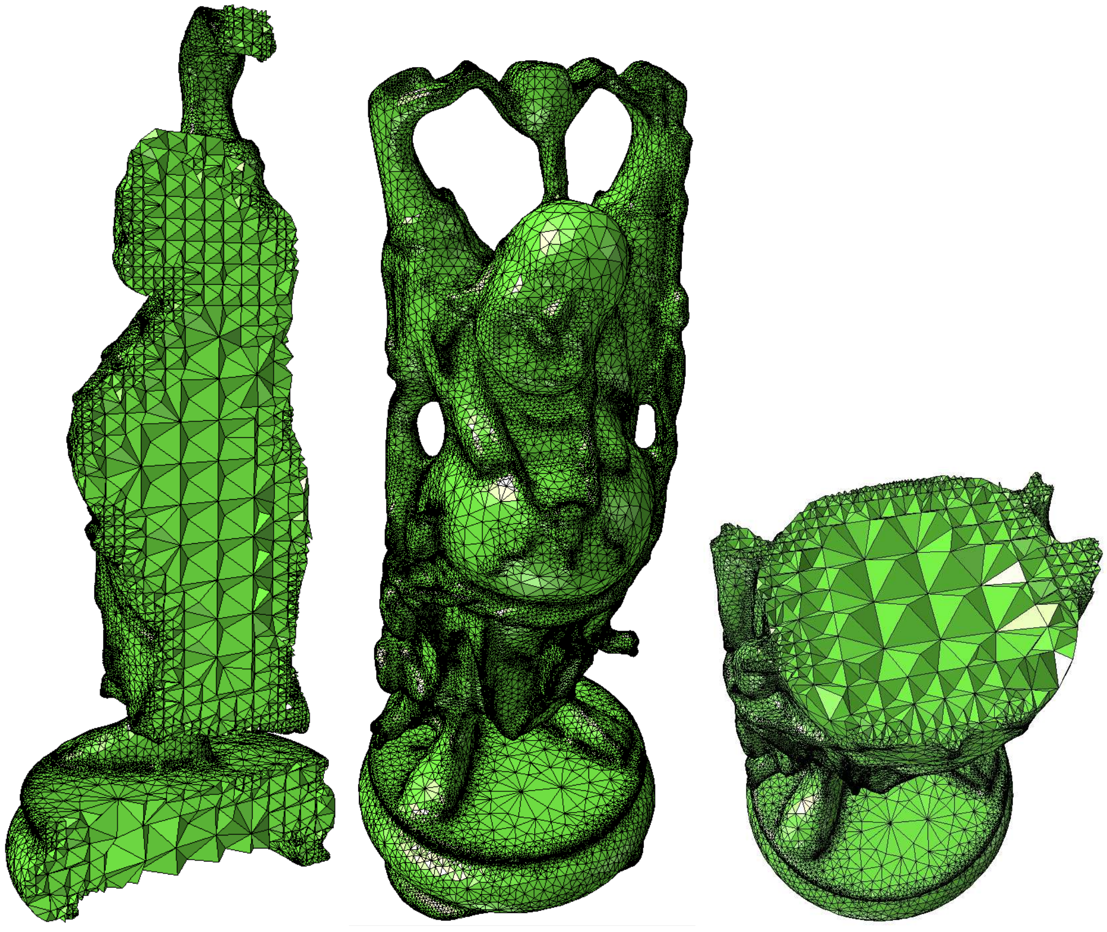

Figure 5.8: Tetrahedral mesh of a model Buddha (900K elements), adaptively refined to resolve features on multiple scales.

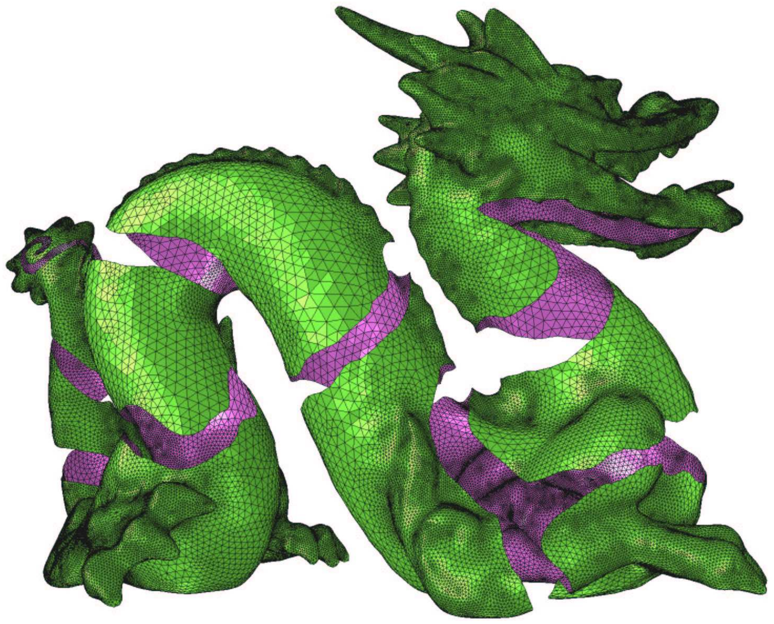

Figure 5.9: Our method is easily extended to create a triangle mesh of a manifold with boundary. The peeled away regions of the dragon were modeled with a second level set.

boundary, making sure to stay on the surface. Note that quality two-dimensional surface meshes with boundary are important for cloth simulation, especially when considering collisions.

Chapter 6

A Sparse Level Set Structure

Level sets have been used in several places in this thesis: for representing collision objects in cloth and rigid body simulations, giving the input to a tetrahedral mesh generation algorithm, and specifying how to trim a surface mesh. To quickly review, by level set I mean an implicit representation of a surface, defined as the set of points where some function $\phi(x, y, z)$ defined over all of space is zero. This function is not given analytically but instead is discretely sampled on a grid and reconstructed with interpolation. Usually it is preferable to make ϕ a signed distance function, positive outside the object and negative inside with the magnitude equal to the distance to the surface. This is typically only enforced approximately with numerical algorithms that seek to make a finite difference approximation of $\nabla\phi$ equal to one. The recent book by Osher and Fedkiw[131] contains a detailed overview of level set algorithms and applications.

One of the major difficulties that level sets present is their storage requirements. Storing signed distance on a full three dimensional grid is highly redundant, as typically only values very close to the surface are actually referenced. Local level set methods[1, 134] reduce the computational cost by only ever dealing with these nearby values, but still use the full grid representation and thus have the same storage problem.

While many geometric applications such as those in this thesis incur large overheads by using three-dimensional grids, it should be mentioned that in some applications, notably involving fluid flow, there are several other variables such as velocity which must be stored on three-dimensional grids and thus the level set signed distance function is a relatively small overhead. However, standard level set algorithms have a difficult time maintaining sharp features. This has been resolved for certain classes of flows (namely those where the level set characteristics aren't destroyed in shocks or created in rarefactions) with the hybrid particle-level set algorithm of [56, 57], but for other uses of level sets in physics, the best route currently may be to refine the level set grid beyond the other variables. This again necessitates the use of efficient storage for the level set to avoid large overhead.

The most popular approach to reduce storage is the use of an octree instead of a uniform grid (see e.g. [158, 159, 60, 136]). Assuming that in any $O(n^3)$ grid there are only $O(n^2)$ cells close to the surface, as is the case for smooth enough geometry[1], it can be seen that an octree grid uses optimal $O(n^2)$ storage.

Unfortunately, the overhead involved in an octree representation isn't negligible. Random queries (as may occur in collision detection, ray-tracing, etc.) are $O(logn)$ instead of $O(1)$, though if there is some spatial coherency in a sequence of queries, paths through the octree may be cached to substantially improve this. More fundamental is the overhead implied by the pointer structures, hidden in the asymptotic $O()$ notion: even with careful memory management there are lots of extra pointers required. For static octree level sets, one can make do with just one pointer per cell (pointing to an array of the children), and possibly even eliminate that pointer for the maximum depth cells using some additional special case code. However, for dynamic level sets, or operations on octrees such as ensuring gradual gradations from coarse cells to fine cells, other pointers are needed, such as pointers to neighboring cells. The worst overhead, however, is computational cost on modern architectures: the amount of indirection required and the lack of sequential memory access makes cache pre-fetching largely ineffective.

The other problem with octrees is that many dynamic level set algorithms cannot easily be implemented, motivating the work on alternatives in [158, 159]. For advecting or reinitializing level sets, it is important to use a high resolution Hamilton-Jacobi method (e.g. 5th order WENO[89]), but these usually require several uniformly spaced values on either side of the surface. Again, at the expense of additional code and more storage overhead, the octree could be arranged to include a larger band of cells around the surface, but it is awkward.

Another data structure that has been used to reduce storage requirements is the run-length encoded volume of Curless and Levoy[43]. Long runs of voxels outside or inside the surface are compressed to just the length of the run. While the double buffering algorithm in that paper allowed efficient updates when proceeding in the scanline order, it is not as efficient for other operations such as random point queries (e.g. in the collision algorithms presented in this thesis), the fast marching method, or local updates as in the sculpting application below. On the other hand, the structured operations of finite difference methods for PDE's would require only a simple generalization of the double buffering technique to handle larger stencils: this may be an attractive alternative in the future for computational physics.

6.1 Sparse Block Grids

I propose a different approach: the sparse block grid. Instead of the octree's many levels of resolution, just two levels are used. A coarse grid contains flags for inside/outside in cells (or optionally, an

[1]This assumption is useful for heuristics, but is not necessarily true, particularly in computer graphics. Typically the grid size is not an independent variable, but instead is chosen to be as coarse as possible while still representing the geometry faithfully. Thus it can be expected that a bit more than $O(n^2)$ cells will be close to the surface.

approximate signed distance for use far from the surface), and pointers to fine subgrids near the surface. Another way of thinking about this is as a large grid divided into blocks, but made sparse so only some of those blocks, the ones near the level set surface, actually exist.

The fundamental question that needs to be answered is how large the fine subgrids, the blocks, should be. For reasons of efficiency, they should be large enough to make the pointer overhead negligible and to fill cache lines so that pre-fetching can be exploited: a reasonable minimum is $5 \times 5 \times 5$. However, they cannot be too large or memory will be wasted. For a c^3 coarse grid with blocks of size f^3, the total memory scales like

$$c^3 + c^2 f^3$$

accounting for the coarse grid in the first time, and the $O(c^2)$ populated blocks in the second. To achieve a resolution equivalent to an n^3 full grid, we need $n = cf$. Minimizing the above expression then gives $c = n^{3/4}$ and $f = n^{1/4}$, with memory scaling like $n^{2+1/4}$. This is a little more than an octree, but not far off, and after accounting for overhead is very competitive for the size of structures in use today. As an example, for $n \approx 1000$ this suggests blocks of size 6^3 and a coarse grid of size 167^3.

As an aside, to handle very high resolution where the $O(n^{2.25})$ scaling isn't good enough, it is simple to add one extra level of grid—that is, very fine sub-grids inside the intermediate sub-grids. A simple scaling analysis shows that for three levels, one wants a coarse grid of size $n^{9/13}$, intermediate sub-grids of size $n^{3/13}$, and fine sub-grids of size $n^{1/13}$, for a total memory use scaling like $n^{2+1/13}$. Adding further levels pushes this down to $n^{2+1/40}$, $n^{2+1/121}$, ... It is questionable whether in the foreseeable future n will ever be large enough to warrant going to four or more levels.

Another option that should be mentioned is to replace the coarse grid with a hash-table that stores pointers to the fine sub-grids and flags for the unpopulated blocks inside the object (but not outside, since there are of course infinitely many such blocks). This allows for easy extension of the domain without *a priori* limits, and since most of the computation will be done with reasonably sized blocks the constant factor overhead for the hash look-up instead of a simple array look-up should be negligible. I have not tested this yet.

Updates to the structure are straightforward. When a block's minimum distance to the surface is too large after a modification of the level set, it is deallocated and replaced with a null pointer and an inside/outside flag determined by the sign of the block's values. When the boundary values of a block next to an unpopulated block are sufficiently close to zero, the neighboring block is allocated and initialized. This initialization uses the existing neighboring blocks for boundary conditions, say with the fast marching method[171, 151] or fast sweeping method[170] applied locally. Due to the hyperbolic nature of the Eikonal equation defining signed distance functions, this is correct: those boundary conditions are all the information that is required for initialization, and the new block should not influence the values in existing blocks.

The one difficulty is when two or more neighboring blocks are created simultaneously: their initialization might not be independent mathematically, with information needing to be transferred either or both ways across the shared boundaries. There are two approaches I suggest. The first is to initialize the blocks in an arbitrary order, effectively allowing information transfer only one way across the shared boundaries. This may lead to discontinuities at the boundaries where information needs to flow the other way, so in a second pass, a PDE-based reinitialization (e.g. [161]) corrects the problem while using the first pass as a good initial approximation for efficiency. The second approach, where higher order accuracy isn't critical but speed is (such as the interactive sculpting tool presented below), is to couple neighboring blocks in the fast sweeping method, as in domain decomposition: sweeps are performed on the blocks alternately, with information passed across the common boundary each time. If convergence is required this may take many more sweeps than a single block would need, but the answer after the usual number of sweeps is generally good enough. If the level set function is periodically reinitialized as a matter of course, problems in the original initialization are quickly eliminated anyhow.

The standard numerical algorithms for local level sets can be rewritten to use this structure with no mathematical changes. The code becomes slightly more complex as a result of having to proceed in blocks, but if the blocks are large enough (say $5 \times 5 \times 5$) so that stencils cannot overlap more than two blocks in any direction, it is quite manageable and definitely preferable compared to octree structures.

6.2 Example Application: Sculpting

This section outlines a use of the sparse block grid structure for interactive sculpting of level sets. While most modeling is done with parametric surfaces, implicit surfaces are a useful complement. Large deformations, topology changes, and boolean CSG operations which be difficult with parameterized models are transparently handled by level sets, and PDE methods provide robust surface operations such as smoothing through curvature motion.

Previous work begins with Galyean and Hughes[61] who used a smoothed characteristic function (instead of signed distance) on full grids with local updates from various abstract tools that smoothly modify the surface locally, controlled by a 3D input device. Wang and Kaufman[181] developed alternative sharp carving and sawing tools controlled by a simple mouse, again operating on a full grid. Bærentzen[8] supported CSG and smooth tools with an octree representation. Frisken et al.[60, 136] moved to signed distance functions on octrees. Bönning and Müller[24] removed the limitation of a pre-determined bounding box with a structures inspired by virtual memory paging. Museth et al.[123] added to a full grid representation advanced surface editing operators based on PDE's.

The tool developed here implements very simple PDE-based operators on top of the sparse block

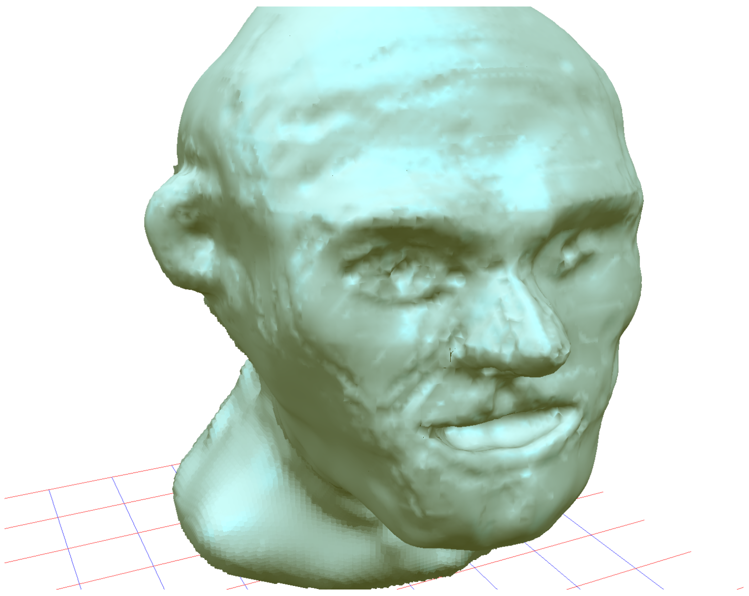

Figure 6.1: A screenshot from the interactive level set sculpting application (at 240^3 resolution).

grid data structure. The mouse is used as the input device: a ray is cast from the eye through the mouse pointer on the screen to the level set surface, and the intersection is used as the base of the tool in use. Raise and lower tools are computed as motion in the normal direction with speed determined by radial distance (a hat function) from the intersection point. Smoothing and roughening tools also use a hat function of radial distance to modulate motion by mean curvature. The level set is continually reinitialized to signed distance using the fast sweeping method on a block-by-block basis, as discussed in the previous section, so that the PDE approximations of the tools stay first order accurate. The radius and magnitude of the hat function controlling the different tools is fully adjustable.

For the interactive rendering, frustum culling is done on the coarse grid, and then an estimate of screen size for each coarse voxel are used to select the rendering method. Coarse voxels far away are rendered with a single point splat of the appropriate size, with the location taken by projecting the center of the voxel onto the level set surface with a normal determined by a central finite difference, and at intermediate distances, each fine grid cell intersected by the level set is rendered with a point splat (see [144] for a more sophisticated multiresolution point splatting algorithm). For close distances, a variation of Surface Nets[66] (an alternative to Marching Cubes[108] for triangulating isosurfaces) is used. This was enough to achieve interactive rates for grids equivalent to 2000^3 on a standard desktop PC, but is certainly not optimal: the triangulation is computed from scratch every frame when most of the time it has only changed locally or simply has been rotated.

An interactive rendering technique that was explored for close-up rendering of isosurfaces is camera space Marching Cubes: the signed distance function is resampled on a grid in camera space aligned with the view frustum (similar to [142]) and Marching Cubes is performed there. This automatically gives a level-of-detail rendering where the sampling density in world space depends on distance to the camera, and gets finer as the camera zooms in. In effect, this is ray-casting on a coarse screen grid with informed interpolation between rays, and thus can very easily benefit from the accelerations natural to ray-casting signed distance functions (frustum culling; occlusion culling; acceleration by taking large steps, i.e. skipping grid cells, when the distance to the surface is large). Unfortunately this suffers from severe resampling problems when the camera is not zoomed in: without prefiltering the signed distance function, disastrous errors may be introduced in the isosurface including spurious holes. Simple convolution filters do not cure this problem, though a nonlinear local minimum filter[83] can at least prevent the introduction of holes though it may significantly overestimate the extent of the isosurface. However, this approach is still useful in debugging level set algorithms, allowing the user to zoom in and see the details of the isocontour of the polynomial reconstruction of the signed distance function between samples.

The final addition to the application is a simple refinement procedure: if more resolution is desired, either the coarse grid or the fine sub-grid size is doubled (the selection is made based on which will use less memory). The tricubic B-spline subdivision rule is used to determine new sample values.

See figure 6.1 for a screenshot of sculpting. Subjectively the tools make it very simple and fairly intuitive to quickly rough out objects, but it is unwieldy for finishing them off and for texturing. This is probably much better done in a parametric modeler.

Final renderings (see figure 6.2) were made with a ray-tracer. Instead of trilinear interpolation, a C^2 tricubic interpolation was used to get a smooth surface without needing to nonphysically adjust normals. Marschner and Lobb[111] explored several possibilities for tricubic interpolation; here I instead used an interpolating C^2 tricubic spline. Starting with the Catmull-Rom C^1 Hermite cubic spline as an initial guess, I used Gauss-Seidel iteration to quickly solve the strongly diagonally dominant linear system for the derivative values needed for C^2 smoothness. As an option in the renderer, the signed distance function was perturbed with noise at a sub-pixel resolution to directly simulate micro-faceted glossy surfaces.

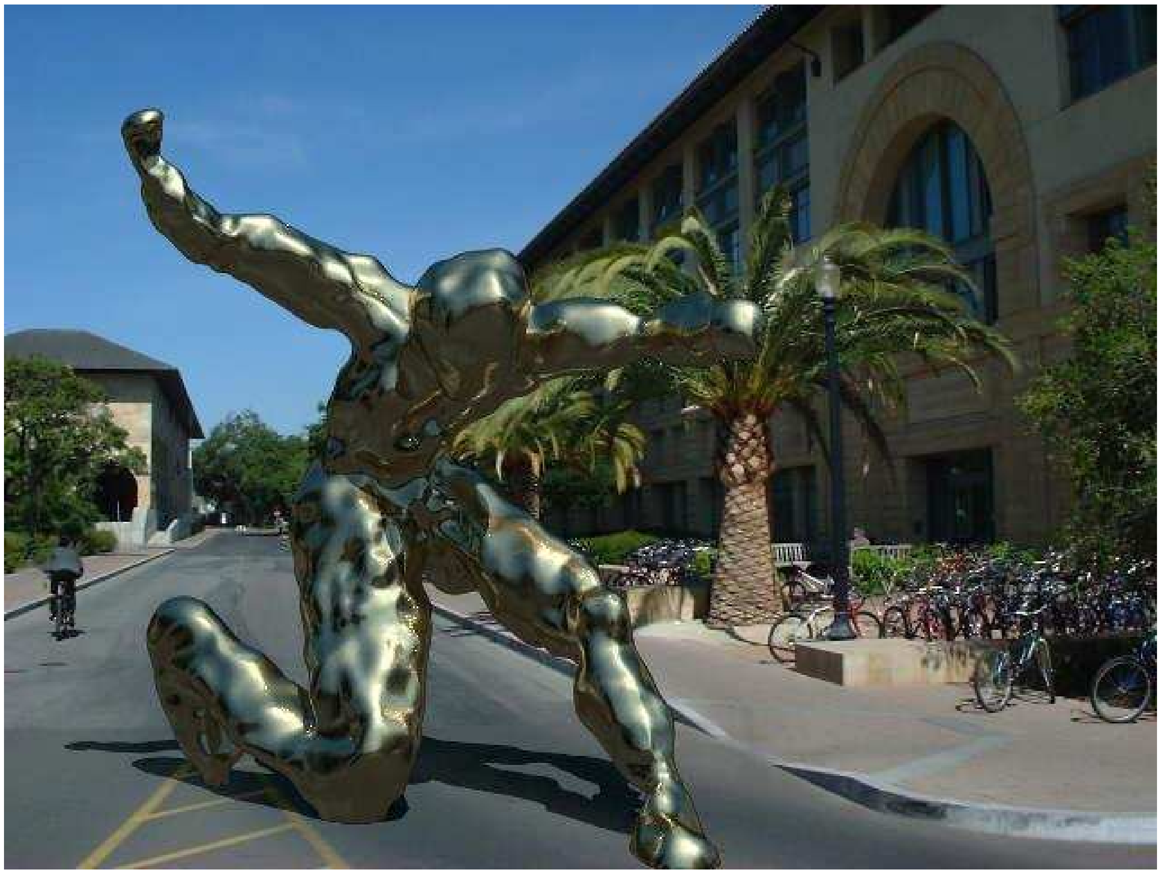

Figure 6.2: A sculpted sparse level set, composited into a real photograph.

Bibliography

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

[2] D. Adalsteinsson and J. A. Sethian. The fast construction of extension velocities in level set methods. *J. Comput. Phys.*, 148:2–22, 1999.

[3] L. Ambrosio and H. M. Soner. Level set approach to mean curvature flow in arbitrary codimension. *Journal of Differential Geometry*, 43:693–737, 1996.

[4] M. Aono. A wrinkle propagation model for cloth. In *Proc. 8th International Conf. of the Computer Graphics Society on CG International '90*, pages 95–115, 1990.

[5] U. M. Ascher and E. Boxerman. On the modified conjugate gradient method in cloth simulation. 2002.

[6] J. Ascough, H. E. Bez, and A. M. Bricis. A simple beam element, large displacement model for the finite element simulation of cloth drape. *J. Text. Inst.*, 87(1):152–165, 1996. part 1.

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

[8] J. A. Bærentzen. Octree-based volume sculpting. In *Proc. IEEE Visualization*, pages 9–12, 1998.

[9] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics (Proc. SIGGRAPH)*, pages 223–232, 1989.

[10] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics (Proc. SIGGRAPH)*, pages 19–28, 1990.

[11] D. Baraff. Coping with friction for non-penetrating rigid body simulation. *Computer Graphics (Proc. SIGGRAPH)*, pages 31–40, 1991.

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

[13] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Computer Graphics (Proc. SIGGRAPH)*, pages 23–34, 1994.

[14] D. Baraff. Interactive simulation of solid rigid bodies. *IEEE Computer Graphics and Applications*, 15(3):63–75, 1995.

[15] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. *Computer Graphics (Proc. SIGGRAPH)*, pages 303–308, 1992.

[16] D. Baraff and A. Witkin. Global methods for simulating contacting flexible bodies. In *Computer Animation Proc.*, pages 1–12. Springer-Verlag, 1994.

[17] D. Baraff and A. Witkin. Large steps in cloth simulation. *Computer Graphics (Proc. SIGGRAPH)*, pages 43–54, 1998.

[18] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Comp. Graphics Forum*, 15(3):387–396, 1996.

[19] R. Barzel, J. F. Hughes, and D. N. Wood. Plausible motion simulation for computer graphics. In *Computer Animation and Simulation '96*, Proc. Eurographics Workshop, pages 183–197, 1996.

[20] M. Bertram, M. Duchaineau, B. Hamann, and K. Joy. Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. In *Proceedings Visualization 2000*, pages 389–396, 2000.

[21] J. Bey. Tetrahedral grid refinement. *Computing*, (55):355–378, 1995.

[22] V. Bhatt and J. Koechling. Three-dimensional frictional rigid-body impact. *ASME J. App. Mech.*, 62:893–898, 1995.

[23] J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):109–116, 1990.

[24] R. Bönning and H. Müller. Interactive sculpturing and visualization of unbounded voxel volumes. In *Proc. 7th ACM Symposium on Solid Modeling and Applications*, pages 212–219, 2002.

[25] F. J. Bossen and P. S. Heckbert. A pliant method for anisotropic mesh generation. In *5th International Meshing Roundtable*, pages 63 – 76, 1996.

[26] D. E. Breen, D. H. House, and M. J. Wozny. Predicting the drape of woven cloth using interacting particles. *Computer Graphics (Proc. SIGGRAPH)*, pages 365–372, 1994.

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

[28] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. *Eurographics/ACM Symp. on Comp. Animation*, 2003.

[29] G. Burns and A. M. Glazer. *Space groups for solid state scientists.* Academic Press, Inc., 1990. (2nd ed.).

[30] S. R. Buss. Accurate and efficient simulation of rigid-body rotations. *J. Comp. Phys*, 164(2), 2000.

[31] E. Caramana, D. Burton, M. Shashkov, and P. Whalen. The construction of compatible hydrodynamics algorithms utilizing conservation of total energy. *J. Comput. Phys.*, 146:227–262, 1998.

[32] M. Carignan, Y. Yang, N. Magnenat-Thalmann, and D. Thalmann. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics (Proc. SIGGRAPH)*, pages 99–104, 1992.

[33] H. Carr, T. Möller, and J. Snoeyink. Simplicial subdivisions and sampling artifacts. In *Proc.IEEE Visualization*, pages 99–106, 2001.

[34] J. T. Chang, J. Jin, and Y. Yu. A practical model for hair mutual interactions. In *Proc. ACM SIGGRAPH Symposium on Computer Animation*, pages 77–80, 2002.

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

[36] B. Chen and M. Govindaraj. A physically based model of fabric drape using flexible shell theory. *Textile Res. J.*, 65(6):324–330, 1995.

[37] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *Journal of the ACM*, 47(5):883–904, 2000.

[38] S. Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. *Computer Graphics (Proc. SIGGRAPH)*, pages 219–228, 2000.

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

[40] J. R. Collier, B. J. Collier, G. O'Toole, and S. M. Sargand. Drape prediction by means of finite-element analysis. *J. Text. Inst.*, 82(1):96–107, 1991.

[41] L. Cooper and S. Maddock. Preventing collapse within mass-spring-damper models of deformable objects. In *The Fifth International Conference in Central Europe on Computer Graphics and Visualization*, 1997.

[42] P. Crossno and E. Angel. Isosurface extraction using particle systems. In *Proceedings Visualization 1997*, pages 495–498, 1997.

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

[44] B. Cutler, J. Dorsey, L. McMillan, M. Müller, and R. Jagnow. A procedural approach to authoring solid models. *ACM Trans. Gr. (Proc. SIGGRAPH)*, 21:302–311, 2002.

[45] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *Proc. First International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 428–438. IEEE, 2002.

[46] H. L. de Cougny and M. S. Shephard. Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 46:1101–1125, 1999.

[47] L. H. de Figueiredo, J. Gomes, D. Terzopoulos, and L. Velho. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of the conference on Graphics interface*, pages 250–257, 1992.

[48] G. Debunne, M. Desbrun, M. Cani, and A. H. Barr. Dynamic real-time deformations using space & time adaptive sampling. *Computer Graphics (Proc. SIGGRAPH)*, 20, 2001.

[49] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. *Computer Graphics (Proc. SIGGRAPH)*, pages 85–94, 1998.

[50] M. Desbrun and M.-P. Gascuel. Animating soft substances with implicit surfaces. *Computer Graphics (Proc. SIGGRAPH)*, pages 287–290, 1995.

[51] M. Desbrun, P. Schröder, and A. H. Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, 1999.

[52] I. Doghri, A. Muller, and R. L. Taylor. A general three-dimensional contact procedure for finite element codes. *Engineering Computations*, 15(2):233–259, 1998.

[53] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (Proc. SIGGRAPH)*, pages 131–137, 1992.

[54] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, pages 52–59, 1996.

[55] J. W. Eischen, S. Deng, and T. G. Clapp. Finite-element modeling and control of flexible fabric parts. *IEEE Computer Graphics and Applications*, pages 71–80, 1996.

[56] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*, 183:83–116, 2002.

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

[58] S. Fisher and M. C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proc. of Eurographics Workshop on Animation and Simulation*, pages 99–111. Eurographics Association, 2001.

[59] L. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40:3979–4002, 1997.

[60] 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. *Computer Graphics (Proc. SIGGRAPH)*, pages 249–254, 2000.

[61] T. A. Galyean and J. F. Hughes. Sculpting: an interactive volumetric modeling technique. *Computer Graphics (Proc. SIGGRAPH)*, pages 267–274, 1991.

[62] L. Gan, N. G. Ly, and G. P. Steven. A study of fabric deformation using nonlinear finite elements. *Textile Res. J.*, 65(11):660–668, 1995.

[63] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. In *Eurographics*, 2000.

[64] R. Garimella and M. Shephard. Boundary layer meshing for viscous flows in complex domains. In *7th International Meshing Roundtable*, pages 107–118, 1998.

[65] M.-P. Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics (Proc. SIGGRAPH)*, pages 313–320, 1993.

[66] S. F. Gibson. Contrained elastic Surface Nets: generating smooth surfaces from binary segmented data. *Proc. Medical Image Computation and Computer Assisted Interventions*, pages 888–898, 1998.

[67] S. F. F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proc. of IEEE Symposium on Volume Visualization*, pages 23–30, 1998.

[68] O. Gloth and R. Vilsmeier. Level sets as input for hybrid mesh generation. In *Proceedings of 9th International Meshing Roundtable*, pages 137–146, 2000.

[69] P. Gnoffo. A vectorized, finite-volume, adaptive-grid algorithm for Navier-Stokes calculations. *Numerical Grid Generation*, pages 819–835, 1982.

[70] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-Tree: a hierarchical structure for rapid interference detection. *Computer Graphics (Proc. SIGGRAPH)*, pages 171–179, 1996.

[71] E. Grinspun, A. Hirani, M. Desbrun, and P. Schröder. Discrete shells. *Eurographics/ACM Symp. on Comp. Animation*, 2003.

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

[73] E. Grinspun and P. Schröder. Normal bounds for subdivision-surface interference detection. In *Proc. of IEEE Scientific Visualization*, pages 333–340. IEEE, 2001.

[74] S. Grosskopf and P. J. Neugebauer. Fitting geometrical deformable models to registered range images. In *European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE)*, pages 266–274, 1998.

[75] R. Grosso, C. Lürig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *Visualization*, pages 387–394, 1997.

[76] E. Guendelman, R. Bridson, and R. Fedkiw. Simulation of non-convex rigid bodies with stacking. *ACM Trans. Gr. (Proc. SIGGRAPH)*, 22, 2003.

[77] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics (Proc. SIGGRAPH)*, pages 299–308, 1988.

[78] P. S. Heckbert. Ray tracing Jell-O brand gelatin. *Computer Graphics (Proc. SIGGRAPH)*, pages 73–74, 1987.

[79] G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs. An implicit finite element method for elastic solids in contact. In *Computer Animation*, 2001.

[80] C. W. Hirt. An arbitrary lagrangian-eulerian computing technique. In *Proc. Second International Conference on Numerical Methods in Fluid Dynamics*, 1970.

[81] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics (Proc. SIGGRAPH)*, pages 295–302, 1994.

[82] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics (Proc. SIGGRAPH)*, pages 19–26, 1993.

[83] K. Hormann, U. Labsik, M. Meister, and G Greiner. Hierarchical extraction of iso-surfaces with semi-regular meshes. In *Symposium on Solid Modeling and Applications*, pages 58–58, 2002.

[84] D. H. House and D. E. Breen, editors. *Cloth modeling and animation.* A. K. Peters, 2000.

[85] P. Howlett and W. T. Hewitt. Mass-spring simulation using adaptive non-active points. In *Computer Graphics Forum*, volume 17, pages 345–354, 1998.

[86] T. J. R. Hughes. *The finite element method: linear static and dynamic finite element analysis.* Prentice Hall, 1987.

[87] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulations. In *Eurographics*, pages 31–45, 1996.

[88] D. L. James and D. K. Pai. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. Gr. (Proc. SIGGRAPH)*, 21, 2002.

[89] G.S-. Jiang and D. Peng. Weighted ENO schemes for Hamilton-Jacobi equations. *SIAM J. Sci. Comput.*, 21(6):2126–2143, 2000.

[90] S. Jimenez and A. Luciani. Animation of interacting objects with collisions and prolonged contacts. In B. Falcidieno and T. L. Kunii, editors, *Modeling in computer graphics—methods and applications*, Proc. of the IFIP WG 5.10 Working Conference, pages 129–141. Springer-Verlag, 1993.

[91] Y. Kang, J.-H. Choi, H.-G. Cho, D.-H. Lee, and C.-J. Park. Real-time animation technique for flexible and thin objects. In *Proc. WSCG*, pages 322–329, 2000.

[92] Y.-M. Kang, J.-H. Choi, H.-G. Cho, and D.-H. Lee. An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer*, 17(3):147–157, 2001.

[93] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. In *International Journal of Computer Vision*, pages 321–331, 1987.

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

[95] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. In *ACM Symp. Comp. Animation*, 2002.

[96] L. P. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. In *Eurographics*, pages 119–130, 1999.

[97] E. Kokkevis, D. Metaxas, and N. Badler. User-controlled physics-based animation for articulated figures. In *Proc. Computer Animation '96*, 1996.

[98] T. L. Kunii and H. Gotoda. Modeling and animation of garment wrinkle formation processes. In *Proc. Computer Animation*, pages 131–147, 1990.

[99] B. Lafleur, N. Magnenat-Thalmann, and D. Thalmann. Cloth animation with self-collision detection. In *Proc. of the Conf. on Modeling in Comp. Graphics*, pages 179–187. Springer, 1991.

[100] A. D. Lewis and R. M. Murray. Variational principles in constrained systems: theory and experiments. *Int'l. J. Nonlinear Mech.*, 30(6):793–815, 1995.

[101] X. Li, S. Teng, and A. Üngör. Biting spheres in 3d. In *8th International Meshing Roundtable*, pages 85–95, 1999.

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

[103] L. Ling, M. Damodaran, and R. K. L. Gay. A quasi-steady force model for animating cloth motion. *Graphics, Design and Visualization*, pages 357–363, 1993.

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

[105] D. W. LLoyd. The analysis of complex fabric deformations. *Mechanics of Flexible Fibre Assemblies*, pages 311–342, 1980.

[106] R. Lohner and J. Cebral. Generation of non-isotropic unstructured grids via directional enrichment. In *2nd Symposium on Trends in Unstructured Mesh Generation*, 1999.

[107] C. Loop. Triangle mesh subdivision with bounded curvature and the convex hull property. Technical Report MSR-TR-2001-24, Microsoft Research, 2001.

[108] W. Lorensen and H. Cline. Marching cubes: A high-resolution 3d surface construction algorithm. *Computer Graphics (Proc. SIGGRAPH)*, 21:168–169, 1987.

[109] N. Magnenat-Thalmann and Y. Yang. Techniques for cloth animation. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *New trends in animation and visualization*, pages 243–256. John Wiley & Sons, 1991.

[110] D. W. Marhefka and D. E. Orin. Simulation of contact using a nonlinear damping model. In *Proc. of the 1996 IEEE Int'l Conf. on Robotics and Automation*, pages 1662–1668. IEEE, 1996.

[111] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of Visualization*, pages 100–107, 1994.

[112] V. J. Milenkovic. Position-based physics: simulation the motion of many highly interacting spheres and polyhedra. *Computer Graphics (Proc. SIGGRAPH)*, pages 129–136, 1996.

[113] V. J. Milenkovic and H. Schmidl. Optimization-based animation. *Computer Graphics (Proc. SIGGRAPH)*, pages 37–46, 2001.

[114] J.V. Miller, D.E. Breen, W.E. Lorensen, R.M. O'Bara, and M.J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Computer Graphics (Proc. SIGGRAPH)*, pages 217–226, 1991.

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

[116] B. Mirtich. Timewarp rigid body simulation. *Computer Graphics (Proc. SIGGRAPH)*, pages 193–200, 2000.

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

[118] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proc. of 1995 symposium on interactive 3d graphics*, pages 181–188, 217, 1995.

[119] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. Submitted to Intl. Meshing Round Table, 2003.

[120] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics (Proc. SIGGRAPH)*, pages 289–298, 1988.

[121] M. Muller, L. McMillan, J. Dorsey, and R. Jagnow. Real-time simulation of deformation and fracture of stiff materials. In *Eurographics*, pages 113–124, 2001.

[122] K. Museth, D. Breen, R. Whitaker, and A. H. Barr. Level set surface editing operators. *ACM Trans. Gr. (Proc. SIGGRAPH)*, pages 330–338, 2002.

[123] K. Museth, D. Breen, R. Whitaker, and S. Mauch. Algorithms for interactive editing of level set models. Technical Report 192, Caltech CACR, 2002.

[124] L. P. Nedel and D. Thalmann. Real time muscle deformations using mass-spring systems. In *Proc. Computer Graphics International*, pages 156–165, 1998.

[125] P. Neugebauer and K. Klein. Adaptive triangulation of objects reconstructed from multiple range images. In *IEEE Visualization*, 1997.

[126] H. N. Ng and R. L. Grimsdale. Computer graphics techniques for modeling cloth. *IEEE Computer Graphics and Applications*, pages 28–41, 1996.

[127] H. N. Ng, R. L. Grimsdale, and W. G. Allen. A system for modelling and visualization of cloth material. *Comput. and Graphics*, 19(3):423–430, 1995.

[128] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins. Graphical modeling of ductile fracture. *ACM Trans. Gr. (Proc. SIGGRAPH)*, 21, 2002.

[129] J. F. O'Brien and J. K. Hodgins. Graphical modeling and animation of brittle fracture. *Computer Graphics (Proc. SIGGRAPH)*, pages 137–146, 1999.

[130] Y. Ohtake and A. G. Belyaev. Dual/primal mesh optimization for polygonized implicit surfaces. In *Proceedings of the seventh ACM symposium on solid modeling and applications*, pages 171–178. ACM Press, 2002.

[131] S. Osher and R. Fedkiw. *Level set methods and dynamic implicit surfaces.* Springer-Verlag, 2002. New York, NY.

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

[133] D. Parks and D. Forsyth. Improved integration for cloth simulation. In *Eurographics*, 2002.

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

[135] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. *Computer Graphics (Proc. SIGGRAPH)*, pages 215–222, 1989.

[136] R. N. Perry and S. F. Frisken. Kizamu: a system for sculpting digital characters. *Computer Graphics (Proc. SIGGRAPH)*, pages 47–56, 2001.

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

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

[139] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–154, May 1995.

[140] X. Provot. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, pages 177–89, 1997.

[141] R. A. Radovitzky and M. Ortiz. Tetrahedral mesh generation based in node insertion in crystal lattice arrangements and advancing-front Delaunay triangulation. *Computer Methods in Applied Mechanics and Engineering*, 187:543–569, 2000.

[142] N. Rasmussen, D. Q. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Trans. Gr. (Proc. SIGGRAPH)*, 22, 2003.

[143] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. In *Eurographics*, 2002.

[144] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. *Computer Graphics (Proc. SIGGRAPH)*, pages 343–352, 2000.

[145] I. A. Sadarjoen and F. H. Post. Deformable surface techniques for field visualization. In *Eurographics*, pages 109–116, 1997.

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

[147] H. Schmidl and V. J. Milenkovic. A fast impulsive contact suite for rigid body simulation. Transactions in Visualization and Computer Graphics, in review, 2002.

[148] J. Schöberl. Netgen - an advancing front 2d/3d mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997.

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

[150] S. Sclaroff and A. Pentland. Generalized implicit functions for computer graphics. *Computer Graphics (Proc. SIGGRAPH)*, pages 247–250, 1991.

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

[152] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal of Numerical Methods Engineering*, 32:709–739, 1991.

[153] J. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proc. Fourteenth Annual Symposium on Computational Geometry*, pages 86–95, 1998.

[154] K. Shimada and D. Gossard. Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing. In *ACM Third Symposium on Solid Modelling and Applications*, pages 409–419, 1995.

[155] K. Sims. Evolving virtual creatures. *Computer Graphics (Proc. SIGGRAPH)*, pages 15–22, 1994.

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

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

[158] J. Strain. Fast tree-based redistancing for level set computations. *J. Comput. Phys.*, (152):664–686, 1999.

[159] J. Strain. Tree methods for moving interfaces. *J. Comput. Phys.*, (151):616–648, 1999.

[160] G. Strang. *An analysis of the finite element method*. Prentice-Hall, Englewood Cliffs, 1973.

[161] M. Sussman, P. Smereka, and S. Osher. A levelset approach for computing solutions to incompressible two-phase flow. *J. Comp. Phys.*, 114:146–159, 1994.

[162] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (Proc. SIGGRAPH)*, pages 185–194, 1992.

[163] J. Teran, S. Salinas-Blemker, V. Ng, and R. Fedkiw. Finite volume methods for the simulation of muscle tissue. *Eurographics/ACM Symp. on Comp. Animation*, 2003.

[164] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, (4):306–331, 1988.

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

[166] D. Terzopoulos, J. Platt, A. H. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics (Proc. SIGGRAPH)*, pages 205–214, 1987.

[167] T. Theußl, T. Möller, and E. Gröller. Optimal regular volume sampling. In *Proc. IEEE Visualization*, pages 91–98, 2001.

[168] J. A. Thingvold and E. Cohen. Physical modeling with B-spline surfaces for interactive design and animation. *Computer Graphics (Proc. SIGGRAPH)*, pages 129–137, 1992.

[169] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Opt.*, 7(1):1–25, 1997.

[170] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao. Fast sweeping algorithms for a class of Hamilton-Jacobi equations. *to appear in SIAM J. Num. Anal.* UCLA CAM Report 01-27.

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

[172] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proc. SIGGRAPH)*, 25:289–298, 1991.

[173] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics (Proc. SIGGRAPH)*, pages 55–64, 1992.

[174] A. Van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *Journal of Graphics Tools*, 3(2):21–42, 1998.

[175] L. Velho, J. Gomes, and D. Terzopoulos. Implicit manifolds, triangulations and dynamics. *Journal of Neural, Parallel and Scientific Computations*, 15(1–2):103–120, 1997.

[176] P. Volino, M. Courchesne, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *Computer Graphics (Proc. SIGGRAPH)*, pages 137–144, 1995.

[177] P. Volino and N. Magnenat-Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In *Proc. of Eurographics*, volume 13 of *Computer Graphics Forum*, pages C–155–166. Eurographics Association, 1994.

[178] P. Volino and N. Magnenat-Thalmann. Developing simulation techniques for an interactive clothing system. In *Proc. of the 1997 International Conf. on Virtual Systems and MultiMedia*, pages 109–118. IEEE, 1997.

[179] P. Volino and N. Magnenat-Thalmann. Implementing fast cloth simulation with collision response. *Computer Graphics International*, pages 257–266, 2000.

[180] P. Volino and N. Magnenat-Thalmann. Comparing efficiency of integration methods for cloth simulation. In *Proc. Computer Graphics International*, pages 265–274, 2001.

[181] S. Wang and A. E. Kaufman. Volume sculpting. In *Proc. Symposium on Interactive Graphics*, pages 151–214. ACM SIGGRAPH, 1995.

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

[183] J. Weil. The synthesis of cloth objects. *Computer Graphics (Proc. SIGGRAPH)*, pages 49–54, 1986.

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

[185] R. T. Whitaker. A level-set approach to 3d reconstruction from range data. *International Journal of Computer Vision*, 29(3):203–231, 1998.

[186] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics (Proc. SIGGRAPH)*, 28:269–277, 1994.

[187] Z. Wood, M. Desbrun, P. Schröder, and D. Breen. Semi-regular mesh extraction from volumes. In *Visualization 2000*, pages 275–282, 2000.

[188] S. Yamakawa and K. Shimada. High quality anisotropic tetrahedral mesh generation via packing ellipsoidal bubbles. In *The 9th International Meshing Roundtable*, pages 263–273, 2000.

[189] D. Zhang and M. M. F. Yuen. Collision detection for clothed human animation. In *Pacific Graphics*, pages 328–337, 2000.

[190] H.-K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *1st IEEE Workshop on Variational and Level Set Methods, 8th Int. Conf. on Computer Vision*, pages 194–202, 2001.