



A Finite Element Method for Animating Large Viscoplastic Flow

Adam W. Bargteil
Carnegie Mellon

Chris Wojtan
Georgia Institute of Technology

Jessica K. Hodgins
Carnegie Mellon

Greg Turk
Georgia Institute of Technology

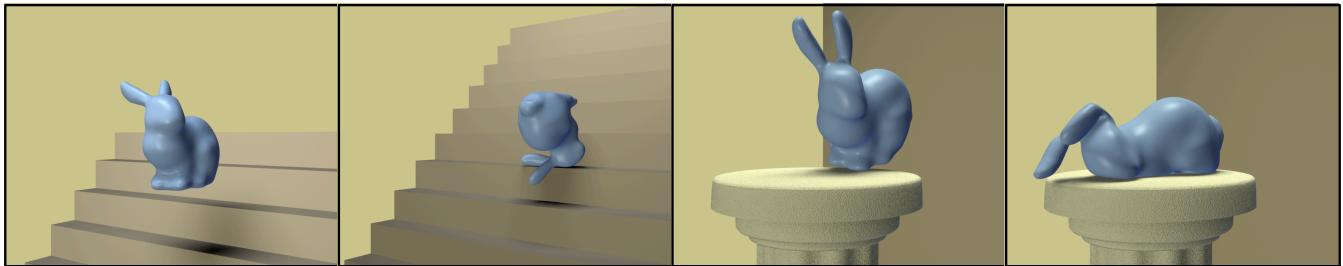


Figure 1: A bunny is dropped down a flight of stairs. Due to work softening, portions of the bunny become soft and droop.

Abstract

We present an extension to Lagrangian finite element methods to allow for large plastic deformations of solid materials. These behaviors are seen in such everyday materials as shampoo, dough, and clay as well as in fantastic gooey and blobby creatures in special effects scenes. To account for plastic deformation, we explicitly update the linear basis functions defined over the finite elements during each simulation step. When these updates cause the basis functions to become ill-conditioned, we remesh the simulation domain to produce a new high-quality finite-element mesh, taking care to preserve the original boundary. We also introduce an enhanced plasticity model that preserves volume and includes creep and work hardening/softening. We demonstrate our approach with simulations of synthetic objects that squish, dent, and flow. To validate our methods, we compare simulation results to videos of real materials.

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.

Keywords: Natural phenomena, physically based animation, deformable models, finite element methods, computational fluid dynamics, viscoplastic, elastoplastic, viscoelastic.

{adamw, jkh}@cs.cmu.edu, {wojtan, turk}@cc.gatech.edu.

From the ACM SIGGRAPH 2007 conference proceedings.

Copyright © 2007 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page or intial screen of the document. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

ACM SIGGRAPH 2007, San Diego, CA

1 Introduction

Over the last decade, computer graphics researchers have developed simulation methods to animate such materials as rigid bodies, thin shells, water, smoke, cloth, and hair. While initial approaches focused on such simplified material models as ideal fluids (no elastic deformation) and ideal solids (no plastic deformation), recently researchers have begun developing techniques to simulate materials that behave according to more sophisticated and physically realistic models. Materials that incorporate both plastic and elastic deformations such as chewing gum, toothpaste, shaving cream, shampoo, bread dough, and modeling clay are frequently encountered in everyday life, while more fantastic materials, such as exploding marshmallow men, melting faces, and slime monsters can be found in special effects. We present extensions to standard finite element methods that allow these techniques to simulate the large plastic flow required to create such examples.

Finite element methods are commonly used to animate elastic bodies in computer graphics. These methods are particularly well-suited to elastic deformations because they compute an explicit deformation function, thereby allowing them to reverse deformations exactly. However, the deformation function becomes ill-conditioned in the presence of large plastic flow and causes the simulation to become numerically unstable. In this paper, we present an extension to Lagrangian finite element methods to allow for large plastic deformations of solid materials. We account for plastic deformation by explicitly updating the linear basis functions defined over the finite elements. Unfortunately, these updates may cause the basis functions to become ill-conditioned. When that occurs, the simulation domain is remeshed and a new high-quality finite-element mesh is constructed. The resulting mesh preserves the details of the boundary while improving the quality of the tetrahedral elements. After remeshing, simulation variables are transferred to the new mesh.

We also introduce an enhanced plasticity model for simulating large viscoplastic deformations. Our model guarantees that the plastic deformation preserves volume, incorporates time-dependence (*viscoplasticity*), and includes work hardening or softening, which occurs when plastic deformation increases or lowers resistance to further plastic deformation.

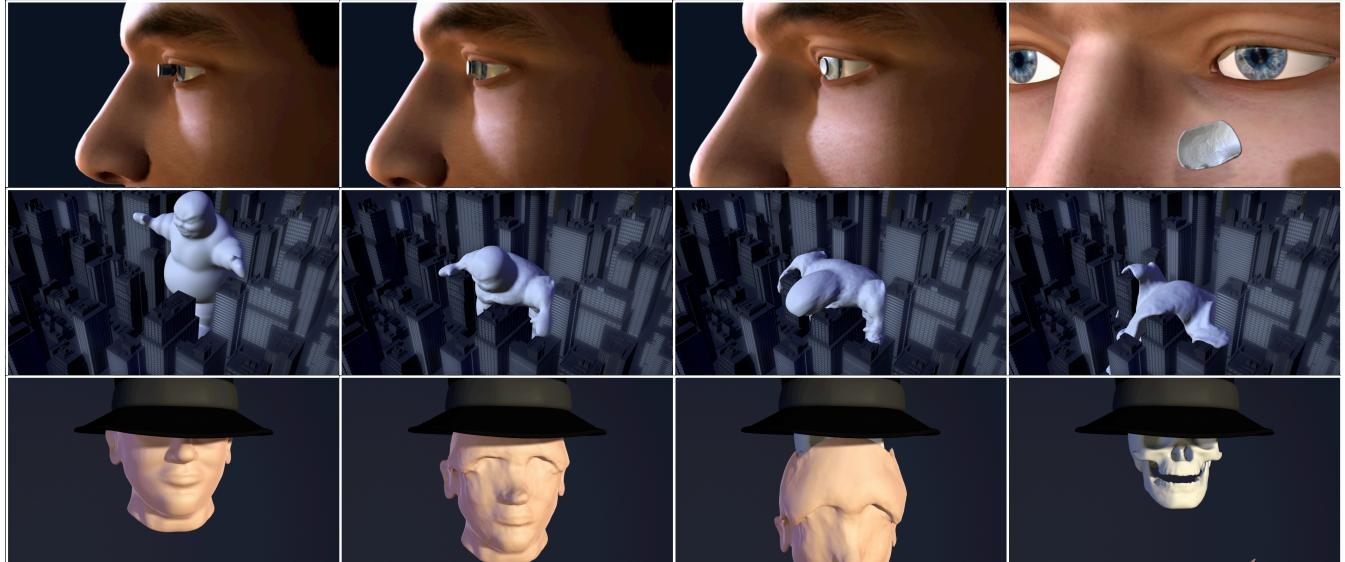


Figure 2: Several animations inspired by special effects. The material properties of the marshmallow man (middle) were changed, producing a noticeable change in behavior part way through the animation to mimic “crossing the streams.”

Our method produces realistic animations of a variety of materials that exhibit complicated behavior, as can be seen in Figures 1 and 2. We also compare our simulation results with real-world video of bread dough and a cornstarch solution.

2 Related Work

Several approaches for simulating viscoplastic materials already exist. Terzopoulos and Fleischer [1988] introduced viscoplastic materials to the computer graphics community. This work was extended by Terzopoulos et al. [1989] to model changes in material properties brought on by heating. Since this early work, three primary approaches have been developed for modeling viscoplastic materials: Eulerian methods, finite element methods, and meshless methods.

Eulerian methods were introduced to graphics by Goktekin et al. [2004] who added elastic forces to a standard fluid simulator. In this approach, the deformation rate is computed from the gradient of the velocity field and then integrated across time to arrive at a measure of deformation. They then applied elastic forces proportional to the divergence of the computed deformation. Naturally, this approach is extremely useful for animating materials that largely behave like fluids but demonstrate limited elastic behavior. However, because no direct deformation function is available and the volumetric shape of the object is not explicitly represented, Eulerian methods cannot reverse deformations exactly, and the shape will drift over time. This approach was also used by Losasso et al. [2006] who extended the model to account for rotation of the advected elasticity.

Finite element methods are commonly used to compute the motion of ideal elastic solids. O’Brien et al. [2002] extended these methods with a simple plasticity model, thereby increasing the range of materials that could be simulated. However, the numerical calculation becomes ill-conditioned when the material undergoes substantial elastic or plastic deformation, a situation that causes the simulation to become

unstable. More recently, Irving et al. [2004], built on the work of Müller et al. [2002] and Müller and Gross [2004], to create a finite element method that robustly handles inverted elements, thereby making finite element methods much more attractive for computer graphics applications. Their approach uses the singular value decomposition to separate the rotational and scale components of the deformation gradient. Arbitrary elastic deformation can be handled by constraining the resulting deformation gradient matrix to be well-conditioned. Unfortunately, this approach does not address the problems that result from ill-conditioned basis matrices. Consequently, it can handle only limited amounts of plastic flow. The primary contribution of our paper is to maintain well-conditioned basis matrices by recomputing the simulation mesh. Although remeshing during finite element simulations has not been considered in the graphics literature, it has been considered in other disciplines [Borouchaki et al. 2005; Mauch et al. 2006]

An interesting alternative to finite element methods was proposed by Clavet et al. [2005]. In their approach, an object is treated as a mass spring system in which the springs are dynamically inserted and removed. This process is similar in spirit to our remeshing procedure. Their springs explicitly model viscous and elastic forces and include a model of plastic flow. They demonstrate a wide range of materials handled by their method.

Meshless simulation methods have proven to be extremely versatile and are capable of simulating a wide variety of materials and phenomena. Early work in meshless methods was done by Desbrun and Gauscuel [1995], who calculated elastic forces using dynamically determined neighborhoods to allow behavior that is similar to plastic flow. Müller et al. [2004] described an effective meshless method for simulating elasto-plastic materials. For small plastic flow, they store the built-up plastic strain and then remove it before computing elastic forces. For situations with large plastic flow, they switch to a method that stores and maintains the elastic strain instead of the plastic strain. They update the elastic strain every timestep by advecting the particles and adding the elastic

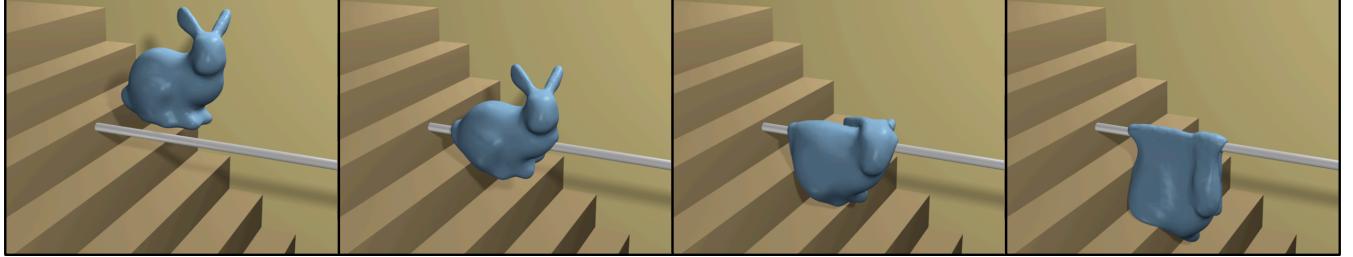


Figure 3: A bunny is dropped on a metal bar. Due to work hardening, portions of the bunny become more elastic and jiggle.

strain generated during the timestep. This approach is similar to the Eulerian methods, but rather than integrate deformation rate across time, as in Eulerian methods, they directly compute deformation during each timestep. However, because they cannot directly compute total deformation, errors in the accumulated deformation may build up over time. Nevertheless, their approach produces excellent results and handles a wide range of materials. Pauly et al. [2005] used a similar approach to plasticity in their work on modeling fracturing materials. They also introduced the idea of updating the shape functions for each simulation particle, although their updates were prompted by changes in topology rather than plastic flow. Keiser et al. [2005] introduced a unified meshless approach to materials running the gamut from solids to fluids. To achieve creep, they enhanced the moderate plasticity model proposed by Müller et al. [2004] with a flow rate.

3 Dynamics Simulation

Our goal is to simulate elastic deformable bodies that undergo very large viscoplastic flow. We begin with a Lagrangian finite element method for elastic bodies based on the one presented by Irving et al. [2004]. We then incorporate a new viscoplasticity model that explicitly updates the basis matrices used by the finite element method. Finally, when these basis matrices become ill-conditioned, we use a tetrahedral meshing technique to generate a new high-quality simulation mesh and transfer simulation variables to this new mesh.

3.1 Finite Element Method

Our finite element simulations require objects discretized into volumetric tetrahedral meshes. Each tetrahedral element has a three-dimensional linear basis that describes its deformation. We denote the world coordinates of the four nodes that define an element by \mathbf{x}_0 , \mathbf{x}_1 , \mathbf{x}_2 , and \mathbf{x}_3 . For every tetrahedron, we then define a matrix \mathbf{X} such that $X_{ij} = x_{ij} - x_{0j}$, where $i = 1, 2, 3$ varies over three of the four nodes in the element and $j = 1, 2, 3$ varies over the three dimensions of space. At the start of the simulation we compute the basis matrix $\boldsymbol{\beta} = \mathbf{X}_0^{-1}$. Given a deformed state, \mathbf{X} , we can then compute the deformation gradient \mathbf{F} for the element as

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial u} = \mathbf{X} \boldsymbol{\beta}. \quad (1)$$

Following Irving et al. [2004] we diagonalize \mathbf{F}

$$\mathbf{F} = \mathbf{U} \hat{\mathbf{F}} \mathbf{V}^T, \quad (2)$$

where U and V are rotations, and compute the first Piola-Kirchhoff stress $\hat{\mathbf{P}}$

$$\hat{\mathbf{P}} = 2\mu(\hat{\mathbf{F}} - \mathbf{I}) + \lambda \text{Tr}(\hat{\mathbf{F}} - \mathbf{I}) \mathbf{I} \quad (3)$$

where λ and μ are material parameters. Finally, we can compute forces due to a single element on node n_i as

$$\mathbf{g}_i = \mathbf{U} \hat{\mathbf{P}} \mathbf{V}^T \sum_j A_j \mathbf{N}_j \quad (4)$$

where the $A_j \mathbf{N}_j$ are the area weighted normals of the faces incident to node n_i , in the element's rest state. See Irving et al. [2004] for further details.

3.2 Plasticity Model

Classical approaches to plasticity [Hill 1950; Simo and Hughes 1998] generally assume an additive model of plasticity, where total strain is divided into plastic and elastic parts:

$$\epsilon_{total} = \epsilon_e + \epsilon_p. \quad (5)$$

This basic model has been adopted by a number of computer graphics researchers [O'Brien et al. 2002; Goktekin et al. 2004; Müller et al. 2004; Keiser et al. 2005; Pauly et al. 2005]. A fundamental calculation in this approach is the computation of the *strain deviation tensor*:

$$\epsilon' = \epsilon - \text{Tr}(\epsilon) \mathbf{I}. \quad (6)$$

The plastic strain is then computed as a percentage of ϵ' . This computation is motivated by the desire to prevent plastic deformation from producing a change in volume, and it is accurate for infinitesimal strains. However, as pointed out by Simo and Hughes [1998] and in graphics by Irving et al. [2004], this decomposition loses its physical meaning for finite strains. A multiplicative plasticity model is more appropriate:

$$\mathbf{F} = \mathbf{F}_e \cdot \mathbf{F}_p. \quad (7)$$

By forcing $\det(\mathbf{F}_p) = 1$, we can guarantee that plastic deformation does not change the volume of the element. Intuitively, this constraint means that when an element becomes thinner in one dimension, other dimensions will grow in such a way that the total volume of the element is conserved.

To compute the plastic strain for a given element, we first use the diagonalized deformation gradient $\hat{\mathbf{F}}$ to compute

$$\hat{\mathbf{F}}^* = (\det(\hat{\mathbf{F}}))^{-1/3} \hat{\mathbf{F}}, \quad (8)$$

where $\det(\hat{\mathbf{F}}^*) = 1$. We then compute

$$\hat{\mathbf{F}}_p = (\hat{\mathbf{F}}^*)^\gamma, \quad (9)$$

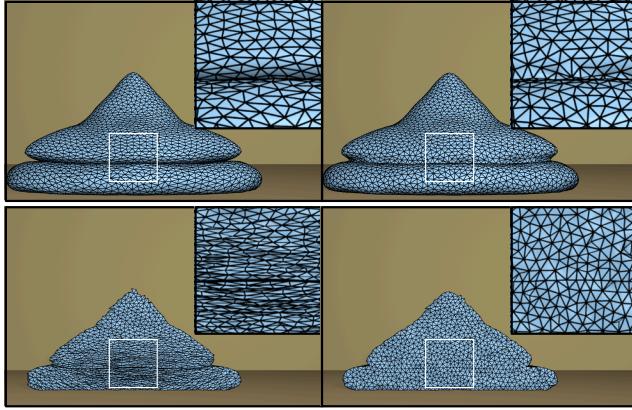


Figure 4: Views of the simulation mesh before and after remeshing. The left column shows the meshes before remeshing; the right column shows the meshes directly after remeshing. The top images show the surface mesh and the bottom images show a cross section of the volume mesh. Notice that remeshing automatically merges the two objects.

where γ is a function of the current stress (\mathbf{P}), the yield stress (P_Y), the flow rate (ν), and hardening parameters (α , K). Because $\hat{\mathbf{F}}^*$ is diagonal, we can easily perform exponentiation by taking the power of the values on the diagonal. The exponent $\gamma(\cdot)$ takes the form

$$\gamma(\mathbf{P}, P_Y, \nu, \alpha, K) = \min\left(\frac{\nu(\|\mathbf{P}\| - P_Y - K\alpha)}{\|\mathbf{P}\|}, 1\right). \quad (10)$$

The $K\alpha$ term in this model enables work hardening and softening. As the simulation progresses, α , which is initialized to zero, is incremented in each element by the norm of the stress,

$$\dot{\alpha} = \|\mathbf{P}\|. \quad (11)$$

The parameter K controls the amount of work hardening or softening. Positive K results in hardening, while negative K results in softening. Work hardening refers to an increase in mechanical strength due to plastic deformation and occurs in many metals. In contrast, materials that experience work softening flow more easily after undergoing plastic deformation. Although materials that demonstrate work softening are somewhat less common in the real world, we expect that such materials will be useful in special effects. This plasticity model has the following three properties: it preserves volume, allows work hardening/softening, and incorporates viscoplasticity or creep with a flow rate parameter.

After the plastic deformation has been computed, we update the basis functions to remove this permanent deformation,

$$\boldsymbol{\beta} := \mathbf{V}\hat{\mathbf{F}}_p^{-1}\mathbf{V}^T. \quad (12)$$

We multiply by the rotation matrix \mathbf{V}^T rather than \mathbf{U}^T to avoid rotating $\mathbf{F} = \mathbf{X}\boldsymbol{\beta}$ when we update the basis matrix. After the basis matrix has been updated, the object no longer has a rest configuration. Although each individual element does have a rest state, which can be found by inverting $\boldsymbol{\beta}$, these rest states are generally inconsistent. Because updating $\boldsymbol{\beta}$ changes the rest state, we must also recompute the area-weighted face-normals used in Equation (4) based on the new rest state. This relationship between plastic deformation and the direction of internal forces is ignored by plasticity models that do not update the finite-element basis functions.

4 Remeshing Deforming Objects

After substantial plastic flow, the $\boldsymbol{\beta}$ matrices will become ill-conditioned. The diagonalization approach introduced by Irving et al. [2004] is effective for handling degenerate and inverted finite elements, but it is not robust to ill-conditioned basis functions. Our solution to this problem is to generate a new tetrahedral mesh when the condition number of the worst $\boldsymbol{\beta}$ matrix exceeds a threshold or when the condition number of a $\boldsymbol{\beta}$ matrix has changed significantly since the last remesh. Remeshing is done between timesteps and, consequently, is independent of the timestep size. See Figure 4 for visualizations of a mesh before and after remeshing.

Our remeshing procedure is based on the variational tetrahedral meshing approach introduced by Alliez et al. [2005]. This approach treats tetrahedral mesh generation as an optimization problem. The following pseudo-code summarizes the approach:

1. Read the input boundary mesh $\partial\Omega$
2. Compute sizing field η
3. Generate initial sites x_i inside $\partial\Omega$
4. Do
5. Construct Delaunay triangulation($\{x_i\}$)
6. Move sites x_i to their optimal positions x_i^*
7. Until (convergence or stopping criterion)
8. Extract interior mesh

The approach of Alliez and colleagues has also been successfully used for smoke simulations by Klingner et al. [2006]. However, our problem of remeshing deforming objects differs from the problems considered by these authors. Alliez et al. [2005] created a static mesh from a high-resolution polygonal mesh and did not address deforming objects. Klingner et al. [2006] considered deforming meshes, but a secondary high-resolution representation of the mesh boundary allowed them to resample the boundary every timestep without smoothing it. During our simulations of plastic flow we have no canonical shape to refer to, and the boundary of the tetrahedral mesh for the current timestep is the *only* representation of the boundary. To avoid excessive surface smoothing and other artifacts, we must take extra care in dealing with the boundary during remeshing.

Handling the boundary is one of the most important and difficult problems in computer graphics simulations, regardless of the simulation method selected. In grid-based Eulerian liquid simulation, researchers have struggled with representing the liquid surface and a wide variety of techniques have been proposed [Enright et al. 2002; Zhu and Bridson 2005; Hieber and Koumoutsakos 2005; Bargteil et al. 2006]. In meshless simulation approaches [Müller et al. 2004; Pauly et al. 2005], researchers generally do not represent the surface with simulation particles. Instead, a high density of surface particles, called *surfels*, are used to ensure a high-quality representation. In our method, we have modified the approach of Alliez and colleagues [2005] to handle deforming boundaries. In particular, we have modified their approach to handling the boundary during vertex optimization (step 6) and when extracting the interior mesh (step 8). Throughout this discussion we will refer to the poor-quality input mesh as the *old mesh* and the mesh generated by the remeshing procedure as the *new mesh*.

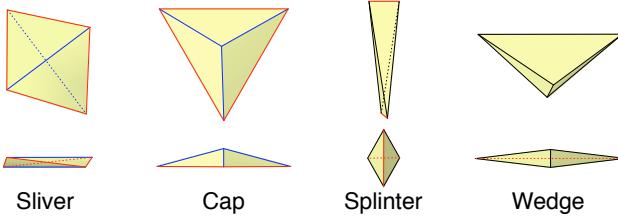


Figure 5: This figure illustrates the four types of tetrahedra we attempt to remove. Small dihedral angles are shown in red and large dihedral angles are shown in blue.

4.1 Optimizing Boundary Vertices

When optimizing vertex locations, vertices on the boundary must be specially handled. Alliez et al. [2005] suggest that the initial surface be finely sampled. Then, during each optimization step, each surface sample s_i searches for the nearest mesh vertex, x_i . Each mesh vertex is then moved to a weighted average of the s_i for which it was the closest mesh vertex (if any such s_i exist). This approach ensures that mesh vertices that are close to the boundary are snapped to points near the boundary, making it likely that the Delaunay triangulation does not include faces that intersect the boundary, but rather includes faces that lie on the boundary. This procedure resamples and smoothes the surface as a side effect. To avoid unwanted smoothing, we apply this procedure only if it moves additional vertices to the surface. In this way we guarantee that no vertices get very close to the boundary without being snapped to the boundary, but we also limit surface smoothing.

4.2 Extracting the Mesh

The Delaunay triangulation creates tetrahedra for the entire convex hull of the point set x_i , but many of these tetrahedra are outside of our nonconvex simulated object and must be deleted. Our goal is to extract a mesh whose surface closely matches the surface of the old mesh. Additionally, we would like the tetrahedra in the new mesh to be high quality. Finally, we require that the surface mesh be manifold. To achieve these goals we perform the following steps:

1. Remove elements outside the surface.
2. Remove poorly shaped elements.
3. Add and remove elements to ensure that the surface is manifold.

The first step is accomplished by computing the volume overlap of each element in the new mesh with elements in the old mesh. We make use of a hierarchical axis-aligned bounding box structure to speed up these computations. Any element that has overlap volume less than half of its volume in the new mesh is removed. After this step, at least half of every element remaining in the new mesh overlaps the old mesh.

Because the meshing procedure is not free to move boundary vertices, it tends to generate slivers and other ill-conditioned elements on the boundary. These elements contain very little volume but negatively affect the simulation; therefore we delete poorly shaped elements in the second step if their removal does not introduce surface artifacts.

We begin by identifying all ill-conditioned elements and classifying them based upon the number and location of their

small dihedral angles. We identify four types of tetrahedra (see Figure 5), using the naming scheme introduced by Cheng et al. [1999]:

- *Slivers* contain four small dihedral angles.
- *Caps* contain three edges with small dihedral angles, all incident to one triangle.
- *Splinters* contain two non-incident edges with small dihedral angles.
- *Wedges* contain exactly one small dihedral angle.

Other types of poorly shaped tetrahedra exist, but as pointed out by Shewchuk [2002], only elements with small dihedral angles negatively affect conditioning, so we ignore the other poorly shaped elements. An edge has a small dihedral angle if the inner product of the unit normals of the two incident faces is less than some threshold (in our implementation, -0.8). Correct classification is extremely important because the rules determining whether a tetrahedron can be safely deleted differ for each class of tetrahedra. Thus, we not only consider how many small dihedral angles a tetrahedron has but also the relative location of these dihedral angles. For example, if a given tetrahedron has three small dihedral angles but they are not all incident to a single face, we treat this tetrahedron as a sliver rather than a cap. Similarly, if a tetrahedron has two small dihedral angles around a face, we treat the tetrahedron as a cap or a sliver depending on whether the face's third dihedral angle is less than or greater than 90°.

After classifying the poorly shaped tetrahedra, we attempt to delete them. Each class of tetrahedra can be deleted if certain criteria are met:

- *Slivers* may be deleted if two faces that share an edge with a large dihedral angle are on the surface.
- *Caps* may be deleted if the face incident to the three small dihedral angles is on the surface.
- *Splinters* may be deleted if any faces are on the surface.
- *Wedges* may be deleted if either face incident to the small dihedral angle is on the surface.

After deleting an element the surface may contain cosmetic artifacts, such as small dimples, but these rules ensure the surface does not contain folds or other artifacts that will affect the stability of the simulation. The algorithm cycles through the lists of classified tetrahedra until no more elements can be safely deleted. Often a handful of slivers remain.

The third step ensures that the surface mesh is manifold. Having a manifold surface mesh avoids problems with self-collisions and allows the creation of signed-distance fields. We guarantee that the surface is manifold by ensuring that each vertex and each edge in the surface mesh is manifold.

Non-manifold vertices are rare but do occur, particularly when the mesh becomes very thin or objects merge. A non-manifold vertex can be identified by considering graphs where the nodes correspond to tetrahedra and where there is an edge between two nodes if the corresponding tetrahedra share a face. There are two such graphs: one constructed from tetrahedra adjacent to the vertex in the current mesh, the other constructed from tetrahedra that have been deleted but were adjacent to the vertex before deletion. The vertex is manifold if, and only if, both these graphs are connected. If one of the graphs is not connected, we remove the smallest connected component. To remove a connected

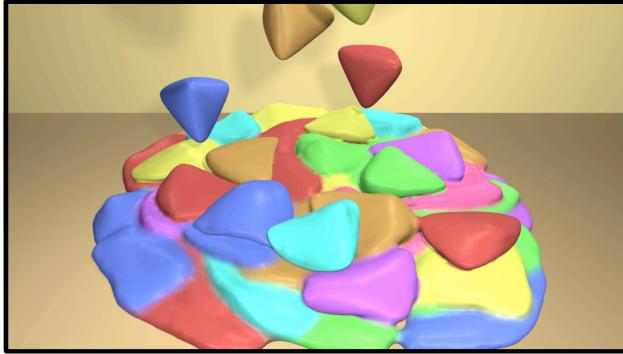


Figure 6: An image from an animation with many topological merges. Each object contains only 4200 elements, but by the end of the animation the system contains more than 200,000 elements.

component from the graph constructed from tetrahedra currently in the mesh, we delete the tetrahedra corresponding to the nodes in the connected component. To remove a connected component from the graph constructed from the deleted tetrahedra, we return tetrahedra to the mesh.

We address non-manifold edges by examining a topological structure called the *link* [Dey et al. 1999]. The *link* of an edge is an ordered list of mesh vertices connected to both endpoints of the edge. Any two consecutive vertices in this list, together with the two edge vertices, specify one of the tetrahedra that surround the edge. If a link has no gaps, the edge is in the interior of the mesh. If a link has one gap, then the edge is on the surface. A link with more than one gap indicates a non-manifold edge. We examine the link of every edge in the mesh. If a non-manifold edge is found, tetrahedra are either inserted or removed, depending on the quality of the potentially added tetrahedra and the volume of the potentially deleted tetrahedra, until the edge is once again manifold.

We repeatedly search for and fix non-manifold vertices and edges until none remain. We avoid infinite loops by disallowing the addition or deletion of tetrahedra that have already been considered. To guarantee that progress can always be made, we always allow the removal of tetrahedra to fix non-manifold vertices, even if these tetrahedra were previously added to fix some other topological problem. We do not have a proof that this process terminates, but we have not found that to be a problem in practice.

During this step of extracting the mesh interior, the topology of the new mesh is created. If in the old mesh two regions were touching or nearly touching, the remesher may add tetrahedra between them, thereby joining the surfaces. Similarly, if a region of the mesh has become very thin, its elements may be deleted, resulting in a hole or tear. Thus, topological changes are handled implicitly by our remeshing procedure. This approach is similar to the level set approaches employed by grid-based Eulerian methods and suffers from the same drawbacks. Explicitly dealing with topological changes, perhaps with the method developed by Brochu [2006], is an area of future work.

4.3 Transferring Simulation Variables

After constructing a new simulation mesh, we transfer the simulation variables from the old mesh to the new one.

There are two types of simulation variables: those stored per element and those stored per node. The per element variables that we must transfer are the deformation gradient \mathbf{F} and the work hardening/softening variable α . Given a tetrahedron in the new mesh, we compute its simulation variables by finding all the tetrahedra in the old mesh with which it overlaps. We then set the new variable to be the average of the simulation variables stored in the old tetrahedra, weighted by the volume of the overlap with each old tetrahedron. While averaging is straightforward for the scalar α , transferring \mathbf{F} is more difficult.

Instead of transferring \mathbf{F} directly, we compute and transfer Green's strain, $\mathbf{G} = 1/2(\mathbf{F}^T \mathbf{F} - \mathbf{I})$. \mathbf{F} can then be computed in each of the new elements by performing an Eigen decomposition of $2\mathbf{G} + \mathbf{I}$. This approach of averaging strain has been used before in the graphics literature by Goktekin et al. [2004], Müller et al. [2004], and Losasso et al. [2006].

After we have transferred \mathbf{F} to the new mesh, we compute a β matrix for each element:

$$\boldsymbol{\beta} = \mathbf{X}^{-1} \mathbf{F}. \quad (13)$$

We then compute the rest state for each element. Specifically, we compute area-weighted face normals (used in the computation of forces, see Equation (4)) and desired volume for each element. Finally, we compute the mass of each node.

We also need to transfer variables stored per node; velocities are the only such variable in our system. If a node in the new mesh sits inside a tetrahedron in the old mesh, we use barycentric interpolation to compute the value for the new node. Otherwise, the new node lies slightly outside the old mesh. In this case, we find the nearest point to the new node on the old surface and interpolate the velocity there.

5 Implementation Details

For self-collisions or collisions between finite element meshes, we run the cloth collision algorithm of Bridson et al. [2002] on the surface of our simulation mesh. This algorithm ensures that our tetrahedral meshes will never self-intersect, an important invariant required to avoid complications during re-meshing. For collisions with rigid bodies and immovable objects, we project penetrating vertices onto the surface of the object and apply friction, similar to the procedure in Irving et al. [2004].

We integrate our simulations through time with the variable timestep Newmark integrator used by Bridson et al. [2003]. To decide when to reduce the timestep in our adaptive integrator, we check for drastic changes in the edge lengths of any tetrahedron. When an edge length has changed by more than some threshold within a single timestep, we back up the simulation and divide the timestep in half. The volume preservation and extra damping due to high plasticity make it difficult to detect instabilities merely by inspecting edge lengths, so we also reduce the timestep whenever sudden accelerations occur. After several successful simulation steps have been executed without encountering instabilities, we double the timestep size.

When computing forces, we do not compute plastic updates for inverted or degenerate elements, similar to how Irving et al. [2004] sacrificed accuracy for stability by altering the elastic forces for inverted elements. This adjustment helps to ensure the stability of our simulations by avoiding unreliable force calculations on elements with negligible volume.

6 Results

We have used our finite element approach to simulate materials that exhibit a wide range of behaviors. The figures and the accompanying video demonstrate our results.

We demonstrate work softening by dropping a bunny down a set of stairs in Figure 1. Each time the bunny strikes a stair, that portion of the bunny softens. By the time the bunny comes to rest, several body parts are very soft while others remain hard. Figure 2 shows several examples of our method inspired by special effects. The models in these examples have on the order of 100,000 elements and interact with complicated collision geometries. Figure 3 shows another example with the bunny model this time falling onto a metal bar. This example also demonstrates work hardening; at the beginning of the simulation the bunny easily deforms plastically, but after limited flow it deforms only elastically.

We have tested our method on examples with more than 200,000 elements (Figure 6). This particular example took more than a week to run on a 3.0 GHz Pentium 4, remeshed about 75 times, and took roughly 13% of the total simulation time for remeshing. The running times of our implementation are long, in part because we have not removed debugging code or optimized our implementation. After optimization, we expect the running speed to be only 10-20% longer than that of Irving et al. [2004]. This additional time should be enough to perform periodic remeshing and the additional matrix operations required to update the rest state.

To test the tunability of our material parameters, we used a high-speed camera (1000 fps) to record the motion of a star-shaped piece of dough as it strikes the ground (Figure 7 bottom). By adjusting the flow rate and the yield stress parameters, we were able to match the basic behavior in a simulation, (Figure 7 top). Note that the material parameters do not need to be kept constant. Some materials, such as cornstarch, will act either in an elastic or in a flowing manner depending on the stress. Figure 8 shows a real (bottom) and a simulated (top) ball of cornstarch. The material bounces when it first strikes the ground due to the high stress, but when it comes to rest it flows readily. To achieve this behavior we extended our plasticity model by varying the flow rate, ν , based on stress, P . At high stresses the flow rate was low and at low stresses the flow rate was high.

7 Discussion

In this work, we have chosen to use a linear strain model, primarily for stability reasons. However, our general approach of updating the elements' basis functions and remeshing and our plasticity model are not dependent on this choice and would apply equally well to St. Venant-Kirchhoff or other strain models as well as to non-linear stress-strain relationships. We also note that our plasticity model is easily generalized by replacing our constants with functions as was done for the cornstarch example (Figure 8).

While we have found our particular remeshing strategy sufficient, we acknowledge that it is far from perfect for solving our problem. For example, though we were careful to preserve the mesh boundary during remeshing, each remeshing step does resample the surface, smoothing it as a side effect. Overly aggressive remeshing does produce noticeable artifacts; however, these artifacts are largely hidden by flow and other motion in the case of moderate remeshing. These problems suggest two areas of future work. Because we can

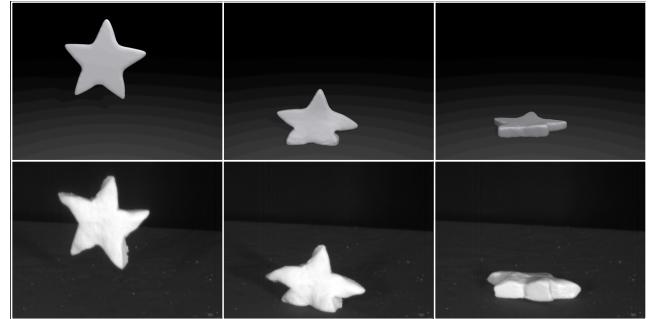


Figure 7: A comparison between simulated (top) and real (bottom) bread dough.

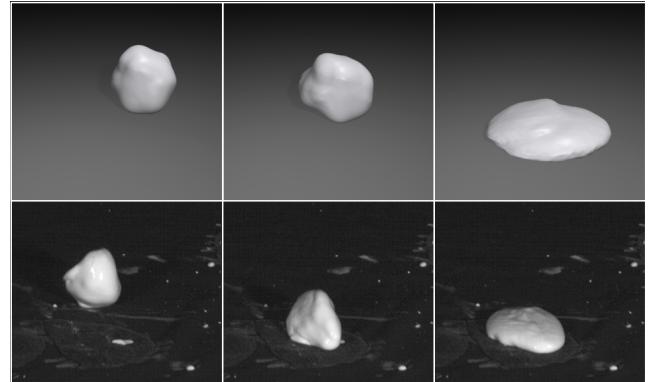


Figure 8: A comparison between simulated (top) and real (bottom) cornstarch solutions. The first bounce induces only elastic deformations; however, under smaller stress, the solution flows, forming a puddle.

measure locally how much volume is lost, we would like to find a way to replace this lost volume. Alternatively, we could use a secondary, high-resolution representation of the surface. The general meshing problem has been receiving more attention in the computational geometry community and the recent work of Shewchuk [2005], Labelle [2006], and Hudson et al. [2006] might lead to alternative approaches to remeshing in finite element simulations.

Several approaches have been used for animating materials that undergo large plastic flow, including Eulerian grids and particle-based approaches. We offer a new approach that uses tetrahedral finite elements, periodic re-meshing, and deformation state inheritance to simulate elastic and plastic deformation. Each of these approaches have different trade-offs in terms of advantages and drawbacks. Our finite element approach can more easily simulate highly elastic and nearly rigid objects than can Eulerian grid methods. Because we retain connectivity between elements, our approach does not incur the cost of nearest neighbor searches that tend to dominate particle-based approaches.

Acknowledgements

We would like to thank Geoffrey Irving for answering our questions, James O'Brien and Bryan Klingner for sharing their code with us, Jonathan Shewchuk for sharing his *Pyramid* software, and Moshe Mahler for sharing his modeling expertise. Thanks to Autodesk for the donation of *Maya*.

licenses. Most of our animations were computed and rendered on a cluster donated by Intel. Several of our animations were rendered with Pixie. Partial funding for this work was provided by NSF grants CCF-0625264 and IIS-0326322 and ONR DURIP #N00014-06-1-0762. Chris Wojtan was supported by an NSF Graduate Research Fellowship.

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.
- BOROUCHAKI, H., LAUG, P., CHEROUAT, A., AND SAA-NOUNI, K. 2005. Adaptive remeshing in large plastic strain with damage. *International Journal for Numerical Methods in Engineering* 63, 1 (February), 1–36.
- 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 *The Proceedings of the ACM SIGGRAPH/Eurographics 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.
- CHENG, S.-W., DEY, T. K., EDELSBRUNNER, H., FACELLO, M. A., AND TENG, S.-H. 1999. Sliver exudation. In *The Proceedings of the Symposium on Computational Geometry*, 1–13.
- CLAVET, S., BEAUDOIN, P., AND POULIN, P. 2005. Particle-based viscoelastic fluid simulation. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 219–228.
- DESBRUN, M., AND GASCUEL, M.-P. 1995. Animating soft substances with implicit surfaces. In *The Proceedings of ACM SIGGRAPH*, 287–290.
- DEY, T., EDELSBRUNNER, H., GUHA, S., AND NEKHAYEV, D. V. 1999. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S.)* 66, 23–45.
- ENRIGHT, D. P., MARSCHNER, S. R., AND FEDKIW, R. P. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph.* 21, 3, 736–744.
- 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.
- HIEBER, S. E., AND KOUMOUTSAKOS, P. 2005. A Lagrangian particle level set method. *J. Comp. Phys.* 210, 1, 342–367.
- HILL, R. 1950. *The Mathematical Theory of Plasticity*. Oxford University Press, Inc.
- HUDSON, B., MILLER, G., AND PHILLIPS, T. 2006. Sparse Voronoi Refinement. In *The Proceedings of the 15th International Meshing Roundtable*.
- IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 131–140.
- 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.
- KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. *ACM Trans. Graph.* 25, 3, 820–825.
- LABELLE, F. 2006. Sliver removal by lattice refinement. In *The Proceedings of the ACM Symposium on Computational Geometry*, 347–356.
- LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM Trans. Graph.* 25, 3, 812–819.
- MAUCH, S., NOELS, L., ZHAO, Z., AND RADOVITZKY, R. 2006. Lagrangian simulation of penetration environments via mesh healing and adaptive optimization. In *The Proceedings of the 25th Army Science Conference*.
- MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *The Proceedings of Graphics Interface*, 239–246.
- MÜLLER, M., DORSEY, J., McMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *The Proceedings of the ACM SIGGRAPH/Eurographics 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 *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 141–151.
- 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.
- SHEWCHUK, J. R. 2002. What is a good linear element? Interpolation, Conditioning, and Quality Measures. In *11th Int. Meshing Roundtable*, 115–126.
- SHEWCHUK, R. 2005. Star splaying: an algorithm for repairing Delaunay triangulations and convex hulls. In *The Proceedings of the Symposium on Computational Geometry*, 237–246.
- SIMO, J., AND HUGHES, T. 1998. *Computational Inelasticity*. Springer-Verlag.
- TERZOPoulos, D., AND FLEISCHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *The Proceedings of ACM SIGGRAPH*, 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.
- ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph.* 24, 3, 965–972.