Fast Viscoelastic Behavior with Thin Features

Chris Wojtan
Georgia Institute of Technology

Greg Turk
Georgia Institute of Technology

Abstract

We introduce a method for efficiently animating a wide range of deformable materials. We combine a high resolution surface mesh with a tetrahedral finite element simulator that makes use of frequent re-meshing. This combination allows for fast and detailed simulations of complex elastic and plastic behavior. We significantly expand the range of physical parameters that can be simulated with a single technique, and the results are free from common artifacts such as volume-loss, smoothing, popping, and the absence of thin features like strands and sheets. Our decision to couple a high resolution surface with low-resolution physics leads to efficient simulation and detailed surface features, and our approach to creating the tetrahedral mesh leads to an order-of-magnitude speedup over previous techniques in the time spent re-meshing. We compute masses, collisions, and surface tension forces on the scale of the fine mesh, which helps avoid visual artifacts due to the differing mesh resolutions. The result is a method that can simulate a large array of different material behaviors with high resolution features in a short amount of time.

Keywords: Deformable models, viscoelastic behavior, finite element method, computational fluid dynamics, free-form deformation, explicit surface.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation.

1 Introduction

Physics-based animation is responsible for many impressively realistic special effects. In recent years, the quality of these effects has greatly improved due to the increased sophistication of numerical techniques for physical simulation. Particular attention has been given to the animation of deformable materials because they are far too complex to animate by hand. In fact, they are often too complex to simulate on a computer without making several approximations and reducing the size of the problem. One of the tasks of computer graphics researchers is to recognize which types of approximations we can make without introducing visually disturbing artifacts.

Embedded techniques for animating deformable bodies are popular because they give the illusion of highly detailed physics with relatively simple computations. Unfortunately, the nature of embedded deformations limits the scope of realistic material behaviors to those that will not significantly alter the original embedding. We present a technique for removing this obstacle and greatly enhancing the range of materials that can be simulated with embedded deformations.

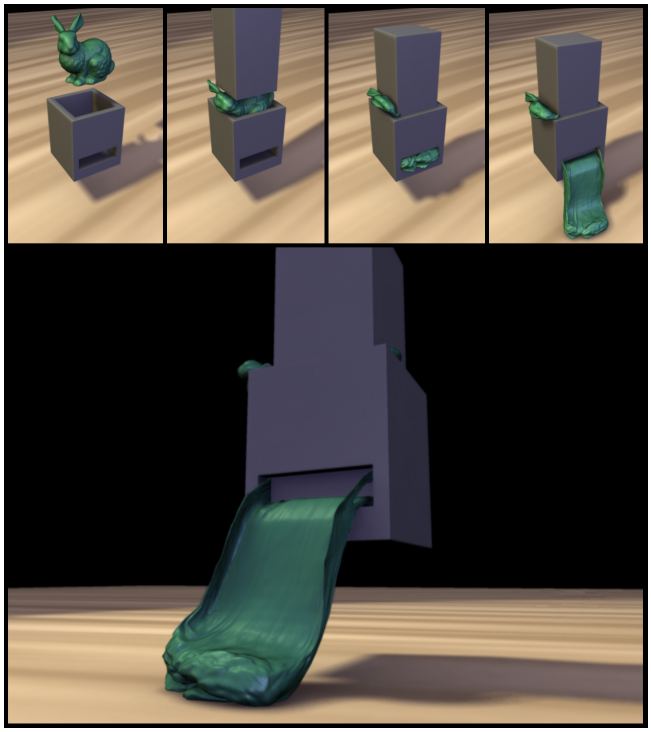


Figure 1: *A stiff plastic Stanford Bunny is forced through a small pressing machine that squishes it into several thin sheets. This result was generated in 30 minutes.*

The heart of our method is a finite element method (FEM) for simulating elasticity and plasticity. We represent the surface of our object with a triangle mesh and embed it into the mesh of tetrahedral elements. Though embedded methods have existed in the literature for years, no research has combined such a technique with frequent recomputation of the coarse control mesh. By re-meshing our underlying FEM mesh whenever simulation quality degrades, we remove several of the barriers preventing embedded mesh techniques from simulating highly plastic behavior. In figure 2, we display a simplified map of the types of materials useful to computer animators. Materials with limited plasticity, like ceramic and rubber, can be modeled with existing embedded FEM techniques, but extremely plastic behaviors like those of water and toothpaste are not possible due to the fixed topology of the control mesh. In this paper, we combine an FEM-driven embedded surface mesh with fast and robust re-meshing, allowing for efficient and stable simulation of behaviors previously unattainable by embedded mesh techniques. In addition, the embedded nature of our method preserves significantly more surface details than existing methods for animating vis-

{wojtan, turk}@cc.gatech.edu.

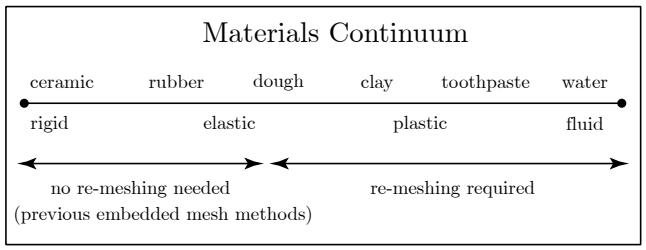


Figure 2: *Current FEM methods with an embedded surface mesh only simulate a limited range of materials because ill-conditioned basis functions result from large plastic flow. Our method recomputes basis functions by re-meshing throughout the simulation, allowing us to simulate a much greater range of materials.*

coelastic flow, and it remains stable when simulating thin features that have eluded previous techniques. Figure 1 shows an example.

We calculate the continuum mechanics forces on an underlying FEM mesh, taking advantage of our knowledge of the detailed surface whenever possible. We then calculate other forces on the surface mesh, including those from collision resolution and surface tension. Finally, we couple these forces together to animate the surface through time. With an embedded surface, we are free to use a non-conforming FEM mesh. We use the Delaunay tetrahedralization of an un-warped body-centered cubic (BCC) lattice that we exploit for much faster mesh creation and point location queries. We sidestep inaccuracies from this low resolution mesh by carefully computing nodal masses whenever necessary. The main contributions of this paper are as follows:

Embedded surface mesh: Our combination of an embedded mesh and a model for large plastic flow with frequent re-meshing preserves surface features while avoiding common visual artifacts such as popping, smoothing, and volume loss.

Accurate sub-element mass computation: We compute nodal masses at the resolution of the surface mesh, which is essential for plausible animation of high-resolution surface details.

Efficient re-meshing: Our remeshing technique is an order of magnitude faster than previous approaches and we reduce the time spent transferring data between meshes by exploiting mesh structure.

High resolution surface tension: We compute surface tension forces on the finely-detailed surface mesh and couple them with the coarser finite element mesh.

Large range of materials: This method efficiently simulates nearly-rigid bricks, dripping slime, and splashing liquid without the need to change numerical solvers or simulation techniques.

Resolution of thin features: We can plausibly animate thin sheets and strands of material without lowering the resolution of the simulation. We have not seen such a combination of detailed results and fast simulation times elsewhere in the community.

2 Related Work

The computer graphics literature is rich with physical simulation techniques. Terzopolous et al. [1988a; 1988b; 1989], pioneered the idea of using deformable models in computer graphics. O'Brien and Hodgins [1999] used a finite element method to animate elasticity and brittle fracture. Later, they added a limited amount of plasticity to the model to produce ductile fracture [O'Brien et al. 2002]. Müller et al. [2002] introduced a co-rotational formulation for stable FEM animation, and Müller and Gross [2004] added plasticity and fracture to the model. Irving et al. [2004] made FEM simulations more stable by allowing for invertible elements, and Irving et al. [2007] enforced an incompressibility condition.

One way to present the illusion of a faster, more detailed simulation is to embed a detailed surface into a coarse control mesh. Sederberg and Parry [1986] introduced this idea of free-form deformation (FFD), and Faloutsos et al. [1997] introduced dynamic FFD for animation. The work of Capell et al. [2002a; 2002b] is closer to ours. They used a coarse FEM mesh to control a fine surface mesh, which gives the illusion of a high-resolution elastic simulation. Müller et al. [2004b] embedded a surface mesh into a finite element simulation to simulate fracture in real time. Molino et al. [2004] applied FFD to the crack front of a fracture simulation, skirting around the stability problem that arises with small or poorly-shaped elements. Sifakis et al. [2007a] improved upon this method by allowing arbitrary cuts within an element. Sifakis et al. [2007b] developed a method that embedded a high-resolution point-sampled surface in a coarse finite element mesh. Like us, they performed collision detection and response using the high-resolution surface. However, they applied forces to the embedded particle directly and updated the finite element mesh using *soft* bindings. In our approach, we apply collision response forces directly to the finite element mesh, corresponding to their *hard* bindings.

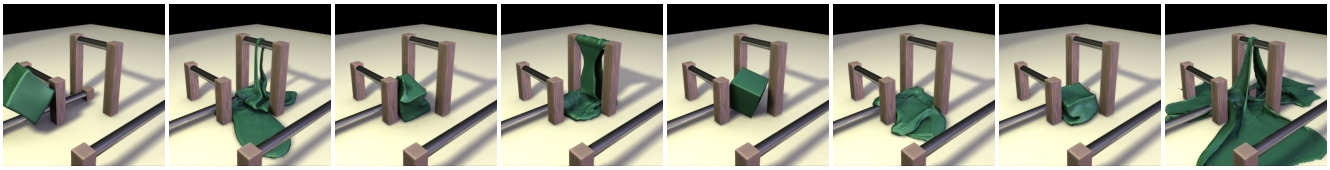
In addition to embedding a surface into a finite element simulation, Müller et al. [2005] showed elastic behavior can be driven through shape matching. Recently, Rivers and James [2007] made this idea even more efficient, and Botsch et al. [2007] applied this idea to shape modeling. Galoppo et al. [2006] used deformation textures to simulate high resolution surface detail with a simplified interior model. The work of Batty et al. [2007] is similar to ours in that they simulate complex phenomena on a coarse grid by calculating sub-grid cell accurate details.

Meshless methods are an interesting alternative to finite elements. Müller et al. [2004a] applied a meshless method to elastic and plastic simulations, and Pauly et al. [2005] showed that these methods are excellent for computing crack fronts in fracture simulations. Keiser et al. [2005] simulated both liquid and solid material with a meshless method.

Our paper addresses the simulation of elasticity, plasticity, and viscosity. Clavet et al. [2005] produced elastic and plastic behaviors with a particle simulation. Goktekin et al. [2004] added elasticity to an Eulerian viscous fluid simulation to produce viscoelastic fluids. Losasso et al. [2006] extended this model by accounting for rotation of elastic terms. In our simulations, we use the method of Bargteil et al. [2007], who added large plastic flow to a finite element simulation by recomputing basis functions as they become ill-conditioned.

Surface tracking is also relevant to our work. By embedding a triangle mesh into a finite element simulation, we have opted for explicit surface tracking. Reynolds [1992] used explicit surface tracking by embedded a high resolution surface into a complex flow field. Brochu [2006] implemented a fluid simulation with an explicit surface driven by a boundary element method, and Jiao [2007] developed an advection scheme for explicit surfaces. Enright et al. [2005] combined explicit particles and an implicit level set to create the particle level set technique, and Bargteil et al. [2006] updated an explicit surface mesh through time using an implicit representation. Mullen et al. [2007] developed an advection scheme for Eulerian fluid that allows them to conserve volume.

Our work also involves generating FEM meshes. The meshing technique by Alliez et al. [2005] employs iterative optimization to conform to a surface mesh. Molino et al. [2003] used a BCC lattice and



$\mu = 1 \times 10^5$ $\beta = 5 \times 10^1$ $P_Y = 1 \times 10^{10}$ $\nu = 0$	$\mu = 2 \times 10^6$ $\beta = 1 \times 10^3$ $P_Y = 0$ $\nu = 1 \times 10^5$	$\mu = 2 \times 10^6$ $\beta = 3 \times 10^3$ $P_Y = 3 \times 10^4$ $\nu = 1 \times 10^{30}$	$\mu = 2 \times 10^5$ $\beta = 1 \times 10^2$ $P_Y = 3 \times 10^3$ $\nu = 2 \times 10^3$	$\mu = 1 \times 10^7$ $\beta = 1 \times 10^4$ $P_Y = 1 \times 10^{10}$ $\nu = 0$	$\mu = 2 \times 10^6$ $\beta = 1 \times 10^2$ $P_Y = 1 \times 10^1$ $\nu = 1 \times 10^3$	$\mu = 5 \times 10^4$ $\beta = 3 \times 10^2$ $P_Y = 5 \times 10^5$ $\nu = 1 \times 10^{10}$	$\mu = 2 \times 10^6$ $\beta = 1 \times 10^2$ $P_Y = 0$ $\nu = 1 \times 10^5$

Figure 3: *Material parameters for a dropped cube, demonstrating a range of behaviors ranging from rigid to fluid-like. Each image is taken from a separate animation with different material parameters. μ is stiffness, β is viscosity, P_Y is yield stress, and ν is flow rate.*

a physics-based optimization strategy to enforce mesh conformity. Recently, Labelle and Shewchuk [2007] also used a BCC lattice and placed guaranteed bounds on the quality of the resulting finite elements. Chentanez et al. [2007] use this technique to animate liquid. We use a BCC lattice as well, but we do not force it to conform to our surface mesh. The high quality of the shapes of our tetrahedra allow us to run stable simulations with large time steps. Please see Shewchuk [2002] for more information on the correlation between tetrahedral shapes and the accuracy and stability of finite elements.

3 Physical Model

To produce our animations of elastic and plastic phenomena, we use the elasticity model of Irving et al. [2004] and the plasticity model of Bargteil et al. [2007]. This plasticity model incorporates creep and work hardening:

$$\hat{\mathbf{F}}_p = (\hat{\mathbf{F}}^*)^\gamma, \tag{1}$$

$$\gamma(\mathbf{P}, P_Y, \nu, \alpha, K) = \min\left(\frac{\nu(||\mathbf{P}|| - P_Y - K\alpha)}{||\mathbf{P}||}, 1\right) \tag{2}$$

where $\hat{\mathbf{F}}_p$ is the diagonalized plastic deformation tensor, $\hat{\mathbf{F}}^*$ is the volume-conserving portion of the diagonalized deformation tensor, $\mathbf{P}$ is the stress tensor, P_Y is the plastic yield point, ν is the flow rate, K is a hardening parameter, and α is a scalar representing the accumulated plastic stress. See Bargteil et al. [2007] and Irving et al. [2004] for more details.

Following Bargteil et al., we allow for arbitrarily large plastic flow by re-meshing whenever the finite element basis functions become ill-conditioned. After creating a new FEM mesh, we copy all old simulation variables to the new, well-conditioned mesh through interpolation and averaging. Unlike previous work we employ an explicit surface mesh, avoiding problems such as volume loss and limited surface detail. We also use a non-conforming tetrahedral FEM mesh based on a BCC lattice (that is, the boundary of the finite element mesh does not precisely align with the material interface). To advance our simulations forward through time, we use a variable time step Newmark integrator, as in Bridson et al. [2003].

4 Embedded Surface Mesh

Our method significantly differs from previous work in the treatment of the surface details of an object. Instead of using the boundary of the FEM mesh as the exterior surface of our simulated material, we provide a separate explicit mesh for the surface. We use this surface mesh for rendering as well as for collision dynamics, but all of the elasticity and plasticity computations come from the finite element simulation.

For each vertex in the surface mesh, we find the tetrahedral element that completely encloses it, and we assign the vertex to that tetrahedron. From this point on, we can express the surface vertex position $\mathbf{x}$ as a linear combination of the position of tetrahedral vertices $\mathbf{x}_i$ using barycentric coordinates b_i:

$$\mathbf{x} = b_1\mathbf{x}_1 + b_2\mathbf{x}_2 + b_3\mathbf{x}_3 + b_4\mathbf{x}_4. \tag{3}$$

This way, when the finite element calculations distort the parent tetrahedron, the surface mesh will distort as well. We can also express the velocity and forces on this surface vertex as a linear combination of the velocities and forces of the parent element's vertices using these same barycentric coordinates.

For collisions with objects in the environment, we use a projected-vertex approach [Irving et al. 2004]. However, instead of computing collisions with FEM nodes, we compute collisions with respect to the embedded surface vertices. If a vertex is involved in a collision, we project the vertex onto the obstacle's surface by moving the entire element such that the collision is resolved. To do this, we define distribution weights for each surface vertex:

$$w_i = \frac{b_i}{b_1^2 + b_2^2 + b_3^2 + b_4^2}, \quad i = 1, 2, 3, 4 \tag{4}$$

where w_i is the distribution weight corresponding to node i of the parent tetrahedron, and b_i is its corresponding barycentric coordinate. Using these distribution weights, we distribute the position change to each of the nodes of the tetrahedron via the equation $\mathbf{x}_i = \mathbf{x}_i + w_i\Delta\mathbf{x}$. Similarly, we add velocity impulses in the presence of friction with $\mathbf{v}_i = \mathbf{v}_i + w_i\Delta\mathbf{v}$. To avoid competing collisions within an element, we only apply the collision resolution to the deepest penetrating vertex in each element. For self collisions and collisions with other deformable bodies, we use the model of Bridson et al. [2002]. We apply impulses to each surface vertex using the distribution weights as described above.

In simulations with large plastic flow, sometimes the surface triangles stretch to be distractingly large, or shrink to the point that they are unnoticeably small. To maintain a quality surface mesh, we subdivide long edges at the midpoint and bisect the two incident triangles. We also perform a simple edge collapse operation if any edge in the mesh is too short. If volume preservation is of utmost importance, one might decide to implement a more sophisticated scheme like that of Lindstrom and Turk [1999].

Lastly, our embedded scheme allows several additional optimizations: because each surface vertex is guaranteed to lie within its parent element, the tetrahedral FEM mesh doubles as a collection of bounding volumes that can greatly speed up collision detection. As the position of a surface vertex is given by the barycentric coordinates within a tetrahedron, there is no need to update the position of a surface vertex unless it is participating in a collision or we are

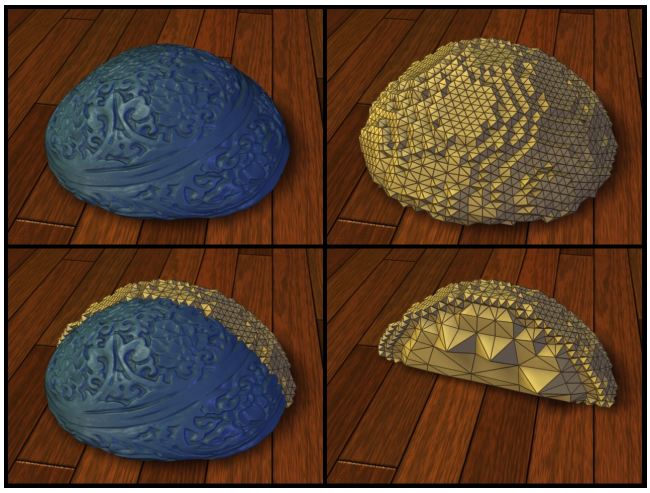


Figure 4: *A high resolution surface mesh (blue) embedded into a low resolution adaptive BCC lattice (gold). We show the cross section of the FEM mesh in the bottom row.*

rendering the surface. If there is no collision with the bounding tetrahedron, there is no need to update any of the surface vertices within it. This idea is especially useful when there are hundreds of surface vertices per element, as in figure 4.

5 Re-meshing

We use a tetrahedral mesh constructed from a BCC lattice [Molino et al. 2003; Labelle and Shewchuk 2007] for our finite element computations. To create the tetrahedralization, we first voxelize our surface mesh onto two offset regular cubic grids. Any voxelization algorithm will suffice; we chose to scan convert the surface mesh by sorting triangles by their x- and y- bounding boxes, casting rays parallel to the z-axis, and counting triangle intersections up to each grid point. This voxelization method is analogous to polygon rasterization, as noted by Nooruddin and Turk [2003]. During voxelization, we classify each lattice node as internal or external with respect to the surface mesh. Finally, we create any BCC tetrahedra that touches an internal node or overlaps any geometry in the surface mesh. For added efficiency, we use graded BCC tetrahedra on an octree as in Labelle and Shewchuk [2007].

We embed our surface mesh into the volumetric tetrahedral mesh, so we do not spend any computational effort modifying our BCC mesh to conform to the surface. Consequently, we do not need to perform any iso-surface evaluations. We never move any BCC nodes and we only briefly deal with the surface, yielding a highly efficient meshing algorithm. As Labelle and Shewchuk [2007] state, although their technique is already quite fast, the mesh generation step of their algorithm took 1% of the total time, while the rest was spent on isosurface evaluation. Based on their timings, using a nonconforming mesh (as we do) is roughly one hundred times faster than the state of the art in conformal meshing.

After re-meshing, we transfer FEM simulation data from the old tetrahedral mesh to the new one. In our simulations, this data consists of a deformation gradient tensor $\mathbf{F}$ and a scalar accumulated stress α stored at each element, and a velocity vector $\mathbf{v}$ stored at each node. To assign data to each element in the new mesh, we compute the average of the data from each overlapping tetrahedron in the old mesh, weighted by the amount of overlap in volume. To transfer node-based data, we first find which tetrahedron in the old mesh encapsulates the node, then use barycentric interpolation to find the new data. Boundary nodes present a slight complication—we do not use a conforming tetrahedral mesh, so some new simulation nodes will lie outside of the old simulation mesh. Barycentric interpolation is not possible here because there is no bounding tetrahedron. Instead, we find the closest element to the node and use barycentric extrapolation. Extrapolation is valid in this case because it preserves the velocity of the embedded surface vertices. We do not do anything special for new elements that only partially overlap the old tetrahedral mesh; a weighted average of overlapping tetrahedra is sufficient for these edge cases.

For the efficient transfer of data between arbitrary meshes, we could use a kd-tree for point location and a hierarchical system of bounding boxes for tetrahedral-overlap queries. Fortunately, we do not need to resort to any of these data structures for efficient transfer of variables after a re-mesh because we can take advantage of the BCC lattice structure of our new undeformed mesh. We perform point location in logarithmic time using a BCC octree (constant time for a uniform lattice), and we save memory by never allocating any auxiliary data structures. We transfer per-node variables by iterating through each tetrahedron in the old deformed mesh and performing a fast lookup into the undeformed BCC lattice of the new mesh to find out which nodes in the new mesh overlap the tetrahedron's bounding box. For each node from the new mesh that is inside of this tetrahedron, we transfer data using barycentric interpolation. Per-element variable transfer is just as efficient: for each tetrahedron in the old mesh, we can quickly find all new tetrahedra that overlap its bounding box by taking advantage of the undeformed lattice structure of the new mesh. We compute the volume of overlap between these new tetrahedra and the old one, and use that volume for the weighted average in the per-element data transfer.

This immense savings in time spent re-meshing might be viewed as unimportant because we only re-mesh sporadically throughout the simulation. After all, Bargteil et al. [2007] reported that they only spent 13% of the simulation time re-meshing. However, re-meshing is nearly free in our case, so we can afford to re-mesh quite frequently, allowing us to simulate phenomena with a much higher flow rate ν in the same amount of time.

6 Exact Mass Computation

We use a lumped mass formulation in our finite element simulation, as described in O'Brien and Hodgins [1999]. This entails computing the mass contribution of each element, then distributing that

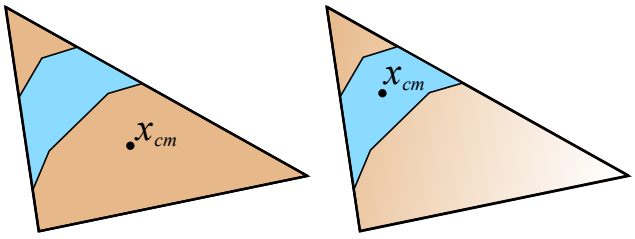


Figure 5: ***Sub-element mass computation****: The blue strip of material is clipped to the element, and then masses are distributed to nodes at the element corners, resulting in a linear density function represented in beige. Naive lumped mass calculations give a uniform density across the element and an incorrect center of mass x_{cm} (left). Our method computes the correct center of mass and density distribution (right).*

Figure 6: *A viscoelastic armadillo drips through a straining device, resulting in several thin strands of slime. Because we compute nodal masses at sub-element resolution, our method maintains physical plausibility in the presence of these small features. This result was simulated at 30 seconds per frame*

mass to the nodes of the FEM mesh by assigning one quarter of the element's mass to each of its nodes. One problem with blindly applying this strategy to our simulation with an embedded mesh is that the mass per element no longer accurately reflects the actual amount of mass we may wish to represent. This is especially problematic with coarse FEM meshes or thin strands of surface material. For example, in the left side of figure 5, a thin strip of material only covers part of the element. By using a naive lumped mass formulation and ignoring the surface mesh, the center of mass actually exists outside of the material, and the total mass in the element grossly overestimates the amount of mass occupied by the surface mesh. This leads to sudden unexpected movements if the center of mass varies significantly after a tetrahedral re-mesh, and the error in total mass unphysically injects momentum into the system during significant plastic flows. A slightly inaccurate mass and inertial moment may not be visually disturbing by themselves, but they may cause noticeable artifacts if we continually recompute inconsistent physical properties on different FEM meshes through time.

Our goal is to calculate mass distributions per element that accurately reflect the properties of the surface mesh. If we ensure that they are correct on a per-element level, then the physical properties of the overall object will be equal before and after a re-mesh event. Our strategy for computing correct nodal masses is to first compute the exact mass and center of mass occupied by the material in each element, then distribute the mass from the exact center of mass of the material within that element. This will ensure that the mass and moment of the element match the mass and moment of the material within it. Interestingly, Batty et al. [2007] used a similar technique in a much different context to achieve sub-grid accurate coupling in an Eulerian fluid simulation.

We compute the mass within an element by clipping the surface mesh to the tetrahedral element, then computing the mass m and center of mass x_{cm} of the resulting closed polyhedron. Our implementation uses the method by Lien et al. [1984] to compute the volume V of the clipped surface mesh. We then compensate for the elastic deformation by dividing by the determinant of the deformation gradient $\mathbf{F}$.

$$m = \frac{\rho V}{|\mathbf{F}|} \tag{5}$$

where ρ is the material density. After we have obtained m and x_{cm}, we compute barycentric coordinates $b_{1,2,3,4}$ and distribution weights $w_{1,2,3,4}$ for x_{cm}, and distribute m to each node of the tetrahedron via $m_i = m_i + w_i m$.

Unfortunately, just as poorly-shaped tetrahedral elements can introduce stability problems, so can small nodal masses. The exact nodal mass computation is ideal for accurate simulations, but proves impractical in the presence of thin features, where masses per node may be near zero. We implemented a minimum node mass $m_{\min}$ as a tunable simulation parameter. Each time we re-mesh, we compute the exact mass for every FEM node and then clamp any unacceptably small mass to $m_{\min}$. This allows the user to strike a balance between accuracy and simulation speed. We have found that overestimations of mass were not noticeable for largely elastic simulations, but large plastic flows leading to thin sheets and strands required more accurate masses. In our most fluid-like simulations, we set $m_{\min}$ to be 5% of the mass that would be normally given to a node surrounded by fully-massive elements.

7 High Resolution Surface Tension

We occasionally wish to compute surface tension forces as well as elastic and plastic forces. We leverage the work by Brochu [2006] to compute momentum-conserving surface tension forces on the high resolution surface. Brochu used triangles as the simulation primitives, while we use vertices, so we distribute per-triangle forces by dividing the force equally among the three vertices of each triangle. The force on a triangle from a neighboring face is a vector in the plane of the neighbor and normal to the shared edge, proportional to the edge length and the surface tension coefficient. So the total surface tension force on each triangle is:

$$\mathbf{f}_T = \sum_{i=1}^{3} \sigma(\mathbf{n}_i \times \mathbf{e}_i) \tag{6}$$

where σ is the surface tension coefficient, $\mathbf{n}_i$ is the normal of the neighboring triangle i and $\mathbf{e}_i$ is a vector representing the shared edge between this triangle and this neighbor, with the edge vectors pointing clockwise around the triangle. The surface tension force on each vertex is one third of the total force from all surrounding triangles:

$$\mathbf{f}_{\text{tension}} = \frac{1}{3} \sum_{j=1}^{n} \mathbf{f}_{T,j} \tag{7}$$

where j iterates through each of the vertex's incident triangles. Once we have a per-vertex surface tension force $\mathbf{f}_{\text{tension}}$, we distribute these forces to the FEM nodes using distribution weights

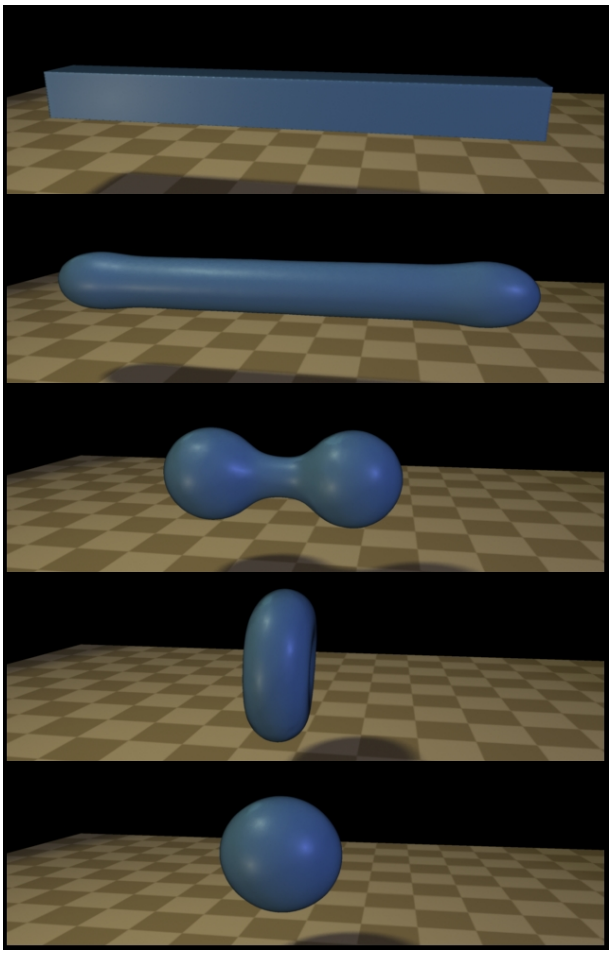


Figure 7: *Surface tension smoothly changes the shape of a rectangular block.*

($\mathbf{f}_i = \mathbf{f}_i + w_i \mathbf{f}_{\text{tension}}$). This suffices to produce a plausible surface tension simulation on the FEM mesh.

These coarse surface tension forces do not change sharp features within an element. To smooth out local features, we use explicit Laplacian smoothing with a scale-dependent umbrella operator [Desbrun et al. 1999]. At each time step, we apply this smoothing and scale it back by a multiple of the surface tension coefficient and the time step size. We used this technique to generate the animation in figure 7.

8 Algorithm Overview

Our algorithm is summed up by the pseudo-code in figure 8. The timestep begins by calculating all of the forces on each FEM node due to elasticity, plasticity, gravity, and surface tension. Then the velocities and positions of the nodes are updated via time integration. If we wish to simulate surface tension, we then apply geometric smoothing and recompute barycentric coordinates of the surface vertices. Then we handle collisions and update surface vertices to account for any changes in position. At the end of the timestep, we decide whether we should re-mesh. A re-mesh event can be triggered by ill-conditioned element basis functions or a significant change in the size or shape of an element. We also force a re-mesh if geometric smoothing has caused a surface vertex to move significantly outside of its parent element. The simulations in this paper

```
Calculate forces (elastic/plastic, gravity,
                  surface tension)
Integrate FEM nodes
If SurfaceTension
  Smooth the surface
  Recompute barycentric coordinates
Update surface vertex positions
Collision detection and resolution
Update surface vertex positions
If RemeshNeeded
  Subdivide large surface triangles
  Collapse short surface edges
  Create new FEM mesh
  Recompute barycentric coordinates
  Transfer variables to new mesh
  Re-calculate mass in FEM nodes
```

Figure 8: *Pseudo-code for one timestep*

re-meshed if any element tripled its condition number since the last re-mesh event, or if any barycentric coordinate dropped below -0.2.

Once we have decided to re-mesh, we first operate on the surface mesh by subdividing large triangles or decimating small ones. Then we create a new mesh using the procedure in section 5. Finally, we recompute barycentric coordinates, transfer simulation variables from the old mesh to the new one, and re-calculate FEM masses.

For the sake of clarity, we did not incorporate any optimizations into figure 8 (such as delaying surface vertex updates as long as possible), and we did not discuss special collision treatment during time integration. For example, our algorithm uses the integrator of Irving et al. [2004], which interweaves collision handling and velocity integration.

9 Results

Figure 4 shows a coarse finite element mesh of 20k tetrahedra used in conjunction with an extremely fine surface mesh of 360k triangles, illustrating how a complex surface works with a nonconformal FEM mesh. In figure 1, we smashed a 70k triangle bunny model into a ream of thin material. The total simulation time was 30 minutes. We chose this example to show how our method can efficiently simulate conditions that would push other techniques to their limits. The armadillo in figure 6 started with 8k elements and 71k surface triangles, finished with 11k elements and 113k triangles, and simulated at 30 seconds per frame. Notice how our method preserves the detailed ridges on the armadillo's face, legs, and back, and notice the thin strands of goo as pieces of his legs slowly drip off. The thinnest strand in the simulation is more than one hundred times smaller than the FEM resolution. Figure 9 shows a simulation of two different colored hands colliding with each other. The distinct meshes stick together but do not merge, illustrating how this method can be used to animate immiscible fluids. Figure 3 shows the range of material parameters capable with this technique. By varying the viscosity, plastic yield, flow rate, and elastic stiffness, we were able to generate rigid bricks, thick viscous ooze, jiggly gelatin, gooey jelly, and splashing fluid. On average, each of the examples in figure 3 took an hour and a half to simulate, while the most extreme fluid example (figure 3, far right) simulated for about 10 hours. It began with 7.5k elements and 12k triangles, and ended with 60k elements and over 600k triangles. We used NVIDIA's Gelato to render our animations. Ambient occlusion was utilized in all animations except for figures 4 and 6.

Though we are proud of the quality and speed of our simulations,

our method could greatly benefit from a GPU implementation and a fully implicit integrator. These additions could speed up some moderately complex examples to real time. We can already run some coarse examples at interactive rates on our unoptimized research code. Alternatively, we believe that a professionally-implemented version of this method could generate detailed production-quality results in tens of minutes.

10 Discussion

We have presented a method for animating a large range of different material parameters with relatively quick simulation times. We designed our method to handle exceptionally thin features without becoming unstable. Our system can even address the thin sheet problem that arises in many fluid simulation techniques. Because the plasticity model forces our tetrahedra to locally preserve volume, volume is approximately conserved even for extremely thin features embedded in the FEM mesh. Figure 3, far right, shows an example of a low-viscosity fluid that has thin sheets everywhere, yet 95% of the volume was retained by the end of the simulation. A grid-based level set method would fail to resolve any of these features, resulting in an aggressive deletion of material. This small amount of volume loss is largely due to our nonconservative edge-collapse operations. Simulations that do not need to collapse small edges typically retain more than 99% of their volume. If volume conservation were especially important, we could enforce incompressibility in our elasticity simulation [Irving et al. 2007].

We have found several benefits to using an explicit embedded surface mesh in our simulations. During the simulation of extreme plastic deformation, we must recompute the FEM mesh often in order to maintain stable basis functions. Without our explicit mesh, temporal continuity across re-meshing events would pose a problem: visible surface smoothing and popping can occur when changing from one conforming FEM mesh to another, which is annoying at best and catastrophic for collision handling at worst. In contrast, our embedded surface mesh remains C^0 continuous with respect to time.

Our re-meshing algorithm executes quickly, but it also allows large simulation time steps. Because the size of the time step is dictated by the worst element in the mesh [Shewchuk 2002], and because every element in our mesh starts in the exact same nearly-optimal shape, we are not limited by a single poorly-shaped tetrahedron. Of course this changes if we decrease the mass of the nodes for improved accuracy, but we have presented the idea of a minimum nodal mass as an intuitive control knob for balancing speed and physical realism.

The biggest limitation of our system is the lack of changes in topology. We can simulate immiscible materials (figure 9), but we cannot yet simulate several objects merging together. In the future, we would like to continue the work started by Brochu [2006] in implementing splitting and merging behaviors with an explicit fluid mesh.

To conclude, our method is efficient and it is capable of simulating a wide variety of materials. With our embedded surface mesh, we can add an arbitrary amount of detail and compute high resolution surface tension forces without increasing FEM resolution. We have also presented an efficient re-meshing algorithm that yields near-perfect tetrahedra, priming the simulation for speed. Our method is also capable of animating surface details like thin sheets and strands that are impossible with other techniques.

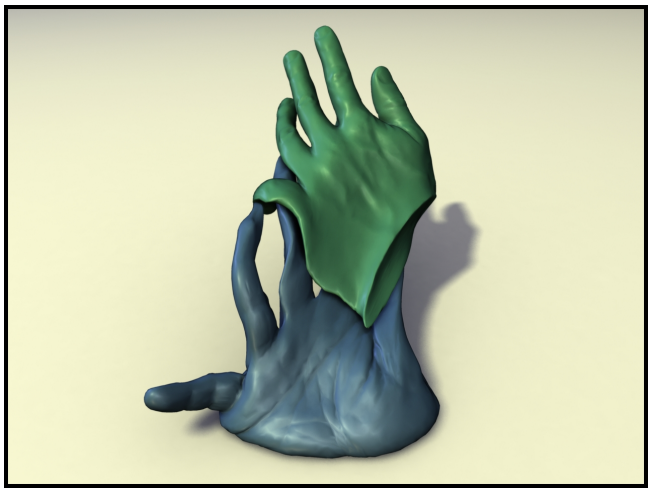


Figure 9: *Two gooey hands stick together in mid flight.*

Acknowledgements

We would like to thank Adam Bargteil, James F. O'Brien, and the members of the Georgia Tech and Carnegie Mellon computer graphics groups for their suggestions and encouragement. This work was funded by an NSF graduate fellowship and by NSF grant CCF-0625264. We are also grateful for an equipment donation from NVIDIA.

References

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Trans. Graph. 24*, 3, 617–625.

BARGTEIL, A. W., GOKTEKIN, T. G., O'BRIEN, J. F., AND STRAIN, J. A. 2006. A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph. 25*, 1, 19–38.

BARGTEIL, A. W., WOJTAN, C., HODGINS, J. K., AND TURK, G. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. 26*, 3, 16:1–16:8.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3, 100:1–100:7.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. 2007. Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3, 339–347.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3, 594–603.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *Proc. Symposium on Computer Animation*, 28–36.

BROCHU, T. 2006. *Fluid Animation with Explicit Surface Meshes and Boundary-Only Dynamics*. Master's thesis, University of British Columbia.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21*, 3, 586–593.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. A multiresolution framework for dynamic

deformations. In *Proc. Symposium on Computer Animation*, 41–47.

CHENTANEZ, N., FELDMAN, B. E., LABELLE, F., O'BRIEN, J. F., AND SHEWCHUK, J. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Proc. Symposium on Computer Animation*, 219–228.

CLAVET, S., BEAUDOIN, P., AND POULIN, P. 2005. Particle-based viscoelastic fluid simulation. In *Proc. Symposium on Computer Animation*, 219–228.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proc. SIGGRAPH*, 317–324.

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2005. A fast and accurate semi-Lagrangian particle level set method. *Computers and Structures 83*, 479–490.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic free-form deformations for animation synthesis. *IEEE TVCG 3*, 3, 201–214.

GALOPPO, N., OTADUY, M., MECKLENBURG, P., GROSS, M., AND LIN, M. 2006. Fast simulation of deformable models in contact using dynamic deformation textures. In *Proc. Symp. on Computer Animation*, 73–82.

GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. 2004. A method for animating viscoelastic fluids. *ACM Trans. Graph. 23*, 3, 463–468.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. In *Proc. Symposium on Computer Animation*, 131–140.

IRVING, G., SCHROEDER, C., AND FEDKIW, R. 2007. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph. 26*, 3, 13:1–13:6.

JIAO, X. 2007. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys. 220*, 2, 612–625.

KEISER, R., ADAMS, B., GASSER, D., BAZZI, P., DUTRÉ, P., AND GROSS, M. 2005. A unified Lagrangian approach to solid-fluid animation. In *the Proceedings of Eurographics Symposium on Point-based Graphics*, 125–133.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph. 26*, 3, 57:1–57:10.

LIEN, S., AND KAJIYA., J. T. 1984. A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE CG&A 4*, 10 (October), 35–41.

LINDSTROM, P., AND TURK, G. 1999. Evaluation of memoryless simplification. *IEEE TVCG 5*, 2, 98–115.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM Trans. Graph. 25*, 3, 812–819.

MOLINO, N., BRIDSON, R., TERAN, J., AND FEDKIW, R. 2003. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *IMR*, 103–114.

MOLINO, N., BAO, Z., AND FEDKIW, R. 2004. A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. 23*, 3, 385–392.

MULLEN, P., MCKENZIE, A., TONG, Y., AND DESBRUN, M. 2007. A variational approach to eulerian geometry processing. *ACM Trans. Graph. 26*, 3, 66.

MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *the Proccedings of Graphics Interface*, 239–246.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *Proc. Symposium on Computer Animation*, 49–54.

MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In *Proc. Symposium on Computer Animation*, 141–151.

MÜLLER, M., TESCHNER, M., AND GROSS, M. 2004. Physically-based simulation of objects represented by surface meshes. In *Computer Graphics International*, 26–33.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3, 471–478.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 191–205.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *the Proceedings of ACM SIGGRAPH 99*, 137–146.

O'BRIEN, J. F., BARGTEIL, A. W., AND HODGINS, J. K. 2002. Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. 21*, 3, 291–294.

PAULY, M., KEISER, R., ADAMS, B., DUTRÉ;, P., GROSS, M., AND GUIBAS, L. J. 2005. Meshless animation of fracturing solids. *ACM Trans. Graph. 24*, 3, 957–964.

REYNOLDS, C. W., 1992. Adaptive polyhedral re-sampling for vertex flow animation, *unpublished*. http://www.red3d.com/cwr/papers/1992/df.html.

RIVERS, A. R., AND JAMES, D. L. 2007. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. 26*, 3, 82:1–82:6.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4, 151–160.

SHEWCHUK, J. R. 2002. What is a good linear element? interpolation, conditioning, and quality measures. In 11^{th} *Int. Meshing Roundtable*, 115–126.

SIFAKIS, E., DER, K. G., AND FEDKIW, R. 2007. Arbitrary cutting of deformable tetrahedralized objects. In *Proc. Symposium on Computer Animation*, 73–80.

SIFAKIS, E., SHINAR, T., IRVING, G., AND FEDKIW, R. 2007. Hybrid simulation of deformable solids. In *Proc. Symposium on Computer Animation*, 81–90.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Deformable models. *The Visual Computer 4*, 306–331.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *the Proceedings of ACM SIGGRAPH 1988*, 269–278.

TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. 1989. Heating and melting deformable models (from goop to glop). In *the Proceedings of Graphics Interface*, 219–226.