A Virtual Node Algorithm for Changing Mesh Topology During Simulation

Neil Molino* Stanford University
Zhaosheng Bao* Stanford University
Ron Fedkiw† Stanford University, Industrial Light + Magic

Abstract

We propose a virtual node algorithm that allows material to separate along arbitrary (possibly branched) piecewise linear paths through a mesh. The material within an element is fragmented by creating several replicas of the element and assigning a portion of real material to each replica. This results in elements that contain both real material and empty regions. The missing material is contained in another copy (or copies) of this element. Our new virtual node algorithm automatically determines the number of replicas and the assignment of material to each. Moreover, it provides the degrees of freedom required to simulate the partially or fully fragmented material in a fashion consistent with the embedded geometry. This approach enables efficient simulation of complex geometry with a simple mesh, i.e. the geometry need not align itself with element boundaries. It also alleviates many shortcomings of traditional Lagrangian simulation techniques for meshes with changing topology. For example, slivers do not require small CFL time step restrictions since they are embedded in well shaped larger elements. To enable robust simulation of embedded geometry, we propose new algorithms for handling rigid body and self collisions. In addition, we present several mechanisms for influencing and controlling fracture with grain boundaries, prescoring, etc. We illustrate our method for both volumetric and thin-shell simulations.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation;

Keywords: changing mesh topology, finite elements, fracture, virtual surgery, sculpting

1 Introduction

Meshes with changing topology are indispensable in numerous applications, e.g. cutting in virtual surgery or haptics, sculpting and modeling in computer aided design, tearing of textiles, fracture, etc. Simulation can complement or obviate the need for difficult-to-perform laboratory experiments. This is particularly necessary if the object is rare, expensive, irreplaceable, or if the material is hazardous. Many problems include fracture as a critical component, e.g. medical applications such as the break-up of kidney stones using lithotripsy, the structural and safety analysis of bridges and buildings, etc. Ships, aircraft, and other metallic structures in harsh environments are particularly susceptible to corrosive cracks and material failure. Moreover, physically based animation of destructive phenomena (i.e. explosions and fracture) enables special effects studios to create content for feature films.

*e-mail: {npmolino, jbao}@stanford.edu
†e-mail: fedkiw@cs.stanford.edu

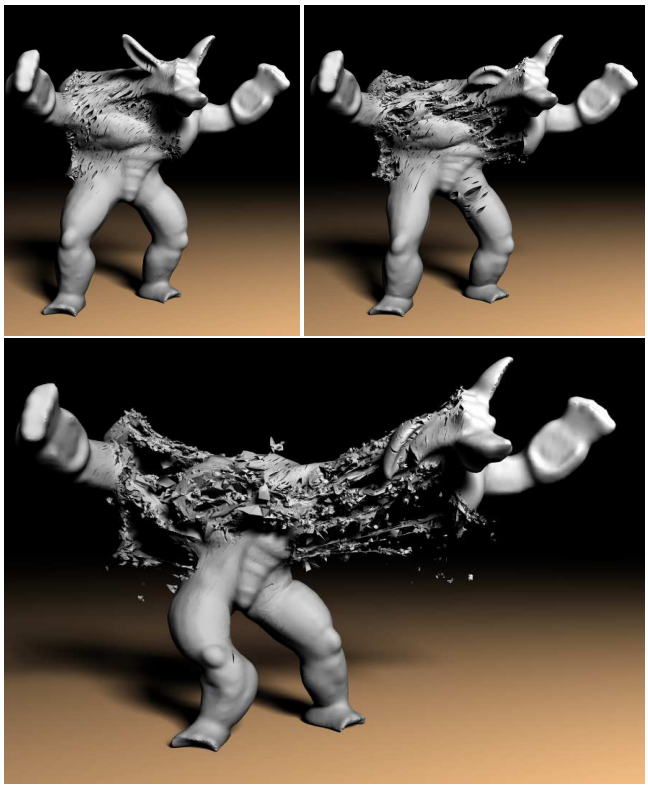

Figure 1: Tearing a plastically deforming high resolution volumetric mesh (380K tetrahedra).

There is an intimate connection between the Lagrangian simulation of a deformable solid and its discretization. Research in Lagrangian simulation can be loosely classified into three categories: mesh generation, simulation, and mesh alteration during simulation. A high-quality mesh is a prerequisite for effective simulation, and poor quality elements adversely affect accuracy, efficiency and stability. Mesh alteration during simulation falls roughly into two categories. Various works have focused on maintaining mesh quality in the face of large deformation, such as the pioneering work of [Hirt et al. 1974] on arbitrary Lagrangian-Eulerian (ALE) methods. Other work has addressed topological changes such as fracture. According to [Tabiei and Wu 2003], there are three main approaches for treating separating material with a Lagrangian mesh: elements can be weakened along cracks so that they stretch arbitrarily while preserving connectivity, the mesh can be split along element boundaries, or continuous remeshing can be performed. Unfortunately, the first two options produce visual artifacts on the scale of the coarse tetrahedral mesh.

Although continuous remeshing was once considered promising, see e.g. [Camacho and Ortiz 1997], many authors no longer advocate it when they move to three spatial dimensions, see e.g. [Ortiz and Pandolfi 1999]. Continuously remeshing the arbitrary situations that arise during fracture can be daunting. And even when remeshing is successful, the result can often be ill-conditioned reducing both accuracy and stability, e.g. producing sliver tetrahedra

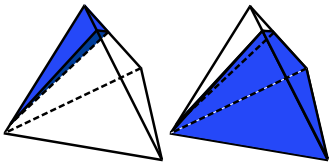

Figure 2: The subelements resulting from a triangle cut: a subtetrahedron (left) and a triangular prism (right). Slivers of material are still simulated with nicely shaped elements.

that severely limit the time step or cause the mesh to invert. [Bessette et al. 2003] points out that many authors address this difficulty by deleting problem elements altogether. However, disappearing tetrahedra destroy the visual accuracy. A nice compromise between fully remeshing and only fracturing between elements was struck by [O'Brien and Hodgins 1999; O'Brien et al. 2002] where cuts are constrained to go through existing nodes. It was further necessary to impose limitations upon the directions of these cracks to avoid backcracking and low quality elements that compromised the stability of their explicit time integration scheme. Even so, their small elements generated time step restrictions that made their calculations quite computationally expensive.

Our goal is to allow element splitting without suffering from the small time step restrictions imposed by sliver elements. We do this by simulating partially void elements (only partially filled with material), e.g. see figure 2. A major advantage of simulating embedded geometry is that it decouples the resolution required for the geometry from the resolution required for the physics, e.g. it is difficult for a boundary conforming mesh to match sharp corners exactly. Also, we can simulate slivers of material without having sliver elements in the simulation mesh.

The key to allowing arbitrary separation of the material represented by a mesh is our virtual node algorithm. Using it, we circumvent many of the difficulties in traditional techniques for fracture calculations whereby the crack geometry is limited to the boundaries of existing elements or the entire region surrounding a crack must be remeshed. By simulating multiple copies of an element, each representing some potion of the material from the original, the simulation mesh remains well-conditioned. There is no need for sliver elements to resolve crack geometry. Therefore, this technique avoids the pitfalls associated with low quality elements forced by remeshing.

2 Related Work

[Terzopoulos et al. 1987] pioneered deformable models in graphics including early work on plasticity and fracture in [Terzopoulos and Fleischer 1988b; Terzopoulos and Fleischer 1988a] where cloth was torn. Other early work on finite element modeling includes [Gourret et al. 1989; Chen and Zeltzer 1992]. Some of the more recent work includes the adaptive framework of [Debunne et al. 2001], the rotation based approach in [Muller et al. 2002], and the finite volume muscle models of [Teran et al. 2003]. Other interesting approaches to the simulation of deformable objects include [James and Pai 2002; James and Fatahalian 2003].

Most graphics researchers simply break connections or springs between elements when the force is high, see e.g. [Norton et al. 1991; Hirota et al. 1998; Mazarak et al. 1999; Smith et al. 2001]. Interesting two dimensional results were obtained in [Neff and Fiume 1999] in the context of blast waves. [Muller et al. 2001] treated objects as rigid bodies between collisions, and used static finite element analysis techniques during collisions. They used the principal stress components to separate tetrahedra occasionally refining large tetrahedra before splitting along element boundaries. Similarly, [Muller and Gross 2004; Muller et al. 2004] fracture between element boundaries in a FFD framework and maintain a watertight

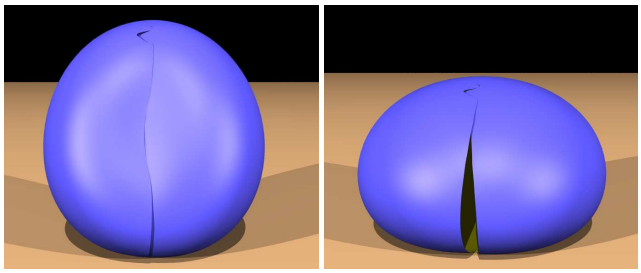

Figure 3: Edge edge collision of a fractured spherical shell (1K triangles).

embedded surface mesh, however their fracture surfaces are quite limited by the coarse simulation mesh. The state of the art in fracture for computer graphics is the work of [O'Brien and Hodgins 1999; Yngve et al. 2000; O'Brien et al. 2002] which used a pseudo principal stress and continuous remeshing.

Although virtual surgery researchers often recommend splitting tetrahedra via subdivision, the resulting tetrahedra can be ill-conditioned and difficult to simulate, see e.g. [Mor and Kanade 2000]. [Bielser and Gross 2000] use semi-implicit time stepping and asynchronous time integration to partially alleviate these difficulties. Moreover, they use masses and springs (as opposed to finite elements) so inverting tetrahedra, although visually displeasing, do not cause the simulation to fail. For more on tetrahedral subdivision, see e.g. [Bielser et al. 2003]. Similarly, for element deletion alternatives, see e.g. [Forest et al. 2002].

Our method also works for cloth and shells. We use a diagonalized finite element model similar to [Etzmuss et al. 2003] for the in plane deformations and the bending model of [Bridson et al. 2003] (see also [Grinspun et al. 2003]), but other bending models such as [Choi and Ko 2002] could be used. For self-collisions, we treat the material surface in the same manner as [Bridson et al. 2002], and propose new methods to modify the parent elements and to keep these two meshes in sync. We note that an untying method for collision handling as in [Baraff et al. 2003] is inappropriate since we often have edge edge collisions that their method does not handle, e.g. see figure 3. We use mixed explicit implicit time integration, although for stiffer shells a fully implicit method as in [Baraff and Witkin 1998] or [Grinspun et al. 2003] might be preferable. Other interesting work on shells includes the adaptive mesh simulation of [Grinspun et al. 2002].

For volumetric collisions, we treat the embedded boundary surface as a triangulated manifold and again use the method of [Bridson et al. 2002], providing new algorithms to make the virtual nodes respond and to keep the two meshes in sync. When we fracture a volumetric object, we cannot ensure collision-freeness (as we can with shells). Thus we have slightly modified [Bridson et al. 2002] to ignore triangles that are interfering. We then use an untangling strategy similar to that in [Baraff et al. 2003], which is best coupled with penalty forces pushing nodes out of tetrahedra as in [O'Brien and Hodgins 1999].

Our method of simulating a parent element with enslaved embedded geometry is essentially a free form deformation (FFD) as proposed in [Sederberg and Parry 1986]. Dynamic FFD's were developed by [Faloutsos et al. 1997]. Later, [Capell et al. 2002b] extended these dynamic FFD's to finite element simulation. A related strategy was also carried out in [Capell et al. 2002a].

3 Virtual Node Algorithm

The virtual node algorithm solves the problem of how to represent and simulate severed or separated material within a mesh. In certain situations, determining how to duplicate elements and create virtual nodes is straightforward, e.g. as in figure 2. However, in other scenarios, making these determinations can be rather complex and

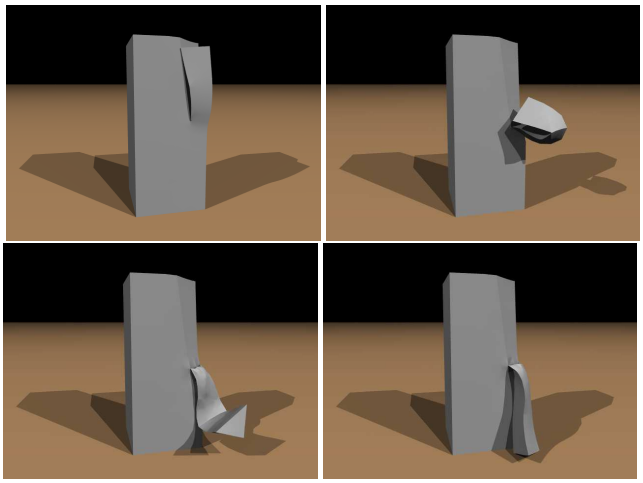

Figure 4: Slicing a volumetric brick with a surgical scalpel (not pictured).

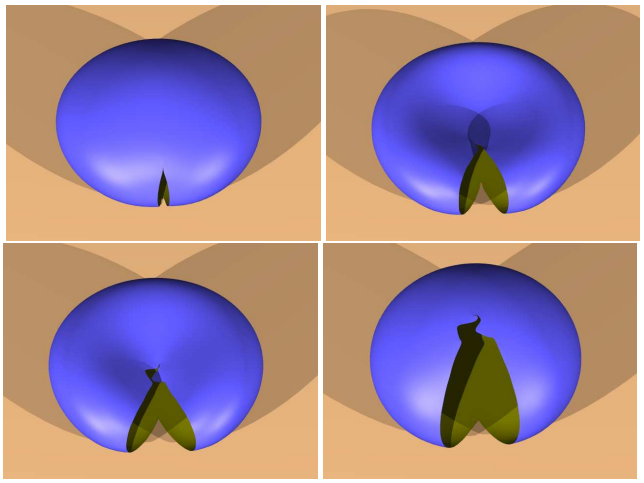

Figure 6: A single fracture initiation propagating through a spherical shell (1K triangles).

moreover requires nonmanifold geometry. Even in the simple case of two triangles shown in figure 5, obtaining the desired degrees of freedom requires three triangles on a single edge. This occurs quite often when cutting partially into a mesh as in figure 4, or as a crack slowly propagates through material as in figure 6. Our virtual node algorithm automatically handles all of these cases. The essence of the virtual node algorithm is simple to describe. For each distinct scoop cut out of a node's one ring, a virtual copy of the central node is created and donated to the nodes within the given scoop to give the mesh the degrees of freedom needed to break apart. This works for both two and three spatial dimensions, for volumes and shells, for arbitrary branching, etc. The algorithm is quite simple and surprisingly general.

A fundamental assumption we make when separating an existing mesh is that each node represents a point sample of the material around it, and as such cannot be separated from a core material contained in its one ring. That is, if the mesh is completely fragmented into its smallest possible units, each one will consist of a real node from the original mesh surrounded by all the elements from its one ring with every (edge connected) neighbor replaced by a new virtual node. See figure 7 (far right). It is obvious how to duplicate elements and construct virtual nodes in the complete fragmentation case, but any viable algorithm needs to treat partial fragmentation as well. The fundamental goal is to determine which parts of an element's one ring are fragmented and which parts are not. This will be necessary in order to treat more complicated scenarios such as that depicted on the left and center of figure 7.

A natural way to determine which portions of a one ring have been fragmented away, and which have not, is to consider scoops out of one rings. If the segmentation boundary completely separates a node from some subset of its neighbors, then the material cores of the neighbors can be thought of as separated from the material core of the central node in question. In particular, the central node donates one virtual node to each independent scoop out of its one ring. See figure 8. The algorithm is carried out in two passes. First,

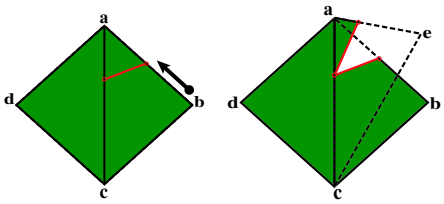

Figure 5: The crack (drawn in red) allows the material to partially open. Node *e* is a virtual copy of node *b* and provides the degrees of freedom that enable the material to separate.

for each node, we determine how many disjoint scoops have been taken out of its one ring and assign a virtual copy of the central node to each scoop. Then in a second pass, we visit every node and look to its neighbors to see if any of them assigned a virtual copy of itself to the node in question. In this second pass, we build the new mesh topology by constructing all the elements in the one ring of this central node using either the real neighbor or the assigned virtual copy when it exists.

In figure 5, only node *b* has a scoop out of its one ring, and thus it makes a copy of itself (node *e*) and assigns this node to node *a* in the first sweep. In the second sweep, node *b* constructs its triangle as usual (attaching to nodes *a* and *c*), but node *a* uses node *e* in place of node *b* when it constructs the triangle to the right. This provides the degrees of freedom necessary for the crack to open. In figure 7 only node *f* has a scoop out of its one ring, and thus it makes a virtual copy of itself (node *g*) and assigns it to node *v*. Then all nodes construct their triangles as usual, except node *v* which uses virtual node *g* when constructing two of its triangles. Again, this provides the degrees of freedom necessary to open the crack.

Although we used triangles to illustrate this method, it carries over identically to tetrahedra. The one ring neighbor nodes are still edge connected, cores of material are subsets of tetrahedral elements, watertight one ring scoops are surfaces as opposed to curves, etc. The algorithm seems to readily extend to other types of meshes as well, e.g. quads, hexahedra, mixed quads and triangles, etc. For brevity, we skipped a few obvious but important details above, e.g. one should not create the same element (i.e. same exact nodes) more than once, new virtual nodes are assigned the positions of real parents except when they are splitting off from already existing and evolved virtual nodes, etc. Also, some choices can be made in the algorithm. For example, we allow all real nodes to maintain their original mass throughout the duration of the simulation, and assign the mass of virtual nodes to be identical to that of their corresponding real node. While more physical mass redistribution techniques exist, this one is highly stable (allowing larger time steps) since all elements keep their original mass distribution regardless of the

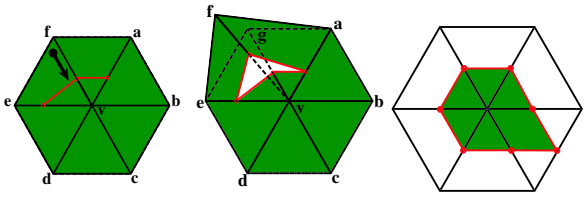

Figure 7: A one ring with a cut that opens (left, center), and a *core of material* (right).

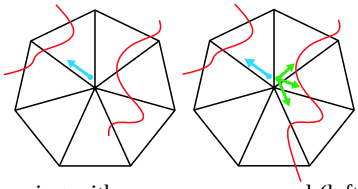

Figure 8: A one ring with one scoop removed (left) and with two scoops removed (right). Each complete scoop receives a distinct virtual node (depicted by the arrows).

presence of slivers, etc. Thus, we have made a conscious choice of stability over accuracy for our graphics applications. That said, we note that it is not unusual for computational physics algorithms to relax mass conservation near lower dimensional interfaces, see e.g. [Fedkiw et al. 1999; Aivazis et al. 2000].

4 Fracturing of Elements

In order to replicate elements and split material between them, one needs to chose a representation for the curves (or surfaces) that partition the material creating new boundaries. Although the virtual node algorithm does not depend on this choice, our data structures, collision algorithms, etc. do, and thus we take a simplified approach using piecewise linear cuts. This choice does not unreasonably restrict the types of cuts we can make through the mesh, as it allows second order accurate representations of the newly formed material boundaries (fracturing along element boundaries is first order accurate). In fact, using only our piecewise planar cuts allows us to both cut along straight paths as shown in figure 4 or sculpt interesting and detailed high resolution geometry as shown in figure 10.

While there are a number of ways to decompose elements with linear cuts, see e.g. [Bielser et al. 2003], we place a few restrictions on this so that the resulting subelements are more readily simulated in regards to both free form deformations and contact and collision algorithms. That is, we restrict all decompositions such that all newly created enslaved or *embedded* nodes appear on an edge of a parent element in the simulation mesh. Moreover, we only allow one embedded node per edge, and find it convenient to construct a data structure that correlates this node with the two *parent* particles on this edge. When a new embedded node is created, its position relative to its parents is defined via a single barycentric coordinate, or *interpolation fraction*, λ. The position and velocity of this enslaved embedded particle is calculated by linearly interpolating these values from the parents.

Triangles may contain up to three embedded nodes. And since we want all material to be associated with a one-ring around a node, we allow a maximum of two cuts in each triangle as shown on the left in figure 9. Thus, we have two types of embedded nodes in any given triangle, those involved with one cut and those involved with two. All possible pairings of these types of nodes across triangle boundaries is shown on the right in figure 9, and one can readily see both the potential for branching as well as the sanctity of the material cores.

A tetrahedron may contain up to six embedded nodes on its

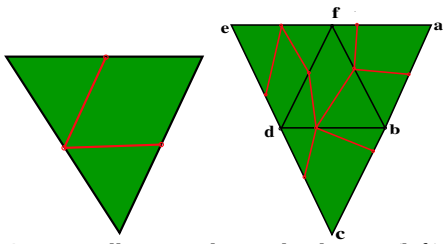

Figure 9: A maximally severed triangle element (left). A fragmented triangle mesh illustrating the different types of junctions of cuts that occur at embedded nodes (right).

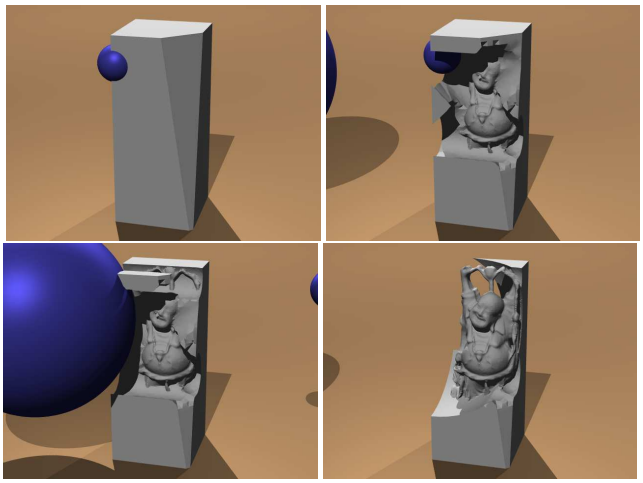

Figure 10: Sculpting a hybrid level set/embedded tetrahedron mesh (500K tetrahedra).

edges. A full decomposition consists of either shaving off three subtetrahedra or splitting the parent element with a quadrilateral and shaving a subtetrahedron off of each piece as is shown in figure 11. In either case, this amounts to three cuts per tetrahedron. When considering all possible pairings of cut tetrahedra across boundaries (i.e. triangle faces), a new case arises that was not present in figure 9. In a triangle, the embedded node splits its edge into two pieces each of which belong to the core of material of one of its parents. The common triangle between two tetrahedra is not as easily partitioned as the boundary edge between two triangles. This common triangle face can be split into a maximum of four subtriangles, and all but the center subtriangle obviously belong to one of the tetrahedron node's material cores. The center subtriangle is not intrinsically part of the core of any of those three tetrahedron nodes, and thus we do not allow it to represent material connections with adjacent tetrahedra. That is, material in the center octahedron (which has this center triangle as a boundary) is connected to the octahedron material in an adjacent element if and only if it is connected via the core of one on the three parent nodes. Otherwise, the center subtriangle is split. See figure 12.

When constructing elements in the virtual node algorithm, one needs to faithfully model all subelement geometry. We do this by giving all replicas the same embedded nodes and cut geometry. This can be seen, for example, in figure 5 where each of the two copies of the triangle to the right contains two embedded nodes and a segment connecting them. Moreover, this segment represents the new material boundary so obviously two are needed. Similarly, see figure 7. We further stress that the concern above about whether or not a center subtriangle connects octahedron material is automatically handled by the virtual node algorithm. When creating new elements and giving all replicas the same embedded nodes and geometry, the octahedrons automatically split apart when the center subtriangles are created twice.

For both rendering and collision handling, it is necessary to identify the boundary of the (non-virtual) material. For triangulated surfaces, this is trivial, except that we add the caveat that all quads (see

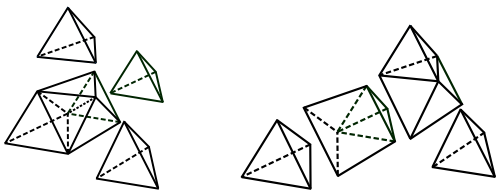

Figure 11: The two different maximally-split configurations for a tetrahedron.

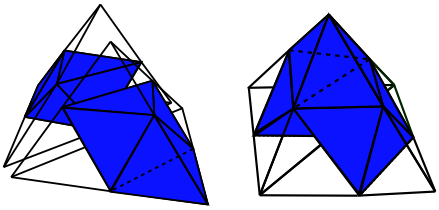

Figure 12: The central subtriangle of the face shared by two tetrahedra is connected if and only if the two central octahedra are associated with the same simulation node. On the right, both central octahedra are associated with the top node. On the left, they are associated with different nodes (and thus split apart).

figure 9, left) are cut into two triangles. The boundary surface of a tetrahedralized volume is constructed via surface cancelation. For every element in the simulation mesh, we determine which portion of it is material and add a triangulation of the subelement's boundary to the list of boundary triangles. If the same triangle is ever added with the opposite orientations, we permanently remove this triangle from the boundary list. Of course, one must be consistent with the triangulation of subelement boundaries for this algorithm to work.

5 Dynamics

There are three kinds of nodes in the mesh: real nodes, virtual nodes and embedded nodes. The elements are comprised of real and virtual nodes, and in principle any deformable model (masses and springs, finite elements, etc.) may be used to simulate them. We use the diagonalized finite element method of [Irving et al. 2004], which is equivalent to the standard finite element method for tetrahedra with non-negligible positive volume. However, this method also works well for both degenerate and inverted elements, which is particularly useful for large deformations, plasticity and fracture. Given both the rest shape and current shape of an element (a triangle or tetrahedron), the constitutive model produces a first Piola-Kirchoff stress P. As shown in [Teran et al. 2003], the force can be accumulated at the nodes by multiplying P by the area weighted face normals of the element in material coordinates. For triangles, out-of-plane forces are computed with the bending model of [Bridson et al. 2003]. In addition, a multiplicative plasticity component (as opposed to the additive plasticity in [O'Brien et al. 2002]) is included to model ductile fracture, see e.g. [Armero and Love 2003].

We use a slight variant of the Newmark time integration scheme presented in [Bridson et al. 2003].

- $\tilde{v}^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, \tilde{v}^{n+1/2})$
- $\tilde{x}^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$
- Process rigid body collisions using $\tilde{x}^{n+1}$ and v^n, producing final positions x^{n+1} and modified velocities $\tilde{v}^n$.

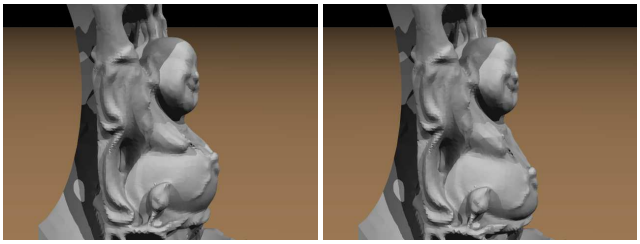

Figure 13: Volumetric simulation of the Buddha sculpted from the block in figure 10. The belly is soft and drops under its own weight (500K tetrahedra).

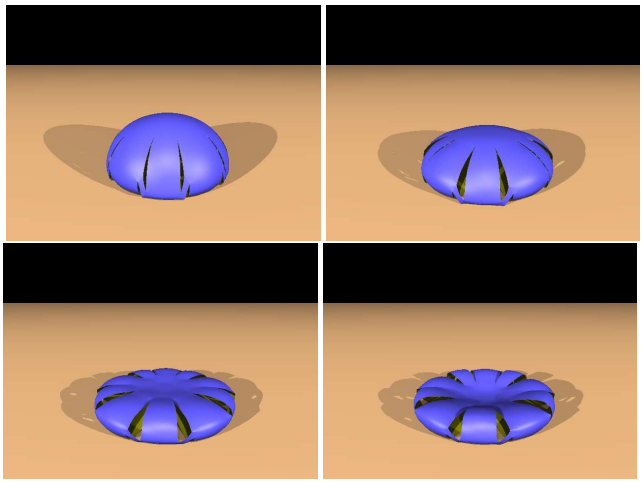

Figure 14: Ductile fracture of a spherical shell (1K triangles).

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

After each position or velocity update, the embedded nodes are updated using linear interpolation from their parent particles.

6 Collisions

Whether working with shells or volumetric objects, section 4 outlines how to construct the boundary mesh of the material. This mesh consists of triangles made up of real nodes and embedded nodes, but no virtual nodes. Moreover, virtual nodes should not collide, since they do not represent material. Also, for efficiency, we do not collide interior nodes of volumetric objects.

6.1 Rigid Body Collisions

First, the rigid body collisions of the real nodes on the material boundary are processed, and then the positions and velocities of the embedded nodes are linearly interpolated. Next, the rigid body collisions of the embedded nodes are processed to obtain a Δx_{emb} and Δv_{emb} for each embedded particle. Δx_{emb} is used to adjust the positions of the two parent particles via

$$x_1 += \frac{1-\lambda}{\lambda^2 + (1-\lambda)^2} \Delta x_{emb}$$
$$x_2 += \frac{\lambda}{\lambda^2 + (1-\lambda)^2} \Delta x_{emb},$$

where λ is the embedded particle's interpolation fraction. Adding $1-\lambda$ times the first equation to λ times the second equation gives $x_{emb} += \Delta x_{emb}$ as desired. The change in velocity is handled similarly. Although this scheme works well, we have found that improved results can be obtained by *not* processing parent particles that are on the boundary surface unless they collide directly. Of course, this throws the embedded particles out of sync, so as a final step we linearly interpolate their new state (position and velocity).

6.2 Self Collisions

We save a collision free material boundary mesh, integrate the computational mesh forward in time for one or more steps, calculate the new state of the material boundary mesh, and then use the algorithm of [Bridson et al. 2002] to process collisions assuming linear trajectories between the initial and final state of the material boundary mesh. The result is a collision free set of positions and velocities for the material boundary mesh. However, since some

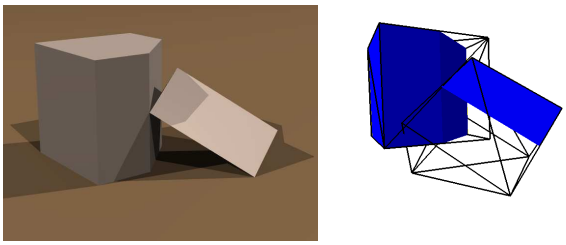

Figure 15: Simple scenario demonstrating the efficacy of our embedded collisions (5 tetrahedra).

of the nodes on the boundary mesh are embedded, reinterpolating the embedded nodes on the material boundary can introduce self-interference. Thus, we do not resync these nodes, but instead save the resulting unsynced mesh as the collision free state. In the next suite of time steps, the target material boundary mesh *is* automatically synced with the computational mesh. Thus, the unsynced mesh will try to return to a synced position, and will succeed in the absence of self collisions, so the drift is temporary. Similar to rigid body collisions, the effects of Δx_{emb} and Δv_{emb} need to be mapped to the parents. However, we stress that *not* processing real nodes on the material surface is even more important here as it can create an interfering state. Thus, for each real or virtual node that is not on the material surface, we first compute the Δx_{emb} of *all* the embedded children for which this node is a parent. Then this parent is simply assigned the average of all the Δx_{emb}'s of its embedded children (similarly for velocity).

Fracture is handled after self collision and outside of the self-collision loop. One of the drawbacks of our collision handling strategy is that our fracture algorithm cracks material *in place* and thus it can be touching (and intersecting due to roundoff errors) immediately after fracturing. So we do not have a collision free state for the collision algorithm. This is readily rectified for shells by perturbing the material in-plane by an amount just larger than round-off error to guarantee a collision freeness. For volumetric objects, no guarantees can be made. So although we attempt to perturb the material slightly after fracture, we also implemented a version of [Bridson et al. 2002] that ignores interfering triangles. Moreover, we use a combination of untying as in [Baraff et al. 2003] and repulsion forces as in [O'Brien and Hodgins 1999] in order to pursue a non-interfering state at which point our modified version of [Bridson et al. 2002] automatically takes over preventing all subsequent collisions.

7 Fracture and Control

We use the Rankine condition [Rankine 1872] of maximal principal stress to decide both whether and how a material fractures. To remove mesh aliasing, we smooth the Cauchy stress σ with a few passes of volume weighted averaging as is common in the literature, see e.g. [Belytschko et al. 2003; Wells and Sluys 2001; Jirasek and Zimmermann 2001; Bouchard et al. 2003]. Then, if the maximum eigenvalue of σ (potentially modified for control) exceeds a threshold for tensile fracture, the element is broken. Secondary and tertiary fractures of the same element are given higher thresholds to help reduce spurious branching. We also use the minimum eigenvalue to model compressive fracture, but with higher thresholds (e.g. [Smith et al. 2001] points out that bonds can be up to eight times stronger during compression). We note that the non-element based "separation tensor" in [O'Brien and Hodgins 1999; O'Brien et al. 2002] is not typical in the literature, but it has the same flavor as the Rankine condition. For each of the possible cuts (e.g. in a tetrahedron there are three quadrilateral cuts and four triangle cuts), we first snap them to be coincident with cuts in adjacent elements to help promote clean fracture similar to [Jirasek and Zimmermann 2001]. Moreover, except for a finite number of initiation points (globally or per region), we disregard potential cuts that do not extend an existing crack. With these constraints, we determine which of the cuts has a normal direction that most closely matches the eigenvector associated with the maximum (or minimum) principal stress.

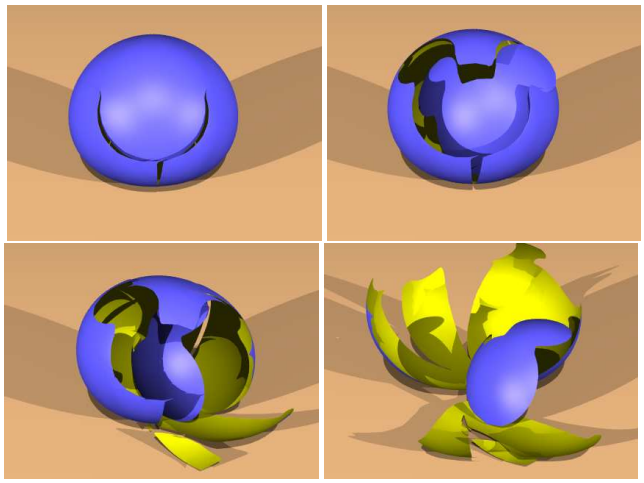

Figure 16: Prescored fracture with a level set of a "mouse" (1K triangles).

Recently, control of natural phenomena has received a lot of attention, see e.g. [Treuille et al. 2003]. We control our fracture simulations by biasing σ's eigenstructure and by prescoring with level sets. In order to promote straight cracking, we add a term of the form $\beta_1 pp^T$, where p is a vector normal to existing cracks in adjacent elements. In order to make the fracture more interesting, heterogeneous material anisotropy is modeled by seeding and growing a number of regions in which a particular direction is assigned. A smoothing postprocess is used to blend these regions together. Then a term of the form $\beta_2 gg^T$ is added as well. With these control structures in place, we apply the Rankine condition to $\beta_0\sigma + \beta_1 pp^T + \beta_2 gg^T$ where, like σ, the β_i can be element dependent. Since level sets can be used to model cracks themselves, see e.g. [Ventura et al. 2003], we designed a level set prescoring control structure that stores level set values at the nodes of the mesh and uses these values to determine how a particular element cracks. Moreover, we can impose conditions such as "no cracking" in a particular region of the level set. For an example of this, see figure 16.

8 Examples

All of our volumetric and surface meshes were generated with the meshing algorithm of [Molino et al. 2003]. Figures 18 and 4 illustrated the ability to make arbitrary linear cuts through a surface and volumetric mesh respectively. Visual accuracy is crucial for graphics, and the clean cuts through a complicated mesh as in figure 18 would not be possible by only breaking at element boundaries (the cut would be jagged). Moreover, there is no reason to limit the visual representation of the geometry to first order accuracy just because the stress is first order accurate.

Once the cut in figure 4 extends entirely through the brick, we use the remaining material for our sculpting application, as demonstrated in figure 10. The sculpting is modeled with level set values on the nodes and the resulting embedded geometry is obtained with a slight variant of marching tetrahedra. This sculpting produces an embedded mesh with well conditioned elements and, as such, is readily simulated. In figure 13, we simulate the jiggling of the sculpted Buddha's belly. Finally, these sculpted meshes may un-

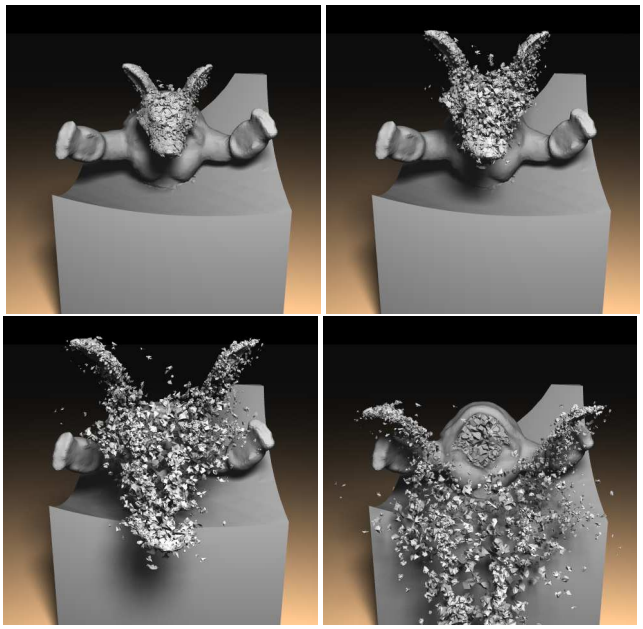

Figure 17: Sculpted geometry admits further topology change during simulation (210K tetrahedra).

dergo further topology change during simulation as shown in figure 17, where the diversity of sub-element geometry provides visual richness. In such scenarios our technique is under-resolved, but other standard methods (such as fracturing to the level of individual elements along element boundaries) is also wildly inaccurate. That is, for example, real material does not fracture into individual tetrahedrons.

Level sets or signed distance functions are used in other sculpting and editing systems as was shown in [Perry and Frisken 2001; Museth et al. 2002; Cutler et al. 2002]. In [Cutler et al. 2002] level sets were used to create layered objects, but to provide a tetrahedral mesh for each layer, they needed a conforming discretization. This creates problematic low-quality tetrahedra which they worked to remove. Our embedded approach does not suffer from these difficulties since no new tetrahedra (only copies) are created. Also, by virtue of having level set values define the geometry, we can make some material (e.g. the Buddha) indelible by limiting ϕ on certain nodes.

All the spherical shells examples (figures 3, 6, 14 and 16) were run with only 1K triangles without hindering our ability to make interesting (and straight) cuts. Moreover, they took only minutes per simulation (about five minutes for soft examples and twenty for stiffer ones). Our algorithm also scales well. For example, simulating the 380K tetrahedra in figure 1 takes about twenty minutes per frame and simulating the 40K triangles in figure 18 runs at roughly ten minutes per frame.

9 Conclusion

We proposed a new virtual node algorithm that allows material to fracture along arbitrary (possibly branched) piecewise linear paths. We illustrated the application of this algorithm to both volumetric solids and thin shells. In addition, we proposed new algorithms to treat both rigid body and self collisions for this embedded mesh structure. Examples of cutting, sculpting and fracture were shown. Moreover, we provided a control strategy for fracture including a level set prescoring mechanism.

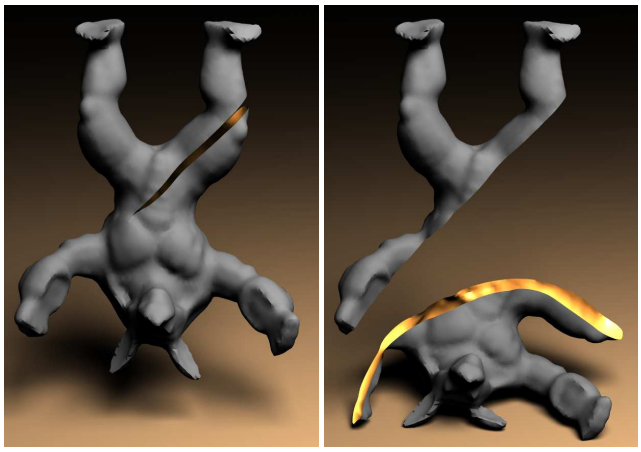

Figure 18: Cutting of a high resolution shell (40K triangles).

10 Acknowledgement

Research supported in part by an ONR YIP award and a PECASE award (ONR N00014-01-1-0620), a Packard Foundation Fellowship, a Sloan Research Fellowship, ONR N00014-03-1-0071, ONR N00014-02-1-0720, ARO DAAD19-03-1-0331, NSF DMS-0106694, NSF ITR-0121288, NSF IIS-0326388, NSF ACI-0323866 and NSF ITR-0205671. In addition, N. M. was supported in part by a Stanford Graduate Fellowships, and Z. B. was supported in part by an NSF Graduate Research Fellowship.

We'd also like to thank Mike Houston for providing computing resources used for rendering.

References

AIVAZIS, M., GODDARD, W., MEIRON, D., ORTIZ, M., POOL, J., AND SHEPHERD, J. 2000. A virtual test facility for simulating the dynamic response of materials. *Comput. in Sci. and Eng. 2*, 42–53.

ARMERO, F., AND LOVE, E. 2003. An arbitrary lagrangian-eulerian finite element method for finite strain plasticity. *Int. J. Num. Meth. Eng. 57*, 471–508.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proc. SIGGRAPH 98*, 1–12.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 862–870.

BELYTSCHKO, T., CHEN, H., XU, J., AND ZI, G. 2003. Dynamic crack propagation based on loss of hyperbolicity and a new discontinuous enrichment. *Int. J. Num. Meth. Eng. 58*, 1873–1905.

BESSETTE, G., BECKER, E., TAYLOR, L., AND LITTLEFIELD, D. 2003. Modeling of impact problems using an h-adaptive, explicit lagrangian finite element method in three dimensions. *Comput. Meth. in Appl. Mech. and Eng. 192*, 1649–1679.

BIELSER, D., AND GROSS, M. 2000. Interactive simulation of surgical cuts. In *Pacific Graph.*, 116–125.

BIELSER, D., GLARDON, P., TESCHNER, M., AND GROSS, M. 2003. A state machine for real-time cutting of tetrahedral meshes. In *Pacific Graph.*, 377–386.

BOUCHARD, P., BAY, F., AND CHASTEL, Y. 2003. Numerical modelling of crack propagation: automatic remeshing and comparison of different criteria. *Comput. Meth. in Appl. Mech. and Eng. 192*, 3887–3908.

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

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 28–36.

CAMACHO, G., AND ORTIZ, M. 1997. Adaptive Lagrangian modelling of ballistic penetration of metallic targets. *Comput. Meth. in Appl. Mech. and Eng. 142*, 269–301.

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

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. A multiresolution framework for dynamic deformations. In *ACM SIGGRAPH Symp. on Comput. Anim.*, ACM Press, 41–48.

CHEN, D., AND ZELTZER, D. 1992. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. *Comput. Graph. (SIGGRAPH Proc.)*, 89–98.

CHOI, K.-J., AND KO, H.-S. 2002. Stable but responsive cloth. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 604–611.

CUTLER, B., DORSEY, J., MCMILLAN, L., MULLER, M., AND JAGNOW, R. 2002. A procedural approach to authoring solid models. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 302–311.

DEBUNNE, G., DESBRUN, M., CANI, M., AND BARR, A. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *Proc. SIGGRAPH 2001*, vol. 20, 31–36.

ETZMUSS, O., KECKEISEN, M., AND STRASSER, W. 2003. A fast finite element solution for cloth modelling. In *Pacific Graph.*, 244–251.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic free-form deformations for animation synthesis. *IEEE Trans. on Vis. and Comput. Graphics 3*, 3, 201–214.

FEDKIW, R., ASLAM, T., MERRIMAN, B., AND OSHER, S. 1999. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys. 152*, 457–492.

FOREST, C., DELINGETTE, H., AND AYACHE, N. 2002. Removing tetrahedra from a manifold mesh. In *Computer Animation*, 225–229.

GOURRET, J.-P., MAGNENAT-THALMANN, N., AND THALMANN, D. 1989. Simulation of object and human skin deformations in a grasping task. *Comput. Graph. (SIGGRAPH Proc.)*, 21–30.

GRINSPUN, E., KRYSL, P., AND SCHRODER, P. 2002. CHARMS: A simple framework for adaptive simulation. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 281–290.

GRINSPUN, E., HIRANI, A., DESBRUN, M., AND SCHRODER, P. 2003. Discrete shells. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 62–67.

HIROTA, K., TANOUE, Y., AND KANEKO, T. 1998. Generation of crack patterns with a physical model. *The Vis. Comput. 14*, 126–187.

HIRT, C., AMSDEN, A., AND COOK, J. 1974. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *J. Comput. Phys. 135*, 227–253.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. *Submitted to the Symposium on Computer Animation (SCA)*.

JAMES, D., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 879–887.

JAMES, D., AND PAI, D. 2002. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 582–585.

JIRASEK, M., AND ZIMMERMANN, T. 2001. Embedded crack model: II. combination with smeared cracks. *Int. J. Num. Meth. Eng. 50*, 1291–1305.

MAZARAK, O., MARTINS, C., AND AMANATIDES, J. 1999. Animating exploding objects. In *Proc. of Graph. Interface 1999*, 211–218.

MOLINO, N., BRIDSON, R., TERAN, J., AND FEDKIW, R. 2003. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable*, 103–114.

MOR, A., AND KANADE, T. 2000. Modifying soft tissue models: progressive cutting with minimal new element creation. In *MICCAI*, 598–607.

MULLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *Grap. Interface*.

MULLER, M., MCMILLAN, L., DORSEY, J., AND JAGNOW, R. 2001. Real-time simulation of deformation and fracture of stiff materials. In *Comput. Anim. and Sim. '01*, Proc. Eurographics Workshop, Eurographics Assoc., 99–111.

MULLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *ACM SIGGRAPH Symp. on Comput. Anim.*, 49–54.

MULLER, M., TESCHNER, M., AND GROSS, M. 2004. Physically-based simulation of objects represented by surface meshes. In *Proc. Comput. Graph. Int.*, 156–165.

MUSETH, K., BREEN, D., WHITAKER, R., AND BARR, A. 2002. Level set surface editing operators. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 330–338.

NEFF, M., AND FIUME, E. 1999. A visual model for blast waves and fracture. In *Proc. of Graph. Interface 1999*, 193–202.

NORTON, A., TURK, G., BACON, B., GERTH, J., AND SWEENEY, P. 1991. Animation of fracture by physical modeling. *Vis. Comput. 7*, 4, 210–219.

O'BRIEN, J., AND HODGINS, J. 1999. Graphical modeling and animation of brittle fracture. In *Proc. SIGGRAPH 99*, vol. 18, 137–146.

O'BRIEN, J., BARGTEIL, A., AND HODGINS, J. 2002. Graphical modeling of ductile fracture. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 291–294.

ORTIZ, M., AND PANDOLFI, A. 1999. Finite-deformation irreversible cohesive elements for three-dimensional crack-propagation analysis. *Int. J. Num. Meth. Eng. 44*, 1267–1282.

PERRY, R., AND FRISKEN, S. 2001. Kizamu: a system for sculpting digital characters. In *Proc. SIGGRAPH 2001*, vol. 20, 47–56.

RANKINE, W. 1872. *Applied mechanics*. Charles Griffen and Company, London.

SEDERBERG, T., AND PARRY, S. 1986. Free-form deformations of solid geometric models. *Comput. Graph. (SIGGRAPH Proc.)*, 151–160.

SMITH, J., WITKIN, A., AND BARAFF, D. 2001. Fast and controllable simulation of the shattering of brittle objects. In *Comput. Graphics Forum*, D. Duke and R. Scopigno, Eds., vol. 20(2). Blackwell Publishing, 81–91.

TABIEI, A., AND WU, J. 2003. Development of the DYNA3D simulation code with automated fracture procedure for brick elements. *Int. J. Num. Meth. Eng. 57*, 1979–2006.

TERAN, J., SALINAS-BLEMKER, S., NG, V., AND FEDKIW, R. 2003. Finite volume methods for the simulation of skeletal muscle. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 68–74.

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. *Comput. Graph. (SIGGRAPH Proc.)*, 269–278.

TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. *Comput. Graph. (Proc. SIGGRAPH 87) 21*, 4, 205–214.

TREUILLE, A., MCNAMARA, A., POPOVIC, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 716–723.

VENTURA, G., BUDYN, E., AND BELYTSCHKO, T. 2003. Vector level sets for desription of propagating cracks in finite elements. *Int. J. Num. Meth. Eng. 58*, 1571–1592.

WELLS, G., AND SLUYS, L. 2001. A new method for modelling cohesive cracks using finite elements. *Int. J. Num. Meth. Eng. 50*, 2667–2682.

YNGVE, G., O'BRIEN, J., AND HODGINS, J. 2000. Animating explosions. In *Proc. SIGGRAPH 2000*, vol. 19, 29–36.