Interactive Deformation Using Modal Analysis with Constraints

Kris K. Hauser Chen Shen James F. O'Brien

EECS, Computer Science Division
University of California, Berkeley

Abstract

Modal analysis provides a powerful tool for efficiently simulating the behavior of deformable objects. This paper shows how manipulation, collision, and other constraints may be implemented easily within a modal framework. Results are presented for several example simulations. These results demonstrate that for many applications the errors introduced by linearization are acceptable, and that the resulting simulations are fast and stable even for complex objects and stiff materials.

Key words: Animation techniques, physically based modeling, simulation, dynamics, deformation, modal analysis, modal synthesis, finite element method, video games, interactive simulation, real-time simulation, constraints, precomputed dynamics.

1 Introduction

Interactive modeling of deformable objects has a wide range of applications from surgical training to video games. Many of these applications require realistic, real-time simulation for complex objects. Unfortunately, the most straightforward simulation methods turn out to be prohibitively expensive for modeling objects of even modest complexity. When the high cost of simulation couples with the reality that CPU cycles must be shared among many tasks, the need for faster, more sophisticated simulation methods becomes clear.

Recently several ingenious techniques for modeling deformable objects have been proposed. Examples include multi-resolution representations that avoid wasting time on irrelevant details (*e.g.* [4, 6, 8]), reformulating the dynamics to make them more stable (*e.g.* [15, 20]), extensive precomputation to minimize runtime costs (*e.g.* [9, 10, 19, 22]), robust integration schemes that afford large time-steps (*e.g.* [3]), and many other approaches that we cannot list here due to space constraints. As of yet, none provides a perfect solution that satisfies the requirements for all interactive applications.

This paper reexamines a technique known as modal analysis that was originally introduced to the graphics community over a decade ago, but has since been largely neglected, with only a couple of notable exceptions (*e.g.* [10, 22, 23]). Like the techniques mentioned above, modal analysis does not provide a perfect solution for every interactive application, but it does provide a solution that suits some applications quite well.

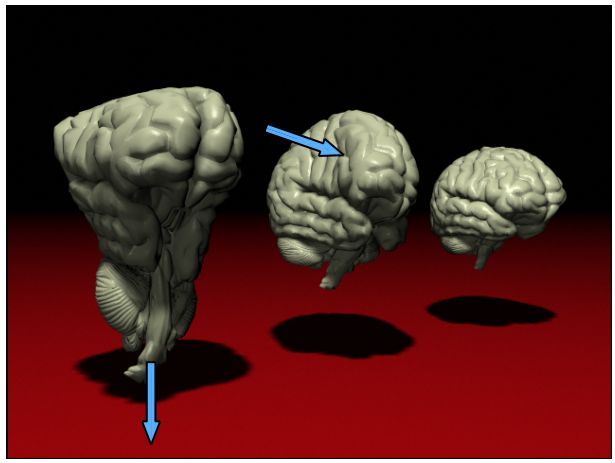

Figure 1: This example demonstrates a complex model being deformed using a modal simulation method. The object furthest from the viewer shows the undeformed configuration. The nearer objects are being deformed by a force indicated by the blue arrows.

The results presented here show that modal analysis can be used effectively to model situations where the deformable object is directly manipulated using constraints and where it interacts with an environment through contact forces. We demonstrate that although linear modal analysis does incur errors because of the inherent linearization of the dynamics, these errors are acceptable in many contexts, particularly when exaggerated cartoon-like deformations are desired. While precomputing the modal decomposition for a complex object may take up to a few hours of precomputation, for applications which make use of fixed content this computational cost only occurs during content development and it is well worth the dramatic increase in runtime performance.

The concepts required to manipulate the modal equations are to a certain extent conceptually difficult to work with but their implementation is surprisingly simple. The results shown in this paper (*e.g.* figure 1) were gener-

ated using an implementation that we have ported to several platforms: SGI IRIX, Windows, Linux, and Sony PlayStation2. On each of these platforms we were able to obtain interactive simulation times even for relatively complex models.

2 Background

Modal analysis is a well established mathematical technique that has been used extensively in mechanical, aerospace, civil, and other engineering disciplines for several decades. To a large extent the work we present in this paper follows as direct application of the methods developed in those fields to the task of interactively simulating deformable solids. There are, however, some issues that are unique to interactive simulation, such as imposing manipulation constraints and computing fast collision responses. This paper focuses on those issues. A discussion of modal analysis and its use with the finite element method can be found in the text by Cook, Malkus, and Plesha [5], and a more detailed discussion of modal analysis, its mathematical theory, and its applications may be found in the text by Maia and Silva [13].

Modal analysis was first introduced to the graphics community in 1989 by Pentland and Williams as a fast method for approximating deformation [19]. They used a hybrid framework, previously described by Terzopoulos and Fleischer [24], that separated the motion of a deformable solid into a rigid component and a deformation component. The deformable component existed in a non-inertial reference frame that moved with the rigid component. To avoid the cost of computing the modes for a particular object Pentland and Williams used linear and quadratic deformation fields defined over a rectilinear volume instead of the object's actual modes and then embedded the object within the region in a fashion similar to a free-form deformation. Although using approximated modes is computationally inexpensive, it only generates reasonable results for compact objects that are well approximated by a rectilinear solid. Pentland and his colleagues also integrated their modal deformation techniques into a interactive modeling system [18].

In 1997 Stam developed a modal method for modeling trees blowing in the wind [23]. Rather than starting with a deformable object, he computed the low-frequency modes from an articulated structure that described the tree. Once the closed-form solutions for each mode were computed, the response of the tree to a stochastic wind field could be computed efficiently.

Most recently, James and Pai implemented a system for computing real-time modal deformations on commodity graphics hardware [10]. They focused on modeling deformable skin and soft tissues attached to moving characters or as background elements in a surgical simulation. Shen and his colleagues have demonstrated an interactive system that could simulate models with over 10,000 vertices on a laptop PC with no special hardware acceleration [22].

Other related work includes sound generation techniques that make use of modal synthesis, and deformation techniques that use global shape functions that have some general similarities to a object's mode shapes. Van den Doel and his colleagues have used both analytically computed modes for simple geometric shapes and sampled modes from real objects to compute realistic sounds for simulated environments [26, 27, 28]. O'Brien and his colleagues developed similar techniques that used numerically computed modes from a finite element description of an object [17]. Examples of deformation techniques using global shape functions include: free-form deformations and their dynamics extensions [7, 21], deformable superquadrics [14], and the boundary element method [9]. Modal bases have also proven to be an efficient way to compactly encode both shapes and deformations [11, 12]. Finally, this paper focuses primarily on integrating manipulation and contact constraints into a modal framework, and there is prior work on applying these types of constraints to flexible body simulations [2].

3 Methods

The mechanical properties of an object can generally be captured by a function that maps the state of the object to a distribution of internal forces. For nearly any non-trivial system this function will be nonlinear and the representation of state will require many variables. Consequently, modeling the object's behavior over time will involve integrating a large, nonlinear system of differential equations. These systems are typically far too complex to be solved analytically, so some type of numerical solution method must be employed.

Modal analysis is the process of taking the nonlinear description of a system, finding a good linear approximation, and then finding a coordinate system that diagonalizes the linear approximation. This process transforms a complicated system of nonlinear equations into a simple set of decoupled linear equations that may be individually solved analytically.

The main benefit of this modal approach is that the behavior of the system can be computed much more efficiently. Because each of the decoupled equations can be solved analytically, the stability limitations that plague numerical integration methods are eliminated. Further, one may examine each of the decoupled components and discard those that are irrelevant to the problem at hand.

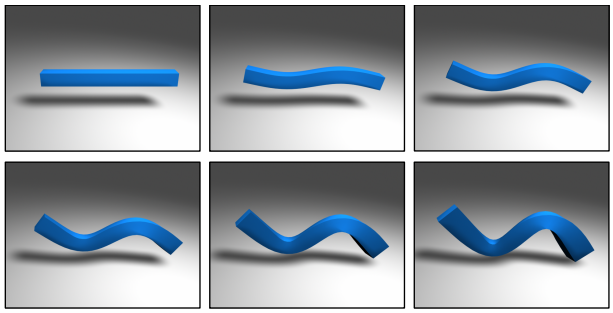

Figure 2: Using a linear formulation to model a bending bar produces acceptable results for small to moderate amounts of deformation. For larger deformations significant amounts of distortion appear. This example shows the deformation corresponding to the bar's second transverse mode.

There are also two drawbacks to a modal approach. First, linearizing the original nonlinear equations means that the solution will only be a first order approximation of the true solution. How objectionable the linearization error is depends on the application and the extent to which the objects deform from their initial configurations. As illustrated by figure 2, small to moderate deformations exhibit little or no noticeable error when casually observed. Even when the errors do grow noticeable, they have a cartoon-like, exaggerated appearance that may actually be desirable for some applications.

The second drawback arises because decoupling the linear system requires computing its eigendecomposition. However we do not believe that this drawback is particularly significant. The content in most interactive applications is constant, so that eigendecompositions can be precomputed during content development and stored with the objects. Furthermore, the linear systems are sparse, so that fast, robust, publicly available codes may be used to efficiently compute the decompositions (*e.g.* TRLAN [29]).

The remainder of this section describes how one computes the modal decomposition for a given object and how that decomposition can be used to efficiently model the object's behavior. Some of this material has been presented elsewhere by others in the graphics community (*e.g.* [10, 19]) but we include it here for completeness. The discussion will focus in particular on including manipulation and collision constraints in the modal framework. An overview of the entire process is shown in figure 3.

3.1 Modal Decomposition

The modal decomposition of a physical system begins with a linear set of equations that describe the system's behavior. In general, the equations describing the system

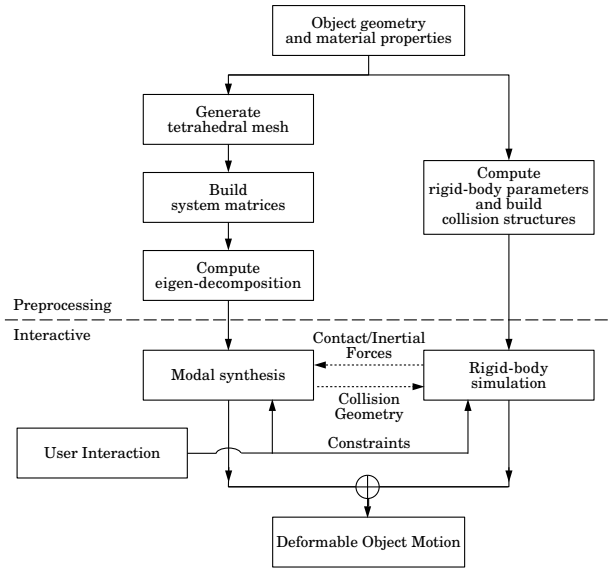

Figure 3: This diagram illustrates both the preprocessing steps used to construct the deformable modal model for an object, and the processes that subsequently generate interactive motion from this description.

may be nonlinear, and one obtains the linear equations by linearizing about some point, typically the rest configuration of the system. The linearized equations have the general form:

$$\boldsymbol{K}\boldsymbol{d} + \boldsymbol{C}\dot{\boldsymbol{d}} + \boldsymbol{M}\ddot{\boldsymbol{d}} = \boldsymbol{f} \ , \qquad (1)$$

where $\boldsymbol{K}$, $\boldsymbol{C}$, and $\boldsymbol{M}$ are respectively known as the system's stiffness, damping, and mass matrices, $\boldsymbol{d}$ and $\boldsymbol{f}$ respectively as the vector of generalized displacements and forces, and an overdot indicates differentiation with respect to time. The physical meaning of the generalized force and displacement vectors, and the method for computing the system matrices will depend on the type of method used for modeling the system. For general finite element methods, we refer the reader to the excellent text by Cook, Malkus, and Plesha [5]. We are using an implementation of the piecewise-linear tetrahedral finite element method described by O'Brien and Hodgins [16]. Details on computing the system matrices appear in [17].

Modal decomposition refers to the process of diagonalizing equation (1). The most general form of modal decomposition can be used for nearly arbitrary systems, but the systems arising from the finite element method we use have a structure that makes them amenable to a simpler manipulation provided we assume that the damping matrix, $\boldsymbol{C}$, is a linear combination of the $\boldsymbol{K}$ and $\boldsymbol{M}$. This restriction is known as Rayleigh damping, and although it is a restriction it still produces results superior to the simple mass damping that is most commonly used in

graphics applications. With these conditions, diagonalizing equation (1) becomes equivalent to solving a generalized symmetric eigenproblem with symmetric, positive-definite matrices. Cook, Malkus, and Plesha describe the process in detail and we only repeat the end result here.

With the restriction of Rayleigh damping equation (1) may be rewritten as:

$$\boldsymbol{K}(\boldsymbol{d} + \alpha_1 \dot{\boldsymbol{d}}) + \boldsymbol{M}(\alpha_2 \dot{\boldsymbol{d}} + \ddot{\boldsymbol{d}}) = \boldsymbol{f} \ , \tag{2}$$

where α_1 and α_2 are the Rayleigh coefficients. Let the columns of $\boldsymbol{W}$ be the solution to the generalized symmetric eigenproblem $\boldsymbol{K}\boldsymbol{x} + \lambda \boldsymbol{M}\boldsymbol{x} = 0$ and $\boldsymbol{\Lambda}$ be the diagonal matrix of eigenvalues[1], then equation (2) may be transformed to:

$$\boldsymbol{\Lambda}(\boldsymbol{z} + \alpha_1 \dot{\boldsymbol{z}}) + (\alpha_2 \dot{\boldsymbol{z}} + \ddot{\boldsymbol{z}}) = \boldsymbol{g} \ , \tag{3}$$

where $\boldsymbol{z} = \boldsymbol{W}^{-1}\boldsymbol{d}$ is the vector of modal coordinates and $\boldsymbol{g} = \boldsymbol{W}^{\mathsf{T}}\boldsymbol{f}$ is the external force vector in the modal coordinate system.

Each row of equation (3) corresponds to a single scalar second-order differential equation:

$$\lambda_i z_i + (\alpha_1 \lambda_i + \alpha_2)\dot{z}_i + \ddot{z}_i = g_i \ . \tag{4}$$

The analytical solutions to each equation are

$$z_i = c_1 e^{t\omega_i^+} + c_2 e^{t\omega_i^-} \tag{5}$$

where c_1 and c_2 are arbitrary (complex) constants, and ω_i is the complex frequency given by

$$\omega_i^{\pm} = \frac{-(\alpha_1 \lambda_i + \alpha_2) \pm \sqrt{(\alpha_1 \lambda_i + \alpha_2)^2 - 4\lambda_i}}{2} \ . \tag{6}$$

The absolute value of the imaginary part of ω_i is the frequency (in radians/second, not Hertz) of the mode, and the real part is the mode's decay rate. In the special case where the term under the radical in equation (6) is zero, we have $\omega_i^+ = \omega_i^-$, which gives the critically damped solution:

$$z_i = c_1 t e^{t\omega_i} + c_2 e^{t\omega_i} \ . \tag{7}$$

The columns of $\boldsymbol{W}$ are the vibrational modes of the object being modeled. (See figure 4.) Each mode has the property that a displacement or velocity over the object that is a scalar multiple of the mode will produce an acceleration that is also a scalar multiple of the mode. This property means that the modes do not interact with each other, which is why decoupling the system into a set of independent oscillators was possible. The eigenvalue for each mode is the ratio of the mode's elastic stiffness to the mode's mass, and it is the square of the mode's natural frequency (in radians per second). In general the eigenvalues will be positive, but for each free body in the system there will be six zero eigenvalues that correspond to the body's six rigid-body modes. The rigid-body eigenvalues are zero because a rigid-body displacement will not generate any elastic forces.

[1] Equivalently let $\boldsymbol{W} = \boldsymbol{L}^{-\mathsf{T}}\boldsymbol{V}$ where $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^{\mathsf{T}}$ (Cholesky decomposition) and $\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathsf{T}} = \boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{L}^{-\mathsf{T}}$ (symmetric eigendecomposition).

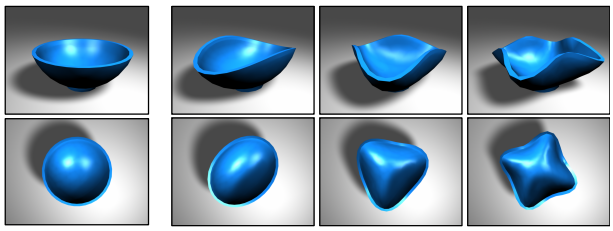

Figure 4: The two rows show a side and top view of a bowl along with three of the bowl's first vibrational modes. The modes selected for the illustration are the first three non-rigid modes with distinct eigenvalues that are excited by a transverse impulse to the bowl's rim.

The decoupled system of equations is *not an approximation* of the original linear system, it will generate exactly the same results as the original linear system. Of course the linear system may have been an approximation to some initial nonlinear one, but any problem that could be solved using equation (2) could also be solved with equation (3). Furthermore, simulation that would have required numerical time integration of equation (1) can now be solved without integration using the analytical solutions in equations (5) or (7).

3.2 Discarding Modes

Although decoupling equation (1) and then solving each of the resulting components analytically provides significant benefits, we can derive additional benefit by considering whether or not each of these components is needed. In particular we can discard modes that will have no significant effect on the phenomena we wish to model.

If the eigenvalue, λ_i, associated with a particular mode is large, then the force required to cause a discernible displacement of that mode will also be large. We can expect that in a given environment there will be both an upper bound on the magnitude of the forces encountered and a lower limit on the amplitude of observable movement. For example, if modeling an indoor environment we would not expect to encounter forces in excess of $60,000\,\mathrm{N}$ (the braking force of a large truck), and we would not be able to observe displacements less than about $0.1\,\mathrm{mm}$. Thus if $||\boldsymbol{w}_i||^2/\lambda_i < \mathrm{min\_res}/\mathrm{max\_frc}$ for some mode then that mode's behavior will be unobservable.

The imaginary part of ω_i determines the frequency that a mode will vibrate at. Modes that vibrate at more than half the display's frame rate will cause temporal aliasing.

Removing modes that are too stiff and/or too high frequency to be observed will not change the appearance of

the resulting simulation, but removing them will greatly reduce the simulation's cost. For most objects that we have worked with, nearly all of the modes are unobservable. A typical result is that an object with several thousand vertices will have many fewer than fifty modes that need to be retained. Furthermore, the number of modes that must be retained is nearly independent from the resolution of the model.

For later convenience let $\bar{\boldsymbol{W}}$ be the matrix $\boldsymbol{W}$ with the *columns* corresponding to the discarded modes removed, and let $\bar{\boldsymbol{W}}^{-1}$ be the matrix $\boldsymbol{W}^{-1}$ with the *rows* corresponding to the discarded modes removed. Note that $\bar{\boldsymbol{W}}^{-1} \neq (\bar{\boldsymbol{W}})^{-1}$, $\bar{\boldsymbol{W}}$ and $\bar{\boldsymbol{W}}^{-1}$ are not square, and $\bar{\boldsymbol{W}}^{-1}\bar{\boldsymbol{W}} = \boldsymbol{I}$ but $\bar{\boldsymbol{W}}\bar{\boldsymbol{W}}^{-1} \neq \boldsymbol{I}$.

3.3 Oscillator Coefficients and Time Steps

The analytical solution for each mode, equation (5), describes how that mode will behave when no external forces are acting on it. Using these solutions, however, requires some way of modeling responses to external forces and of setting initial conditions.

Given a set of initial conditions described by the node positions, $\boldsymbol{d}_0$, and their velocities, $\dot{\boldsymbol{d}}_0$, setting the oscillators to match those conditions requires finding appropriate values for the coefficients c_1 and c_2. First, the initial conditions are transformed to modal coordinates: $\boldsymbol{z}_0 = \bar{\boldsymbol{W}}^{-1}\boldsymbol{d}_0$ and $\dot{\boldsymbol{z}}_0 = \bar{\boldsymbol{W}}^{-1}\dot{\boldsymbol{d}}_0$. For each mode, c_1 and c_2 are given by

$$c_1 = \frac{z_0}{2} + \frac{(\alpha_1\lambda_i + \alpha_2)z_0 + 2\dot{z}_0}{2\sqrt{(\alpha_1\lambda_i + \alpha_2)^2 - 4\lambda_i}} \quad (8)$$

$$c_2 = \frac{z_0}{2} - \frac{(\alpha_1\lambda_i + \alpha_2)z_0 + 2\dot{z}_0}{2\sqrt{(\alpha_1\lambda_i + \alpha_2)^2 - 4\lambda_i}} \; . \quad (9)$$

For the critically damped case c_1 and c_2 are given by

$$c_1 = \frac{(\alpha_1\lambda_i + \alpha_2)z_0}{2} + \dot{z}_0 \quad (10)$$

$$c_2 = z_0 \; . \quad (11)$$

Note that if the $\omega_i^{\pm}$ are real then c_1 and c_2 will also be real. If the $\omega_i^{\pm}$ are complex then the $\omega_i^{\pm}$ and the c_1 and c_2 will be complex conjugate pairs. In either case equation (6) will evaluate to a real value.

To compute the response of a mode to an impulse delivered at $t = 0$, first transform the impulse to modal coordinates with $\Delta t\boldsymbol{g} = \Delta t\bar{\boldsymbol{W}}^{\mathsf{T}}\boldsymbol{f}$ and then compute c_1 and c_2 as shown above with z_0 set to zero and $\dot{z}_0$ replaced by $\Delta t g$. Because the modes behave linearly, the response of the system to forces applied at an arbitrary time may be computed by time-shifting this impulse response and adding it to the existing values.

Because $ce^{(t+\Delta t)\omega} = (ce^{t\omega})e^{\Delta t\omega}$, the state of each oscillator can be stored simply as a pair of complex numbers that reflect the current values of $c_1e^{t\omega^+}$ and $c_2e^{t\omega^-}$. Each time the system is advanced forward in time, these values get multiplied by $e^{\Delta t\omega^{\pm}}$. If Δt is constant then the step multiplier for each mode may be cached to avoid the cost of evaluating an exponential. Impulses applied to the system simply require adding the appropriate values to each oscillator's state. Finally, modes where ω^+ and ω^- are complex conjugate pairs (*i.e.* underdamped modes) can be reduced to only a single oscillator.

3.4 Constraints

Although we can compute the behavior of the decomposed system extremely efficiently, the method is not particularly useful unless it accommodates manipulation constraints and collision response. When working with the original system constraints on the node positions are nearly trivial to implement. Collision response requires more sophistication but still is conceptually straightforward. Unfortunately, applying these same constraints in the modal basis requires moving between the node positions and modal coordinates which can be unintuitive. Matters are further complicated because if we have discarded any modes then the transformations will be non-invertible.

3.4.1 Interactive Manipulation

If we wish to include continual constraints on part of the system, the optimal way to do so is to remove those degrees of freedom prior to performing the modal decomposition. Examples demonstrating this approach can be seen in James and Pai's modal method for modeling tissue deformation [10], and in our deformable sheet example. (See accompanying animations.) Using this approach for dynamic constraints, however, would require recomputing the eigendecomposition each time a constraint was added or removed from the system. James and Pai accomplished something similar for a boundary element method using Sherman-Morrison-Woodbury updates but we do not know of any corresponding incremental update scheme for an eigensystem [9].

Instead we apply manipulation constraints to the decomposed system. Let ψ be the set of degrees of freedom in the original system that we wish to constrain, and let ϕ be the places where we are willing to apply forces in order to enforce the constraints. For a manipulation task where a point on the object is being dragged we would typically have $\phi = \psi$ but we will not require it. Let $\boldsymbol{d}_\psi$ or $\boldsymbol{f}_\phi$ denote the displacement or force vectors where all except the elements corresponding to ϕ or ψ have been removed. Similarly, let $\bar{\boldsymbol{W}}_\psi$ be $\bar{\boldsymbol{W}}$ where all the rows not in ψ have been removed and let $\bar{\boldsymbol{W}}_\phi^{\mathsf{T}}$ be $\bar{\boldsymbol{W}}^{\mathsf{T}}$ where all the columns but for those in ϕ have been removed. Finally, let $\ddot{\boldsymbol{d}}_\psi^*$ be the desired accelerations at the constraint locations. By combining $\ddot{\boldsymbol{d}} = \bar{\boldsymbol{W}}\ddot{\boldsymbol{z}}$, $\boldsymbol{g} = \bar{\boldsymbol{W}}^{\mathsf{T}}\boldsymbol{f}$ and a bit

of manipulation we obtain:

$$\ddot{\boldsymbol{d}}^*_\psi = \bar{\boldsymbol{W}}_\psi(\ddot{\boldsymbol{z}} + \bar{\boldsymbol{W}}^{\mathsf{T}}_\phi \boldsymbol{f}_\phi) \ . \tag{12}$$

Solving for $\boldsymbol{f}_\phi$ yields:

$$\boldsymbol{f}_\phi = \left(\bar{\boldsymbol{W}}_\psi \bar{\boldsymbol{W}}^{\mathsf{T}}_\phi\right)^{-\mathsf{P}} \left(\ddot{\boldsymbol{d}}^*_\psi - \bar{\boldsymbol{W}}_\psi \ddot{\boldsymbol{z}}\right) \ , \tag{13}$$

where $\cdot^{-\mathsf{P}}$ denotes a pseudoinverse. Velocity constraints only differ in that $\boldsymbol{f}_\phi$ gets replaced by an impulse, *e.g.* $\Delta t \boldsymbol{f}_\phi$ and we have:

$$\boldsymbol{f}_\phi = \frac{1}{\Delta t}\left(\bar{\boldsymbol{W}}_\psi \bar{\boldsymbol{W}}^{\mathsf{T}}_\phi\right)^{-\mathsf{P}} \left(\dot{\boldsymbol{d}}^*_\psi - \bar{\boldsymbol{W}}_\psi \dot{\boldsymbol{z}}\right) \ . \tag{14}$$

Position constraints can be enforced in a similar fashion so long as we adjust for how each mode will evolve over the interval while the force is applied:

$$\boldsymbol{f}_\phi = \frac{2}{\Delta t^2}\left(\bar{\boldsymbol{W}}_\psi \boldsymbol{S} \bar{\boldsymbol{W}}^{\mathsf{T}}_\phi\right)^{-\mathsf{P}} \left(\boldsymbol{d}^*_\psi - \bar{\boldsymbol{W}}_\psi \boldsymbol{z}\right) \ , \tag{15}$$

where $\boldsymbol{S}$ the diagonal matrix with components given by

$$s_{ii} = \frac{e^{\Delta t \omega_i^+} - e^{\Delta t \omega_i^-}}{|\sqrt{(\alpha_1 \lambda_i + \alpha_2)^2 - 4\lambda_i}|} \tag{16}$$

that compensates for the motion of each mode during the interval.

3.4.2 Dynamics Simulation

Implementing a deformable dynamics simulator for free bodies using modal analysis can be accomplished by combining the modal simulation with a standard rigid-body dynamics simulator. The modal system is embedded in a rigid-body reference frame, and both systems evolve over time. The two systems interact with each other though inertial effects. The modal system should experience centrifugal and coriolis forces as the rigid-body moves, and the inertial moments of the rigid-body will change as the modal system deforms. Unless the object is rotating rapidly, neither effect will be significant so we omit them. They could be included at an additional computational cost. Inertial effects due to translational and rotational acceleration of the rigid-body frame do not need to be modeled explicitly so long as the forces generating those accelerations are also applied to the modal system.

Because we are modeling deformable objects, a collision detection method optimized for use with rigid-body simulations requires some modification because precomputed data structures will become invalid as the object deforms. The method we are using employs a hierarchy of axis-aligned bounding boxes, aligned to the world axes, to efficiently find potential collisions. The tree is initially constructed based on the undeformed shape of the object. Each leaf node in the tree corresponds to one of the primitives that makes up the object, and the bounding box at that node encloses the primitive. The bounding boxes of interior nodes encompass the union of their children. The tree's topology is chosen to minimize the overlap among the interior nodes. Once the object deforms the tree will become invalid, but recomputing the tree's topology every time-step would be prohibitively expensive. Instead we use an update scheme similar to one described by van den Bergen [25]. After each time-step the bounding boxes are updated, but the tree's topology does not change. If we expected arbitrary deformation, this could result in a very poorly structured tree, but because the extent of deformation is limited we have found this approach to work quite well.

Using these trees the collision system can efficiently determine contact points and a normal for each contact. For collisions between an object and a ground plane, the collision normal is simply the plane's normal. For collisions between objects, we look at involved tetrahedra to determine a normal based on their overlap [16]. We have found that each physical contact site may produce several pairs of colliding primitives. To reduce the computation when using constraint-based collisions we cluster nearby collision points and treat each cluster as a single collision point.

We have implemented collision response using both a penalty-based method and using constraints. As one would expect, the penalty methods require less work per time-step, achieving real-time performance, but stiff penalty coefficients can lead to instability. The constraint-based method requires more work per time-step, but it is more stable. Because the modal system will allow arbitrarily large time-steps in the absence of external influences we prefer the more stable constraint-based methods.

To implement penalty methods, when a point on a surface violates one of the penalty constraints, a force proportional to the magnitude of the violation is applied at that point. Transforming the forces to modal coordinates and then applying the force to the modal system is done as described previously. The penalty force should be applied to both the modal and the rigid-body systems.

Constraint-based collisions require a more complex implementation, but we find that they produce better results. First, when a collision occurs, the simulation is backed up to the point during the time-step when the objects first came into contact. Then contact forces are calculated as the minimal outward normal force to ensure that the objects will not continue to penetrate. These are determined by solving a linear programming problem for the normal forces at all contact points. Baraff details an efficient method for solving for the required forces [1].

Constraint methods are often used in traditional rigid-body simulations only to solve for resting contact, while

impulses are used to calculate elastic response. Elastic components of the response can be handled differently in our modal simulation, because the elastic behavior of the modal system models them directly. We first enforce a velocity constraint that solves for an impulse to ensure that none of the contact velocities are negative, then secondly it enforces an acceleration constraint that solves for a force to ensure that none of the contact accelerations are negative. The derivation of these methods requires equations relating the change in velocity and acceleration with respect to an applied impulse and acceleration, respectively.

Let $\boldsymbol{p}_l$ be the location of a contact point on an object expressed in the local coordinate frame of the rigid body. This location will be a linear function of the modal coordinates so that:

$$\boldsymbol{p}_l = \boldsymbol{U}\boldsymbol{W}\boldsymbol{z} \ , \tag{17}$$

where $\boldsymbol{U}$ is a matrix that averages the appropriate node locations based on the barycentric location of $\boldsymbol{p}$ in one of the surface triangles. The location in world coordinates is given by

$$\boldsymbol{p}_w = \boldsymbol{t} + \boldsymbol{R}\boldsymbol{p}_l \ , \tag{18}$$

where $\boldsymbol{t}$ and $\boldsymbol{R}$ are the translation and rotation matrices for the rigid-body frame. Differentiating with respect to time to obtain the world velocity and acceleration of $\boldsymbol{p}$ yields:

$$\dot{\boldsymbol{p}}_w = \dot{\boldsymbol{t}} + \boldsymbol{R}[\boldsymbol{\omega}]\boldsymbol{p}_l + \boldsymbol{R}\dot{\boldsymbol{p}}_l \ , \tag{19}$$

$$\ddot{\boldsymbol{p}}_w = \ddot{\boldsymbol{t}} + \boldsymbol{R}[\boldsymbol{\omega}][\boldsymbol{\omega}]\boldsymbol{p}_l + \boldsymbol{R}[\boldsymbol{\alpha}]\boldsymbol{p}_l + 2\boldsymbol{R}[\boldsymbol{\omega}]\dot{\boldsymbol{p}}_l + \boldsymbol{R}\ddot{\boldsymbol{p}}_l \ , \tag{20}$$

where $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ are the rigid-body's angular velocity and acceleration[2]. The notation $[\boldsymbol{a}]$ denotes the skew-symmetric matrix such that $[\boldsymbol{a}]\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b} = -[\boldsymbol{b}]\boldsymbol{a}$.

Differentiating equation (19) with respect to an applied impulse allows us to obtain the change in velocity generated by a constraint force over a time interval:

$$\Delta\dot{\boldsymbol{p}}_w = \Delta t \left(\frac{1}{m}\boldsymbol{f}_w + \boldsymbol{R}[\boldsymbol{H}^{-1}\boldsymbol{\tau}_l]\boldsymbol{p}_l + \boldsymbol{R}\boldsymbol{U}\boldsymbol{W}\bar{\boldsymbol{W}}^{\mathsf{T}}\boldsymbol{f}_l \right) \tag{21}$$

where $\boldsymbol{H}$ is the object's inertia matrix and $\boldsymbol{\tau}$ is the torque generated by $\boldsymbol{f}$. Differentiating equation (20) with respect to an applied force produces a similar result for the change in acceleration at the contact point. These equations are linear in $\boldsymbol{f}$, and can be used similarly to solve for position, joint, and collision constraints. Position constraints require that a point's velocity and acceleration are zero. Joint constraints require relative velocities and accelerations are zero, merely requiring a subtraction of the proper terms. Collision constraints require the normal components of relative velocities and accelerations are nonnegative, and only solve for the nonnegative normal force magnitude. All constraints are solved simultaneously as a linear program. Solving cannot always be done in real-time if there are a large number of contact points, although system response does remain interactive.

[2]In order to adhere to common convention we are reusing ω and α, that were previously used for the modal frequencies and Rayleigh damping coefficients. The intended meaning should be clear from context and the presence/absence of bold notation.

Example	Fig	Verts.	Nodes	Tets.	Modes	Time
Brain	1	18,847	304	997	40	68.5 sec
Dodo	5	336	113	295	40	6.2 sec
Bunny	8	2,633	37,114	15,507	32	24 min
Sphere	video	66	80	282	40	2.9 sec
Sheet	video	195	195	486	20	14.4 sec
Bat	video	241	310	1,030	20	68.9 sec

Table 1: This table list the number of vertices in the rendered models, the number of nodes and elements in the finite element models, the number of modes retained, and the time required to compute the decomposition for some of the demonstration objects.

We model friction at the contacts using a simplified Coulomb friction model. The system computes a force opposite the tangential velocity at the contact points. The magnitude of the force equals the magnitude of the normal force multiplied by a friction coefficient. If the friction force causes the predicted tangential velocity to be reversed then it is limited to the force that would cause no slipping. If interactivity can be sacrificed, a more precise method would be to add an additional no-slip constraint to be re-solved with the other constraints. We find our heuristic reasonable for producing plausible friction effects.

4 Results

We have implemented a system that models deformable objects using a hybrid formulation that combines rigid-body motion with deformation computed using modal analysis. Objects may be interactively manipulated by the user with both penalty forces and displacement constraints. The modal objects may collide with each other and with their environment. Collisions can be treated with either penalty forces or constraints, and objects may also be attached together using joint constraints. Table 1 lists several of the models we have used to demonstrate our results and shows the geometric and kinematic complexity of the models along with how much precomputation time was required to perform the modal decomposition for each model.

The brain model in figure 1 demonstrates pulling and pushing using force application. Force vectors are projected into the modal basis, modifying the modal state, and then are projected out, resulting in realistic deformation. The images in figure 6 and figure 7 show pulling and pushing using manipulation constraints. Typically, up to

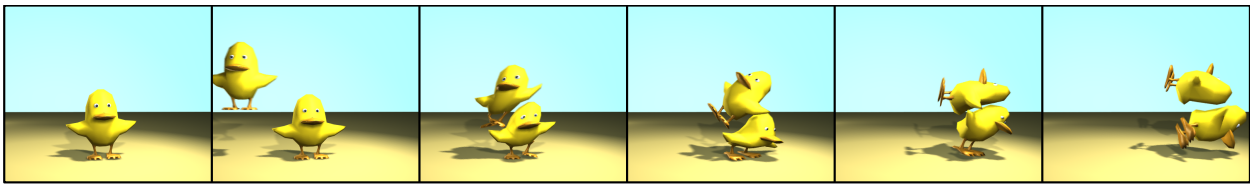

Figure 5: This image sequence shows frames from an animation of a pair of objects colliding with each other. Each object is a hybrid simulation that incorporates a rigid and a deformable (modal) component.

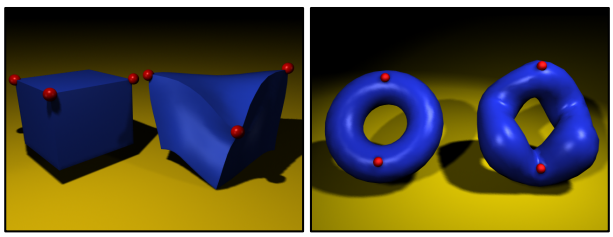

Figure 6: These images shows how constraints can be used to deform objects. The object on the left of each image shows the object prior to deformation, and the right object shows the results after the red constraint points have been moved.

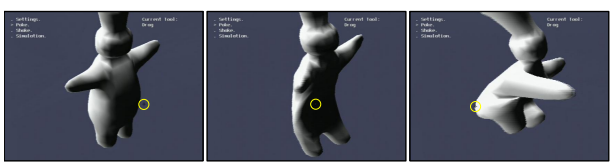

Figure 7: These images are screen shots from an application running natively on a Sony PlayStation2. The yellow circle highlights the cursor that the user is using to poke and pull an elastic figure.

around 10 points on the model can be constrained in real-time on a moderate speed computer (300 MHz Pentium II or Sony Playstation2). A limit is reached because the solutions to equation (13) and equation (15) require a relatively expensive computation of singular value decompositions, which cannot be calculated in real-time once the matrices become too large.

We have created several animations (see supplemental materials) demonstrating this system, each simulated interactively for moderately complex objects. The results appear plausible, and resemble animations that might be simulated using more straightforward but more computationally expensive methods. The bottlenecks in hybrid modal/rigid-body simulation are collision detection and solving the linear program for the constraints. To reduce the computation used in solving the linear program, the extent of contact point clustering may be tweaked to sacrifice accuracy for speed. Figures 5 and 8 show objects involved in collisions with a ground plane and each other.

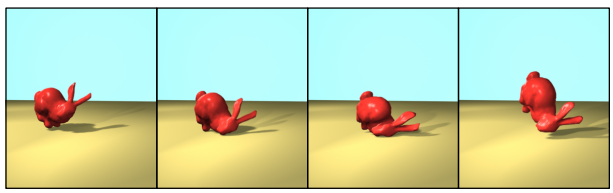

Figure 8: A sequence of images showing the Stanford Bunny model bouncing off a ground plane.

As with other methods based on tetrahedral finite elements, we can embed high-resolution or non-manifold surfaces inside a tetrahedral volume model. The benefits of this technique are that the surface shading and texturing can be specified independently from the dynamics, and poorly constructed "polygon-soup" models may be used. Both the brain model in figure 1, an extremely complex object, and the "dodo" model in figure 5, a non-manifold object, are modeled in this way. The "dodo" model also demonstrates non-uniform material properties: the legs and beak are made of a stiffer material than the rest of the body.

5 Conclusions

Modal analysis has been shown to be a useful tool for interactively producing realistic simulations of elastic deformation. Both the analytic calculation of modal amplitudes using complex oscillators and the removal of high-frequency modes have a stabilizing effect on simulations, allowing for large time steps to be taken.

Despite the approximation of linearity in modal analysis, the simulation results are quite plausible for most objects. The exceptions are long, thin, or highly deformable objects, where nonlinear behavior dominates the expected behavior. Despite these specific drawbacks, many objects can be manipulated quite efficiently and realistically using modal models.

The already small costs of modal analysis can be reduced further by leveraging graphics hardware, as shown by James and Pai [10] or our own implementation on the Sony PlayStation2. Using such hardware, CPU costs can be reduced to modifying mode amplitudes during evolution of time steps, projection of forces, and application of manipulation constraints.

We recognize that there are many implementation details that cannot fit into this paper, so we have released the source code for our Linux implementation under the GNU License. It is our hope that making this code available will encourage others to work with modal simulation methods. The code may be accessed at www.cs.berkeley.edu/∼job/Projects/ModeDef.

Acknowledgments

The authors thank Christine Gatchalian for her help with the models used in the example, the other members of the Berkeley Graphics Group for their support, and the reviewers for their insightful comments.

This work was supported with contributions from Sony Computer Entertainment America, Intel Corporation, and Pixar Animation Studios, and with NFS grant CCR-0204377 and State of California MICRO grant 02–055.

References

[1] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of SIGGRAPH 94*, pages 23–34, July 1994.

[2] David Baraff and Andrew P. Witkin. Dynamics simulation of non-penetrating flexible bodies. In *Proceedings of SIGGRAPH 92*, pages 303–308, July 1992.

[3] David Baraff and Andrew P. Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98*, pages 43–54, July 1998.

[4] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *Proceedings of SIGGRAPH 2002*, July 2002.

[5] Robert D. Cook, David S. Malkus, and Michael E. Plesha. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, New York, third edition, 1989.

[6] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space and time adaptive sampling. In *Proceedings of SIGGRAPH 2002*, pages 31–36, July 2002.

[7] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, July 1997.

[8] Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms: A simple framework for adaptive simulation. In *Proceedings of SIGGRAPH 2002*, pages 281–290, July 2002.

[9] Doug L. James and Dinesh K. Pai. Artdefo - accurate real time deformable objects. In *Proceedings of ACM SIGGRAPH 99*, pages 65–72, August 1999.

[10] Doug L. James and Dinesh K. Pai. Dyrt: Dynamic response textures for real time deformation simulation with graphics hardware. In *Proceedings of SIGGRAPH 2002*, pages 582–585, July 2002.

[11] Zachi Karni and Craig Gotsman. 3D mesh compression using fixed spectral bases. In *Graphics Interface 2001*, pages 1–8, June 2001.

[12] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: Real time large deformation character skinning in hardware. In *Proceedings of the ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 153–160, July 2002.

[13] Nuno M. M. Maia and Julio M. M. Silva, editors. *Theoretical and Experimental Modal Analysis*. Research Studies Press, Hertfordshire, England, 1998.

[14] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. In *Proceedings of ACM SIGGRAPH 92*, pages 309–312, July 1992.

[15] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 49–54, July 2002.

[16] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH 99*, pages 137–146, August 1999.

[17] James F. O'Brien, Chen Shen, and Christine M. Gatchalian. Synthesizing sounds from rigid-body simulations. In *Proceedings of the ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 175–181, July 2002.

[18] Alex Pentland, Irfa Essa, Martin Friedmann, Bradley Horowitz, and Stan Sclaroff. The thingworld modeling system: Virtual sculpting by modal forces. In *1990 Symposium on Interactive 3D Graphics*, pages 143–144, March 1990.

[19] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of SIGGRAPH 89*, pages 215–222, July 1989.

[20] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface 95*, pages 147–154, May 1995.

[21] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In *Proceedings of ACM SIGGRAPH 86*, pages 151–160, August 1986.

[22] Chen Shen, Kris K. Hauser, Christine M. Gatchalian, and James F. O'Brien. Modal analysis for real-time viscoelastic deformation. In *Proceedings of SIGGRAPH 2002*.

[23] Jos Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum*, 16(3):159–164, August 1997.

[24] Demetri Terzopoulos and Kurt Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, 1988.

[25] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[26] Kees van den Doel, Paul G. Kry, and Dinesh K. Pai. Foley automatic: Physically-based sound effects for interactive simulation and animation. In *Proceedings of SIGGRAPH 2001*, pages 537–544, August 2001.

[27] Kees van den Doel and Dinesh K. Pai. Synthesis of shape dependent sounds with physical modeling. In *Proceedings of the International Conference on Auditory Display (ICAD)*, 1996.

[28] Kees van den Doel and Dinesh K. Pai. The sounds of physical shapes. *Presence*, 7(4):382–395, 1998.

[29] Kesheng Wu and Horst Simon. TRLAN user guide. Technical Report LBNL-42953, Lawrence Berkeley National Laboratory, 1999.

Graphical Modeling and Animation of Brittle Fracture

James F. O'Brien Jessica K. Hodgins

GVU Center and College of Computing
Georgia Institute of Technology

Abstract

In this paper, we augment existing techniques for simulating flexible objects to include models for crack initiation and propagation in three-dimensional volumes. By analyzing the stress tensors computed over a finite element model, the simulation determines where cracks should initiate and in what directions they should propagate. We demonstrate our results with animations of breaking bowls, cracking walls, and objects that fracture when they collide. By varying the shape of the objects, the material properties, and the initial conditions of the simulations, we can create strikingly different effects ranging from a wall that shatters when it is hit by a wrecking ball to a bowl that breaks in two when it is dropped on edge.

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: Animation techniques, physically based modeling, simulation, dynamics, fracture, cracking, deformation, finite element method.

1 Introduction

With the introduction in 1998 of simulated water in *Antz* [5, 14] and clothing in *Geri's Game* [4, 15], passive simulation was clearly demonstrated to be a viable technique for commercial animation. The appeal of using simulation for objects without an internal source of energy is not surprising, as passive objects tend to have many degrees of freedom, making keyframing or motion capture difficult. Furthermore, while passive objects are often essential to the plot of an animation and to the appearance or mood of the piece, they are not characters with their concomitant requirements for control over the subtle details of the motion. Therefore, simulations in which the motion is controlled only by initial conditions, physical equations, and material parameters are often sufficient to produce appealing animations of passive objects.

College of Computing, Georgia Institute of Technology, Atlanta, GA 30332. job@acm.org, jkh@cc.gatech.edu.

SIGGRAPH 99, Los Angeles, CA USA

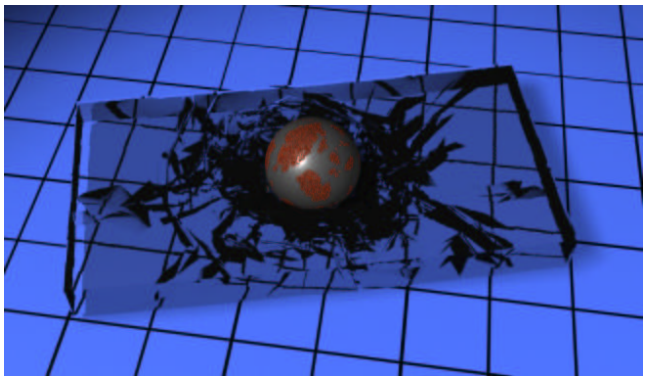

Figure 1: Slab of simulated glass that has been shattered by a heavy weight.

Our approach to animating breaking objects is based on linear elastic fracture mechanics. We model three-dimensional volumes using a finite element method that is based on techniques presented in the computer graphics and mechanical engineering literature [3, 6, 18]. By analyzing the stresses created as a volumetric object deforms, the simulation determines where cracks should begin and in what directions they should propagate. The system accommodates arbitrary propagation directions by dynamically retesselating the mesh. Because cracks are not limited to element boundaries, the models can form irregularly shaped shards and edges as they shatter.

We demonstrate the power of this approach with the following examples: a glass slab that shatters when a weight is dropped onto it (Figure 1), an adobe wall that crumbles under the impact of a wrecking ball (Figure 9), a series of bowls that break when they hit the floor (Figure 11), and objects that break when they collide with each other (Figure 14). To assess the realism of this approach, we compare high-speed video images of a physical bowl dropping onto concrete and a simulated version of the same event (Figure 13).

2 Background

In the computer graphics literature, two previous techniques have been developed for modeling dynamic, deformation-induced fracture. In 1988, Terzopoulos and Fleischer [18, 19] presented a general technique for modeling viscoelastic and plastic deformations. Their method used three fundamental metric tensors to define energy functions that measured deformation over curves, surfaces, and volumes. These energy functions provided the basis for a continuous deformation model that they simulated using a variety of discretization methods. One of their methods made use of a finite differencing technique defined by controlled continuity splines [17]. This formulation allowed them to demonstrate how certain fracture effects could be modeled by setting the elastic coefficients between adjacent nodes to zero whenever the distance between the nodes exceeded a threshold. They demonstrated this technique with sheets paper and cloth that could be torn apart.

In 1991, Norton and his colleagues presented a technique for animating 3D solid objects that broke when subjected to large strains [12]. They simulated a teapot that shattered when dropped onto a table. Their technique used a spring and mass system to model the behavior of the object. When the distance between two attached mass points exceeded a threshold, the simulation severed the spring connection between them. To avoid having flexible strings of partially connected material hanging from the object, their simulation broke an entire cube of springs at once.

Two limitations are inherent in both of these methods. First, when the material fails, the exact location and orientation of the fracture are not known. Rather the failure is defined as the entire connection between two nodes, and the orientation of the fracture plane is left undefined. As a result, these techniques can only realistically model effects that occur on a scale much larger than the inter-node spacing.

The second limitation is that fracture surfaces are restricted to the boundaries in the initial mesh structure. As a result, the fracture pattern exhibits directional artifacts, similar to the "jaggies" that occur when rasterizing a polygonal edge. These artifacts are particularly noticeable when the discretization follows a regular pattern. If an irregular mesh is used, then the artifacts may be partially masked, but the fractures will still be forced onto a path that follows the element boundaries so that the object can break apart only along predefined facets.

Other relevant work in the computer graphics literature includes techniques for modeling static crack patterns and fractures induced by explosions. Hirota and colleagues described how phenomena such as the static crack patterns created by drying mud can be modeled using a mass and spring system attached to an immobile substrate [8]. Mazarak et al. use a voxel-based approach to model solid objects that break apart when they encounter a spherical blast wave [9]. Neff and Fiume use a recursive pattern generator to divide a planar region into polygonal shards that fly apart when acted on by a spherical blast wave [10].

Fracture has been studied more extensively in the mechanics literature, and many techniques have been developed for simulating and analyzing the behavior of materials as they fail. A number of theories may be used to describe when and how a fracture will develop or propagate, and these theories have been employed with various numerical methods including finite element and finite difference methods, boundary integral equations, and molecular particle simulations. A comprehensive review of this work can be found in the book by Anderson [1] and the survey article by Nishioka [11].

Although simulation is used to model fracture both in computer graphics and in engineering, the requirements of the two fields are very different. Engineering applications require that the simulation predict real-world behaviors in an accurate and reliable fashion. In computer animation, what matters is how the fracture looks, how difficult it was to make it look that way, and how long it took. Although the technique presented in this paper was developed using traditional engineering tools, it is an animation technique and relies on a number of simplifications that would be unacceptable in an engineering context.

3 Deformations

Fractures arise in materials due to internal stresses created as the material deforms. Our goal is to model these fractures. In order to do so, however, we must first be able to model the deformations that cause them. To provide a suitable framework for modeling fractures, the deformation method must provide information about the magnitude and orientation of the internal stresses, and whether they are tensile or compressive. We would also like to avoid deformation methods in which directional artifacts appear in the stress patterns and propagate to the resulting fracture patterns.

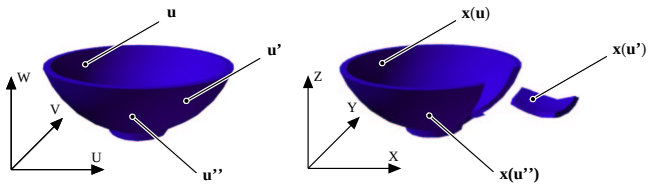

Figure 2: The material coordinates define a 3D parameterization of the object. The function $\boldsymbol{x}(\boldsymbol{u})$ maps points from their location in the material coordinate frame to their location in the world coordinates. A fracture corresponds to a discontinuity in $\boldsymbol{x}(\boldsymbol{u})$.

We derive our deformation technique by defining a set of differential equations that describe the aggregate behavior of the material in a continuous fashion, and 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 derived in this fashion. The one presented here was designed to be simple, fast, and suitable for fracture modeling.

3.1 Continuous Model

Our continuous model is based on continuum mechanics, and an excellent introduction to this area can be found in the text by Fung [7]. 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 that compose the material can be modeled as a continuous media. Although this assumption is often valid for modeling deformations, macroscopic fractures can be significantly influenced by effects that occur at small scales where this assumption may not be valid. Because we are interested in graphical appearance rather than rigorous physical correctness, we will put this issue aside and assume that a continuum model is adequate.

We begin the description of the continuous model by defining material coordinates that parameterize the volume of space occupied by the object being modeled. Let $\boldsymbol{u} = [u, v, w]^{\mathsf{T}}$ be a vector in $\Re^3$ that denotes a location in the material coordinate frame as shown in Figure 2. The deformation of the material is defined by the function $\boldsymbol{x}(\boldsymbol{u}) = [x, y, z]^{\mathsf{T}}$ that maps locations in the material coordinate frame to locations in world coordinates. In areas where material exists, $\boldsymbol{x}(\boldsymbol{u})$ is continuous, except across a finite number of surfaces within the volume that correspond to fractures in the material. In areas where there is no material, $\boldsymbol{x}(\boldsymbol{u})$ is undefined.

We make use of Green's strain tensor, $\boldsymbol{\epsilon}$, to measure the local deformation of the material [6]. It can be represented as a 3×3 symmetric matrix defined by

$$\epsilon_{ij} = \left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} \right) - \delta_{ij} \tag{1}$$

where δ_{ij} is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1 & : \quad i = j \\ 0 & : \quad i \neq j \end{cases}. \tag{2}$$

This strain metric only measures deformation; it is invariant with respect to rigid body transformations applied to $\boldsymbol{x}$ and vanishes when the material is not deformed. It has been used extensively in the engineering literature. Because it is a tensor, its invariants do not depend on the orientation of the material coordinate or world systems. The Euclidean metric tensor used by Terzopoulos and Fleischer [18] differs only by the δ_{ij} term.

In addition to the strain tensor, we make use of the strain rate tensor, $\boldsymbol{\nu}$, which measures the rate at which the strain is changing.

It can be derived by taking the time derivative of (1) and is defined by

$$\nu_{ij} = \left(\frac{\partial \boldsymbol{x}}{\partial u_i} \cdot \frac{\partial \dot{\boldsymbol{x}}}{\partial u_j} \right) + \left(\frac{\partial \dot{\boldsymbol{x}}}{\partial u_i} \cdot \frac{\partial \boldsymbol{x}}{\partial u_j} \right) \tag{3}$$

where an over dot indicates a derivative with respect to time such that $\dot{\boldsymbol{x}}$ is the material velocity expressed in world coordinates.

The strain and strain rate tensors provide the raw information that is required to compute internal elastic and damping forces, but they do not take into account the properties of the material. The stress tensor, $\boldsymbol{\sigma}$, combines the basic information from the strain and strain rate with the material properties and determines forces internal to the material. Like the strain and strain rate tensors, the stress tensor can be represented as a 3×3 symmetric matrix. It has two components: the elastic stress due to strain, $\boldsymbol{\sigma}^{(\epsilon)}$, and the viscous stress due to strain rate, $\boldsymbol{\sigma}^{(\nu)}$. The total internal stress, is the sum of these two components with

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{(\epsilon)} + \boldsymbol{\sigma}^{(\nu)} \ . \tag{4}$$

The elastic stress and viscous stress are respectively functions of the strain and strain rate. In their most general linear forms, they are defined as

$$\sigma_{ij}^{(\epsilon)} = \sum_{k=1}^{3} \sum_{l=1}^{3} C_{ijkl}\, \epsilon_{kl} \tag{5}$$

$$\sigma_{ij}^{(\nu)} = \sum_{k=1}^{3} \sum_{l=1}^{3} D_{ijkl}\, \nu_{kl} \tag{6}$$

where $\boldsymbol{C}$ is a set of the 81 elastic coefficients that relate the elements of $\boldsymbol{\epsilon}$ to the elements $\boldsymbol{\sigma}^{(\epsilon)}$, and $\boldsymbol{D}$ is a set of the 81 damping coefficients.[1]

Because both $\boldsymbol{\epsilon}$ and $\boldsymbol{\sigma}^{(\epsilon)}$ are symmetric, many of the coefficients in $\boldsymbol{C}$ are either redundant or constrained, and $\boldsymbol{C}$ can be reduced to 36 independent values that relate the six independent values of $\boldsymbol{\epsilon}$ to the six independent values of $\boldsymbol{\sigma}^{(\epsilon)}$. If we impose the additional constraint that the material is isotropic, then $\boldsymbol{C}$ collapses further to only two independent values, μ and λ, which are the Lamé constants of the material. Equation (5) then reduces to

$$\sigma_{ij}^{(\epsilon)} = \sum_{k=1}^{3} \lambda \epsilon_{kk} \delta_{ij} + 2\mu \epsilon_{ij} \ . \tag{7}$$

The material's rigidity is determined by the value of μ, and the resistance to changes in volume (dilation) is controlled by λ.

Similarly, $\boldsymbol{D}$ can be reduced to two independent values, ϕ and ψ and (6) then reduces to

$$\sigma_{ij}^{(\nu)} = \sum_{k=1}^{3} \phi \nu_{kk} \delta_{ij} + 2\psi \nu_{ij} \ . \tag{8}$$

The parameters μ and λ will control how quickly the material dissipates internal kinetic energy. Since $\boldsymbol{\sigma}^{(\nu)}$ is derived from the rate at which ϵ is changing, $\boldsymbol{\sigma}^{(\nu)}$ will not damp motions that are locally rigid, and has the desirable property of dissipating only internal vibrations.

Once we have the strain, strain rate, and stress tensors, we can compute the elastic potential density, η, and the damping potential density, κ, at any point in the material using

$$\eta = \frac{1}{2} \sum_{i=1}^{3} \sum_{j=1}^{3} \sigma_{ij}^{(\epsilon)} \epsilon_{ij} \ , \tag{9}$$

[1] Actually $\boldsymbol{C}$ and $\boldsymbol{D}$ are themselves rank four tensors, and (5) and (6) are normally expressed in this form so that $\boldsymbol{C}$ and $\boldsymbol{D}$ will follow the standard rules for coordinate transforms.

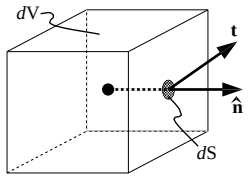

Figure 3: Given a point in the material, the traction, $\boldsymbol{t}$, that acts on the surface element, $d\mathrm{S}$, of a differential volume, $d\mathrm{V}$, centered around the point with outward unit normal, $\hat{\boldsymbol{n}}$, is given by $\boldsymbol{t} = \boldsymbol{\sigma}\, \hat{\boldsymbol{n}}$.

$$\kappa = \frac{1}{2} \sum_{i=1}^{3} \sum_{j=1}^{3} \sigma_{ij}^{(\nu)} \nu_{ij} \ . \tag{10}$$

These quantities can be integrated over the volume of the material to obtain the total elastic and damping potentials. The elastic potential is the internal elastic energy of the material. The damping potential is related to the kinetic energy of the material after subtracting any rigid body motion and normalizing for the material's density.

The stress can also be used to compute the forces acting internal to the material at any location. Let $\hat{\boldsymbol{n}}$ be an outward unit normal direction of a differential volume centered about a point in the material. (See Figure 3.) The traction (force per unit area), $\boldsymbol{t}$, acting on a face perpendicular to the normal is then given by

$$\boldsymbol{t} = \boldsymbol{\sigma}\, \hat{\boldsymbol{n}} \ . \tag{11}$$

The examples in this paper were generated using this isotropic formulation. However, these techniques do not make use of the strain or strain rate tensors directly; rather they rely only on the stress. Switching to the anisotropic formulation, or even to a non-linear stress to strain relation, would not require any significant changes.

3.2 Finite Element Discretization

Before we can model a material's behavior using this continuous model, it must be discretized in a way that is suitable for computer simulation. Two commonly used techniques are the finite difference and finite element methods.

A finite difference method divides the domain of the material into a regular lattice and then uses numerical differencing to approximate the spatial derivatives required to compute the strain and strain rate tensors. This approach is well suited for problems with a regular structure but becomes complicated when the structure is irregular.

A finite element method partitions the domain of the material into distinct sub-domains, or elements as shown in Figure 4. Within each element, the material is described locally by a function with some finite number of parameters. The function is decomposed into a set of orthogonal shape, or basis, functions that are each associated with one of the nodes on the boundary of the element. Adjacent elements will have nodes in common, so that the mesh defines a piecewise function over the entire material domain.

Our discretization employs tetrahedral finite elements with linear polynomial shape functions. By using a finite element method, the mesh can be locally aligned with the fracture surfaces, thus avoiding the previously mentioned artifacts. Just as triangles can be used to approximate any surface, tetrahedra can be used to approximate arbitrary volumes. Additionally, when tetrahedra are split along a fracture plane, the resulting pieces can be decomposed exactly into more tetrahedra.

We chose to use linear elements because higher-order elements are not cost effective for modeling fracture boundaries. Although higher-order polynomials provide individual elements with many

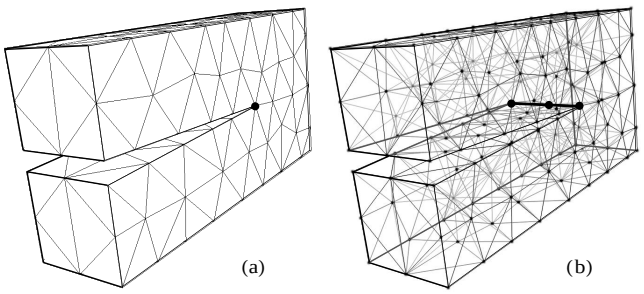

Figure 4: Tetrahedral mesh for a simple object. In **(a)**, only the external faces of the tetrahedra are drawn; in **(b)** the internal structure is shown.

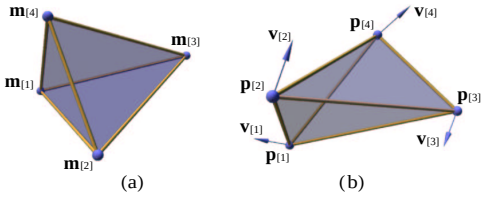

Figure 5: A tetrahedral element is defined by its four nodes. Each node has **(a)** a location in the material coordinate system and **(b)** a position and velocity in the world coordinate system.

degrees of freedom for deformation, they have few degrees of freedom for modeling fracture because the shape of a fracture is defined as a boundary in material coordinates. In contrast, with linear tetrahedra, each degree of freedom in the world space corresponds to a degree of freedom in the material coordinates. Furthermore, whenever an element is created, its basis functions must be computed. For high-degree polynomials, this computation is relatively expensive. For systems where the mesh is constant, the cost is amortized over the course of the simulation. However, as fractures develop and parts of the object are remeshed, the computation of basis matrices can become significant.

Each tetrahedral element is defined by four nodes. A node has a position in the material coordinates, $\boldsymbol{m}$, a position in the world coordinates, $\boldsymbol{p}$, and a velocity in world coordinates, $\boldsymbol{v}$. We will refer to the nodes of a given element by indexing with square brackets. For example, $\boldsymbol{m}_{[2]}$ is the position in material coordinates of the element's second node. (See Figure 5.)

Barycentric coordinates provide a natural way to define the linear shape functions within an element. Let $\boldsymbol{b} = [b_1, b_2, b_3, b_4]^{\mathsf{T}}$ be barycentric coordinates defined in terms of the element's material coordinates so that

$$\begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{m}_{[1]} & \boldsymbol{m}_{[2]} & \boldsymbol{m}_{[3]} & \boldsymbol{m}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{b} \ . \tag{12}$$

These barycentric coordinates may also be used to interpolate the node's world position and velocity with

$$\begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_{[1]} & \boldsymbol{p}_{[2]} & \boldsymbol{p}_{[3]} & \boldsymbol{p}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{b} \tag{13}$$

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{[1]} & \boldsymbol{v}_{[2]} & \boldsymbol{v}_{[3]} & \boldsymbol{v}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix} \boldsymbol{b} \ . \tag{14}$$

To determine the barycentric coordinates of a point within the element specified by its material coordinates, we invert (12) and obtain

$$\boldsymbol{b} = \boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \tag{15}$$

where $\boldsymbol{\beta}$ is defined by

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{m}_{[1]} & \boldsymbol{m}_{[2]} & \boldsymbol{m}_{[3]} & \boldsymbol{m}_{[4]} \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \ . \tag{16}$$

Combining (15) with (13) and (14) yields functions that interpolate the world position and velocity within the element in terms of the material coordinates:

$$\boldsymbol{x}(\boldsymbol{u}) = \boldsymbol{P}\,\boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \tag{17}$$

$$\dot{\boldsymbol{x}}(\boldsymbol{u}) = \boldsymbol{V}\,\boldsymbol{\beta} \begin{bmatrix} \boldsymbol{u} \\ 1 \end{bmatrix} \tag{18}$$

where $\boldsymbol{P}$ and $\boldsymbol{V}$ are defined as

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_{[1]}\, \boldsymbol{p}_{[2]}\, \boldsymbol{p}_{[3]}\, \boldsymbol{p}_{[4]} \end{bmatrix} \tag{19}$$

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_{[1]}\, \boldsymbol{v}_{[2]}\, \boldsymbol{v}_{[3]}\, \boldsymbol{v}_{[4]} \end{bmatrix} \ . \tag{20}$$

Note that the rows of $\boldsymbol{\beta}$ are the coefficients of the shape functions, and $\boldsymbol{\beta}$ needs to be computed only when an element is created or the material coordinates of its nodes change. For non-degenerate elements, the matrix in (16) is guaranteed to be non-singular, however elements that are nearly co-planar will cause $\boldsymbol{\beta}$ to be ill-conditioned and adversely affect the numerical stability of the system.

Computing the values of $\boldsymbol{\epsilon}$ and $\boldsymbol{\nu}$ within the element requires the first partials of $\boldsymbol{x}$ with respect to $\boldsymbol{u}$:

$$\frac{\partial \boldsymbol{x}}{\partial u_i} = \boldsymbol{P}\,\boldsymbol{\beta}\,\boldsymbol{\delta}_i \tag{21}$$

$$\frac{\partial \dot{\boldsymbol{x}}}{\partial u_i} = \boldsymbol{V}\,\boldsymbol{\beta}\,\boldsymbol{\delta}_i \tag{22}$$

where

$$\boldsymbol{\delta}_i = [\delta_{i1}\ \delta_{i2}\ \delta_{i3}\ 0]^{\mathsf{T}} \ . \tag{23}$$

Because the element's shape functions are linear, these partials are constant within the element.

The element will exert elastic and damping forces on its nodes. The elastic force on the ith node, $\boldsymbol{f}^{(\epsilon)}_{[i]}$, is defined as the negative partial of the elastic potential density, η, with respect to $\boldsymbol{p}_{[i]}$ integrated over the volume of the element. Given $\boldsymbol{\sigma}^{(\epsilon)}$, $\boldsymbol{\beta}$, and the positions in world space of the four nodes we can compute the elastic force by

$$\boldsymbol{f}^{(\epsilon)}_{[i]} = -\frac{\mathrm{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl}\beta_{ik}\sigma^{(\epsilon)}_{kl} \tag{24}$$

where

$$\mathrm{vol} = \frac{1}{6}[(\boldsymbol{m}_{[2]} - \boldsymbol{m}_{[1]}) \times (\boldsymbol{m}_{[3]} - \boldsymbol{m}_{[1]})] \cdot (\boldsymbol{m}_{[4]} - \boldsymbol{m}_{[1]}) \ . \tag{25}$$

Similarly, the damping force on the ith node, $\boldsymbol{f}^{(\nu)}_{[i]}$, is defined as the partial of the damping potential density, κ, with respect to $\boldsymbol{v}_{[i]}$ integrated over the volume of the element. This quantity can be computed with

$$\boldsymbol{f}^{(\nu)}_{[i]} = -\frac{\mathrm{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl}\beta_{ik}\sigma^{(\nu)}_{kl} \ . \tag{26}$$

Summing these two forces, the total internal force that an element exerts on a node is

$$\boldsymbol{f}^{\mathrm{el}}_{[i]} = -\frac{\mathrm{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl}\beta_{ik}\sigma_{kl} \ , \tag{27}$$

and the total internal force acting on the node is obtained by summing the forces exerted by all elements that are attached to the node.

As the element is compressed to less than about 30% of its material volume, the gradient of η and κ start to vanish causing the resisting forces to fall off. We have not found this to be a problem as even the more squishy of the materials that we have modeled conserve their volume to within a few percent.

Using a lumped mass formulation, the mass contributed by an element to each one of its nodes is determined by integrating the material density, ρ, over the element shape function associated with that node. In the case of tetrahedral elements with linear shape functions, this mass contribution is simply $\rho\,\mathrm{vol}/4$.

The derivations above are sufficient for a simulation that uses an explicit integration scheme. Additional work, including computing the Jacobian of the internal forces, is necessary for implicit integration scheme. (See for example [2] and [3].)

3.3 Collisions

In addition to the forces internal to the material, the system computes collision forces. The collision forces are computed using a penalty method that is applied when two elements intersect or if an element violates another constraint such as the ground. Although penalty methods are often criticized for creating stiff equations, we have found that for the materials we are modeling the internal forces are at least as stiff as the penalty forces. Penalty forces have the advantage of being very fast to compute. We have experimented with two different penalty criteria: node penetration and overlap volume. The examples presented in this paper were computed with the node penetration criteria; additional examples on the conference proceedings CD-ROM were computed with the overlap volume criteria.

The node penetration criteria sets the penalty force to be proportional to the distance that a node has penetrated into another element. The penalty force acts in the direction normal to the penetrated surface. The reaction force is distributed over the penetrated element's nodes so that the net force and moment are the negation of the penalty force and moment acting at the penetrating node. This test will not catch all collisions, and undetected intersecting tetrahedra may become locked together. It is however, fast to compute, easy to implement, and adequate for situations that do not involve complex collision interactions.

The overlap volume criteria is more robust than the node penetration method, but it is also slower to compute and more complex to implement. The intersection of two tetrahedral elements is computed by clipping the faces of each tetrahedron against the other. The resulting polyhedron is then used to compute the volume and center of mass of the intersecting region. The area weighted normals of the faces of the polyhedron that are contributed by one of the tetrahedra are summed to compute the direction that the penalty force acts in. A similar computation can be performed for the other tetrahedra, or equivalently the direction can be negated. Provided that neither tetrahedra is completely contained within the other, this criteria is more robust than the node penetration criteria. Additionally, the forces computed with this method do not depend on the object tessellation.

Computing the intersections within the mesh can be very expensive, and we use a bounding hierarchy scheme with cached traversals to help reduce this cost.

4 Fracture Modeling

Our fracture model is based on the theory of linear elastic fracture mechanics [1]. The primary distinction between this and other theories of fracture is that the region of plasticity near the crack tip[2] is neglected. Because we are not modeling the energy dissipated by this plastic region, modeled materials will be brittle. This statement does not mean that they are weak; rather the term "brittle" refers to the fact that once the material has begun to fail, the fractures will have a strong tendency to propagate across the material as they are driven by the internally stored elastic energy.

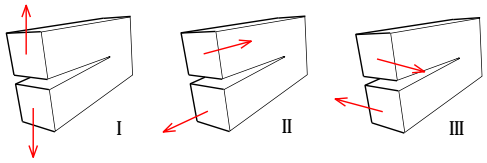

Figure 6: Three loading modes that can be experienced by a crack. Mode I: Opening, Mode II: In-Plane Shear, and Mode III: Out-of-Plane Shear. Adapted from Anderson [1].

There are three loading modes by which forces can be applied to a crack causing it to open further. (See Figure 6.) In most circumstances, some combination of these modes will be active, producing a mixed mode load at the crack tip. For all three cases, as well as mixed mode situations, the behavior of the crack can be resolved by analyzing the forces acting at the crack tip: tensile forces that are opposed by other tensile forces will cause the crack to continue in a direction that is perpendicular to the direction of largest tensile load, and conversely, compressive loads will tend to arrest a crack to which they are perpendicular.

The finite element model describes the surface of a fracture with elements that are adjacent in material coordinates but that do not share nodes across the internal surface. The curve that represents the crack tip is then implicitly defined in a piecewise linear fashion by the nodes that border the fracture surface, and further extension of the crack may be determined by analyzing the internal forces acting on these nodes.

We will also use the element nodes to determine where a crack should be initiated. While this strategy could potentially introduce unpleasant artifacts, we note that because the surface of an object is defined by a polygonal boundary (the outer faces of the tetrahedra) there will always be a node located at any concavities. Because concavities are precisely the locations where cracks commonly begin, we believe that this decision is acceptable.

Our fracture algorithm is as follows: after every time step, the system resolves the internal forces acting on all nodes into their tensile and compressive components, discarding any unbalanced portions. At each node, the resulting forces are then used to form a tensor that describes how the internal forces are acting to separate that node. If the action is sufficiently large, the node is split into two distinct nodes and a fracture plane is computed. 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. Any cached values, such as the node mass or the element shape functions, are recomputed for the affected elements and nodes. With this technique, the location of a fracture or crack tip need not be explicitly recorded unless this information is useful for some other purpose, such as rendering.

[2]The term "crack tip" implies that the fracture will have a single point at its tip. In general, the front of the crack will not be a single point; rather it will be a curve that defines the boundary of the surface discontinuity in material coordinates. (See Figure 4.) Nevertheless, we will refer to this front as the crack tip.

4.1 Force Decomposition

The forces acting on a node are decomposed by first separating the element stress tensors into tensile and compressive components. For a given element in the mesh, let $\mathsf{v}^i(\boldsymbol{\sigma})$, with $i \in \{1,2,3\}$, be the ith eigenvalue of $\boldsymbol{\sigma}$, and let $\hat{\mathbf{n}}^i(\boldsymbol{\sigma})$ be the corresponding unit length eigenvector. Positive eigenvalues correspond to tensile stresses and negative ones to compressive stresses. Since $\boldsymbol{\sigma}$ is real and symmetric, it will have three real, not necessarily unique, eigenvalues. In the case where an eigenvalue has multiplicity greater than one, the eigenvectors are selected arbitrarily to orthogonally span the appropriate subspace [13].

Given a vector $\boldsymbol{a}$ in $\Re^3$, we can construct a 3×3 symmetric matrix, $\mathbf{m}(\boldsymbol{a})$ that has $|\boldsymbol{a}|$ as an eigenvalue with $\boldsymbol{a}$ as the corresponding eigenvector, and with the other two eigenvalues equal to zero. This matrix is defined by

$$\mathbf{m}(\boldsymbol{a}) = \begin{cases} \boldsymbol{a}\,\boldsymbol{a}^{\mathsf{T}}/|\boldsymbol{a}| & : \quad \boldsymbol{a} \neq \mathbf{0} \\ \mathbf{0} & : \quad \boldsymbol{a} = \mathbf{0}\ . \end{cases} \tag{28}$$

The tensile component, $\boldsymbol{\sigma}^+$, and compressive component, $\boldsymbol{\sigma}^-$, of the stress within the element can now be computed by

$$\boldsymbol{\sigma}^+ = \sum_{i=1}^{3} \max(0, \mathsf{v}^i(\boldsymbol{\sigma}))\ \mathbf{m}(\hat{\mathbf{n}}^i(\boldsymbol{\sigma})) \tag{29}$$

$$\boldsymbol{\sigma}^- = \sum_{i=1}^{3} \min(0, \mathsf{v}^i(\boldsymbol{\sigma}))\ \mathbf{m}(\hat{\mathbf{n}}^i(\boldsymbol{\sigma}))\ . \tag{30}$$

Using this decomposition, the force that an element exerts on a node can be separated into a tensile component, $\boldsymbol{f}^+_{[i]}$, and a compressive component, $\boldsymbol{f}^-_{[i]}$. This separation is done by reevaluating the internal forces exerted on the nodes using (27) with $\boldsymbol{\sigma}^+$ or $\boldsymbol{\sigma}^-$ substituted for $\boldsymbol{\sigma}$. Thus the tensile component is

$$\boldsymbol{f}^+_{[i]} = -\frac{\mathrm{vol}}{2} \sum_{j=1}^{4} \boldsymbol{p}_{[j]} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl}\beta_{ik}\sigma^+_{kl}\ . \tag{31}$$

The compressive component can be computed similarly, but because $\boldsymbol{\sigma} = \boldsymbol{\sigma}^+ + \boldsymbol{\sigma}^-$, it can be computed more efficiently using $\boldsymbol{f}_{[i]} = \boldsymbol{f}^+_{[i]} + \boldsymbol{f}^-_{[i]}$.

Each node will now have a set of tensile and a set of compressive forces that are exerted by the elements attached to it. For a given node, we denote these sets as $\{\boldsymbol{f}^+\}$ and $\{\boldsymbol{f}^-\}$ respectively. The unbalanced tensile load, $\boldsymbol{f}^+$ is simply the sum over $\{\boldsymbol{f}^+\}$, and the unbalanced compressive load, $\boldsymbol{f}^-$, is the sum over $\{\boldsymbol{f}^-\}$.

4.2 The Separation Tensor

We describe the forces acting at the nodes using a stress variant that we call the separation tensor, $\boldsymbol{\varsigma}$. The separation tensor is formed from the balanced tensile and compressive forces acting at each node and is computed by

$$\boldsymbol{\varsigma} = \frac{1}{2}\left(-\mathbf{m}(\boldsymbol{f}^+) + \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^+\}} \mathbf{m}(\boldsymbol{f}) + \mathbf{m}(\boldsymbol{f}^-) - \sum_{\boldsymbol{f}\in\{\boldsymbol{f}^-\}} \mathbf{m}(\boldsymbol{f}) \right)\ . \tag{32}$$

It does not respond to unbalanced actions that would produce a rigid translation, and is invariant with respect to transformations of both the material and world coordinate systems.

The separation tensor is used directly to determine whether a fracture should occur at a node. Let v^+ be the largest positive eigenvalue of $\boldsymbol{\varsigma}$. If v^+ is greater than the material toughness, τ, then the material will fail at the node. The orientation in world coordinates of the fracture plane is perpendicular to $\hat{\mathbf{n}}^+$, the eigenvector of $\boldsymbol{\varsigma}$ that corresponds to v^+. In the case where multiple eigenvalues are greater than τ, multiple fracture planes may be generated by first generating the plane for the largest value, remeshing (see below), and then recomputing the new value for $\boldsymbol{\varsigma}$ and proceeding as above.

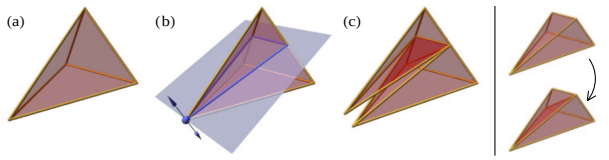

Figure 7: Diagram showing how an element is split by the fracture plane. **(a)** The initial tetrahedral element. **(b)** The splitting node and fracture plane are shown in blue. **(c)** The element is split along the fracture plane into two polyhedra that are then decomposed into tetrahedra. Note that the two nodes created from the splitting node are co-located, the geometric displacement shown in **(c)** only illustrates the location of the fracture discontinuity.

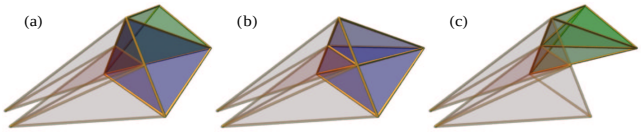

Figure 8: Elements that are adjacent to an element that has been split by a fracture plane must also be split to maintain mesh consistency. **(a)** Neighboring tetrahedra prior to split. **(b)** Face neighbor after split. **(c)** Edge neighbor after split.

4.3 Local Remeshing

Once the simulation has determined the location and orientation of a new fracture plane, the mesh must be modified to reflect the new discontinuity. It is important that the orientation of the fracture be preserved, as approximating it with the existing element boundaries would create undesirable artifacts. To avoid this potential difficulty, the algorithm remeshes the local area surrounding the new fracture by splitting elements that intersect the fracture plane and modifying neighboring elements to ensure that the mesh stays self-consistent.

First, the node where the fracture originates is replicated so that there are now two nodes, n^+ and n^- with the same material position, world position, and velocity. The masses will be recalculated later. The discontinuity passes "between" the two co-located nodes. The positive side of the fracture plane is associated with n^+ and the negative side with n^-.

Next, all elements that were attached to the original node are examined, comparing the world location of their nodes to the fracture plane. If an element is not intersected by the fracture plane, then it is reassigned to either n^+ or n^- depending on which side of the plane it lies.

If the element is intersected by the fracture plane, it is split along the plane. (See Figure 7.) A new node is created along each edge that intersects the plane. Because all elements must be tetrahedra, in general each intersected element will be split into three tetrahedra. One of the tetrahedra will be assigned to one side of the plane and the other two to the other side. Because the two tetrahedra that are on the same side of the plane both share either n^+ or n^-, the discontinuity does not pass between them.

In addition to the elements that were attached to the original node, it may be necessary to split other elements so that the mesh

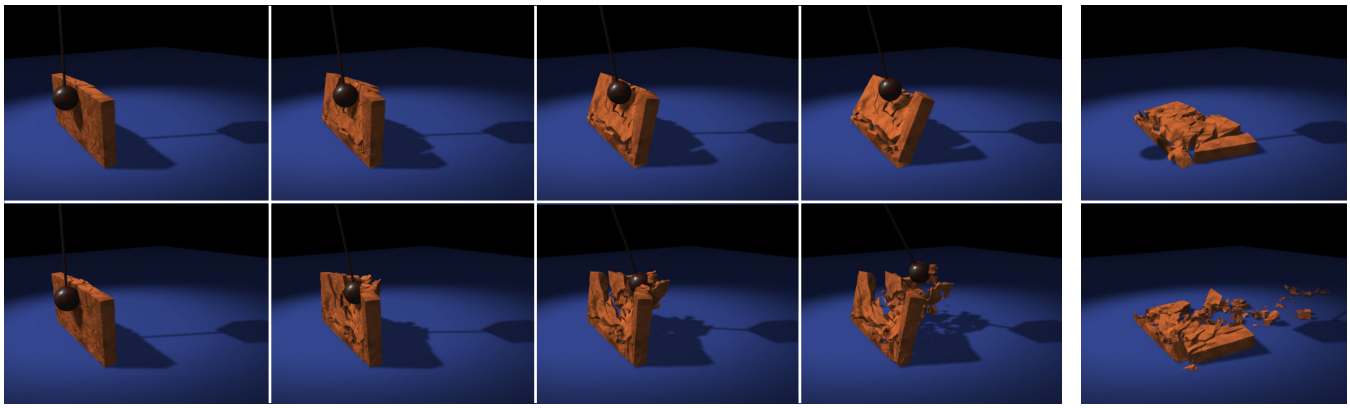

Figure 9: Two adobe walls that are struck by wrecking balls. Both walls are attached to the ground. The ball in the second row has $50\times$ the mass of the first. Images are spaced apart 133.3 ms in the first row and 66.6 ms in the second. The rightmost images show the final configurations.

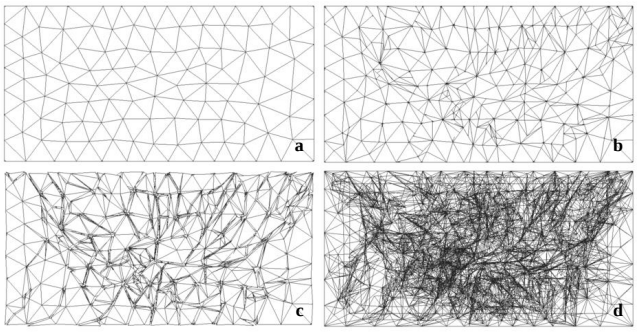

Figure 10: Mesh for adobe wall. **(a)** The facing surface of the initial mesh used to generate the wall shown in Figure 9. **(b)** The mesh after being struck by the wrecking ball, reassembled. **(c)** Same as (b), with the cracks emphasized. **(d)** Internal fractures shown as wireframe.

stays consistent. In particular, an element must be split if the face or edge between it and another element that was attached to the original node has been split. (See Figure 8.) To prevent the remeshing from cascading across the entire mesh, these splits are done so that the new tetrahedra use only the original nodes and the nodes created by the intersection splits. Because no new nodes are created, the effect of the local remeshing is limited to the elements that are attached to the node where the fracture originated and their immediate neighbors. Because the tetrahedra formed by the secondary splits do not attach to either n^+ or n^-, the discontinuity does not pass between them.

Finally, after the local remeshing has been completed, any cached values that have become invalid must be recomputed. In our implementation, these values include the element basis matrix, $\boldsymbol{\beta}$, and the node masses.

Two additional subtleties must also be considered. The first subtlety occurs when an intersection split involves an edge that is formed only by tetrahedra attached to the node where the crack originated. When this happens, the fracture has reached a boundary in the material, and the discontinuity should pass through the edge. Remeshing occurs as above, except that two nodes are created on the edge and one is assigned to each side of the discontinuity.

Second, the fracture plane may pass arbitrarily close to an existing node producing arbitrarily ill-conditioned tetrahedra. To avoid this, we employ two thresholds, one the distance between the fracture plane and an existing node, and the other on the angle between the fracture plane and a line from the node where the split originated to the existing node. If either of these thresholds are not met, then the intersection split is snapped to the existing node. In our results, we have used thresholds of 5 mm and 0.1 radians.

5 Results and Discussion

To demonstrate some of the effects that can be generated with this fracture technique, we have animated a number of scenes that involve objects breaking. Figure 1 shows a plate of glass that has had a heavy weight dropped on it. The area in the immediate vicinity of the impact has been crushed into many small fragments. Further away from the weight, a pattern of radial cracks has developed.

Figure 9 shows two walls being struck by wrecking balls. In the first sequence, the wall develops a network of cracks as it absorbs most of the ball's energy during the initial impact. In the second sequence, the ball's mass is $50\times$ greater, and the wall shatters when it is struck. The mesh used to generate the wall sequences is shown in Figure 10. The initial mesh contains only 338 nodes and 1109 elements. By the end of the sequence, the mesh has grown to 6892 nodes and 8275 elements. These additional nodes and elements are created where fractures occur; a uniform mesh would require many times this number of nodes and elements to achieve a similar result.

Figure 11 shows the final frames from four animations of bowls that were dropped onto a hard surface. Other than the toughness, τ, of the material, the four simulations are identical. The first bowl develops only a few cracks; the weakest breaks into many pieces.

Because this system works with solid tetrahedral volumes rather than with the polygonal boundary representations created by most modeling packages, boundary models must be converted before they can be used. A number of systems are available for creating tetrahedral meshes from polygonal boundaries. The models that we used in these examples were generated either from a CSG description or a polygonal boundary representation using NETGEN, a publicly available mesh generation package [16].

Although our approach avoids the "jaggy" artifacts in the fracture patterns caused by the underlying mesh, there remain ways in which the results of a simulation are influenced by the mesh structure. The most obvious is that the deformation of the material is limited by the degrees of freedom in the mesh, which in turn limits how the material can fracture. This limitation will occur with any discrete system. The technique also limits where a fracture may ini-

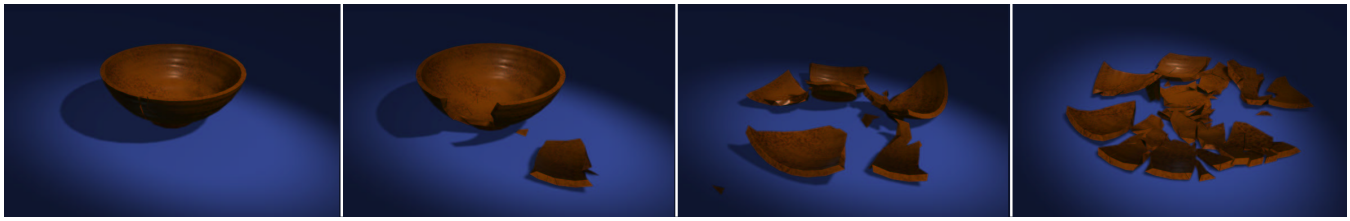

Figure 11: Bowls with successively lower toughness values, τ. Each of the bowls were dropped from the same height. Other than τ, the bowls have same material properties.

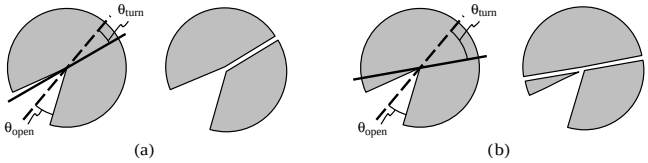

Figure 12: Back-cracking during fracture advance. The dashed line is the axis of the existing crack. Cracks advance by splitting elements along a fracture plane, shown as a solid line, computed from the separation tensor. **(a)** If the crack does not turn sharply, then only elements in front of the tip will be split. **(b)** If the crack turns at too sharp an angle, then the backwards direction may not fall inside of the existing failure and a spurious bifurcation will occur.

tiate by examining only the existing nodes. This assumption means that very coarse mesh sizes might behave in an unintuitive fashion. However, nodes correspond to the locations 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 is related to the speed at which a crack propagates. Currently, the distance that a fracture may travel during a time step is determined by the size of the existing mesh elements. The crack may either split an element or not; it cannot travel only a fraction of the distance across an element. If a crack were being opened slowly by an applied load on a model with a coarse resolution mesh, this limitation would lead to a "button popping" effect where the crack would travel across one element, pause until the stress built up again, and then move across the next element. A second type of artifact may occur if the crack's speed should be significantly greater than the element width divided by the simulation time step. In this case, a high stress area will race ahead of the crack tip, causing spontaneous failures to occur in the material. Although we have not observed these phenomena in our examples, developing an algorithm that allows a fracture to propagate arbitrary distances is an area for future work.

Another limitation stems from the fact that while the fracture plane's orientation is well defined, the crack tip's forward direction is not. As shown in Figure 12, if the cracks turns at an angle greater than half the angle at the crack tip, then a secondary fracture will develop in the opposite direction to the crack's advance. While this effect is likely present in some of our examples, it does not appear to have a significant impact on the quality of the results. If the artifacts were to be a problem, they could be suppressed by explicitly tracking the fracture propagation directions within the mesh.

The simulation parameters used to generate the examples in this paper are listed in Table 1 along with the computation time required to generate one second of animation. While the material density values, ρ, are appropriate for glass, stone, or ceramic, we used values for the Lamé constants, λ and μ, that are significantly less than those of real materials. Larger values would make the simulated materials appear stiffer, but would also require smaller time steps. The values that we have selected represent a compromise between realistic behavior and reasonable computation time.

Our current implementation can switch between either a forward Euler integration scheme or a second order Taylor integrator. Both of these techniques are explicit integration schemes, and subject to stability limits that require very small time steps for stiff materials. Although semi-implicit integration methods have error bounds similar to those of explicit methods, the semi-implicit integrators tend to drive errors towards zero rather than infinity so that they are stable at much larger time steps. Other researchers have shown that by taking advantage of this property, a semi-implicit integrator can be used to realize speed ups of two or three orders of magnitude when modeling object deformation [2]. Unfortunately, it may be difficult to realize these same improvements when fracture propagation is part of the simulation. As discussed above, the crack speed is limited in inverse proportion to the time step size, and the large time steps that might be afforded by a semi-implicit integrator could cause spontaneous material failure to proceed crack advance. We are currently investigating how our methods may be modified to be compatible with large time step integration schemes.

Many materials and objects in the real world are not homogeneous, and it would be interesting to develop graphical models for animating them as they fail. For example, a brick wall is made up of mortar and bricks arranged in a regular fashion, and if simulated in a situation like our wall example, a distinct pattern would be created. Similarly, the connection between a handmade cup and its handle is often weak because of the way in which the handle is attached.

One way to assess the realism of an animation technique is by comparing it with the real world. Figure 13 shows high-speed video footage of a physical bowl as it falls onto its edge compared to our imitation of the real-world scene. Although the two sets of fracture patterns are clearly different, the simulated bowl has some qualitative similarities to 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.

Acknowledgments

The authors would like to thank Wayne L. Wooten of Pixar Animation Studios for lighting, shading, and rendering the images for many of the figures in this paper. We would also like to thank Ari Glezer and Bojan Vukasinovic of the School Mechanical Engineering at the Georgia Institute of Technology for their assistance and the use of the high-speed video equipment. Finally, we would like to thank those in the Animation Lab who lent a hand to ensure that we made the submission deadline.

This project was supported in part by NSF 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.

Example	Figure	Material Parameters $\lambda\,(N/m^2)$	$\mu\,(N/m^2)$	$\phi\,(Ns/m^2)$	$\psi\,(Ns/m^2)$	$\rho\,(kg/m^3)$	$\tau\,(N/m^2)$	Minutes of Computation Time per Simulation Second Minimum	Maximum	Average
Glass	1	1.04×10^8	1.04×10^8	0	6760	2588	10140	75	667	273
Wall #1	9.a	6.03×10^8	1.21×10^8	3015	6030	2309	6030	75	562	399
Wall #2	9.b	0	1.81×10^8	0	18090	2309	6030	75	2317	1098
Bowl #1	11.a	2.65×10^6	3.97×10^6	264	397	1013	52.9	90	120	109
Bowl #2	11.b	2.65×10^6	3.97×10^6	264	397	1013	39.6	82	135	115
Bowl #3	11.c	2.65×10^6	3.97×10^6	264	397	1013	33.1	90	150	127
Bowl #4	11.d	2.65×10^6	3.97×10^6	264	397	1013	13.2	82	187	156
Comp. Bowl	13	0	5.29×10^7	0	198	1013	106	247	390	347
The End	14	0	9.21×10^6	0	9.2	705	73.6	622	6667	4665

Table 1: Material parameters and simulation times for examples. The times listed reflect the total number of minutes required to compute one second of simulated data, including graphics and file I/O. Times were measured on an SGI O2 with a 195 MHz MIPS R10K processor.

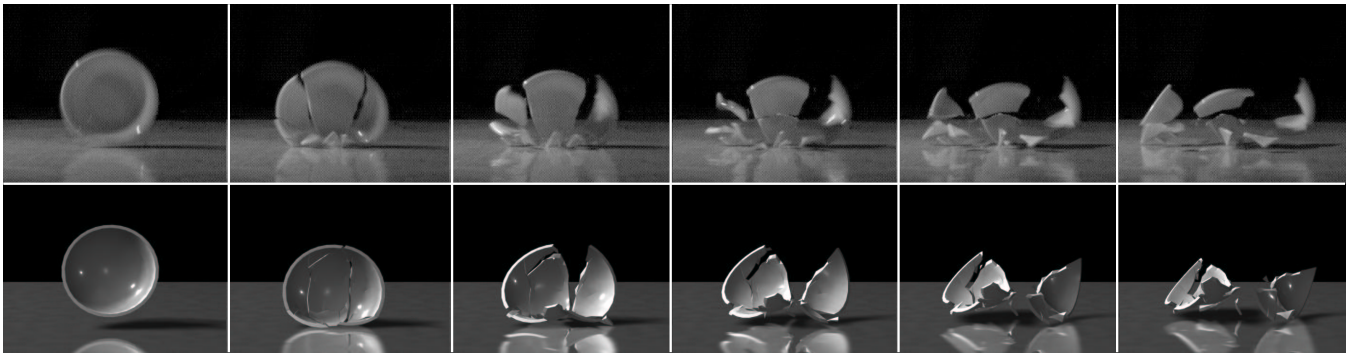

Figure 13: Comparison of a real-world event and simulation. The top row shows high-speed video images of a physical ceramic bowl dropped from approximately one meter onto a hard surface. The bottom row is the output from a simulation where we attempted to match the initial conditions of the physical bowl. Video images are 8 ms apart. Simulation images are 13 ms apart.

References

[1] T. L. Anderson. *Fracture Mechanics: Fundamentals and Applications*. CRC Press, Boca Raton, second edition, 1995.

[2] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998.

[3] R. D. Cook, D. S. Malkus, and M. E. Plesha. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, New York, third edition, 1989.

[4] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 85–94. ACM SIGGRAPH, Addison Wesley, July 1998.

[5] N. Foster and D. Metaxas. Realistic animation of liquids. In *Graphics Interface '96*, pages 204–212, May 1996.

[6] Y. C. Fung. *Foundations of Solid Mechanics*. Prentice-Hall, Englewood Cliffs, N.J., 1965.

[7] Y. C. Fung. *A First Course in Continuum Mechanics*. Prentice-Hall, Englewood Cliffs, N.J., 1969.

[8] K. Hirota, Y. Tanoue, and T. Kaneko. Generation of crack patterns with a physical model. *The Visual Computer*, 14:126–137, 1998.

[9] O. Mazarak, C. Martins, and J. Amanatides. Animating exploding objects. In *Graphics Interface '99*, June 1999.

[10] M. Neff and E. Fiume. A visual model for blast waves and fracture. In *Graphics Interface '99*, June 1999.

[11] T. Nishioka. Computational dynamic fracture mechanics. *International Journal of Fracture*, 86:127–159, 1997.

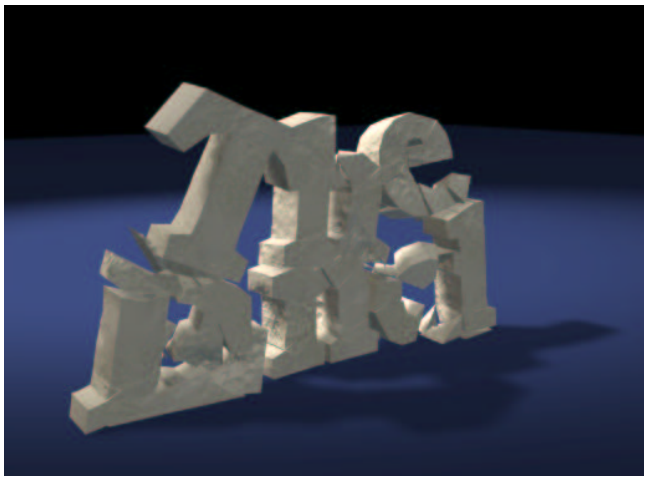

Figure 14: Several breakable objects that were dropped from a height.

[12] A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *The Visual Computer*, 7:210–217, 1991.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1994.

[14] B. Robertson. Antz-piration. *Computer Graphics World*, 21(10), 1998.

[15] B. Robertson. Meet Geri: The new face of animation. *Computer Graphics World*, 21(2), 1998.

[16] J. Schöberl. NETGEN – An advancing front 2D/3D–mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997.

[17] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(4):413–424, July 1986.

[18] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4:306–331, 1988.

[19] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 269–278, August 1988.

SIGGRAPH 2002

Graphical Modeling and Animation of Ductile Fracture

James F. O'Brien
University of California, Berkeley

Adam W. Bargteil
University of California, Berkeley

Jessica K. Hodgins
Carnegie Mellon University

Abstract

In this paper, we describe a method for realistically animating ductile fracture in common solid materials such as plastics and metals. The effects that characterize ductile fracture occur due to interaction between plastic yielding and the fracture process. By modeling this interaction, our ductile fracture method can generate realistic motion for a much wider range of materials than could be realized with a purely brittle model. This method directly extends our prior work on brittle fracture [O'Brien and Hodgins, SIGGRAPH 99]. We show that adapting that method to ductile as well as brittle materials requires only a simple to implement modification that is computationally inexpensive. This paper describes this modification and presents results demonstrating some of the effects that may be realized with it.

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: Animation techniques, physically based modeling, simulation, dynamics, fracture, cracking, deformation, finite element method, ductile fracture, plasticity.

1 Introduction

As techniques for generating photorealistic computer rendered images have improved, the use of physically based animation to generate special effects in film, television, and games has become increasingly common. Physically based animation techniques have proven to be particularly useful for violent or destructive effects that would be impractical or expensive to achieve using other methods. For example, when creating effects for the film *Pearl Harbor*, Industrial Light and Magic made extensive use of simulation methods for modeling the destruction of ships, planes, and other structures [Duncan, 2001].

Animating objects as they break, crack, tear, or in general fracture appears to be an obvious place where physically based modeling should be useful, particularly if the object is expensive, irreplaceable, or if breaking it would be hazardous. However even the most general of current techniques for animating fracture are limited to modeling only brittle materials.

The term *brittle* does not mean that a material is fragile. It means that the material experiences only elastic deformation before fracture. Few real materials are truly brittle. In contrast, *ductile* materials behave elastically up to a point and then experience some amount of plastic deformation before fracture. When brittle materials fracture, they shatter. However, ductile materials demonstrate a much wider range of fracture behaviors. (See figures 1 and 3.) This wider range of behaviors arises due to the interaction of plastic energy absorption with the fracture process.

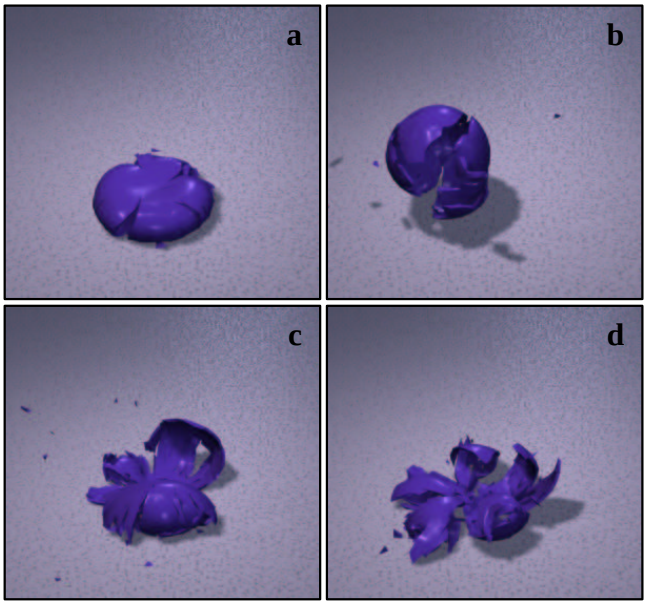

Figure 1: Four hollow balls that have been dropped onto a hard surface. The ball in (a) flattens out and visibly demonstrates plastic yielding. The other three do not show an appreciable amount of plastic deformation, but the manner in which they split and tear, as opposed to shattering, arises from of the interaction between plastic yielding and the fracture process.

This paper describes a method suitable for modeling ductile fracture in common solid materials such as plastics or metals. This method directly extends our prior technique presented in [O'Brien and Hodgins, 1999] for modeling brittle fracture. Adapting that technique to ductile as well as brittle materials requires only a simple to implement and computationally inexpensive modification. This extension dramatically expands the range of materials that may be modeled. For the sake of brevity, this paper describes only this modification and presents results demonstrating some of the effects that may be realized with it.

2 Related Work

The primary contribution of this paper is extending [O'Brien and Hodgins, 1999] to include ductile fracture by adding a plasticity model to the underlying finite-element method. The plasticity model we describe is not novel. It consists of the von Mises yield criterion, simple kinematic work hardening, and a finite yield limit [Fung, 1965]. This plasticity model is similar to the one used in [Terzopoulos and Fleischer, 1988a] and [Terzopoulos and Fleischer, 1988b]. The primary differences between their model and the one presented here are that this model realistically preserves vol-

E-mail: job@cs.berkeley.edu, adamb@cs.berkeley.edu, jkh@cs.cmu.edu

From the ACM SIGGRAPH 2002 conference proceedings.

ACM SIGGRAPH 2002, San Antonio, TX

ume and it includes a second elastic regime once a limit on the amount of plastic flow has been exceeded.

Although ductile fracture has not been widely addressed in the graphics literature, several other graphics researchers have investigated brittle fracture. In [Terzopoulos and Fleischer, 1988a] and [Terzopoulos and Fleischer, 1988b] a finite differencing scheme was used to model tearing sheets of cloth-like material. Work by [Norton et al., 1991] used a mass/spring system to model a breaking teapot. Fracture in the context of explosions was explored by [Mazarak et al., 1999], [Neff and Fiume, 1999], and [Yngve et al., 2000]. Most recently [Smith et al., 2001] used constraint-based methods for modeling brittle fracture.

Outside the graphics literature, both brittle and ductile fracture have been investigated extensively. A comprehensive review of this work can be found in [Anderson, 1995].

3 Ductile *vs.* Brittle Fracture

The common usage of the terms *elastic* and *brittle* differs substantially from their technical meanings. For example, *elastic* is often used incorrectly as a synonym for *flexible*, and the term *brittle* as a synonym for *fragile* [Merriam-Webster, 1998]. The technically correct definition of an elastic material refers to a material that returns to its original configuration when deforming forces have been removed. The ratio between the magnitude of a force and the amount of deformation it induces, that is how easily the material deforms, is the *compliance* of the material and it is irrelevant to whether or not the material is elastic. Although no real material is perfectly elastic, both natural rubber and common glass are examples of nearly elastic materials. Rubber's elastic behavior is obvious while glass appears to be rigid. A brittle material is simply one that behaves elastically up until the point where it fractures.

In contrast to an elastic material, a plastic material will not return to its original configuration once deforming forces have been removed. When a material, such as lead, bends and then holds its new shape, it demonstrates plastic behavior. As previously stated, real materials do not behave perfectly elastically. Real materials can be deformed only to a limited extent before they will no longer return to their original configuration. This limit is known as the material's elastic limit or yield point. When the elastic limit has been exceeded, the material enters a plastic regime and begins to experience plastic flow. Eventually, at the failure threshold, it fractures.

The terms *brittle* and *ductile* relate to the relative values of the elastic limit and failure threshold. If the failure threshold nearly coincides with the elastic limit, then the material will experience only negligible plastic deformation before fracture. The term *brittle* refers to such a material. In contrast, for a *ductile* material the failure threshold is significantly larger than the elastic limit so that as the material deforms it experiences an elastic regime, followed a plastic regime, and then finally fracture.

The significance of the distinction between ductile and brittle materials arises because elastic deformation stores energy whereas plastic deformation dissipates it. When a brittle material is deformed to its failure threshold, the majority of the energy used to deform it has been stored as elastic potential. When fracture occurs, the energy is released and it tends to drive the fracture further into the material. Thus, even though a large or small force may be required to start a crack in a brittle material (depending on its toughness), once the crack is started only a small amount of energy is required to push it further. In contrast, a ductile material requires significantly more work to propagate a crack because energy is being absorbed by plastic deformation. As a result, brittle materials tend to shatter, whereas ductile ones tend to tear.

In general the underlying causes of plasticity are fairly complicated and they give rise to a number of phenomena. For example, the energy absorbed by plastic deformation does not simply vanish and it may result in effects such as fatigue weakening. However for the purposes of animating failure events that occur over relatively short periods of time, the most significant effect of plasticity is how it directly effects fracture propagation, and the methods discussed here focus on modeling those effects efficiently. Additional information about mathematical models of deformation and plasticity can be found in [Fung, 1965; Fung, 1969] and [Han and Reddy, 1999]. Additional information concerning both brittle and ductile fracture may be found in [Anderson, 1995].

4 Modeling Ductility

The dynamic fracture propagation technique described in [O'Brien and Hodgins, 1999] models the fracture process using a simple tetrahedral finite-element method, rules for fracture initiation and propagation, and procedures for automatic remeshing as a crack advances. The quality of the results produced with that method is sufficient for graphical applications, and the only limitation that makes it unsuitable for modeling fracture in ductile materials is that the continuum model does not account for plastic deformation.

Extending that model to account for plasticity may be accomplished by simply redefining the strain metric used to compute element stresses. This change has only a local impact on the fracture algorithm, and so we will not repeat the details of the method which appear in [O'Brien and Hodgins, 1999]. Instead we describe only the modifications that should be made to the algorithm:

- The elastic strain, $\boldsymbol{\epsilon}^e$ defined in section 4.1 of this paper, takes the place of the total strain, $\boldsymbol{\epsilon}$, when computing the elastic stress.
- A routine for updating the plastic strain, described in section 4.2 of this paper, must be called during every integration step.

Even though this extension requires only incremental modifications to the previous method, it significantly extends the range of materials that may be realistically modeled. Furthermore, as our examples will demonstrate, small amounts of plastic yielding can dramatically effect the overall appearance of fracture patterns in a material, even though the plastic deformation itself cannot be observed directly. We feel that the significant relationship between plasticity and the appearance of fracture in most materials makes modeling plasticity a required component of any general system for animating fracture.

4.1 Decomposing Strain

The first step towards modeling plastic deformation requires separating the strain into two components:

$$\boldsymbol{\epsilon} = \boldsymbol{\epsilon}^p + \boldsymbol{\epsilon}^e \tag{1}$$

where $\boldsymbol{\epsilon}$ is the total strain, $\boldsymbol{\epsilon}^p$ is the strain due to plastic deformation, and $\boldsymbol{\epsilon}^e$ is the strain due to elastic deformation. The total strain is a purely geometric measure, it indicates how much the local shape of an object has changed from some initial reference configuration and it may be computed from the material's current configuration. (See [O'Brien and Hodgins, 1999] for computation of Green's strain.) The plastic strain reflects how the material's rest shape has been permanently distorted and it is part of the material's state. Initially, the plastic strain is zero[1] and it will evolve according to an update rule as the simulation progresses. Because the total and plastic strains are known at any given time, equation (1) may be used to compute the elastic strain.

4.2 Plastic Update

The algorithm for modeling the evolution of the plastic strain consists of a yield condition that must be met before plastic deformation occurs and a rule for computing plastic flow once the yield criterion has been met. We employ von Mises's yield criterion for the condition under which plastic flow will begin [Fung, 1965]. Our method for updating the plastic strain assumes that the rate of plastic flow in the material is close enough to its rate of deformation so that plastic flow can be updated instantaneously. This assumption precludes modeling phenomena such as creep and relaxation, but

[1] A non-zero initial value for the plastic strain could be used to model an object that has already experienced plastic deformation.

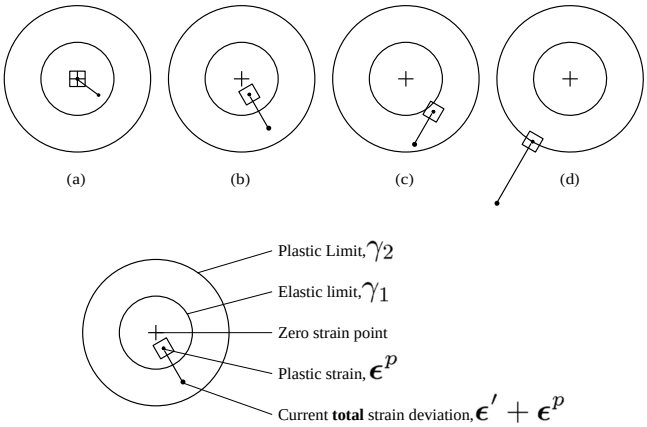

Figure 2: These diagrams illustrate the behavior of the plasticity model. **(a)** Elastic deformation. **(b)** and **(c)** Plastic deformation. **(d)** Limit of plastic yield. (See explanation in the text.)

under most circumstances these phenomena do not significantly effect fracture behavior. We also ignore the weakening of a material due to repeated plastic deformation known as fatigue. While fatigue often plays a significant role in the failure of mechanisms and structures, a previously fatigued object may be modeled by locally adjusting its toughness and plastic limits.

The von Mises yield criterion is based on the deviation of the elastic strain given by

$$\boldsymbol{\epsilon}' = \boldsymbol{\epsilon}^e - \frac{\mathrm{Tr}\,(\boldsymbol{\epsilon}^e)}{3}\boldsymbol{I} \tag{2}$$

where $\mathrm{Tr}\,(\cdot)$ is the trace of a matrix and $\boldsymbol{I}$ is the identity matrix. By averaging out the sum of the diagonal terms, the elastic strain deviation reflects only the portion of the elastic strain that is due to shape distortion and it excludes dilation. Excluding dilation makes the plastic deformation insensitive to hydrostatic pressure and will prevent the material from changing its volume which would generate unnatural behavior.

The yield criterion compares the magnitude of the elastic strain deviation (Frobenius norm) to a material constant, γ_1:

$$\gamma_1 < ||\boldsymbol{\epsilon}'|| \,. \tag{3}$$

Together equations (2) and (3) define the von Mises yield criterion [Fung, 1965]. If this condition is met then plastic deformation will occur. We compute the base change in plastic deformation according to:

$$\Delta\boldsymbol{\epsilon}^p = \frac{||\boldsymbol{\epsilon}'|| - \gamma_1}{||\boldsymbol{\epsilon}'||}\boldsymbol{\epsilon}' \,. \tag{4}$$

A limit on the total amount of plastic deformation that can be withstood by the material, γ_2, is enforced by updating the plastic strain at every time-step according to:

$$\boldsymbol{\epsilon}^p := (\boldsymbol{\epsilon}^p + \Delta\boldsymbol{\epsilon}^p)\min\left(1, \frac{\gamma_2}{||\boldsymbol{\epsilon}^p + \Delta\boldsymbol{\epsilon}^p||}\right) \,. \tag{5}$$

The behavior of this plasticity model is illustrated by figure 2. The image plane represents a two-dimensional projection of the five-dimensional space of strain deviations.[2] The plastic strain behaves as if it were being dragged by the total strain using a rope of length γ_1. The difference between the plastic strain's and the total strain's locations represents the current elastic strain. A barrier at radius γ_2 restricts the motion of the plastic strain, but not the total strain. An elastic force (stress) attracts the total strain to the plastic strain, but not the plastic strain to the total strain. As shown in figure 2.c, the plastic deformation will depend on the history of the total strain's movement.

[2] For three-dimensional objects, strain is a 3×3 symmetric tensor with nine components. Because of symmetry, only six of these components are independent. Equation (2) removes one degree of freedom, leaving five.

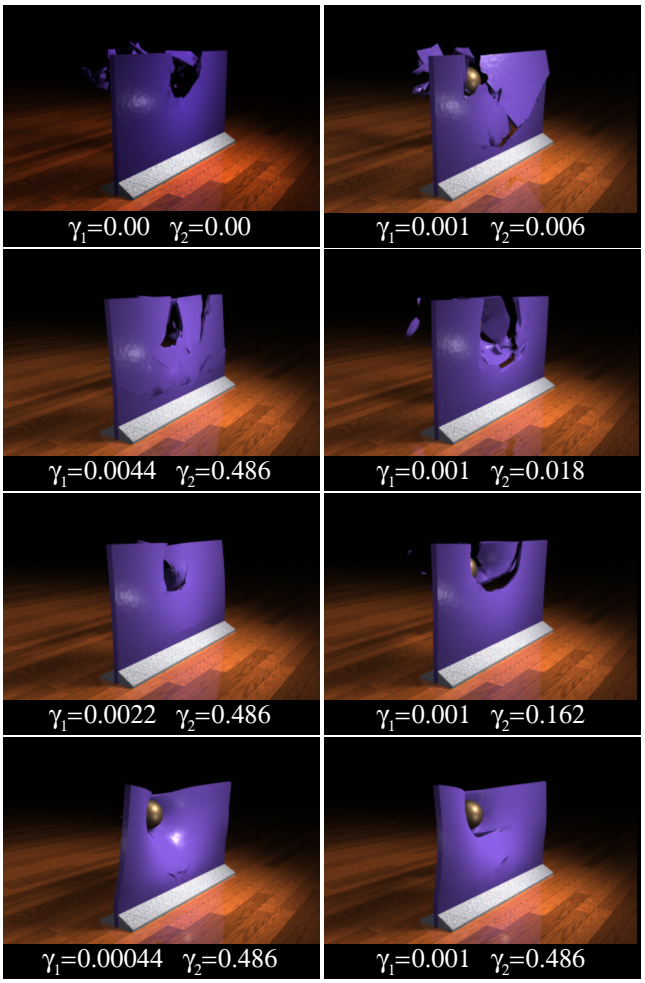

Figure 3: Images showing the results of simulating a set of eight thin walls with different material parameters as they are each struck by a heavy projectile. A purely brittle material is shown in the top-left. The others images demonstrate how varying the plasticity of the material can produce a range of effects.

5 Results and Discussion

Figure 3 shows a set of thin walls that have been struck by a heavy weight. The walls are clamped at the bottom, and they experience collision forces due to contacts with the ground plane, the weight, and self-collisions. The top-left image in figure 3 with ($\gamma_1 = \gamma_2 = 0$) shows the behavior of a purely brittle material. The other images in figure 3 show some examples that demonstrate the effects of different plastic parameter values. In the left column γ_1 has been varied while γ_2 was held fixed. The right column demonstrates the result of varying γ_2 while γ_1 was held fixed. Some of the images, such as the bottom-right with ($\gamma_1 = 0.001, \gamma_2 = 0.486$), demonstrate obvious amounts of plastic yielding. However, plasticity also plays a significant role in the images where plastic yielding is not obviously visible. For example, ($\gamma_1 = 0.001, \gamma_2 = 0.162$) shows only a small part of the wall being torn away largely intact, and ($\gamma_1 = 0.001, \gamma_2 = 0.006$) shows the wall breaking into several large pieces. Both of these behaviors demonstrate how the fracture

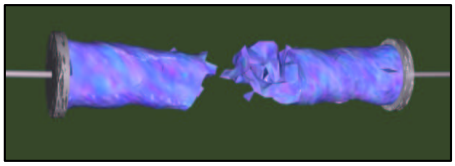

Figure 4: A solid cylinder that experiences ductile fracture when it is pulled apart.

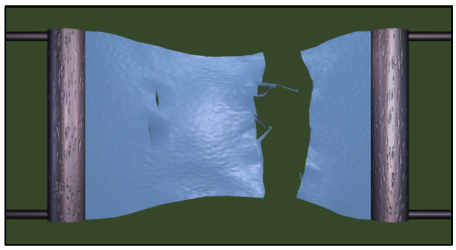

Figure 5: A thin sheet that has been torn apart.

process can be affected by otherwise unnoticeable amounts of plastic deformation. The proceedings DVD contains animations that further illustrate the behaviors depicted in figure 3 as well as the behaviors shown in the other figures.

Figure 4 shows a solid cylinder tearing as it is pulled and twisted apart. Figures 5 and 7 show the ductile fracture that results when other objects are ripped apart.

One way to assess the realism of an animation technique is by comparing it with the real world. Figure 6 shows a real clay slab that has been struck by a spherical projectile and a simulated slab of plastic material that has also be struck by a spherical projectile. Although the two images have obvious differences, the holes left by the projectiles demonstrate qualitative similarities.

While modifying the computation of the element stresses to use the elastic strain instead of the total strain requires only minor changes to an existing code, the change may also have an effect on the integration scheme. Our implementation uses an explicit integrator that takes adaptive time steps. The step size is determined by monitoring the total energy to ensure that the system is not going unstable. We compared the size of steps taken when simulating a purely elastic material to those taken when simulating a material that was identical except that the plasticity code had been enabled. During periods when collisions were occurring, both simulations took similar-sized integration steps. At other times, however, the average step size for the plastic material was approximately twice that of the purely elastic one. This result is not surprising because plastic deformation absorbs energy implying that it should tend to help stabilize the system, but it is only a single test on a single set of parameters and further tests would need to be done before any more general statement could be made.

The deformation model we implemented allows a regime of elastic deformation, followed by a plastic regime, and then possibly followed by a second elastic regime. While this model suffices for many materials, other materials, such as woven fabrics, may go through multiple cycles of elastic and plastic behavior. We have also worked only with a linear relationship between elastic strain and stress. While a linear model adequately describes many materials, other materials such as biological tissues demonstrate distinctly non-linear elastic behavior. Developing adequate graphical models for these types of materials remains an area for future work.

Acknowledgments

The authors thank Cindy Grimm for the use of the model shown in figure 7, and the members of the Berkeley Graphics Group for their helpful criticism and comments. This work was supported in part by Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, a research grant from the Okawa Foundation, and NSF grant CCR-0204377.

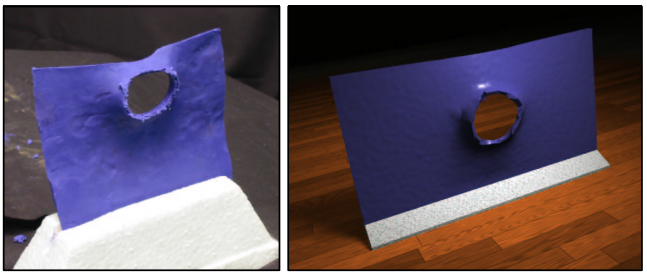

Figure 6: A comparison showing (left) a real clay slab that has been punctured by a spherical projectile and (right) a similar result generated with our method.

References

ANDERSON, T. L. 1995. *Fracture Mechanics: Fundamentals and Applications*, second ed. CRC Press, Boca Raton. 292

DUNCAN, J. 2001. More war. *Cinefex 86* (July), 64–97. 291

FUNG, Y. C. 1965. *Foundations of Solid Mechanics*. Prentice-Hall, Englewood Cliffs, N.J. 291, 292, 293

FUNG, Y. C. 1969. *A First Course in Continuum Mechanics*. Prentice-Hall, Englewood Cliffs, N.J. 292

HAN, W., AND REDDY, B. D. 1999. *Plasticity: Mathematical Theory and Numerical Analysis*. Interdisciplinary Applied Mathematics. Springer-Verlag, New York. 292

MAZARAK, O., MARTINS, C., AND AMANATIDES, J. 1999. Animating exploding objects. In the proceedings of *Graphics Interface '99*. 292

MERRIAM-WEBSTER, Ed. 1998. *Merriam–Webster's Collegiate Dictionary*, 10th ed. International Thomson Publishing, Springfield, Mass. 292

NEFF, M., AND FIUME, E. 1999. A visual model for blast waves and fracture. In the proceedings of *Graphics Interface '99*. 292

NORTON, A., TURK, G., BACON, B., GERTH, J., AND SWEENEY, P. 1991. Animation of fracture by physical modeling. *The Visual Computer 7*, 210–217. 292

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In the proceedings of *ACM SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 137–146. 291, 292

SMITH, J., WITKIN, A., AND BARAFF, D. 2001. Fast and controllable simulation of the shattering of brittle objects. *Computer Graphics Forum 20*, 2, 81–91. 292

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Deformable models. *The Visual Computer 4*, 306–331. 291, 292

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In the proceedings of *ACM SIGGRAPH 88*, Computer Graphics Proceedings, Annual Conference Series, 269–278. 291, 292

YNGVE, G. D., O'BRIEN, J. F., AND HODGINS, J. K. 2000. Animating explosions. In the proceedings of *ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 29–36. 292

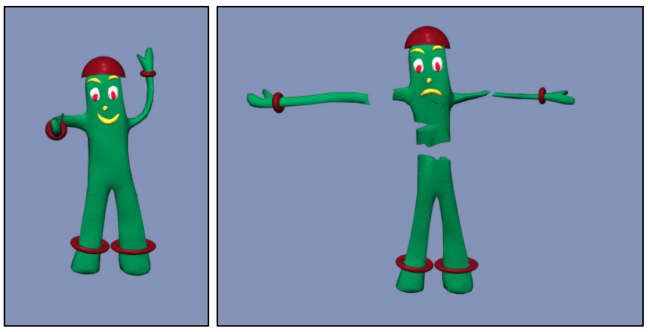

Figure 7: A cartoon character being dismembered by a red torture device.

Synthesizing Sounds from Rigid-Body Simulations

James F. O'Brien Chen Shen Christine M. Gatchalian

EECS, Computer Science Division
University of California, Berkeley

Abstract

This paper describes a real-time technique for generating realistic and compelling sounds that correspond to the motions of rigid objects. By numerically precomputing the shape and frequencies of an object's deformation modes, audio can be synthesized interactively directly from the force data generated by a standard rigid-body simulation. Using sparse-matrix eigen-decomposition methods, the deformation modes can be computed efficiently even for large meshes. This approach allows us to accurately model the sounds generated by arbitrarily shaped objects based only on a geometric description of the objects and a handful of material parameters. We validate our method by comparing results from a simulated set of wind chimes to audio measurements taken from a real set.

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; H.5.5 [Information Interfaces and Presentation]: Sound and Music Computing—Signal analysis, synthesis, and processing

Keywords: Sound modeling, physically based modeling, simulation, surface vibrations, dynamics, animation techniques, finite element method, modal synthesis, modal analysis.

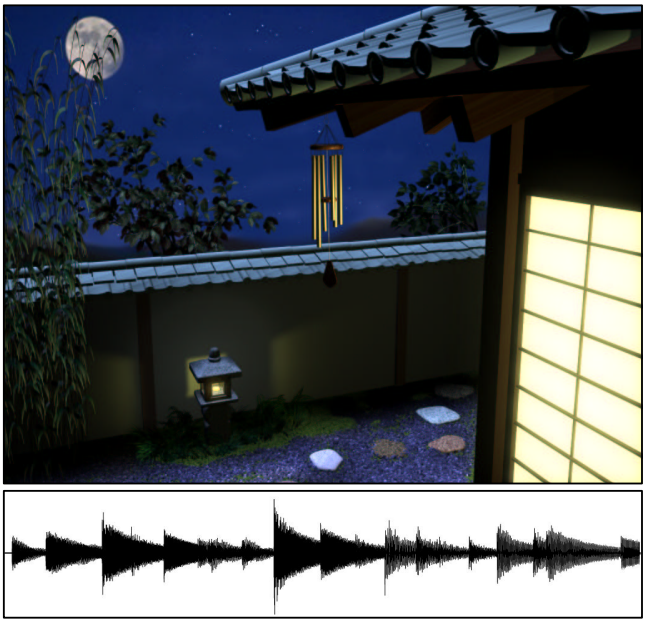

Figure 1: A synthetic environment containing a set of simulated wind chimes. Both the motion of the chimes and the corresponding audio can be computed at interactive speeds.

1 Introduction

One of the central goals for the field of computer graphics is the compelling portrayal of realistic synthetic environments. However, generating convincing animations of scenes such as that shown in figure 1 requires depicting not only the visual aspects of the scene, but its audio components as well. While constructing a soundtrack by hand often provides a feasible option for animations that are generated off line, interactive applications increasingly rely on physically based simulation techniques to generate animated motions in real-time and these applications require methods for generating the corresponding audio in real-time as well.

One class of simulation method that has found widespread use in real-time applications is rigid-body simulations. Because rigid bodies are made up of incompliant materials, they experience only small-amplitude deformations during interactions with their environment. Explicitly discarding these small deformations allows rigid-body simulators to model a system's remaining degrees of freedom efficiently. However, although visually insignificant, it is the vibration of these small-amplitude deformations that generates the sounds heard from these objects.

This paper describes a real-time technique for generating realistic and compelling sounds that correspond to the motions generated by rigid-body simulation methods. Precomputing the shape and frequencies of an object's deformation modes allows that object's vibrational response to contact forces to be efficiently computed at runtime. The vibrational response is then used directly to compute the corresponding audio. Our technique computes an object's deformation modes numerically by performing an eigen-decomposition of the system matrices from a finite element model of the object. This approach allows us to accurately model the sounds generated by arbitrarily shaped objects based on a geometric description of the object and a handful of material parameters. The diagram in figure 2 provides an overview of this process.

2 Background

The technique presented in this paper is closely related to previous methods developed by van den Doel, Kry, and Pai. The concept of using the vibrational modes of an object for generating sound was originally introduced to the graphics community in [van den Doel

job@cs.berkeley.edu, csh@cs.berkeley.edu, tine@cs.berkeley.edu

ACM SIGGRAPH Symposium on Computer Animation 2002

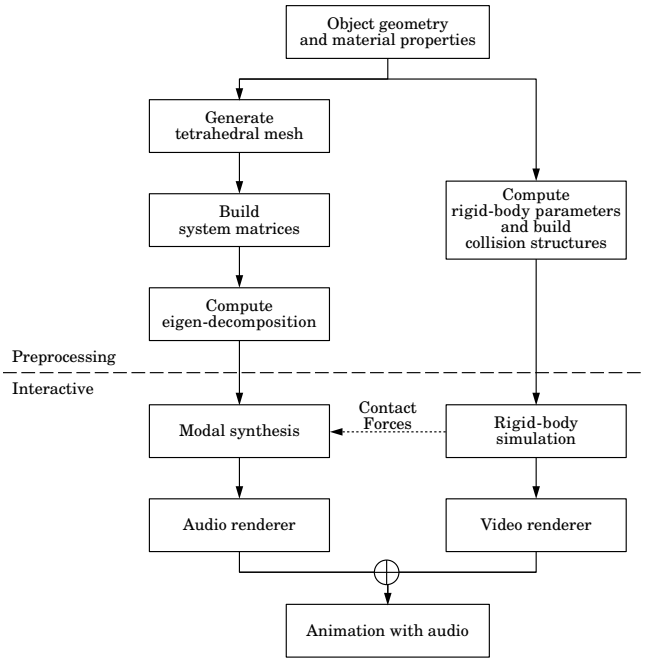

Figure 2: This diagram illustrates both the preprocessing steps that are used to construct the audio/visual model for an object, and the processes that subsequently generate sound and motion from this description at interactive speeds.

and Pai, 1996] and [van den Doel and Pai, 1998]. They showed that the analytically computed vibrational modes of simply shaped objects, such as square plates or cylindrical rods, could be used to generate sound maps over the object surfaces. They constructed a system that computed realistic contact sounds when an interactive user indicated a point on the surfaces of the modeled objects. Because the maps were generated from the mode shapes, the system correctly captured the variations that arise when objects are struck at different locations. More recently, [van den Doel et al., 2001] described a method that uses recorded data to construct sound maps over the surface of an object. In addition to allowing a user to interactively generate sounds by tapping the object surfaces, they used contact forces from a real-time rigid-body simulation to excite the sampled modes.

Our work builds on the ideas of these two previous methods by extending the range of objects that can be modeled to include ones that are neither simple shapes nor available to be measured. The analytically computed modes used in [van den Doel and Pai, 1996] and [van den Doel and Pai, 1998] are the continuous equivalent of the numerically computed discretized modes described in Section 3.1 of this paper. Numerical computation allows determining the modes for essentially arbitrary shapes as opposed to a few simple shapes, and it makes fewer assumptions about the underling differential equations. Our method for driving the sound generation from a rigid-body simulation is essentially identical to the method used in [van den Doel et al., 2001], but we have found that useful results can be generated using "off-the-shelf" rigid-body simulators and that a special contact method is not necessarily required unless one wishes to produce rubbing or scraping sounds.

Another related method for generating sound has been described in [O'Brien et al., 2001]. Their approach uses a nonlinear finite element method to explicitly model the response of an object to external forces. Audio is generated by analyzing the computed surface behavior and then applying a set of filters to the computed motion for extracting frequency components that fall within the audible range. Unlike the method detailed in this paper and the previously described methods of van den Doel and his colleagues, the use of a nonlinear finite element method allows them to model sounds that arise from nonlinear behaviors such as buckling. The main limitation of their method is that it requires large amounts of computation. In contrast, our method can accurately model only sounds produced by linear phenomena, but it can compute these sounds in real-time.

In addition to the above physically motivated work on sound generation, other prior work in the graphics community has focused on sound propagation and heuristic approaches to sound generation. Producing synchronized soundtracks for animations was addressed in [Takala and Hahn, 1992] and [Hahn et al., 1995]. For modeling tearing cloth, [Terzopoulos and Fleischer, 1988] generated soundtracks by playing a pre-recorded sound whenever a connection in a spring mesh failed. Work described in [Savioja et al., 1997] focused on creating virtual musical performances in virtual spaces using physically derived models of musical instruments and acoustic ray-tracing for spatialization of the sound sources. Other researchers have developed methods for correctly modeling reflections and transmissions within the sonic environment [Funkhouser et al., 1998; Funkhouser et al., 1999; Min and Funkhouser, 2000; Tsingos et al., 2001].

The method described in this paper is also related to previous work in the graphics community on modeling deformable objects. The idea of decoupling an object's rigid-body behavior from it's elastic deformation was proposed in [Terzopoulos and Fleischer, 1988] as an efficient method for modeling deformable objects. This idea was extended in [Pentland and Williams, 1989] by using modal analysis for modeling deformable objects, although instead of actually using the object's vibrational modes they approximated them with arbitrary linear and quadratic deformation fields.

Outside the field of computer graphics, an extensive amount of research on sound modeling has been conducted in the digital sound and music communities. There the focus has been primarily on accurately modeling the sounds generated by musical instruments, including the fine subtleties that distinguish high-quality instruments. A comprehensive review of the work that has been done in those areas can be found in [Cook, 2002].

3 Methods

The mechanical dynamics of a solid physical object can be decomposed into two components: idealized rigid-body motions and elastic deformations. An object is referred to as being rigid, or incompliant, if its response to typical interactions includes only negligible deformations. For example, when a person taps the side of a drinking glass it flexes slightly but the amplitude of this deformation is small enough to be unobservable by sight. However, this small deformation may be observable by hearing. In particular, if the elastic properties of the glass are such that the small deformation induced by the tap results in vibrations at frequencies between approximately 20 and $20,000$ Hz, then the small pressure fluctuations caused by the oscillating deformation will be heard as sound. For further information on the physical process of sound generation we refer the reader to [Kinsler et al., 2000].

3.1 Modal Analysis

Our method for modeling the sounds generated by rigid objects makes use of a well studied technique known as modal analysis. This section presents a brief overview of modal analysis and provides the framework for describing our methods. We refer the reader to [Cook et al., 1989] for additional information on modal analysis.

A physical system that has been discretized using a finite element, finite differencing, or other similar method can be expressed in the following general form:

$$\boldsymbol{\mathcal{K}}(\boldsymbol{d}) + \boldsymbol{\mathcal{C}}(\boldsymbol{d}, \dot{\boldsymbol{d}}) + \boldsymbol{\mathcal{M}}(\ddot{\boldsymbol{d}}) = \boldsymbol{f} \tag{1}$$

where $\boldsymbol{d}$ is the vector of node displacements, an overdot indicates a derivative with respect to time, $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{C}}$ are nonlinear functions that respectively determine the internal forces due to node displacements and node velocities, $\boldsymbol{\mathcal{M}}$ maps node accelerations to node momenta, and $\boldsymbol{f}$ represents any other (*e.g.* external) forces. Typically, the forces determined by $\boldsymbol{\mathcal{K}}$ are internal elastic forces and $\boldsymbol{\mathcal{C}}$ determines damping forces.

In general, equation (1) is nonlinear, however if we assume that the displacements are small then we may linearize about the system's rest configuration giving:

$$\boldsymbol{K}\boldsymbol{d} + \boldsymbol{C}\dot{\boldsymbol{d}} + \boldsymbol{M}\ddot{\boldsymbol{d}} = \boldsymbol{f} \tag{2}$$

where $\boldsymbol{K}$, $\boldsymbol{C}$, and $\boldsymbol{M}$ are respectively known as the system's stiffness, damping, and mass matrices. For the physical systems corresponding to solid objects, all three matrices are real and symmetric. Both $\boldsymbol{K}$ and $\boldsymbol{C}$ are positive semi-definite, and $\boldsymbol{M}$ is positive definite. Linearizing in this fashion is consistent with our goal of modeling the small-amplitude, high-frequency vibrations in solid objects that produce sound. Unfortunately, the linearized system cannot model the rotational components of rigid-body motion. We will put this issue aside for now, but later we will return to it and show how the rigid-body modes can be decoupled from all other modes.

Once we have the linearized system, the next step in the modal analysis is to perform a series of manipulations that will diagonalize equation (2). To facilitate this process, we will first assume that $\boldsymbol{C} = \alpha_1\boldsymbol{K} + \alpha_2\boldsymbol{M}$ for some α_1 and α_2. Expressing the damping matrix as a linear combination of the stiffness and mass matrices is known as Raleigh damping. Although this assumption simplifies diagonalization while still producing good results, it is not strictly necessary. A more general assumption, known as proportional damping, that expresses the damping matrix as a linear combination of powers of the stiffness and mass matrices would also be diagonalized by the process described below but the equations would be more cumbersome. Additionally, even if for some reason $\boldsymbol{C}$ must be arbitrary, then other, slightly more complicated, methods are available for decoupling equation (2) [Anderson et al., 1999; Bai et al., 2000].

Replacing $\boldsymbol{C}$ with $\alpha_1\boldsymbol{K} + \alpha_2\boldsymbol{M}$ gives:

$$\boldsymbol{K}(\boldsymbol{d} + \alpha_1\dot{\boldsymbol{d}}) + \boldsymbol{M}(\alpha_2\dot{\boldsymbol{d}} + \ddot{\boldsymbol{d}}) = \boldsymbol{f}\ . \tag{3}$$

Since $\boldsymbol{M}$ is symmetric and positive definite, it may be decomposed using a Cholesky factorization so that $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^{\mathsf{T}}$. If we introduce another variable, $\boldsymbol{y} = \boldsymbol{L}^{\mathsf{T}}\boldsymbol{d}$, and then rewrite equation (3) in terms of $\boldsymbol{y}$ after pre-multiplying by $\boldsymbol{L}^{-1}$ we then have:

$$\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{L}^{-\mathsf{T}}(\boldsymbol{y} + \alpha_1\dot{\boldsymbol{y}}) + (\alpha_2\dot{\boldsymbol{y}} + \ddot{\boldsymbol{y}}) = \boldsymbol{L}^{-1}\boldsymbol{f}\ . \tag{4}$$

The real and symmetric matrix $\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{L}^{-\mathsf{T}}$ can be decomposed into $\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{L}^{-\mathsf{T}} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathsf{T}}$ where $\boldsymbol{V}$ is the orthogonal matrix whose columns are the eigenvectors of $\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{L}^{-\mathsf{T}}$ and $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues. Introducing another variable, $\boldsymbol{z} = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{y}$, and pre-multiplying by $\boldsymbol{V}^{\mathsf{T}}$ transforms equation (4) into:

$$\boldsymbol{\Lambda}(\boldsymbol{z} + \alpha_1\dot{\boldsymbol{z}}) + (\alpha_2\dot{\boldsymbol{z}} + \ddot{\boldsymbol{z}}) = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{L}^{-1}\boldsymbol{f} \tag{5}$$

which can be rearranged to give:

$$\boldsymbol{\Lambda}\boldsymbol{z} + (\alpha_1\boldsymbol{\Lambda} + \alpha_2\boldsymbol{I})\dot{\boldsymbol{z}} + \ddot{\boldsymbol{z}} = \boldsymbol{g} \tag{6}$$

where $\boldsymbol{g} = \boldsymbol{V}^{\mathsf{T}}\boldsymbol{L}^{-1}\boldsymbol{f}$.

At this point the original linear system of equation (3) has been diagonalized into a set of decoupled oscillators. The i'th row of equation (6) is the scalar second-order differential equation:

$$\lambda_i z_i + (\alpha_1\lambda_i + \alpha_2)\dot{z}_i + \ddot{z}_i = g_i \tag{7}$$

where λ_i is the i'th entry of the diagonal matrix $\boldsymbol{\Lambda}$. Equation (7) may be solved by numerical integration or it may be solved more efficiently using the analytic solution:

$$z_i = c_1 e^{t\omega_i^+} + c_2 e^{t\omega_i^-} \tag{8}$$

where c_1 and c_2 are arbitrary (complex) constants, and ω_i is the complex frequency given by

$$\omega_i^{\pm} = \frac{-(\alpha_1\lambda_i + \alpha_2) \pm \sqrt{(\alpha_1\lambda_i + \alpha_2)^2 - 4\lambda_i}}{2}\ . \tag{9}$$

The absolute value of the imaginary part of ω_i is the frequency (in radians/second, not Hertz) of the mode, and the real part is the mode's decay rate.

The decoupled system of equation (6) is *not an approximation* of the original linear system in equation (3), it is *exactly* the same as the original linear system. Of course the linear system was an approximation of the original nonlinear one, but any problem that could be solved using equation (3) could also be solved with equation (6).

The columns of $\boldsymbol{L}^{-\mathsf{T}}\boldsymbol{V}$ are the vibrational modes of the object being modeled. (See figure 3.) Each mode has the property that a displacement or velocity over the object that is a scalar multiple of the mode will produce an acceleration that is also a scalar multiple of the mode. This property means that the modes do not interact with each other, which is why decoupling the system into a set of independent oscillators was possible. The eigenvalue for each mode is the ratio of the mode's elastic stiffness to the mode's mass, and it is the square of the mode's natural frequency (in radians per second). In general the eigenvalues will be nonzero, but for each free body in the system there will be six zero eigenvalues that correspond to the body's six rigid-body modes. The rigid-body eigenvalues are zero because a rigid-body displacement will not generate any elastic forces.

3.2 Rigid Body Simulation

As discussed previously, the rigid-body modes for an object do not interact with the object's deformation modes provided the amount of elastic deformation experienced by the object is small.[1] Additionally, small-amplitude elastic deformations will not significantly effect the rigid-body collisions between objects. These observations allow us to model the rigid-body behavior of the objects in almost the same way as if we were not interested in generating audio. The only change that must be made to the rigid-body simulation is that information about contact forces must be gathered and exported to another process that will generate the audio. Of course, hearing the results of the rigid-body simulation, in addition to seeing them, may reveal previously unnoticed inadequacies of the simulator, but we have not found this to be a problem with the simulation engines we have worked with.

We have implemented our system using two existing rigid-body simulation engines that were not originally designed for generating audio. Our choice of engines was motivated by what systems were readily available and how well they were able to model the scenarios we wished to test. The first is a commercial software package, Vortex, sold by Critical Mass Labs. The second system we are using had been previously written by Okan Arikan, a graduate student

[1] Actually, the requirement was that all displacements be small, including displacements corresponding to the rigid-body modes. The translation modes are inherently linear so they cannot interact with the elastic modes regardless of their magnitude, but for a rapidly rotating body there will be some coupling between the rotation modes and the elastic ones. Unless the object is rotating very rapidly or experiencing large angular accelerations, the coupling between rotation and elastic modes with frequencies in the audible range will be negligible, so we ignore this interaction.

not involved in this project. No special changes were made to either package other then instrumenting them to allow reporting collision forces.

3.3 Deformation Model

Once the task of modeling the rigid-body modes has been delegated to a rigid-body simulator, the remaining elastic deformation modes can be used for generating audio. Because we are interested in modeling sounds from incompliant objects, we can use the modal decomposition methods described in Section 3.1 to compute their behavior efficiently. However before we can perform a modal decomposition, we must first select a deformable modeling method that can be used to generate the $\boldsymbol{K}$, $\boldsymbol{C}$, and $\boldsymbol{M}$ matrices.

The method we are using for modeling deformable behavior is the tetrahedral finite element method described by [O'Brien and Hodgins, 1999] for modeling fracture propagation, and subsequently used in [O'Brien et al., 2001] for modeling nonlinear audio generation. As discussed by O'Brien, Cook, and Essl, a variety of methods could be used, including spring/mass systems or finite differences methods. We selected this finite element method because their previous results show that it is accurate enough for generating compelling audio.

Computing the global stiffness and mass matrices proceeds by first computing individual 12×12 stiffness and mass matrices for each element and then assembling the results to form the global matrices. From [O'Brien and Hodgins, 1999] the nonlinear node forces are given by:

$$f_{[i]a} = -\frac{\mathrm{vol}}{2} \sum_{j=1}^{4} p_{[j]a} \sum_{k=1}^{3} \sum_{l=1}^{3} \beta_{jl} \beta_{ik} \sigma_{kl} \tag{10}$$

where $f_{[i]a}$ is the a'th component of the force exerted on the i'th node of the element, vol is the volume of the element, p are the node positions, β is the element basis matrix, and σ is the stress tensor within the element. Details for computing β and σ appear in [O'Brien and Hodgins, 1999].

The element stiffness matrix, $\boldsymbol{k}$, is computed by taking the partials of $\boldsymbol{f}$ and evaluating them at zero displacement:

$$k_{[ij]ab} = \left. \frac{\partial f_{[i]a}}{\partial p_{[j]b}} \right|_{\boldsymbol{p}=\boldsymbol{p}_{\mathrm{rest}}} \tag{11}$$

$$= -\frac{\mathrm{vol}}{2} (\lambda \beta_{ia} \beta_{jb} + \mu \beta_{ib} \beta_{ja} + \mu \sum_{k=1}^{3} \beta_{ik} \beta_{jk} \delta_{ab}) \tag{12}$$

where δ is the Kronecker delta, and λ and μ are the material's Lamé constants.[2] This is the exactly the same matrix that would have resulted if Cauchy's infinitesimal strain had been used in place of Green's strain, however with Cauchy's strain the partials would be constant with respect to node position so that it would not matter where they were evaluated.

The element mass matrix, $\boldsymbol{m}$ is computed by taking the second partials of the kinetic energy within the element with respect to the node velocities, which turns out to be constant with respect to node position and velocity:

$$\boldsymbol{m}_{[ij]ab} = \frac{\partial^2 \kappa}{\partial \dot{p}_{[i]a} \partial \dot{p}_{[j]b}} \tag{13}$$

$$= \frac{\rho\,\mathrm{vol}}{20} (1 + \delta_{ij}) \delta_{ab} \tag{14}$$

[2] Unfortunately, the symbol λ is conventionally used both to indicate one of the system eigenvalues and the first Lamé constant. In this paper it should be clear from context (and the presence or absence of a subscript) what the symbol is referring to.

where κ is the kinetic energy within the element, an overdot represents a derivative with respect to time (*i.e.* $\dot{p}$ are node velocities), and ρ is the material's density.

The global stiffness and mass matrices, $\boldsymbol{K}$ and $\boldsymbol{M}$, are built by assembling the element matrices. Assuming that we are working with three-dimensional objects, each of the global matrices will be $3N \times 3N$ where N is the number of nodes in the finite element mesh. Each entry in each of the 12×12 element matrices is accumulated into the corresponding entry of the global matrix.

Since each node in a tetrahedral mesh will share an element with only a small number of the other nodes, the global matrices will be very sparse. This sparseness means that an eigen decomposition of $\boldsymbol{K}$ can be performed efficiently using sparse matrix algorithms. Unfortunately, the Cholesky decomposition tends to generate a dense $\boldsymbol{L}$ matrix even when $\boldsymbol{M}$ is originally sparse, and as a result computing $\boldsymbol{L}^{-1}$ may be costly and $\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{L}^{-\mathsf{T}}$ will be densified.

Dense matrix algorithms can be used for systems up to approximately 1000 nodes, but beyond that we suggest using an alternate mass matrix that does not generate a dense Cholesky decomposition. The alternate mass matrix, known as a lumped mass matrix, simply shifts the sum of each row onto the diagonal:

$$\boldsymbol{m}^{\mathrm{lumped}}_{[ij]ab} = \frac{\rho\,\mathrm{vol}}{4} \delta_{ij} \delta_{ab} \,. \tag{15}$$

Because the element mass matrices are diagonal, the global mass matrix will be as well, and its Cholesky decomposition will also be diagonal: $\boldsymbol{L}$ will be a diagonal matrix whose entries are simply the square root of the entries of the lumped $\boldsymbol{M}$. For small systems generated by coarse meshes, the errors introduced by mass lumping may be significant. However, as the mesh gets finer the errors introduced by lumping quickly become insignificant [Cook et al., 1989]. Luckily, the large systems corresponding to fine meshes are precisely the ones that require the sparse solvers facilitated by mass lumping. Our implementation includes both dense and sparse decomposition routines and we use whichever is appropriate to the size of a particular system. For dense decompositions, we use the routines from LAPACK [Anderson et al., 1999], and for sparse decompositions we use the TRLan package [Wu and Simon, 1999]. The method used for each of our examples, along with computation times and the number of nodes, is listed in table 1.

The use of Raleigh damping was another simplification that we made to facilitate decoupling equation (2). In [O'Brien and Hodgins, 1999] they used a nonlinear stiffness-proportional damping term based on the strain rate with parameters ϕ and ψ. Raleigh damping is equivalent to a linearization of this damping term with the additional constraint that $\frac{\lambda}{\phi} = \frac{\mu}{\psi}$, and the Raleigh parameter α_1 should be set to this ratio to generate equivalent results. O'Brien and Hodgins did not discuss a mass proportional damping term, but setting α_2 to a non-zero value would be equivalent to including a $(-\alpha_2 \dot{d}_i m_i)$ damping force on each node.

Even with sparse matrix methods, computing a system decomposition still requires a significant amount of time, so it is worth noting that certain changes may be made without recomputing the decomposition. The damping parameters, α_1 and α_2, have no effect on the decomposition, so the only work involved when changing them is re-evaluating equation (9). Changing the material's density does not change the mode shapes, it only scales the eigenvalues by the inverse of the scale factor applied to the density. Similarly, scaling the Lamé constants both by the same scale factor (*i.e.* so that the ratio between λ and μ is preserved) only scales the eigenvalues by the same ratio. Changing the ratio between the Lamé constants, changing the shape of the object, or modifying the mesh all require recomputing the decomposition.

3.4 Sound Generation

Once all of the computational machinery described above is available, the actual process of computing audio matching the motion

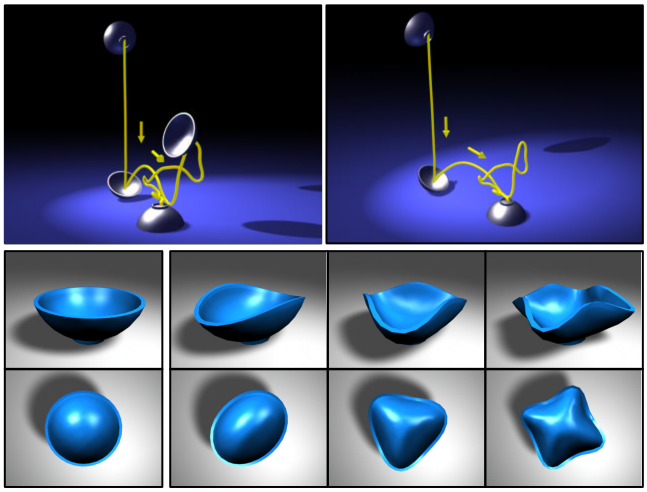

Figure 3: The top shows a multi-exposure image from an animation of a bowl falling onto a hard surface with the path of the bowl's center traced by a yellow curve. Only the bowl is sounding. The two bottom rows show a side and top view of the bowl along with three of the bowl's first vibrational modes. (The modes selected for the illustration are the first three non-rigid ones with distinct eigenvalues that are excited by a transverse impulse to the bowl's rim.)

from a rigid-body simulation is both straightforward to implement and computationally efficient:

1. A rigid-body simulation is set up for the desired scenario.
2. For each object in the simulation, the system matrices are assembled and decomposed into their vibrational modes (*i.e.* the columns of $\boldsymbol{L}^{-\mathsf{T}}\boldsymbol{V}$).
3. For each object in the simulation, only the columns of $\boldsymbol{L}^{-\mathsf{T}}\boldsymbol{V}$ corresponding to $|\mathrm{Im}(\omega_i)|$ in the range $3.18 \ldots 3{,}180\,\mathrm{Rad/s}$ (20...20,000 Hz) are retained, the rest are discarded (or if the sparse method is used, never computed).
4. As the rigid body simulation runs, collision forces are projected onto the retained modes. The response of each mode is modeled using equation (8).
5. Each mode response is scaled according to how it moves the objects surface and the scaled responses are then summed together.
6. Finally, the result is output to the computer's audio device.

In practice, not all of the modes in the audible range need to be retained. As discussed in [van den Doel et al., 2001], high-quality results can easily be obtained using only the first 800 or fewer modes.

A mode's response to a projected impulse is given by equation (8) with

$$c_1 = \frac{2\Delta t g_i}{\omega_i^+ - \omega_i^-} \tag{16}$$

$$c_2 = \frac{2\Delta t g_i}{\omega_i^- - \omega_i^+} \tag{17}$$

where Δt is the interval over which the projected force is applied, and t is time relative to when the impulse was applied. Substituting these values of c_1 and c_2 into equation (8), recalling that only modes with $|\mathrm{Im}(\omega_i)|$ in the range $3.18 \ldots 3{,}180\,\mathrm{Rad/s}$ are used, and then simplifying yields

$$z_i = \frac{2\Delta t g_i}{|\mathrm{Im}(\omega_i)|} e^{t\mathrm{Re}(\omega_i)} \sin(t|\mathrm{Im}(\omega_i)|) \quad . \tag{18}$$

Evaluating equation (18) for every audio sample is inefficient. By noting that $e^{\omega(t+s)} = e^{\omega t} e^{\omega s}$, the value of the oscillator at one audio sample can be computed from the previous value using only a single complex multiply. Additionally, as a mode is excited at subsequent times by different contact forces, the additional excitations can be modeled by simply adding the new value to the oscillator's current value. Because the cost of modeling additional impulses is essentially zero, the forces from the rigid-body simulation may be convolved with a Gaussian kernel to model the effect of soft collisions, or with a noise function to model small-scale roughness that is below the resolution of the rigid-body simulator [van den Doel et al., 2001]. Our results were generated using the former.

A method for modeling the coupling between vibrations in an object and vibrations in the surrounding air is described in [O'Brien et al., 2001]. Unfortunately, their method is too slow for real-time use. We compute an approximate coupling coefficient for each mode by summing the amount of normal displacement generated by that mode over the surface of the object multiplied by the mode's frequency. The coupling coefficient for each mode multiplies the result computed by that mode's oscillator and the sum of the scaled oscillators is the final sound generated by the system. A result of this simplification is all objects are treated as omni-directional sources.

4 Results and Discussion

We have built a system that implements the methods described above and used it to generate a number of demonstrative examples. Table 1 lists the parameters that were used in each of the examples, and the video tape accompanying this paper contains animations that exhibit the sounds and motions produced.

To test how well the computed results match real objects, we generated the wind chimes shown in figure 1. These chimes were modeled based on measurements from a real set of chimes. Each tube is a hollow cylinder $1.25\,\mathrm{cm}$ in radius with a nominal wall thickness of $1\,\mathrm{mm}$. The measured lengths of the chimes are listed in table 2. We computed the modal decomposition for each chime using reference parameters for aluminum. The resulting base frequencies matched measured ones to within 2% error. However, the real chimes were slightly out of tune, so we tuned the simulated set by adjusting the tube lengths so that they were within $\pm 1\,\mathrm{Hz}$ of the correct (D scale) tuning.

Figure 3 shows a bowl model that was used for two of the examples. The modal decomposition of the bowl was computed once with material parameters for aluminum and again with material parameters for wood (oak). Two animations were created, both with the same rigid-body motion but with the two sound tracks generated from the two different modal decompositions. The resulting audio (refer to video tape) captures the general characteristics of both materials as well as details such as the sound produced as the bowl rolls on its edge. Figure 3 also illustrates the mode-shapes for three of the bowl's vibrational modes by showing the results of applying the mode as a displacement over the bowl's original shape.

An example generated using a more complex model consists of bunny figurines falling through a chute. (See figure 4.) Both the bunny and the shelves in the chute generate sounds when struck. The shelves are made of plastic, metal, and wood. The bunny is ceramic. The tetrahedral bunny model was generated by meshing the region between the surface of the Stanford Bunny model and an interior offset surface to create a hollow figure with finite thickness walls, as shown on the right side of figure 4. The right side of

Example	Figure	λ (Pa)	μ (Pa)	α_1	α_2	ρ (Kg/m^3)	Base Freq. (Hz)	Decay	Num. Nodes	Method	Precompute
Chime(D3)	1	4.98×10^{10}	2.57×10^{10}	1×10^{-7}	0	2700	587.4	0.6	18796	Sparse	2h 24min
Bowl #1	3	4.98×10^{9}	2.57×10^{9}	1×10^{-7}	30	2700	551.3	15.6	387	Dense	4min 12sec
Bowl #2	–	5.00×10^{8}	1.00×10^{8}	8×10^{-6}	50	750	216.7	22.4	387	Dense	4min 12sec
Bunny (Ceramic)	4	3.99×10^{9}	2.05×10^{9}	1×10^{-6}	10	2700	855.9	19.5	37114	Sparse	4h 40min
Plastic Shelf	4	2.49×10^{10}	1.28×10^{10}	1×10^{-6}	50	2700	488.9	29.7	361	Sparse	30sec
Aluminum Shelf	4	4.98×10^{10}	2.56×10^{10}	1×10^{-7}	0	2700	691.5	0.9	361	Sparse	30sec
Wood Shelf	4	5.00×10^{8}	1.00×10^{8}	8×10^{-6}	50	750	154.6	28.8	361	Sparse	30sec
Bunny (Metal)	–	4.99×10^{10}	2.56×10^{10}	1×10^{-7}	0	2700	855.9	19.5	37114	Sparse	4h 40min
Blocks	5	5.00×10^{8}	1.00×10^{8}	8×10^{-6}	50	550	1596.2	428.1	1160	Dense	5h 28min
Boxes	5	5.00×10^{8}	1.00×10^{8}	8×10^{-6}	50	550	159.1	49.0	1160	Dense	5h 28min
The End (T)	6	1.49×10^{9}	7.70×10^{8}	2×10^{-7}	30	2700	247.7	15.2	71	Dense	42sec

Table 1: This table lists parameters that were used for each example object, the resulting frequency and decay for the object's primary mode, the number of nodes in the tetrahedral mesh, the method used for the modal decomposition, and the amount of time required to compute the decomposition. Once the model decomposition has been computed, all of the above examples can generate audio in real-time. For "Chimes" and "The End," the information listed is for the D3 tube and the letter T.

	Ideal	Measured		Computed	Adjusted	
Note	Freq.	Length	Freq.	Freq.	Length	Freq.
D3	587.33	.505	585.8	589.17	.5061	587.40
E3	659.26	.475	656.0	665.03	.4770	659.27
G3	783.99	.435	781.8	787.01	.4366	784.06
A4	880.00	.410	877.5	884.70	.4115	879.36
B4	987.77	.388	982.5	984.75	.3878	987.32
D4	1174.66	.353	1167.0	1186.88	.3548	1174.67

Table 2: The notes and ideal frequencies listed indicate the values specified by the manufacturer of the real wind chimes. The measured values were taken from the real wind chimes. The computed frequencies are what our model produced using the parameters from table 1 and the measured lengths. The adjusted values indicate the length and resulting frequency of the simulated chimes after tuning. Lengths are in meters and frequencies in Hertz.

figure 4 also shows the results of projecting a pair of impulses onto the retained modes of the bunny model.

The blocks and boxes shown in figure 5 illustrate how scale can effect the resulting audio. Both the boxes and blocks are geometrically similar: hollow cubes with a wall thickness of 5% their width. However, the boxes are $10\times$ the size of the blocks. While the different scales are subtly revealed by the rigid-body motions (by the rate of acceleration with respect to the object sizes), the sounds produced by the two sets of objects are distinctly different, and the difference provides a clear cue as to the size of the objects.

As we discussed previously, similarities exist between the approach we have presented here and that presented in [van den Doel et al., 2001]. The main difference between the two methods is that we synthesize audio from only geometry and material properties whereas their system makes use of extensive measurements of a given object's response to impacts. Each of these methods presents distinct advantages: by relying on recorded data their method may easily match a given object, but our method is applicable when no real object or no robotic measuring devices are available. One direction that might be worth pursuing would be using their measured data for a given object to infer material parameters that could then be applied to the geometry of a different object. This approach might allow audio models for an entire set of cooking pots, for example, to be generated from measurements of a single pot in the set. It might also allow us to determine the sound made by a novel bell design, based on data from bells of similar materials, before we actually make the bell. Based on the good correspondence between our synthetic chimes and the physical set, we are optimistic about this direction of future work.

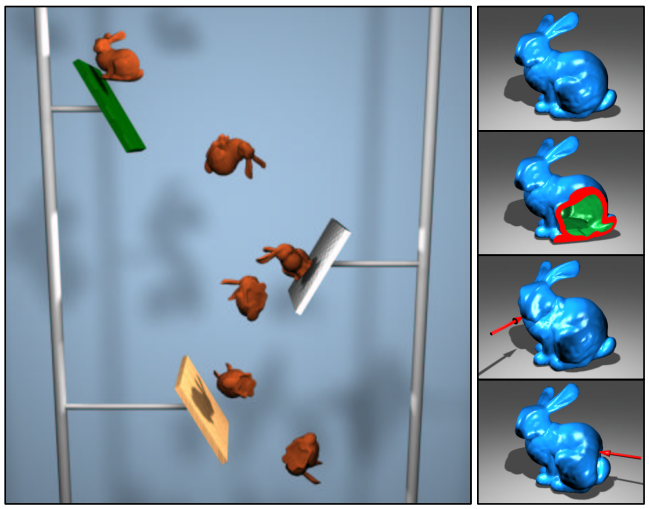

Figure 4: The left side of this figure shows an image from animation of several bunnies falling through a chute. Both the bunnies and the shelves are sounding. The images on the right show in order from top to bottom: the exterior of the bunny model, a cut-away revealing the wall thickness and hollow interior, modal response to an impulse on the bunny's nose, and the modal response to an impulse on the bunny's back. The impulse responses are greatly exaggerated for illustration.

Although the resolution of the mesh can affect the resulting audio, we have found that even very coarse meshes may be used for generating acceptable results. The meshes used for each of the letters shown in figure 6 are very coarse, yet the resulting audio is still acceptable. We have found that low mesh resolution tends to shift frequencies higher and may add a "hollow" quality to the sound. The frequency shifting may be partially compensated for by simply modifying the material parameters (*e.g.* raising the density) to compensate, so it will only be a problem if one is attempting to match a particular object (as we were for the wind chimes).

Although the modal decompositions may require up to a few hours of computation, this work needs only to be done once for a given object and audio can then be generated interactively. By precomputing the modal decomposition and storing it with an object, the approach we have presented could easily be applied to interactive applications such as video games that already employ rigid-body simulation methods. Additionally, because our method

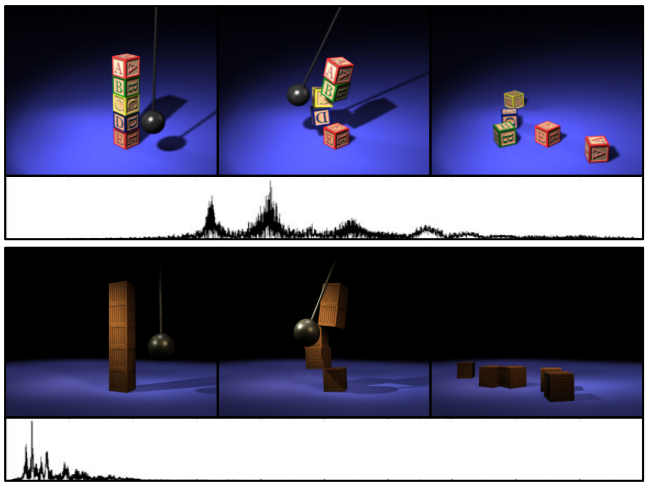

Figure 5: The top images show a stack of 5 cm blocks being knocked over. The images on the bottom show a stack of 50 cm boxes being knocked over. Only the blocks and boxes are sounding. Other than the $10\times$ scale, both models are identical. The plots below each sequence show the frequency content of the resulting audio, indicating a significant difference in the sounds. (The horizontal axis ranges from 0 to 5000 Hz, the vertical axes are autoscaled independently.)

requires only a geometric model and a handful of material parameters, the extra effort required to generated the audio model of a given object is minimal.

Acknowledgments

The authors thank Okan Arikan for allowing us to use his rigid body simulation code, and the members of the Berkeley Graphics Group for their helpful criticism and comments. This work was supported in part by Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, a research grant from the Okawa Foundation, and NSF grant CCR-0204377.

References

ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MAKENNEY, A., AND SORENSEN, D. 1999. *LAPACK Users' Guide*, third ed. Siam, Philadelphia.

BAI, Z., DEMMEL, J., DONGARRA, J., RUHE, A., AND VAN DER VORST, H., Eds. 2000. *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia.

COOK, R. D., MALKUS, D. S., AND PLESHA, M. E. 1989. *Concepts and Applications of Finite Element Analysis*, third ed. John Wiley & Sons, New York.

COOK, P. R. 2002. *Real Sound Synthesis for Interactive Applications*. A. K. Peters, Ltd., Natick, Massachusetts.

FUNKHOUSER, T., CARLBOM, I., ELKO, G., PINGALI, G., SONDHI, M., AND WEST, J. 1998. A beam tracing approach to acoustic modeling for interactive virtual environments. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 21–32.

FUNKHOUSER, T., MIN, P., AND CARLBOM, I. 1999. Real-time acoustic modeling for distributed virtual environments. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 365–374.

HAHN, J., GEIGEL, J., LEE, J., GRITZ, L., TAKALA, T., AND MISHRA, S. 1995. An integrated approach to sound and motion. *Journal of Visualization and Computer Animation 6*, 2, 109–123.

KINSLER, L. E., FREY, A. R., COPPENS, A. B., AND SANDERS, J. V. 2000. *Fundamentals of Acoustics*, fourth ed. John Wiley & Sons, New York.

MIN, P., AND FUNKHOUSER, T. 2000. Priority-driven acoustic modeling for virtual environments. *Computer Graphics Forum 19*, 3 (Aug.).

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 137–146.

O'BRIEN, J. F., COOK, P. R., AND ESSL, G. 2001. Synthesizing sounds from physically based motion. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 529–536.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of SIGGRAPH 89*, Computer Graphics Proceedings, Annual Conference Series, 215–222.

SAVIOJA, L., HUOPANIEMI, J., LOKKI, T., AND VÄÄNÄNEN, R. 1997. Virtual environment simulation - advances in the DIVA project. In *Proceedings of the International Conference on Auditory Display (ICAD)*, 43–46.

TAKALA, T., AND HAHN, J. 1992. Sound rendering. In *Proceedings of SIGGRAPH 92*, Computer Graphics Proceedings, Annual Conference Series, 211–220.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Deformable models. *The Visual Computer 4*, 6, 306–331.

TSINGOS, N., FUNKHOUSER, T., NGAN, A., AND CARLBOM, I. 2001. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 545–552.

VAN DEN DOEL, K., AND PAI, D. K. 1996. Synthesis of shape dependent sounds with physical modeling. In *Proceedings of the International Conference on Auditory Display (ICAD)*.

VAN DEN DOEL, K., AND PAI, D. K. 1998. The sounds of physical shapes. *Presence 7*, 4, 382–395.

VAN DEN DOEL, K., KRY, P. G., AND PAI, D. K. 2001. Foley automatic: Physically-based sound effects for interactive simulation and animation. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 537–544.

WU, K., AND SIMON, H. 1999. TRLAN user guide. Tech. Rep. LBNL-42953, Lawrence Berkeley National Laboratory.

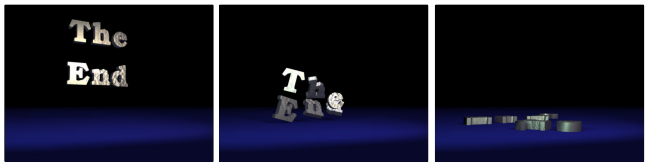

Figure 6: Select frames from an animation of the words "The End" falling onto a hard surface. Both the letters and the surface are sounding.