# Two recent papers in physical simulation

Benjamin Schroeder

December 4, 2008

Today we'll discuss two papers:

Fast Viscoelastic Behavior with Thin Features
Chris Wojtan and Greg Turk
SIGGRAPH 2008

Two-way Coupling of Rigid and Deformable Bodies
Tamar Shinar, Craig Schroeder, and Ronald Fedkiw
SCA 2008

Both of these papers are (broadly) about physically-based simulation of deformable objects. The first one presents a technique for animating highly plastic objects which develop thin sheets and strands of material; the second one discusses how to couple deformable-object and rigid-body simulations while retaining the advantages of both.

*Fast Viscoelastic Behavior with Thin Features* describes a model for deformable objects based on a coarse inner mesh and a refined surface mesh. The model is applicable over a wide range of materials.

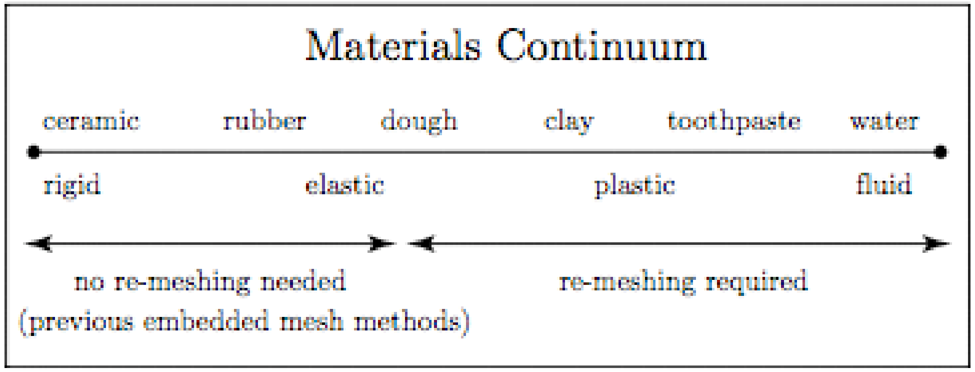


**Figure 2:** *Current FEM methods with an embedded surface mesh only simulate a limited range of materials because ill-conditioned basis functions result from large plastic flow. Our method recomputes basis functions by re-meshing throughout the simulation, allowing us to simulate a much greater range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

The model includes a fast re-meshing scheme; this allows for re-meshing to be performed frequently, which enables the use of the coarse mesh in highly plastic situations.

Most of the simulation takes place using the coarse mesh, for efficiency; several important quantities are computed on the smooth mesh in order to maintain realism.

Objects are modeled (on the coarse level) as tetrahedral finite element meshes.

Each tetrahedral element contributes forces to its nodes depending on how it is deformed from its rest position.

The forces are based on a deformation gradient **F**. The force $\mathbf{g}_i$ on a node i due to some tetrahedron is given by

$$\mathbf{g}_i = -\mathbf{P}(A_1\mathbf{N}_1 + A_2\mathbf{N}_2 + A_3\mathbf{N}_3)/3$$

where
each $A_k$ is the area of a face, and $\mathbf{N}_k$ a normal;
**P** is the first Piola-Kirchoff stress of the node,
which maps from material-space normals to
world-space traction vectors. **P** is a function of **F**.

($A_k$ and $\mathbf{N}_k$ are given in material-space coordinates.)

For more details, including a discussion of particular models for computing the stress, see Irving et al, *Invertible Finite Elements for Robust Simulation of Large Deformation*, 2004.

**F** can be decomposed into an elastic part, which acts to restore nodes to their rest position, and a plastic part, which represents permanent changes.

The plastic part of **F** is given by

$$\hat{\mathbf{F}}_p = (\hat{\mathbf{F}}^*)^\gamma;$$

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

Where

$\hat{\mathbf{F}}^*$ is the volume-conserving portion of the diagonalized deformation tensor;
P is the stress tensor (as before);
$P_Y$ is the plastic yield point,
v is the flow rate,
K is a hardening parameter,
and α represents accumulated plastic stress.

Since plastic change is permanent, the plastic part of F is used to recompute the basis functions for the elements of the tetrahedral mesh, changing the rest state of each element.

For more details, see Bargteil et al, *A Finite Element Method For Animating Large Viscoplastic Flow*, 2007.

The surface of the object is modeled explicitly, at a finer level of detail than the coarse finite element mesh, with a triangular surface mesh. This mesh is used for collision handling and for rendering.

Each vertex in the surface mesh is assigned to a tetrahedral element (which completely encloses it) in the coarse mesh. The vertex position, **x**, is then expressed in terms of the tetrahedral vertex positions $\mathbf{x}_i$,

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

where $\mathbf{b}_i$ are the barycentric coordinates of the surface vertex in material coordinates. This allows the rendering position of the vertex as well as velocities and forces to be expressed in terms of the parent tetrahedron.

Collisions are handled using the surface mesh, by pushing the vertex out of the colliding object; this requires moving the entire parent element. The contribution to each node is weighted; the weight for node *i* is

$$w_i = \frac{b_i}{b_1^2 + b_2^2 + b_3^2 + b_4^2}.$$

The same weights are used to add velocity impulses during friction handling.

Note that a parent element may have many surface vertices associated with it. To avoid conflicts, collision handling is only applied using information from the deepest penetrating vertex. All surface vertices for an element, of course, must be updated during a collision.

Since surface vertices are completely enclosed by their parents, the parents can act as bounding objects. This can greatly speed up collision detection, especially for complex surface meshes.

The coarse mesh is constructed using the Delaunay tetrahedralization of a body-centered cubic lattice. This involves voxelizing the surface mesh onto two offset regular cubic grids, then creating appropriate BCC tetrahedra. Tetrahedra are classified as internal or external during this process, and appropriate surface embedding is noted.

Previous approaches have spent much computational time ensuring that the BCC mesh conforms to the surface. Since rendering and collisions are handled here using the surface mesh, there is no need to do this, which the authors report is around 100 times faster than creating a conformal mesh. The speed allows for more frequent re-meshing, which allows the simulation of greater flow rates. (Re-meshing must be performed in order to prevent the element basis functions from becoming ill-conditioned - for example, if elements become flat - causing the simulation to be unstable.)

The mesh generation approach, and the timings used in the analysis, are from Labelle and Shewchuk, *Isosurface stuffing: fast tetrahedral meshes with good dihedral angles*, 2007.

Re-meshing requires the transfer of the deformation gradient F, stress accumulation α, and node velocities $\mathbf{v}_i$ from old elements to new ones. This is done by considering overlapping elements and using the amount of overlap to weight the transfer. The node-based velocities are transferred by considering which old element a new node lies in, and using barycentric interpolation between the old nodes to calculate the new node's velocity.

The mass of each node in an element is calculated using a "lumped-mass" formulation: each element's mass is calculated, and each node of an element is assigned one quarter of that mass.

However, if the tetrahedral mesh is especially coarse or if a local feature is especially thin, this might cause gross overestimation of the local mass for the object. This can lead to errors during re-meshing (since the center of mass can move) as well as during significant plastic flows.

Mass is therefore assigned according to the correct center of mass, computed by clipping the surface mesh to an element and calculating the center of mass of the resulting closed polyhedron. The mass is itself adjusted to compensate for elastic deformation (see below), and distributed to the nodes using barycentric weights based on the new center of mass.

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

Very small masses can also cause problems; a minimum nodal mass is therefore used.

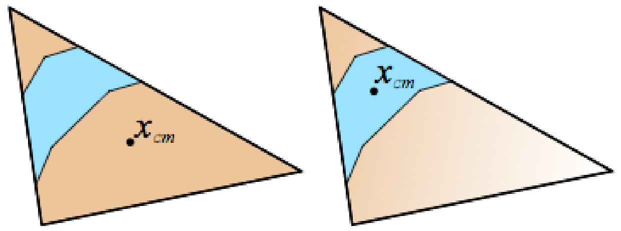


**Figure 5:** ***Sub-element mass computation:*** *The blue strip of material is clipped to the element, and then masses are distributed to nodes at the element corners, resulting in a linear density function represented in beige. Naive lumped mass calculations give a uniform density across the element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

The authors report simulation times for the below figure, with 70k surface triangles, at 30 minutes. (All times appear to be for CPU simulation, without rendering.)

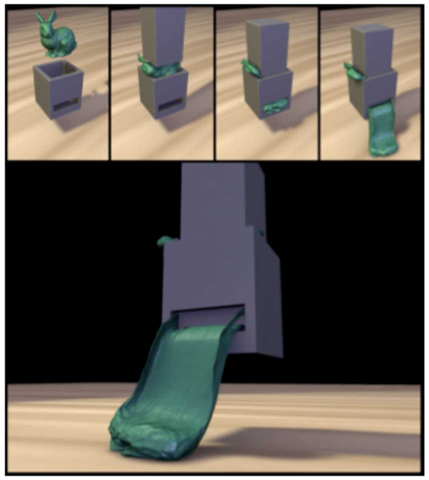
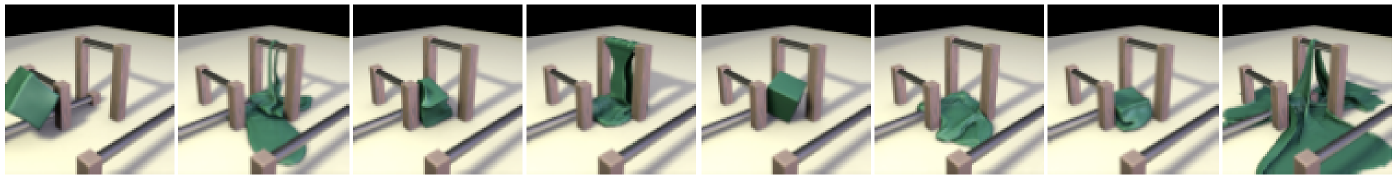


**Figure 1:** *A stiff plastic Stanford Bunny is forced through a small pressing machine that squishes it into several thin sheets. This result was generated in 30 minutes.*

On average, the examples in the below figure took around 1.5 hours to simulate, with the very fluid example on the far right taking about 10 hours.



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

**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. $\mu$ is stiffness, $\beta$ is viscosity, $P_Y$ is yield stress, and $\nu$ is flow rate.*

*Two-way Coupling of Rigid and Deformable Bodies* describes a fully coupled scheme for including rigid and deformable objects in the same simulation. This is desirable for simple mixed-object situations, such as those we saw in the previous paper, but also for more sophisticated animations such as skeletal control of a character with realistic skin and flesh, which then interacts with forces from its environment.

One common approach to this problem is to use individually good methods for simulating each kind of object, and to run them in an interleaved fashion. This can lead to stability problems due to the lag in communication between the two methods, similar to problems one sees with explicit integration of (for example) stiff systems.

Another approach is to use a method that can handle both kinds of objects; this has been done with, for example, SPH. However, many algorithms developed for one kind of object, such as articulation or proportional derivative control for rigid skeletons, are then difficult to use.

This paper proposes a scheme for coupling a rigid-body simulation and a deformable-object simulation at each substep of a time integration algorithm. By considering the subtleties of each part of the coupling, the scheme remains stable while allowing appropriate algorithms to be used with each kind of object.

The time integration scheme used is a Newmark method, which separates position and velocity updates. Such schemes allow for such things as making the position update with a proposed velocity, different than the final ones; the authors give the example of using constraint-violating components to correct drift in the proposed velocity, but removing these in the final update. Furthermore, Newmark schemes are well-supported by algorithms from computational mechanics.

The scheme is composed of five major steps:

1. Advance velocity $\mathbf{v}^n \rightarrow \tilde{\mathbf{v}}^{n+\frac{1}{2}}$
2. Apply collisions $\mathbf{v}^n \rightarrow \hat{\mathbf{v}}^n, \tilde{\mathbf{v}}^{n+\frac{1}{2}} \rightarrow \hat{\mathbf{v}}^{n+\frac{1}{2}}$
3. Apply contact and constraint forces $\hat{\mathbf{v}}^{n+\frac{1}{2}} \rightarrow \mathbf{v}^{n+\frac{1}{2}}$
4. Advance positions $\mathbf{x}^n \rightarrow \mathbf{x}^{n+1}$ using $\mathbf{v}^{n+\frac{1}{2}}, \hat{\mathbf{v}}^n \rightarrow \bar{\mathbf{v}}^n$
5. Advance velocity $\bar{\mathbf{v}}^n \rightarrow \mathbf{v}^{n+1}$

We will briefly discuss each of these steps as well as their constituent sub-steps.

Step 1: Advance velocity

This step is intended to predict the velocity to be used to update positions.

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

Starred variables here and throughout indicate temporary quantities used only within a single major step.

Steps 2 and 3 are optional, and apply to the simulation of such things as incompressible solids and cloth.

## Step 2: Collisions

In this step, velocities are adjusted for collisions (rigid/rigid, rigid/deformable, deformable/deformable). This step comes before contact handling to allow for bouncing of elastically colliding bodies.

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

Collisions are detected by evolving positions forward in time and looking for interferences. Collisions are processed iteratively (details are in the paper).

Post-stabilization here involves ensuring that collision velocities do not violate joint constraints.

Rigid/deformable collisions are processed as inelastic rigid/rigid collisions between the rigid body and the colliding particles of the deformable bodies. Deformable/deformable collisions are processed at a separate step.

## Step 3: Contact and constraint forces

This step accounts for contact and constraint processing, including articulation and PD control (although these are optional).

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

Rigid/deformable pairs are treated as for collisions.

Step 4: Advance positions

Now that we have adjusted velocities for collisions, constraints, and contact, this step evolves bodies to their final positions.

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

Deformable/deformable collisions are handled here through interpenetration resolution, although other algorithms could be applied elsewhere.

Step 5: Advance velocity

This step updates velocities to their final values.

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