These fracture patterns propagate arbitrarily in 3D solid objects as they break, crack, or tear realistically.

Animating Fracture

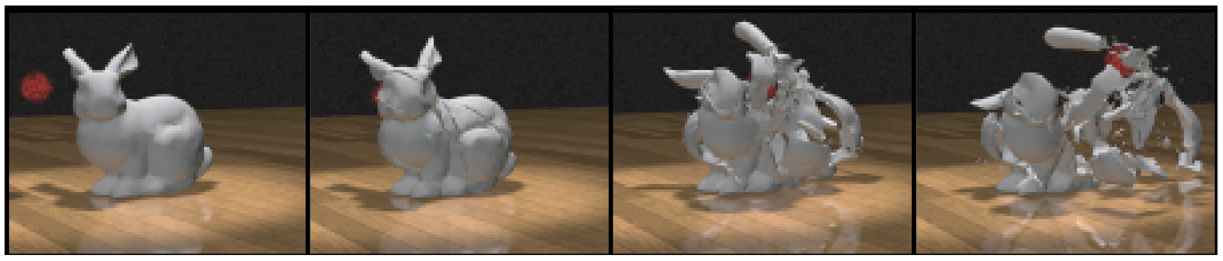
James F. O'Brien and Jessica K. Hodgins

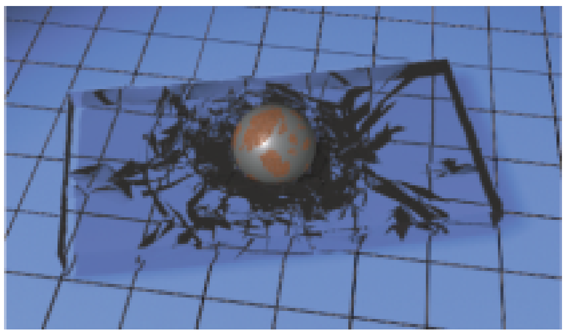
The task of specifying the motion of even a simple animated object like a bouncing ball is surprisingly difficult. This difficulty is due, in part, to our highly developed human skill of observing movement and quickly detecting motion that is unnatural or implausible. Moreover, the motion of many objects is complex, and specifying their movement requires the generation of a great deal of data. For example, cloth can bend and twist in many ways, and the breaking bunny statue in Figure 1a involves hundreds of individual shards.

Animators use three main techniques to generate synthetic motion: keyframing (manually specifying motion); motion capture (using data recorded from actors); and procedural methods (using computer algorithms to generate motion). Keyframing and motion capture both require that motion be specified by some external source. In contrast, procedural methods compute original motion automatically. Many procedural methods are based on informal heuristics, but a subclass known as "physically based modeling" employs numerical simulations of physical systems to generate synthetic motion of virtual objects. With the introduction of simulated water in the 1998 feature film *Antz* [7] (see Foster and Metaxas's "Modeling Water for Computer Animation" in this section) and clothing in the 1999 *Stuart Little* [8], physically based modeling was clearly demonstrated to be a viable technique for commercial animation.

Physically based modeling is especially effective for animating passive objects, because these objects are inanimate and lack an internal source of energy. The advantage of using simulation is not surprising, as these objects tend to have many degrees of freedom, making keyframing or motion capture difficult. Moreover, while passive objects are often essential to the plot of an animation and to its appearance or mood, they are not characters and do not require the same control over the subtle details of their motion. Therefore, simulations in which motion is

Figure 1. Images produced by our technique: (top) a hollow ceramic bunny as it is struck by a heavy, fast-moving weight and (bottom) a plate of simulated glass shattered by a heavy weight.





controlled only by initial conditions, physical equations, and material parameters are often sufficient to produce appealing animations of passive objects.

The computer graphics literature includes many examples of passive systems modeled with simulation. For example, computational fluid dynamics models have been used to animate splashing water, rising smoke, and explosions in the air. Footprints and other patterns left by objects falling on the ground have been modeled by representing the ground surface as a height field. Clothing, hair, and other flexible objects have been approximated with spring and mass systems or collections of beam elements.

In 1999, we developed a simulation technique that uses nonlinear finite element analysis and elastic fracture mechanics to compute physically plausible motion for 3D solid objects as they break, crack, or tear [6]. When these objects deform beyond their mechanical limits, the system automatically determines where fractures should begin and in what directions they should propagate. Our technique allows fractures to propagate in arbitrary directions by dynamically restructuring the elements of a tetrahedral mesh. Because cracks are not limited to the original element boundaries, the objects can form irregularly shaped shards and edges as they shatter. The result is realistic fracture patterns (see Figure 1b).

In the computer graphics literature, two previous techniques were developed for modeling dynamic fracture caused by deformations. In 1988, Demetri Terzopoulos and Kurt Fleischer at Schlumberger Research introduced a general technique for animating viscoelastic and plastic deformations [10]. They modeled certain fracture effects by severing the connection between two nodes when the distance between them exceeded a threshold. They demonstrated this technique by simulating sheets of paper and cloth that could be torn apart. In 1991, Alan Norton and his colleagues at IBM Research broke a model of a china teapot using a similar technique [5]. Both of these animation techniques produced good results, especially considering the computational resources available at the time. But both also allow fractures only along the boundaries in the initial mesh structure, creating artifacts in the crack pattern. These artifacts are especially noticeable when the mesh used to define the objects follows a regular pattern, creating an effect similar to the "jaggies" that occur when rasterizing a polygonal edge.

Fracture has been studied more extensively in the mechanics literature. Fundamentally, a material fractures when the forces acting on an atomic level are sufficiently large to overcome the interatomic bonds holding the material together. The mechanical literature contains theories that abstract this small-scale description of fracture to a macroscopic level where it can be used with a continuum model. A comprehensive review of this work can be found in [1] and a good survey in [4].

The requirements for simulation techniques in graphics and in engineering are quite different from one another. Engineering applications require the simulation be able to accurately predict real-world behaviors. In computer animation, simulations of physical phenomena are tools that allow the animator to realize a preconceived behavior. As a result, numerical accuracy is less important, while issues relating to visual appearance, ease of use, and computational efficiency are critical. Although the techniques described here draw heavily from the fields of fracture mechanics and finite element analysis, the differing requirements in graphics and engineering applications allow simulation techniques for computer animation to make use of simplifications that would be unacceptable in an engineering context. Conversely, some of the assumptions used in engineering applications, such as symmetry, are not acceptable for computer animation.

Creating Fractures As a Material Deforms

Our goal in computer animation is to model fractures created as a material deforms. First, we need a model of the material's deformation providing information about the magnitude and orientation of the material's internal stresses and whether they are compressive or, conversely, tensile. We therefore derived our deformation technique by defining a set of differential equations describing the aggregate behavior of the material in a continuous fashion, then using a finite element method to discretize these equations for computer simulation. This approach is fairly standard, and many different deformation models can be arrived at in this fashion.

The continuous model is based on continuum mechanics (see [2] for an excellent introduction to this area). The primary assumption in the continuum approach is that the scale of the effects being modeled is significantly greater than the scale of the material's composition. Therefore, the behavior of the molecules, grains, or particles composing the material can be mathematically modeled as a continuous media. Although this assumption is often reasonable for modeling deformations, macroscopic fractures can be significantly influenced by effects occurring at small scales where this assumption may not be valid. For example, microscopic scratches in a drinking glass can concentrate stress, causing fracture in situations in which a new, unscratched glass would not break. Because we are interested in graphical appearance, rather than rigorous physical correctness, we assume a continuum model is adequate.

Strain, which measures how much a material has deformed, can be defined in a number of different ways; in our work, we use the strain formulation due to Green [3]. Green's nonlinear metric measures only deformation; it is invariant with respect to rigid-body transformations and vanishes when the material is not deformed. In addition to the strain tensor, we use the strain rate tensor, which measures the rate at which the strain is changing. It is defined as the time derivative of strain and exhibits the same invariant properties as the strain tensor.

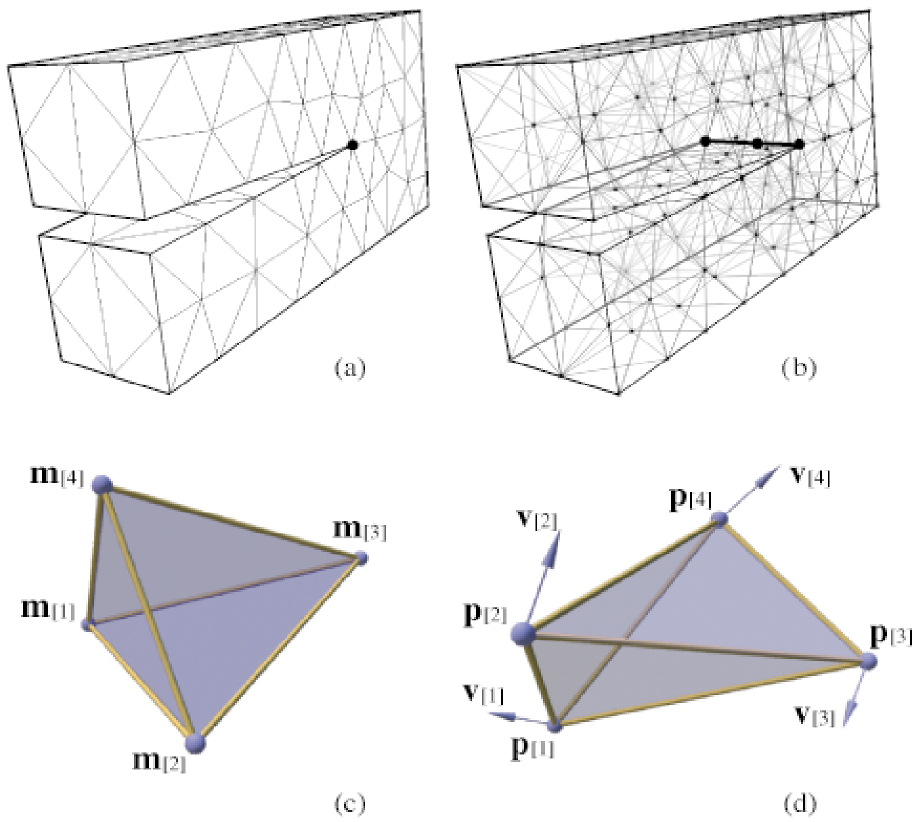
Because cracks are not limited to the original element boundaries, the objects can form irregularly shaped shards and edges as they shatter.

Although the strain and strain rate tensors provide the raw information required to compute internal elastic and damping forces, they do not account for the properties of the material. The stress tensor combines the basic information from the strain and strain rate with the material properties in order to determine forces internal to the material. For many materials, a linear relationship among the components of the stress and strain tensors is adequate. When a material is isotropic, the linear relationship is "parameterized" by two independent variables—μ and λ—known as the Lamé constants of the material. The material's rigidity is determined by the value of μ, while λ controls the resistance to changes in volume (dilation). Similar parameters relate the strain rate to damping stress.

When the stress is computed directly from the strain in this way, the material behaves elastically—meaning the force exerted depends only on how much the material has been deformed. For example, the force exerted by an ideal spring is determined completely by how far the spring has been stretched. In contrast, the stress in a plastic material depends on how the material has been deformed in the past. A nonideal spring stretched too far deforms plastically and bends. Once the spring is bent, the force exerted depends on both how far it is currently stretched and how it was stretched previously.

Although an elastic relationship between stress and strain is adequate for defining the behavior of most

Figure 2. Tetrahedral mesh for a simple object. In (a), only the external faces of the tetrahedra are drawn; (b) shows the internal structure. A tetrahedral element is defined by its four nodes, each with (c) a location in the material coordinate system and (d) a position and velocity in the world coordinate system.



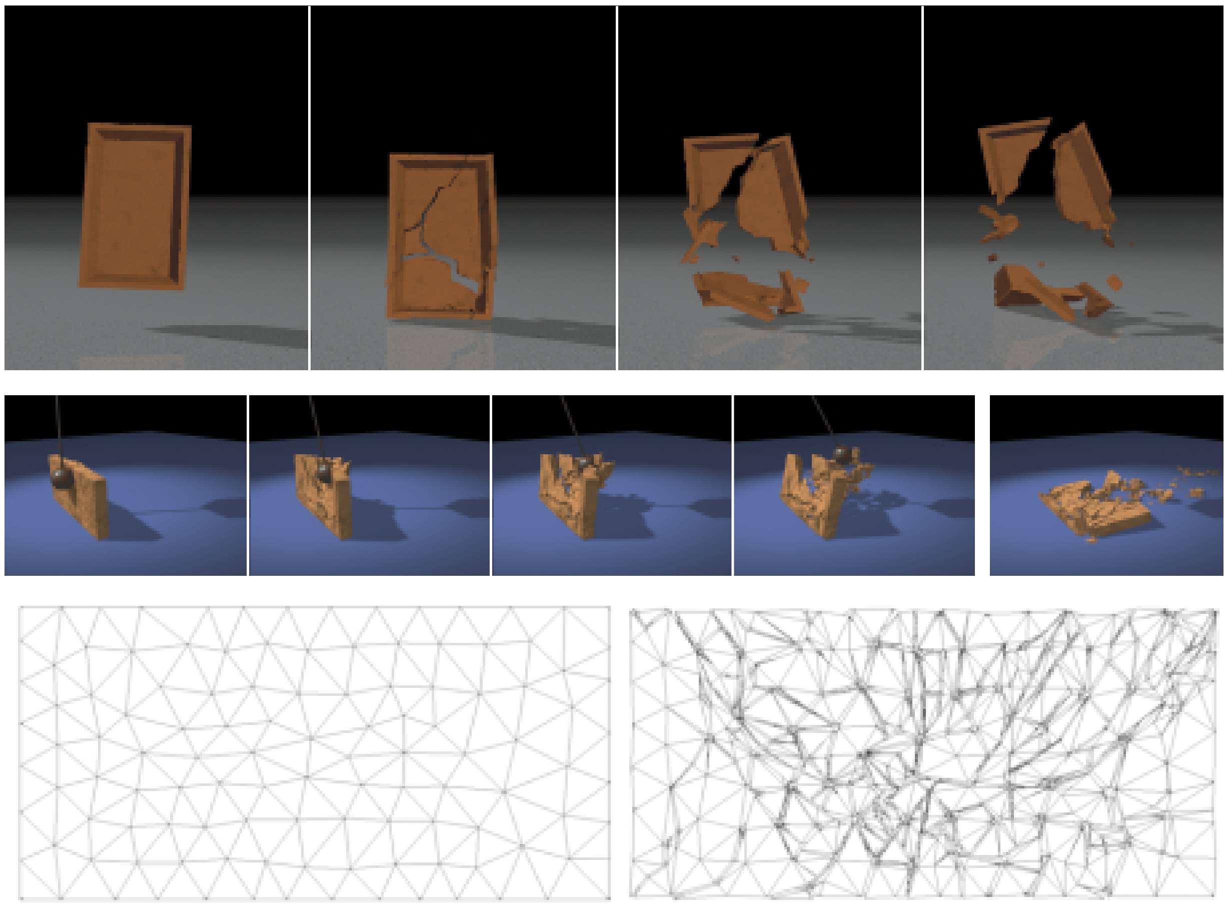
materials, plasticity also plays an important role in fracture. Ductile materials, including many metals, that tear do so largely because of the material's plasticity. Plastic behavior can be added by separating the strain into two components—one elastic and one plastic. The plastic strain is subtracted from the total strain, yielding the elastic strain, which is then used to compute the stress according to the elastic stress-to-strain relationship.

The definitions used to compute such quantities as the strain and the strain rate assume the existence of a function that maps an object from some reference state to its deformed configuration. We use a finite element method to perform this mapping by tesselating, or decomposing, the material into distinct elements, as shown in Figure 2. Within each element, the material is described locally by a function with a finite number of parameters associated with the nodes of the element. Our discretization method employs tetrahedral finite elements with linear polynomial shape functions. Just as triangles can be used to approximate any surface, tetrahedra can be used to approximate arbitrary volumes. Adjacent elements have nodes in common, so the mesh defines a piecewise linear function over the entire material domain.

Each tetrahedral element is defined by four nodes, each with a position in the material coordinates, a position in the world coordinates, and a velocity in world coordinates (see Figures 2c and 2d.) The continuous functions that determine strain, strain rate, plastic strain, and stress within the object are now described in a piecewise fashion in terms of the values at the nodes. The stress is used to compute the internal forces acting on the nodes. These forces determine the accelerations, and the motion of the deformable object is computed by numerically integrating the entire system forward in time.

Although our system works with solid tetrahedral volumes, rather than with the polygonal boundary representations created by most modeling packages, a number of commercial and public-domain systems are available for creating tetrahedral meshes from polygonal boundaries. The models we used were generated either from a constructive solid-geometry description or a polygonal-boundary representation using NETGEN, a

Figure 3. First row: A terra-cotta tray dropped onto a hard floor (images spaced 20ms apart). Second row: An adobe wall struck by a wrecking ball. The bottom edge of the wall is attached to the ground plane; the farthest-right image shows the final configuration (images spaced 66.6ms apart). Third row: Mesh for the adobe wall; (left) the facing surface of the initial mesh used to generate the wall; (right) the reassembled mesh after being struck by the wrecking ball.



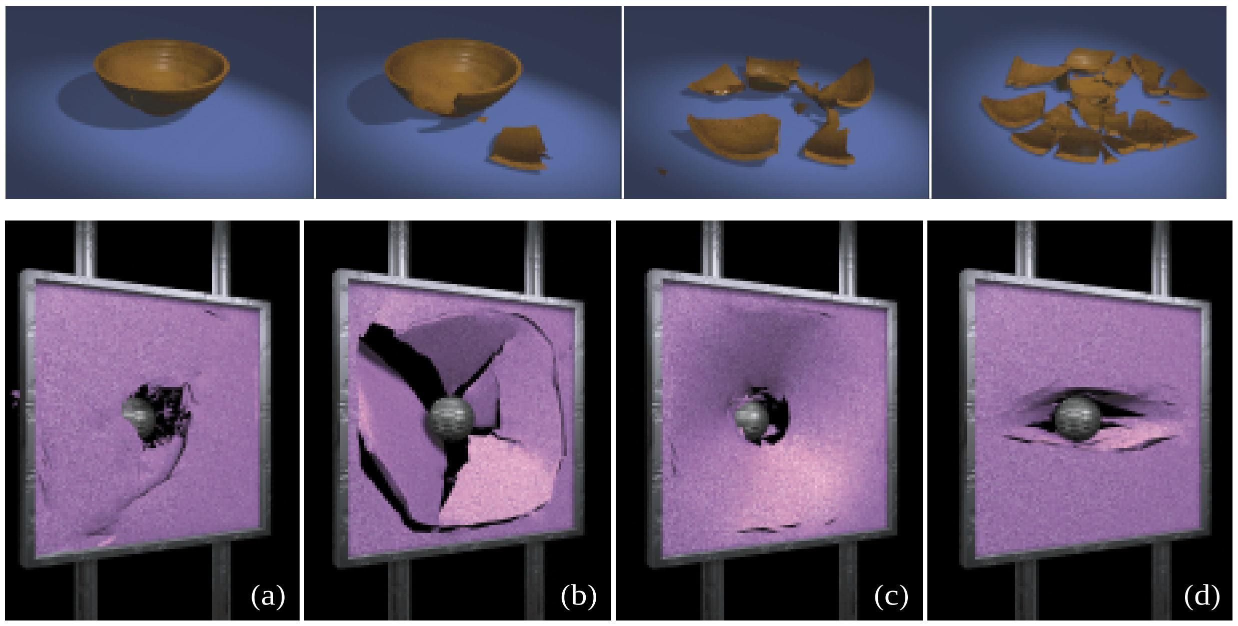
publicly available mesh-generation package [9].

In addition to the forces internal to the material, the system computes collision forces when two elements intersect or if an element violates a geometric constraint, such as the ground plane. Because determining which elements intersect can be computationally very expensive, we use a dynamic hierarchy of bounding boxes with cached traversals to reduce the cost.

To compute collision forces, we use a penalty method that applies a force based on the volume of the region formed by the intersection of two objects. The overlapping volume is computed by clipping the faces of each tetrahedron against the other. A penalty force proportional to the volume, acting at the center of mass of the intersecting region, is applied to the two tetrahedra. Provided that neither tetrahedra is completely contained within the other, this criteria is quite robust, and the forces computed with this method do not depend on the object's tessellation.

Our fracture algorithm works in the following way: After every time step, the system resolves the internal forces acting on all nodes into their tensile and compressive components. At each node, the resulting forces are then used to form a tensor describing how the internal forces are acting to cause a fracture at that node. If one of the eigenvalues, or principle modes, of this separation tensor is sufficiently large, a fracture plane is computed. The resistance of a material to fracture is characterized by the material's toughness parameter. When a fracture occurs, the node is split in two, and all elements attached to the node are divided along the plane, with the resulting tetrahedra assigned to one or the other incarnations of the split node, thus creating a discontinuity in the material. The algorithm re-meshes the local area surrounding the new fracture by splitting elements intersecting the fracture plane and modifying neighboring elements to ensure the mesh remains self-consistent. This re-meshing

Figure 4. First row: Bowls with successively lower toughness values and otherwise identical material properties, each dropped from the same height. Second row: A series of sheets struck by a heavy projectile: (a) stiff, very brittle material; (b) low-density, slightly flexible, but still brittle, material; (c) stiff material that deforms plastically; and (d) another material that yields plastically but with material properties that are directionally dependent, or "anisotropic."



preserves the orientation of the fracture plane and avoids the artifacts produced by other methods.

A Range of Effects

We have animated a number of scenes involving the breaking of objects to demonstrate the range of effects that can be generated with this technique. For example, Figure 1a shows a simulation of a hollow, ceramic model of the Stanford Bunny as it is struck by a heavy, fast-moving weight. Figure 1b shows a plate of glass after a heavy weight, and has been dropped on it. The area near the impact is crushed into many small fragments; farther away, a pattern of radial cracks has developed.

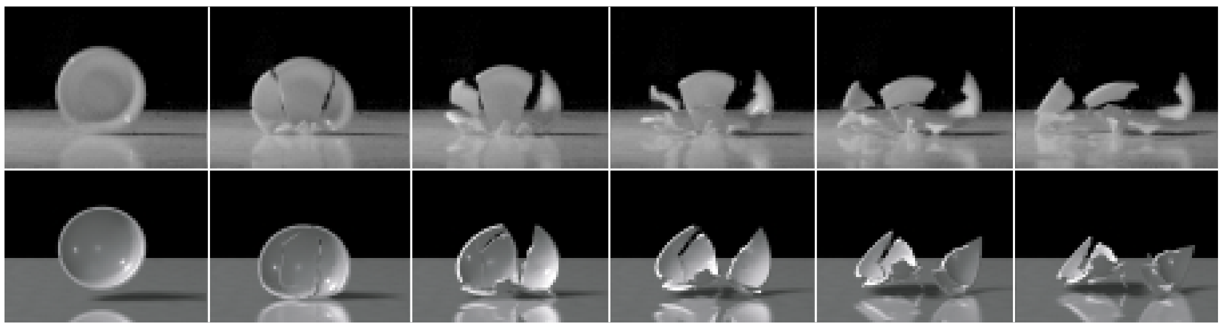
The first row of Figure 3 shows an object dropped onto a hard surface. The second and third rows show a wall being struck by a wrecking ball and the mesh used to generate the wall sequence. The initial mesh contains only 338 nodes and 1,109 elements. By the end of the sequence, the mesh has grown to 6,892 nodes and 8,275 elements, because additional nodes and elements are created wherever fractures occur. A uniform mesh would require many times this number of nodes and elements to achieve a similar result.

The simulation parameters correspond to physical quantities that can be adjusted by the animator to achieve the desired effect. The first row of Figure 4 shows the final frames from four animations of bowls dropped onto a hard surface. Other than the toughness of the material, the four simulations are identical. A more varied set of materials is shown in the second row of the figure.

The physical parameters of the simulation allow a wide range of effects to be modeled, but selecting the combinations of parameters needed to produce a desired result is often quite difficult. Although many parameters, including a material's stiffness, have intuitive meanings when viewed in isolation, the interactions among parameters can be quite complex. Many of the quantities an animator may want to control, including the violence with which an object shatters, do not correspond directly to any single parameter. One solution to this problem is to develop mappings from intuitive parameters, such as "shattering violence," to sets of simulation parameters automatically adjusted to produce the desired effect.

Our approach avoids the "jaggy" artifacts in the fracture patterns caused by the underlying mesh, even though the results of the simulation are still influenced by the mesh structure. The deformation of the material is constrained by the degrees of freedom in the mesh, which in turn dictates how the material can fracture. This limitation occurs with any discrete system. Our technique also restricts the particular location a fracture can start by examining only the existing nodes. As a result, very coarse mesh sizes

Figure 5. Comparison of a real-world event and a simulation. Top row: High-speed video images of a real physical ceramic bowl dropped from approximately one meter onto a hard surface. Bottom row: Output from a simulation in which we attempted to match the initial conditions of the physical bowl. (Video images 8ms apart; simulation images 13ms apart.)



might behave unintuitively. However, nodes occur at the concavities and sharp features where a fracture is most likely to begin. Therefore, with a reasonable grid size, this limitation is not a serious handicap.

A more serious limitation involves the speed at which a crack propagates, because the distance a fracture can travel during a time step is determined by the size of the existing mesh elements. The crack may either split or not split an element; it cannot travel only a fraction of the distance across an element but has to travel all the way. If the crack's speed is significantly greater than the element width divided by the simulation time step, then a high-stress area would race ahead of the crack tip, causing spontaneous failures to occur in the material. We have developed heuristics for dealing with this situation by approximating the stresses at the new crack tip and allowing the crack to advance across multiple elements in a single time step.

A second type of artifact can occur if a crack were being opened slowly by an applied load on a model with a coarse resolution mesh. This scenario would lead to a "button popping" effect, whereby the crack travels across one element, pauses until the stress builds up again, then moves across the next element. Although we have not observed this phenomena in our examples, developing an algorithm allowing smooth, slow fracture propagation is an area for future work.

Our goal is the realistic animation of fracture. However, assessing subjective quantities, such as realism, in the appearance of a fracture animation is quite difficult. One possible way to do so is to compare computer-generated results with images from the real world. Figure 5 contrasts high-speed video footage of a real bowl as it falls onto its edge with our imitation of the real-world scene. Although the two sets of fracture patterns are clearly different, the simulated bowl shares some qualitative similarities with the real one. Both initially fail along the leading edge where they strike the ground and subsequently develop vertical cracks before breaking into several large pieces.

For more about our work on animating fracture and animated sequences corresponding to the images here, see www.gvu.gatech.edu/animation/fracture/. C

References

1. Anderson, T. *Fracture Mechanics: Fundamentals and Applications, 2nd Ed.* CRC Press, Boca Raton, Fla., 1995.
2. Fung, Y. *A First Course in Continuum Mechanics.* Prentice-Hall, Englewood Cliffs, N.J., 1969.
3. Fung, Y. *Foundations of Solid Mechanics.* Prentice-Hall, Englewood Cliffs, N.J., 1965.
4. Nishioka, T. Computational dynamic fracture mechanics. *Int. J. Fract. 86,* 2 (1997), 127–159.
5. Norton, A., Turk, G., Bacon, B., Gerth, J., and Sweeney, P. Animation of fracture by physical modeling. *Vis. Comput.* 7, 4 (July 1991), 210–217.
6. O'Brien, J. and Hodgins, J. Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH'99* (Los Angeles, Calif., Aug. 8–13). ACM Press, New York, 1999, 137–146.
7. Robertson, B. *Antz*-piration. *Comput. Graph. World 21,* 10 (Oct. 1998).
8. Robertson, B. Building a better mouse. *Comput. Graph. World 22,* 12 (Dec. 1999).
9. Schöberl, J. NETGEN—An advancing front 2D/3D-mesh generator based on abstract rules. *Comput. Visualiz. Sci.* 1 (1997), 41–52; see also www.sfb013.uni-linz.ac.at/~joachim/netgen/.
10. Terzopoulos, D. and Fleischer, K. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proceedings of SIGGRAPH'88* (Atlanta, Ga., Aug. 1–5). ACM Press, New York, 1988, 269–278.

JAMES O'BRIEN (job@acm.org) is a Ph.D. candidate in the College of Computing at the Georgia Institute of Technology and a member of the Graphics, Visualization & Usability Center.

JESSICA HODGINS (jkh@cc.gatech.edu) is an associate professor in the College of Computing and Graphics, Visualization & Usability Center at the Georgia Institute of Technology, where she heads the Animation Lab.

This research is supported in part by National Science Foundation NYI Grant No. IRI-9457621, Mitsubishi Electric Research Laboratory, and a Packard Fellowship. The first author was supported by a fellowship from the Intel Foundation.

Thanks to Wayne L. Wooten, Pixar Animation Studios, for helping light, shade, and render the images in this article.