Volume 20, (2001), Number yyy pp.

Fast and Controllable Simulation of the Shattering of Brittle Objects

Jeffrey Smith and Andrew Witkin[†] and David Baraff[‡]

Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A.

Abstract

We present a method for the rapid and controllable simulation of the shattering of brittle objects under impact. An object is represented as a set of point masses connected by distance-preserving linear constraints. This use of constraints, rather than stiff springs, gains us a significant advantage in speed while still retaining fine control over the fracturing behavior. The forces exerted by these constraints during impact are computed using Lagrange multipliers. These constraint forces are then used to determine when and where the object will break, and to calculate the velocities of the newly created fragments. We present the details of our technique together with examples illustrating its use.

An earlier version of this paper was presented at Graphics Interface 2000, held in Montreal, Canada.

Keywords: Physically based modeling; computer animation; impact; brittle materials

1. Introduction

Realistic animation of breaking objects is difficult to do well using the traditional computer animation techniques of hand modeling and key-framing. This difficulty arises from the fact that the breaking of an object typically creates many small, interlocking pieces. The complexity and number of these fragments makes modeling them by hand all but impossible, but the distinctive look of a shattered object prevents the use of simple short-cuts, such as slicing the surface of an object into faces or the use of RenderMan shaders.

Consequently, the simulation of breaking and shattering has received some attention within the graphics community. An early attempt at modeling fracture is given in Terzopoulus and Fleischer[?], where they presented a technique for modeling viscoelastic and plastic deformations. While not specifically intended to model the breaking of brittle objects, their work allowed the simulation of tearing cloth and paper with techniques that could have be applied to this task. In 1991, Norton et al. [?] described a technique specifically for modeling the breaking of three-dimensional objects wherein the object to be broken was subdivided into a set of equally sized cubes attached to one another with springs. Unfortunately, their use of an elastic network invited massive computational expense for large objects.

Most recently, O'Brien and Hodgins[?] used continuum mechanics techniques developed in mechanical and civil engineering to model the behavior of brittle fracture, including crack initiation and propagation. Although their approach did not use an explicit lattice of springs attached to point masses, it was similarly computationally expensive. Specifically, their use of Euler and second-order Taylor integrators restricted the time step of the physical simulation to extremely small values. Due to their re-meshing technique, the timing of the examples presented in their paper is difficult to directly compare with our own. As a rough comparison, their "wall #2" mesh, with a final total of 8275 elements, took an average of 1098 seconds of computation per simulation second on an 195 MHz R10000 processor. Running a similarly sized model (8421 tetrahedra with 16474 shared faces) with the same impact and fragmentation characteristics took roughly 90 seconds with our technique.

Unsurprisingly, the fields of condensed-matter physics and materials science have examined the topic of brittle fracture more thoroughly. Note that the term "brittle," as used in materials science literature and this paper, means that a substance does not undergo significant plastic (reversible) de-

[†] Now at Pixar Animation Studios

[‡] Now at Pixar Animation Studios

formation before breaking (see, for example, Cherepanov[?]). That is, a brittle object will not bend much under stress, but will either resist almost completely or break catastrophically. Unfortunately, the simulation methods used in these fields are often inappropriate for graphics applications. In general, materials scientists are interested in predicting with great accuracy how and when an object will shatter, whereas a computer animator is more concerned with generating realistic-looking behavior in a reasonable amount of time. Consequently, the level of physical detail used in materials science simulations is much higher, and often of a different nature, than we require.

One commonly used approach in this field is the lattice model (Arabi[?], Chung[?], Donze[?]). This method, similar to that used by Norton et al. [?], models objects as a lattice of points or point masses connected by stiff springs. During simulation, the extension of each spring, or some other potential function of the particle displacements, is computed. Depending on the model, either every spring exceeding its extension or potential limit, or only the most egregious violator, is removed. The state of the system is cleared and the process repeated until the object falls apart, or until no new elements are being broken.

A significant disadvantage of material science lattice solutions is that they almost universally use three-dimensional systems of stiff springs combined with explicit numerical integration methods. The step sizes for this type of simulation must be on the order of the inverse of the speed of sound in the material being simulated, and thus the computational expense can be high. (Even with the use of implicit integration methods, the simulation of a lattice of stiff springs entails a computational cost as least equal to that of Lagrange multipliers.) For example, in Chung[?], the simulation (with an explicit integration method) of an object with 2701 lattice-links is done in 137 time-steps, each taking 86 seconds on a 75MHz MIPS R8000, for a total of three and a half hours. A comparable simulation with our method on the same hardware would take roughly five and a half minutes.

In this paper, we present a fast and controllable method for simulating the fracture of brittle objects for animation. This method differs from the majority of the reviewed literature in that represent the object as a system of point masses connected by workless, distance-preserving constraints, rather than as a lattice of stiff springs. Our use of rigid constraints follows from an abstraction of brittle material properties and allows us to solve for the forces exerted by these elements during impact much more quickly than using explicit methods and an elastic mesh. We compute our solution by constructing a large, sparse, linear system which we solve using a conjugate gradient descent method. The constraint forces, once calculated, indicate when and where the object will break. This information is then used to construct the fragments of the broken object from the original geometry and to solve for the final linear and angular velocities of these

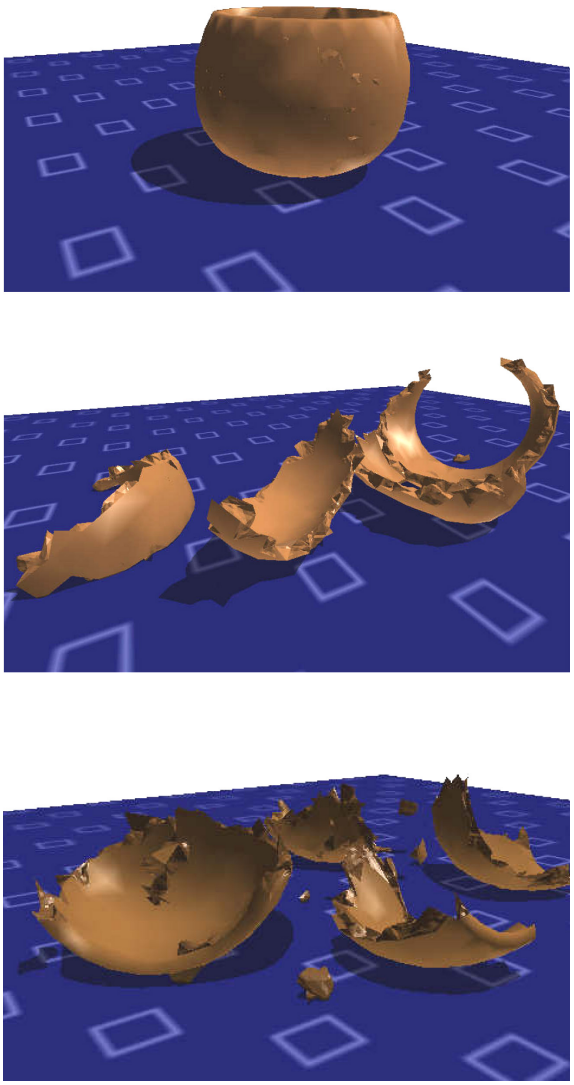

Figure 1: *Glazed ceramic bowl, before (top) and after being broken with two different constraint-strength distributions*

bodies. In addition to advantages of speed, our system allows the user to easily control the size, shape and number of fragments while still yielding realistic-looking output, making it well-suited for use in animation.

2. Modelling

As mentioned above, our method is roughly based on the elastic networks common in material science literature. However, instead of a three-dimensional mesh of springs we use a lattice of rigid constraints to connect point masses. We are motivated in this choice by both speed considerations and the nature of ideal brittle materials.

Consider the naive system of points and springs. Because we are simulating brittle objects, these springs must be very stiff: stiff enough that no visible flexing (plastic deformation) takes place during high-momentum impact. As the brittleness of an object increases, the stiffness of these springs increase, and the displacements they undergo during impact decrease. In the limit, then, for an ideally brittle material we would be forced to model springs which are infinitely stiff and undergo infinitesimal displacements.

Instead, we idealize these stiff springs as distance-preserving constraints and, rather than calculating displacements, calculate the forces that these constraints exert in response to an applied impulse. Our use of distance-preserving constraints allows a faster method of solution (discussed in section **??**) than the use of explicit numeric integration, while retaining both realism and a large amount of user control.

2.1. Constructing the Model

The first step in our simulation is the construction of a solid model — consisting of a collection of simple polyhedra — from the initial description of the object to be shattered. Since this initial description is usually a set of vertices and faces describing only the surface, we first add a number of well-distributed vertices inside this surface. As a rule of thumb, we generally add enough vertices (uniformly distributed) to triple the total number in the model. The user can vary the number and distribution of these added vertices in order to affect the fracturing behavior of the object by changing the shape and size of its potential fragments.

After the addition of internal vertices, we would like to perform a constrained Delaunay tetrahedralization; that is, one such that exterior faces of exterior tetrahedra correspond to the faces in the original surface description. However, since constrained tetrahedralization is a very difficult problem, we cannot always accomplish this goal. If our constrained tetrahedralization fails, we fall back to a non-constrained tetrahedralization (using the freely available package *qhull*[?]) followed by post-processing to clean

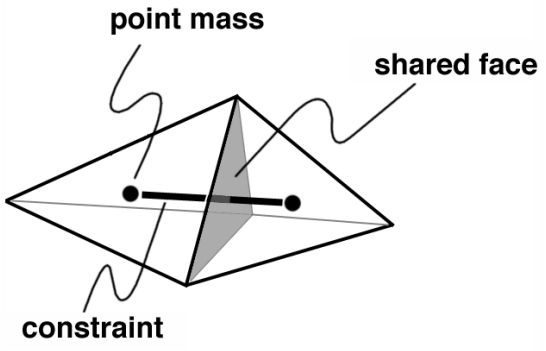

Figure 2: *Two tetrahedra and their point/constraint complement*

up the model. This clean-up is required because a non-constrained tetrahedralization will (unless the object is convex) result in tetrahedra that extend beyond the surface of the original model. We must therefore test every tetrahedron to see if it extends significantly beyond the boundary of the original surface. We accomplish this check using two tests:

1. We test to see if the centroid of any tetrahedron lies outside the surface, discarding any which do.
2. We then test the remaining tetrahedra to see if any of their faces are outside the surface. Specifically, we test if the center of a face lies too far outside the original surface mesh. "Too far," in this case, is defined as more than one quarter of the smallest dimension of the tetrahedra under question.

After removing all surface-crossing tetrahedra, we transform this new, solid model into a lattice representation — our final model. This lattice representation is simply the Voronoi complement (or "dual") of the solid object, which represents the tetrahedra as point masses and their connections (shared faces) as rigid constraints (figure **??**). Each point mass is placed where the center of the tetrahedron it represents was located and the mass of each point is determined by the volume of the tetrahedron and the density of the object in that volume.

The rigid constraints connecting these point masses have an associated ultimate strength which corresponds physically to the strength of the bond between two "micro-fragments" of the original object. If the tensile or compressive forces across a constraint exceeds this strength then the bond will be broken. The breaking strength of a constraint is determined though a combination of user-specified functions and a pair of simple heuristics based upon the the geometry of the two tetrahedra the constraint is "gluing" together. In our model, the strength of a constraint scales with the volume of the connected tetrahedra and with the area of their shared face. By relating the strength of a constraint to the size and shape of the tetrahedra it connects, the breaking

behavior will be influenced by the geometry of the object, which is necessary for physically realistic results.

In addition to these simple geometry-based heuristics, the user may add a procedural variation to the constraint strengths. For example, simple cleaving planes may be added by systematically reducing the constraint strength along a cross-section of the object, or nodes of great strength may be created which will result in these regions remaining intact after the rest of the object is shattered. We have also achieved good results using noise and turbulence functions, as described in Perlin[?]. Much of the flexibility of our model comes from an appropriate choice of the function that determines the constraint strengths.

3. Simulation

Our approach to the simulation of fracture is a simple one, intended to avoid the computational expense and complexity of a full dynamic simulation while preserving physical realism. Although the time course of impacts can be as little as 100 microseconds, the speed of sound is brittle materials is typically several thousand meters per second (5100 meters/second in common glass and between 3500 and 7000 meters/second in hard stone). Given that the objects we wish to shatter are of moderate size (usually on the order of 10 cm on a side), the time to equilibrate internal forces (transmitted at the speed of sound) is on the order of one microsecond. Because the duration of a typical impact is so much longer than the time it takes the internal stresses to reach equilibrium, we make a quasi-static loading approximation, and can safely use global solution methods to calculate the forces between elements of the solid.

3.1. Fundamentals of the Simulation

We formulate the problem of calculating the forces being exerted by the rigid constraints as one of solving for Lagrange multipliers in the following simplification of the constraint force equation (For a derivation of this equation, see Witkin and Baraff[?] or Witkin, Gleicher and Welch[?]):

$$JWJ^T\vec{\lambda} = -JWQ \tag{1}$$

where W is the inverse mass matrix and Q is the global force vector, containing information on what forces are being exerted on which particles by the impact. The matrix J is defined as

$$J = \frac{\partial C}{\partial p}$$

where C is the "constraint vector": a vector of functions — one for each constraint in the system — whose values are zero if the constraint is satisfied and non-zero otherwise. If we wish to introduce prior material stresses, the initial constraint vector may be given non-zero entries and equation **??** becomes

$$JWJ^T\vec{\lambda} = -JWQ - kC$$

where k is some unit-normalizing factor.

Each constraint function is of the form

$$C_i(p_a, p_b) = \|p_a - p_b\| - d_i$$

where p_a and p_b are the locations of the two particles connected to constraint i, and d_i is the length of the constraint.

The vector $\vec{\lambda}$, which is found by solving equation **??**, contains the values of the forces being generated by each distance-preserving constraint and is used to determine which constraints should be broken. Specifically, if a constraint is found to be exerting a force greater than its strength, it is removed. It should be noted that intergranular bonds in brittle materials are eight times stronger under compression than during extension[?], and this must be accounted for in our breaking decision-rule.

With $\vec{\lambda}$ in hand, it is a simple matter to calculate $\hat{Q}$:

$$\hat{Q} = J^T\vec{\lambda} \tag{2}$$

which is the vector containing the forces being exerted by each particle in reaction to the applied forces, Q. The values of $\hat{Q}$ are added to this global force vector, giving us the total forces being experienced by each particle in our simulation. These force values can be used for physical simulation of the aftermath of fracture or, if we are using a multiple-step solution (see section **??**), as the input for another iteration of our algorithm.

3.2. Physically Realizable Solutions

The system that we are solving:

$$JWJ^T\lambda = b \tag{3}$$

is under-constrained in the sense that for a given b, there are any number of $\vec{\lambda}$'s which satisfy the equation. However, an arbitrary vector $\vec{\lambda}$ does not necessarily correspond to a physically realizable set of constraint forces between connected particles. Given this fact, how can we be certain that our solution to equation **??** is the physically realizable one?

First we note that those solutions which are physically meaningful have a particular structure. Consider again the connections between particles to be stiff springs. In this case, the only internal forces that can arise are those that have been generated due to some displacement δp of the particles. These displacements in turn correspond to a vector of spring tensions $\vec{\lambda} = J\delta p$. We can therefore see that all physically realizable $\vec{\lambda}$'s can be written as $\vec{\lambda} = J\delta p$ for some displacement δp. (We could parameterize by δp, but our solution would still have to satisfy equation **??** and our system would be more complex.) Stated a different way, any physically realizable $\vec{\lambda}$ must lie within the column-space of J and thus also in the column-space of JWJ^T (regardless of J's rank; see Strang[?] for details).

Note, though, that any solution $\vec{\lambda}$ of equation **??** that lies

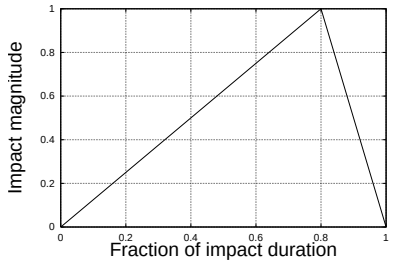

Figure 3: *Impact magnitude versus time*

in the column space of JWJ^T is a minimum-norm solution. Thus, physically realizable solutions are equivalent to minimum-norm solutions, and since the minimum-norm solution to a linear equation is unique (Strang?), so is the physically realizable solution. Therefore, a solution method that finds a minimum-norm solution of equation **??** is guaranteed to give us the unique physically realizable solution $\vec{\lambda}$.

We use the conjugate gradient descent method to solve for the minimum-norm solution of our system. Not only will it give us the correct solution, as shown above, but it exploits the sparsity of the JWJ^T matrix to give us fast solution times?.

3.3. Multiple-Step Solutions

It would appear that the simulation of an impact could be done with a single-step solution for $\hat{Q}$. However, our use of a global solution method would permit constraints to "transmit" forces of arbitrary strength before being removed, whereas we desire the constraints to be able to transmit no more force than their breaking strength would allow. Visually, a single-iteration solution results in the pulverization of a large volume surrounding the impact without the distinctive shards and fragments we desire.

Instead of a single iteration, however, we can solve for $\hat{Q}$ in multiple steps, increasing the impact force with each iteration. In this way we can slowly ramp up the magnitude of the impact so that we are certain that no constraint transmits a force greater, to within some ε, than its breaking strength. By gradually increasing the magnitude of the impact force, we are imposing a pseudo time-course upon our simulation. That is, rather than simulating an impact as a single, zero-time impulse, we are creating a more realistic impact history. For all examples given in this paper, we used the simple piecewise linear function shown in figure **??** as our impact schedule.

The time to equilibrate the forces within a brittle object is much less than the duration of the impact. Therefore, we can safely chop this duration into smaller segments without losing the ability to solve with a global method. In practice, we have found that between 10 and 50 iterations of this loop yields acceptable results. Increasing the number of iterations brings little or no change in the fracturing behavior.

3.4. Crack Growth

Another important feature of brittle fracture that we would like to capture in our simulation is the growth of cracks. In brittle materials, the energy required to start a new crack of length l is significantly higher than the energy required to lengthen an existing crack by the same distance (see, for example, Lawn?). This behavior is the major reason why glass — despite its material homogeneity — breaks into large, polygonal shards under impact rather than turning into a cloud of tiny fragments.

In order to encourage the growth of pre-existing cracks, we modify our multi-step algorithm. When we remove a newly broken constraint, we weaken the constraints around it that correspond to faces which adjoin the just-broken constraint. Thus, in the next iteration it is more likely that these constraints will break than constraints with an equal initial breaking-strength that are not connected to a pre-existing crack. Specifying the form of this function allows the user to control the desirability of creating new cracks versus spreading existing flaws.

To illustrate this effect, three examples were generated using the same model — a simple rectangular plank — the only difference between the simulations being the crack growth function used. The model used contained 3962 tetrahedra with 7096 shared faces. Constraint strengths varied between 90.2 and 541.0, having been generated with a combination of a turbulence function and the geometric heuristics described in section **??**. Although these objects all have the same initial geometry and constraint values and are broken with the same impact, significantly different results were produced. Figure **??** shows (from above) the aftermath of this test solid being fractured with no crack growth function. Here, dark lines show the edges of the top-facing tetrahedral faces and white lines indicate crack boundaries. We can see from this picture that the cracks which resulted in the fragmentation of this object have not spread far beyond the immediate impact location (the tip of the triangle). This object is shown in 3D in figure **??**, with the different fragments assigned varying colors.

Figures **??** and **??** show the results of the same test, but with a crack growth function that reduces constraint strengths by up to a factor of two. Specifically, the function

$$s^i_{new} = s^i_{old}(1.0 - \sin(2\theta + \frac{\pi}{2}))$$

where θ is the angle (between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$) between some constraint broken in the previous time-step and the neighboring constraint i. The values s^i_{old} and s^i_{new} are the old and new constraint strengths respectively. We can see that this crack growth function has encouraged the creation of more frag-

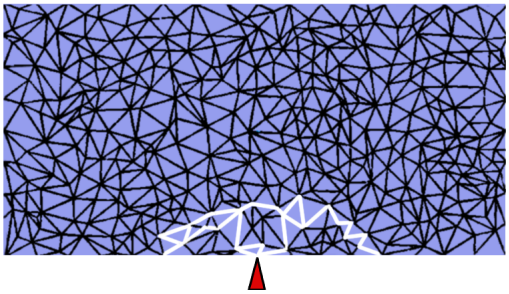

Figure 4: *Top view of broken plank, showing cracks between tetrahedra*

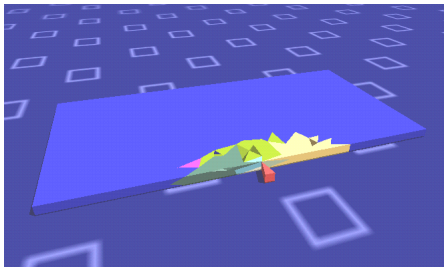

Figure 5: *Broken plank, showing fragments resulting from impact with no crack growth*

ments, and has permitted parts of the object further from the impact site to break.

Finally, figures **??** and **??** show the results of our test object being broken again, but with a more extreme crack growth function. This function reduces constraint strengths by up to a factor of 1000:

$$s^i_{new} = s^i_{old}(0.5005 - 0.4995\sin(4\theta + \frac{\pi}{2})) \qquad (4)$$

Not surprisingly, cracks have propagated deeply into the

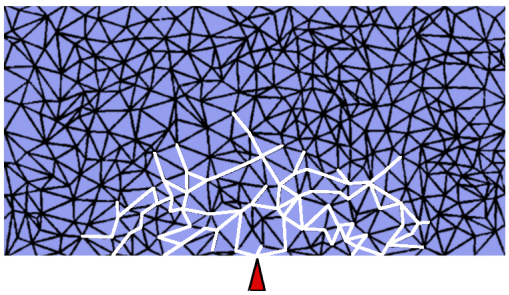

Figure 6: *Top view of broken plank, showing moderate crack growth*

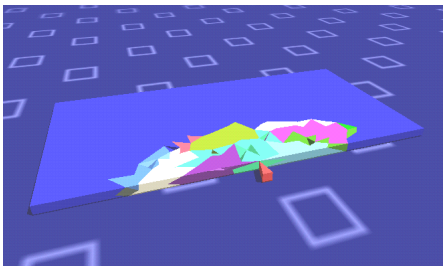

Figure 7: *Broken plank, showing fragments resulting from impact with moderate crack growth*

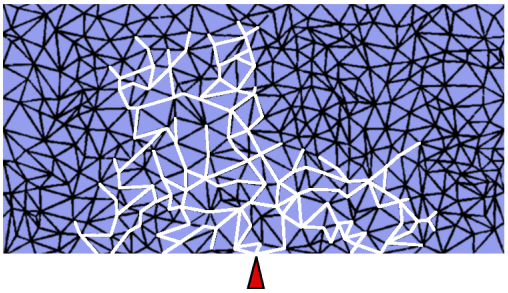

Figure 8: *Top view of broken block, showing dramatic crack growth*

solid and have caused it to break into many more pieces. As can be seen from these three examples, even simple changes in the crack growth function can significantly alter our results, allowing the user further control over the material properties of the object.

3.5. Using the Results

The simulation outlined above produces as output a new point/constraint model, consisting of the same point masses as in the input, but with fewer constraints. Using a region-coloring algorithm, it is a simple matter to determine the

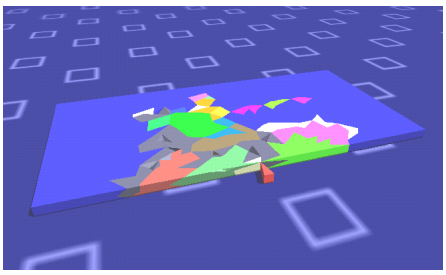

Figure 9: *Broken plank, showing fragments resulting from impact with major crack growth*

connectivity of this new model, and to label the separate fragments. Since each point mass in this model corresponds to a tetrahedron in the original solid model, we can then easily construct the set of solids (made of tetrahedra) which are the fragments of the original object.

At this point we may have, in addition to large pieces of the broken object, many hundreds (or thousands, if the model is large enough or the impact violent enough) of fragments that consist of only one tetrahedron. These tiny fragments, which we refer to as "dust," are generally discarded in order to speed up the physical simulation of the aftermath of the impact. Specifically, we calculate the volume of each fragment and discard those that consist of only one tetrahedron and whose volume is below some threshold. This clean-up can be done with minimal impact on the realism of the final solution, since individual tetrahedra should be quite small in comparison to the size of the original object.

The final step in our algorithm is to calculate the velocities (both linear and angular) of the remaining fragments. The final forces exerted on each point mass in the point/constraint model reside in $\hat{Q}$, calculated by equation **??**. The forces on point masses correspond to forces on the centroids of each tetrahedra in the solid. Thus, given more than two tetrahedra in a fragment, we can easily calculate the linear and angular velocities of that solid:

Given a point i, we know that

$$\dot{p}_i = v_i + \omega_i \times p_i$$

where $\dot{p}_i$ is the velocity of the point, v_i is the strictly linear velocity, ω_i is the angular velocity and p_i is its position. Thus, if we have a solid comprised of three or more tetrahedra with centroids p_0, p_1 and p_2, we can separate v_{solid} from ω_{solid} by solving the following simultaneous equation:

$$\begin{bmatrix} I & -p_0^* \\ I & -p_1^* \\ I & -p_2^* \end{bmatrix} \begin{bmatrix} v_{solid} \\ \omega_{solid} \end{bmatrix} = \begin{bmatrix} \dot{p}_0 \\ \dot{p}_1 \\ \dot{p}_2 \end{bmatrix}$$

where I is the 3 by 3 identity matrix and p^* is the dual (or "cross") matrix:

$$\begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix}$$

With these velocities in hand, we can perform a dynamic physical simulation to produce an animation of the aftermath of shattering.

3.6. Display Issues

As noted earlier, a significant problem with the display of fractured objects is that the underlying polyhedral mesh (in our case, tetrahedra) can be distractingly apparent. The simplest solution to this problem, of course, is to more densely sample the initial object, creating a larger number of smaller tetrahedra. However, using a higher resolution tetrahedralization quickly increases the computational expense (see figures **??** and **??**). However, there is no need to use precisely the same data for the physical simulation and for the display. Similar to Norton et al. [?], wherein the cubic nature of the solid "cells" was partially masked using splines, we can add visual detail to our fracture surfaces by fractally subdividing the display geometry (see figure **??**). This subdivision process is done on a shared geometry database so that "touching" faces on different fragments will still fit together properly. A better technique, and one that we plan to investigate in the future, would be a multi-resolution simulation combined with re-meshing, such as that described in O'Brien and Hodgins[?].

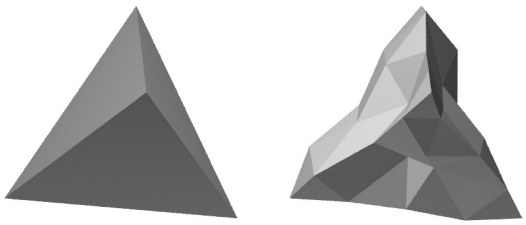

Figure 10: *Original and fractally subdivided tetrahedra*

4. Results and Discussion

We have described a simple, physically motivated model for the rapid simulation of brittle fracture. The following examples illustrate the output of our work and demonstrate some of the different fracturing behaviors and material properties that can be simulated.

4.1. Wine glass

The examples shown in figure **??** were generated from the same geometric data: a wine glass modeled as 3422 tetrahedra with 6447 shared faces. Differences in fracturing behavior were produced by changing the function that determined the strengths of the constraints. More specifically, the constraint strengths were determined by a combination of a thresholded turbulence function and the geometric heuristics outlined in section **??**. Each glass was broken with a single impact at the point where it struck the floor after falling.

4.2. Clay pot

Figure 1 shows before and after images of a pot, made from glazed earthenware. This model was constructed from 6902 tetrahedra, with 13150 shared faces. In the middle image, the initially homogeneous constraints were alternately strengthened and weakened along the vertical axis. This variation

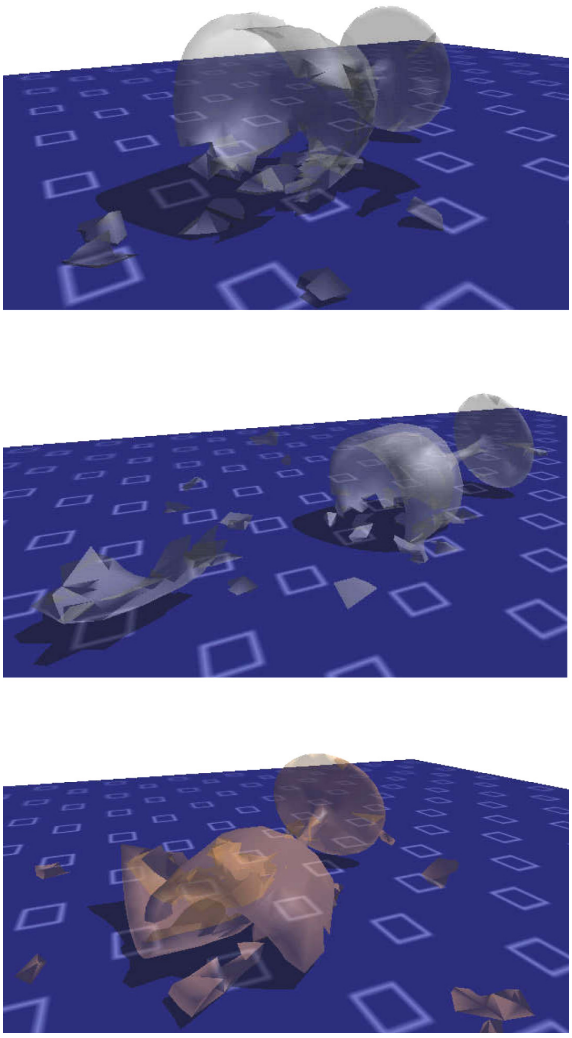

Figure 11: *Three broken wine glasses, demonstrating different fracture behavior*

yields the characteristic breaking behavior of pottery created without a wheel out of a single coil of clay. The lower image shows the same geometric model, but with constraints modified only by our geometric heuristics and a mild turbulence function, which yields a very different set of fragments.

4.3. Glass table

Figure **??** shows a sequence of four images of a ceramic bowl, sitting on a thick glass table, broken by the impact of a falling bowling ball. For this example, the strength of the constraints in both broken objects (the bowl and the tabletop) were initially homogeneous and modified only by our

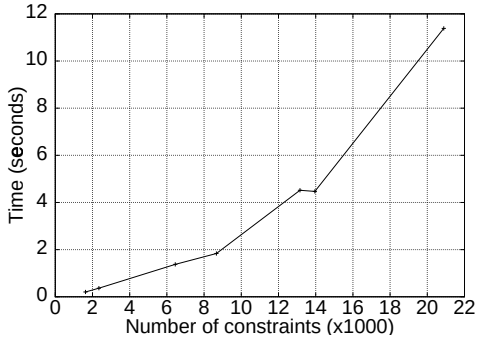

Figure 12: *Time per impact step versus total number of constraints*

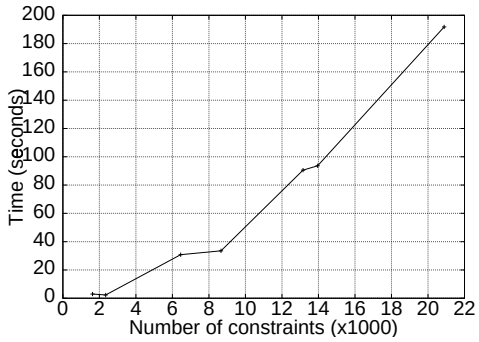

Figure 13: *Time required to construct fragments versus total number of constraints*

simple geometric heuristics. The crack growth function used in the table was that described in equation **??** which contributed to the formation of the long, narrow glass-like fragments.

4.4. Timing

Two major steps are involved in the destruction and subsequent animation of a shattered object: the impact calculation, and the reconstruction of the new fragments' surfaces afterwards. Figure **??** shows the amount of time required for each impact step calculation as a function of the the number of constraints (shared faces) in the lattice model. (All timing was done on a 195 MHz R10000 SGI Octane.)

As can be seen, even for relatively large objects the impact simulation is computationally inexpensive. Since we are repeatedly solving a sparse linear system with the conjugate gradient method, our computational cost is $O(m\sqrt{k})$, where m is the number of non-zero entries in JWJ^T, and k is the ratio of the largest to the smallest eigenvalues of this matrix[2].

Reconstruction of the fragments after impact requires similarly few resources. Figure **??** shows the time required to construct the surfaces and velocities of the new fragments

after impact. These times could be significantly reduced by the use of a more efficient algorithm for separating the fragments from one another. Even with this inefficiency, however, the total time required to break and reconstruct models of moderate size (several thousand constraints) is only a few minutes.

5. Conclusion

We have presented a fast and controllable method for the simulation of the shattering of brittle objects. By framing the problem in terms of distance-preserving constraints rather than stiff springs, we have avoided expensive explicit solution methods while retaining physical accuracy. Furthermore, our method allows simple control over the ultimate number, size and shape of the fragments by adjusting the strength of the constraints throughout the body or by changing the nature of crack growth within the material. Combined with speed and accuracy, this controllability makes our method useful for the otherwise difficult task of animating complex, realistic shattering.

References

1. S. Arbabi and M. Sahimi. Elastic properties of three-dimensional percolation networks with stretching and bond-bending forces. *Physical Review B*, 38(10):7173–7176, 1988.
2. C.B. Barber, D.P. Dobkin, and H.T Huhdanpaa. The quickhull algorithm for convex hulls. In *ACM Transactions on Mathematical Software*, 1996.
3. G.P. Cherepanov. *Mechanics of Brittle Fracture*. McGraw-Hill, 1979.
4. J.W. Chung, A. Roos, and J. Th. M. De Hosson. Fracture of disordered three-dimensional spring networks: A computer simulation methodology. *Physical Review B*, 54:15094–15100, 21.
5. F. Donze and S.-A. Magnier. Formulation of a 3-d numerical model of brittle behavior. *Geophysical Journal International*, 122(3):709–802, 1995.
6. A.A. Griffith. The theory of rupture. *The Proceedings of The First International Congress of Applied Mechanics*, 1924.
7. B. Lawn. *Fracture of Brittle Solids*, chapter one: "The Griffith concept". Cambridge University Press, 1993.
8. A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *Visual Computing*, 7(4):210–219, 1991.
9. J. O'Brien and J. Hodgins. Graphical modeling and animation of brittle fracture. *SIGGRAPH 99 Conference Proceedings*, 33:287–296, 1999.
10. K. Perlin. An image synthesizer. *SIGGRAPH 85 Conference Proceedings*, 19(3):287–296, 1985.
11. Jonathan R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, Aug. 1994.
12. G. Strang. *Linear Algebra and its Applications*. Harcourt Brace Jovanovich, 1988.
13. D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH 88 Conference Proceedings*, 22:287–296, 1988.
14. A. Witkin and D. Baraff. *Physically Based Modeling: Principles and Practice*, chapter Physically Based Modeling. SIGGRAPH Course Notes, ACM SIGGRAPH, 1997.
15. A. Witkin, M. Gleicher, and W Welch. Interactive dynamics. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, volume 24, pages 11–21, March 1990.

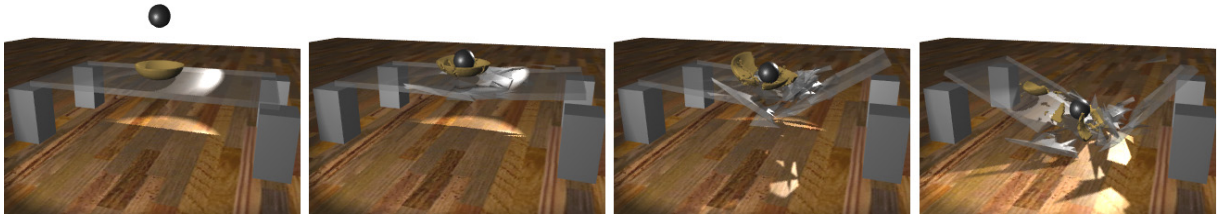

Figure 14: *A bowling ball is dropped onto a ceramic bowl that is sitting on a thick glass table.*

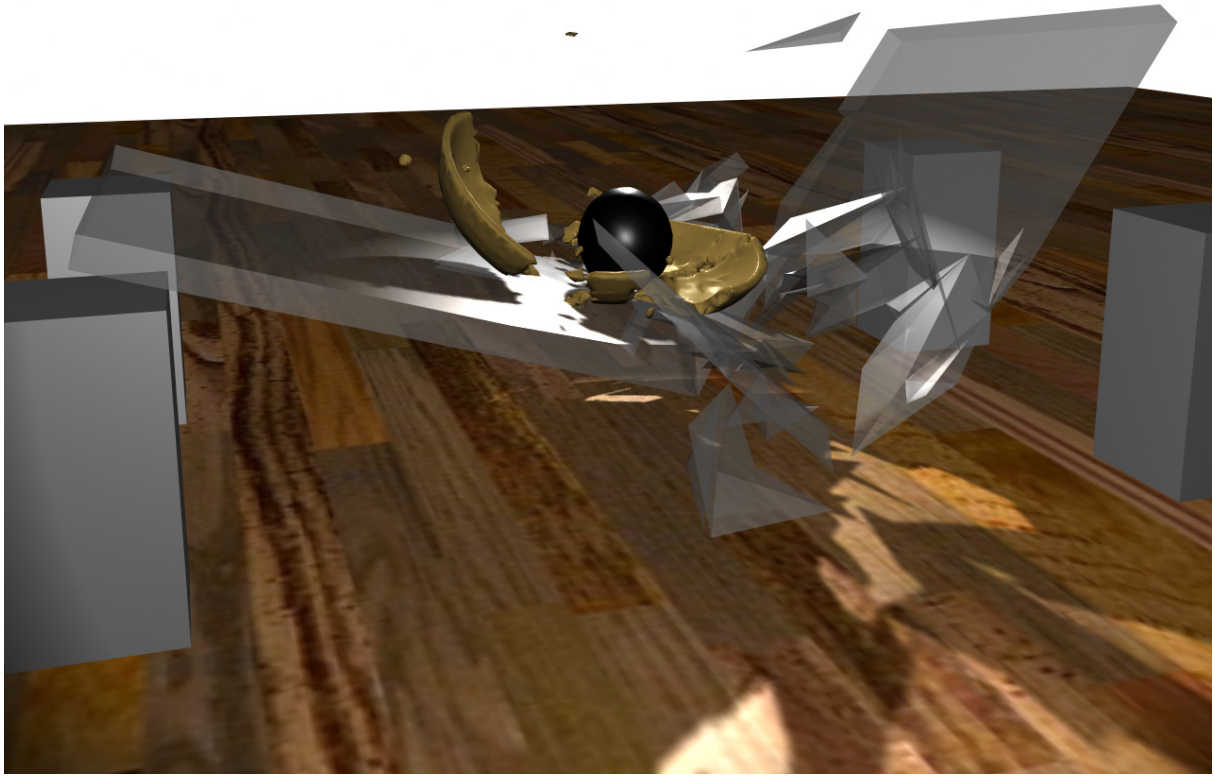

Figure 15: *A close-up of the breaking table and bowl*