The Visual Computer

Animation of fracture by physical modeling

Alan Norton, Greg Turk *,
Bob Bacon, John Gerth,
and Paula Sweeney

IBM T.J. Watson Research Center,
Yorktown Heights, NY 10598, USA

The breaking of solid objects, like glass or pottery, poses a complex problem for computer animation. We present our methods of using physical simulation to drive the animation of breaking objects. Breakage is obtained in a three-dimensional flexible model as the limit of elastic behavior. This article describes three principal features of the model: a breakage model, a collision-detection/response scheme, and a geometric modeling method. We use networks of point masses connected by springs to represent physical objects that can bend and break. We present efficient collision-detection algorithms, appropriate for simulating the collisions between the various pieces that interact in breakage. The capability of modeling real objects is provided by a technique of building up composite structures from simple lattice models. We applied these methods to animate the breaking of a teapot and other dishware activities in the animation *Tipsy Turvy* shown at Siggraph '89. Animation techniques that rely on physical simulation to control the motion of objects are discussed, and further topics for research are presented.

Key words: Modeling – Animation – Deformation – Collision detection – Simulation – Dynamics

* *Current address:* Department of Computer Science, University of North Carolina, Chapel Hill, NC 27514, USA

Introduction

Realism in computer animation depends on having objects move in a natural way. Computer simulation of physical behavior, or physically based modeling, has recently been successfully applied to various aspects of generating realistic motion. Rigid body simulation has been used to drive the animation of colliding objects (Hahn 1988). Flexible objects have been modeled as two- and three-dimensional meshes (Miller 1988; Platt and Barr 1988; Terzopoulos and Witkin 1988; Terzopoulos et al. 1987; Terzopoulos and Fleischer 1988) using either finite elements or springs.

In this paper we present a physically based model that was designed to animate solid objects breaking. The model we use for breakage of solids is similar to the 2D breakage model reported by Terzopoulos and Fleischer (1988). We model breakage as the limit of elastic behavior: when objects are sufficiently deformed, they will break at the place where the stress is greatest. Fractures therefore emerge dynamically as a natural result of stress accumulating and being released in breakage.

For such a model to succeed, we determined that it was necessary to handle a number of interrelated phenomena in a coherent way. Not only did elastic behavior and fracture need to be included in the model, but it was also necessary to incorporate a collision-detection/response model sufficiently robust to show realistic interaction between the broken fragments resulting from the breakage. Realistic object behavior requires other forces, such as gravity and friction. The materials we simulated needed to be molded into shapes appropriate to represent everyday objects, e.g., teapots, which could then be ray traced to show distortion and fragmentation of their structure.

We present a framework for physical modeling that meets these objectives. The physical behavior of solids is approximated by a three-dimensional grid. Masses are associated with vertices of the grid; interactions between neighboring vertices are obtained by simulating spring-like forces.

The individual components of this model are generally not unique; we have used various aspects of simulation, modeling, and collision detection previously presented (Miller 1988; Terzopoulos and Fleischer 1988; Hahn 1988; Moore and Wilhelms 1988). Our contribution is integrating a variety of established techniques in physically based modeling, showing how they can be applied together to achieve 3D breakage of brittle structures and other animation capabilities. In this process, we made pragmatic choices and learned lessons that

The Visual Computer (1991) 7:210–219

will be of value to others interested in realistic computer animation.
The last section of this paper discusses several major issues that need further work before flexible/breaking object simulation is an established computer-animation technique. We give suggestions for research directions and present the techniques we developed for animation control of flexible objects.

Physical modeling of solid structures

Dynamic solid models

To model realistic physical behavior by computer, it is necessary to establish a structure that will be used for the simulation. We define here a model that combines a geometric and dynamic description of objects and can be used to drive both simulation and computer graphics animation.
A dynamic solid model consists of a network or graph

$$N(t)=\{n_i(t), b_{ij}(t)\}$$

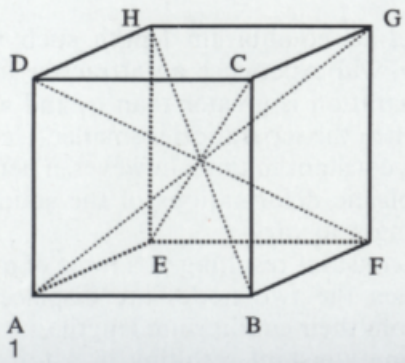
consisting of nodes $\{n_i(t)\}$ as the vertices of the graph and bonds $\{b_{ij}(t)\}$ as the edges of the graph. The network can vary as a function of time t by associating a number of time-dependent parameters with nodes and bonds. For discrete-time simulation, the values of such time-varying parameters are updated each time step.

Each node n_i has associated with it:
- Mass m_i.
- *Position* R_i, a 3-vector.
- Velocity V_i, another 3-vector.

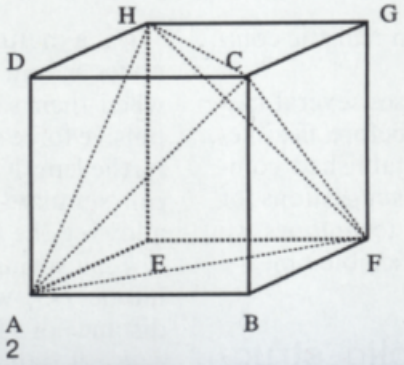
Both R_i and V_i are functions of time. These values are chosen to be consistent with a system of physical units; we have used centimeter-gram-second (cgs) units. One may optionally associate additional parameters with nodes, depending on the structure being simulated.
A bond b_{ij} connects node n_i with node n_j. We allow at most one bond between a given pair of nodes (n_i, n_j). Each bond has an associated force function F_{ij} representing the mutual attraction or repulsion associated with the pair (n_i, n_j). This force is derived from the physical model being used. For example, we compute the force from a number of scalar functions (of time) associated with bonds, including:
- l_{ij}, a natural or equilibrium length such that nodes n_i and n_j will experience an attractive force when their separation is greater than l_{ij} and a repulsive force when the separation is smaller. Usually, the length is constant in time; however, a person can simulate plastic deformations of the solid by allowing the length to vary.
- k_{ij}, a spring constant resulting in a force of magnitude rk_{ij} when the two nodes are displaced a distance of r from their equilibrium length.
- d_{ij}, a damping constant resulting in a force of magnitude vd_{ij} when the relative velocity of the two nodes is v. This can be used to dissipate internal vibrations in the material.
- A breaking threshold t_{ij}, which determines how much the bond is stretched before it "breaks." Breakage occurs when the ratio of stretched length to equilibrium length is more than $1+t_{ij}$. With large values of t_{ij}, the substance can be made unbreakable, and small values can cause it to crumble under slight deformation. The breaking thresholds can be randomly varied within a homogeneous solid so as to cause irregular fracture patterns.
- A Boolean variable s_{ij} indicates whether the given bond has been broken or not. A broken bond will not exert an attractive force when the nodes are separated beyond the equilibrium length, but continues to exert a repulsive force when the nodes become sufficiently close.

Geometry

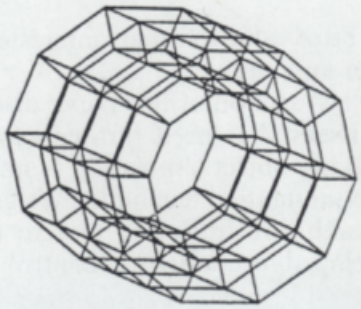
In the above definition of dynamic solid model no mention was made of geometry. To construct a dynamic solid model of a given object requires that both the macrogeometry (shape of the object) and microgeometry (or local internal structure) be modeled. Such a model could be regarded as a coarse approximation to the atomic structure of a solid. Computational limits force us to have nodes (atoms) representing volumes as large as a cubic centimeter of the material, and the choose bonds approximating the interaction between small (but not infinitesimal) nearby volumes.
We have chosen to start with a microgeometry based on a cubic lattice and use this structure as basic material for constructing larger objects. We have illustrated one cube in such a lattice in Fig. 1. The eight nodes associated with such a cube are labelled A, B, C, D, E, F, G, H. The 12 edges of the cube have associated bonds connecting each



1



2



3

4

5

6

7

Fig. 1. Cube with associated bonds. A spring is associated with each of the 12 edges of the cube as well as with each of the four internal diagonals

Fig. 2. Cube with face-diagonals. A spring is associated with each of the 12 edges of the cube as well as with a diagonal on each of the six faccs. Symmetry (but not more stability) can be obtained by associating springs with the other six face diagonals

Fig. 3. A cylinder is modeled with a lattice of deformed cubes

Fig. 4. A flexible teapot deforms as it collides with the floor

Fig. 5. A teapot lurches as if in response to a hiccup when gravity is increased and then suddenly decreased

Fig. 6. A teapot is propelled vertically as a consequence of tightening its springs

Fig. 7. A teapot breaks when it falls on its spout

vertex to three others; e.g., A is connected to B, D, and E. In addition, we introduce four bonds between diagonally opposite vertices occurring between A and G, B and H, C and E, D and F. These bonds on internal diagonals add rigidity to the structure, although the four diagonals shown here are not adequate to constrain all degrees of freedom of a cubic lattice. (There is a 3D set of deformations of the cube vertices that is opposed quadratically but not linearly by the 16 springs.) We also experimented using bonds on face diagonals in a configuration that does constrain all degrees of freedom, as shown in Fig. 2. The more stable configuration showed only minor differences in behavior under simulation.

Multiple cubes can be adjoined to form a lattice. One construction technique is to approximate any given shape with cubes in a lattice based on the Euclidean coordinate system. This method, however, results in staircase-like objects, and immense models would be required to mask grid effects. In order to construct objects of various shapes, we relax the requirement that the lattice elements be precise cubes, i.e., the edges are not required to be of equal length. We can distort the lattice elements so as to approximate nonrectilinear surfaces. For example, a cylinder can be constructed by rotating a set of squares about an axis. The resulting lattice elements are only approximately cubic, as shown in Fig. 3.

Dynamics

Suppose we are given an environment consisting of various geometric shapes realized as dynamic solid models. Assuming Newtonian mechanics, we compute the evolution of such a system in time by integrating Newton's second law, $F=ma$. All the masses in such a system are associated with nodes of the dynamic solid model. Newton's law requires that each node be accelerated according to the total forces applied to it. We have chosen to compute this solution by Euler's method: at each time step t and for each node n_i, the total applied force F is computed based on spring forces, gravity, friction, etc. This determines the acceleration $a=\frac{F}{m_i}$ of that node, resulting in a change in the velocity

$$V_i(t+\Delta t)=V_i(t)+a\Delta t.$$

The position is updated based on the current velocity

$$R_i(t+\Delta t)=R_i(t)+V_i(t)\Delta t.$$

Many forces must be included to model a realistic physical object. These include the interconnecting forces F_{ij} as required of a dynamic solid model. In addition, there are other forces outside the network structure, including gravity, friction, collision interactions, and forces imposed by objects external to the model.

We have used the following approximations for the forces in our model:

– Gravity. This is taken to be a constant downward force of magnitude $m_i g$, i.e., mass times the acceleration of gravity.

– Internal forces. The force of interaction between two adjacent nodes in the model is modeled after Hooke's law for springs. Suppose that the equilibrium distance between two nodes is D. The spring force between them is the sum $kX+dV$ where X is a vector in the direction of the difference in position between the nodes and has a magnitude equal to the difference between the current separation of the nodes and their original separation D. The spring constant k must be chosen appropriate to the stiffness of the material; as k becomes larger, the substance becomes stiffer Vector V is the component of the vector difference in the velocities of the two nodes in the direction of the vector X, and represents the rate at which the nodes are moving apart. It is multiplied by the damping constant d to procedure a force that tends to resist the continued relative motion. The damping force tends to dissipate the internal vibrations of the system and can be chosen appropriate for the material. [See Feynman (1965) for a general discussion of the use of spring forces in modeling the internal dynamics of materials.]

– External forces. We use another spring model to represent interactions with immovable objects, e.g., the upward force exerted force exerted by the floor. Such objects provide a repulsive force of the form $kX+dV$, where the vector X is taken to be the distance that the node intrudes into the immovable object and V is the velocity of that intrusion. There is no force applied if the node does not intrude. The spring constant in this case defines the stiffness of the immovable object and can be tuned to model for example the difference between a vinyl and concrete floor.

– Collisions. In addition to interacting with external objects, it is necessary to model the interaction when different chunks of dynamic solid model come in contact. If this force was not modeled, pieces of a breaking object would travel right through the object without interacting. Our collision model uses a variant of the penalty method: a repulsive (spring-like) force is applied when it is determined that two objects have interpenetrated one another. The collision-detection algorithms will be described in the next section.
– Friction. When two objects have been determined to intrude into one another, e.g., parts of a dynamic solid model or immovable objects, a force is introduced in the direction opposing lateral motion between them and proportional to the magnitude of the intrusion. (In physics friction between two rigid objects ordinarily modeled as proportional to the normal component of the applied force. In our model the same effect is obtained, because the magnitude of intrusion is the consequence of and proportional to the normal component of the force between them.)

Collision detection

Collision detection and response is an important aspect of simulation systems for computer graphics. Real-world objects do not pass through one another, but instead bounce off each other or cause stress that may result in breaking. The major issue in collision detection is speed: the naive algorithm compares all parts of one objects against all parts of another object to determine whether the two objects are interpenetrating. Detecting collisions can be the dominating factor in execution speed in a simulation system (Hahn 1988), so it is important to reduce the time complexity of the detection algorithm.
The method we have chosen for detecting and responding to collisions is node based rather than polygon based. Our model for collisions has each node repel all other nodes that belong to different pieces of a dynamic solid model. The method is simple in that the forces acting on a node due to collisions can just be summed with other forces that act upon the node. A drawback is that it is difficult to position objects in stable continuous contact with one another, as is the case when one object rests on top of another.
We require that collisions be detected not only between separate objects but also between pieces of objects that were once part of the same object. Collision forces should not, however, be generated between points that lie near each other *within* the same object. We consider the collection of all objects in a scene as a graph consisting of the nodes connected by the bonds that are the graph's edges. Two nodes may repel each other if they belong to disjointed components of this graph. Therefore, we must label each node to specify to which graph component it belongs. The graph must be relabeled any time bonds have been broken to identify any piece that may have broken of from dynamic solid model. A depth-first search can be used to label the connected components of a graph.
Once each node is labeled, a locality search is performed to find nodes that are within the repulsive range of a given node. Because the nodes in our models are relatively uniform in density, we use uniform spatial subdivision for this locality search. We partition the space into a cubic lattice in which each cube of the lattice contains a pointer to all nodes that fall within the cube. (These cubes should not be confused with the cells that form the structure of a dynamic solid model.) The size of the cubes is chosen to be large enough so that all nodes that could be repelled by a given node will lie in the same cube or in an adjacent cube to the cube containing the node. Once the nodes have been placed in the cubic partition, a search of nearby cubes is performed for each node to find nodes that should be repelled. Only those nearby nodes that are from a different component than the given node will be repelled. To speed this search for nearby nodes from other components, each cube has a tag specifying the component labels of the nodes it contains. This tag helps avoid comparing a node with the nodes in a cube that are all from the same component. To save space, the cubes are stored in a hash table instead of in a three-dimensional array [see, for example, Bentley and Friedman (1979)]. In the simulations performed, we have found that the time required to complete collision detection grows linearly with the number of nodes in the scene.
Once it is determined that two nodes are in collision, a collision response force is applied. Each of the two nodes has a radius associated with it that was the distance to the nearest neighbor in its original position in the model. The force between two nodes is a spring-like repulsive force,

as if a spring of natural length equal to the sum of the two radii were interposed between the two nodes.

The above model does not detect collisions between different parts of the same connected component. We have also implemented a version where we allow nodes within the same component to repel each other. This prevents self-intersection of objects that can bend back on themselves, such as a rope or cloth, at the expense of detecting additional collisions between nodes.

Breakage modeling

Our approach to dynamic simulation using dynamic solid models was designed to accommodate simulation of breaking objects. We use a simplified fracture model to provide a rough approximation of fracturing materials.

To model accurately how macroscopic shape distortions result in microscopic structural changes in microsecond time intervals would require excessive computation. We therefore accept limits in the smallest time and distance scales used in the simulation; the distance scale is the distance between nodes in a dynamic solid model and the time scale is the simulation step size required for the stiffness of the object.

Given these constraints, the model we describe should not be expected to model accurately the finest details of fracture, but only to show convincingly the coarser aspects of how the physical distortion of a structure can result in its destruction.

Breakage threshold. A simple model for breakage defines a threshold such that when the force applied to a bond is greater than the threshold, then the bond breaks. Studies have shown that materials are much more easily broken when stretched than when compressed [see, e.g. Crandall et al. (1978)]. Therefore, we chose to make bonds break only on stretching, and not on compression. The breakage threshold t_{ij} is compared with the stretch of the bond. If the bond is stretched more than $(1+t_{ij})$ this its starting length, then we say it is broken and set the appropriate Boolean variable s_{ij} at 1.

The breakage of real objects is not usually along regular seams, but tends to be jagged, unless the object is crystalline and breaks along the symmetry of a crystal. Breakage normally tends to occur at weak spots or defects in the structure. We simulate this effect by randomly varying the breakage threshold for the various bonds. If a bond has an unusually low threshold, that represents a defect in the structure that will easily fracture if subjected to a big force.

Cells. The breakage model described above, with individual bonds breaking when sufficiently distorted, is not adequate. If some, but not all, bonds of a cube break, then the cube loses much of its structural stability. This can result in flexible strings of material remaining after a fracture. This defect was remedied by introducing the notion of a cell. A cell is a set of nodes together with all the bonds that connect two nodes in the cell. A cell should be thought of as the basic structural entity that disintegrates under breakage.

Whenever any one bond in a cell is broken, that cell itself is said to be broken. Conversely, whenever all cells containing a bond are broken, we cause that bond to break as well. The result of this algorithm is that the fragmentation of the structure occurs in cellular units rather than just among individual bonds.

For dynamic solid models based on cubic lattices, the appropriate notion of a cell is the set of bonds and nodes associated with the vertices, edges, and diagonals of one cube, as pictured in Figs. 1 or 2.

Collision effects. Spurious effects can emerge during fracture if the collision response model is not designed to be consistent with the internal bonds of the structure. When fracture occurs, collision forces instantly replace the internal forces of the material. If those forces are greater than the binding forces, a chain reaction can be set off, ripping the structure apart. This problem is solved by designing the fracture model to guarantee that breakage does not result in an increase in kinetic energy. We specify the repulsive collision force to be a spring force based on a spring of natural length no greater than the distance to the nearest node in the original solid model. If a material is to break, the spring constant used in collision response must be chosen to be no greater than the constant of the material's internal bonds.

Geometric modeling

It is simple to create a dynamic solid model of simple objects, such as a hollow cylinder or a rect-

angular block. We can build these shapes directly from elements of a cubic lattice, perhaps slightly distorted to match the given shape. We allow the lattice to deviate from exact cubes so that the surface of an object is smooth instead of staircased, as this issue becomes important for rendering. A more complex object is built by joining together more than one such cubic lattice by a "gluing" operation. In this way dynamic solid models of arbitrary topology can be created.

We currently have a small set of simple shapes that we use to build models. All of these shapes are topologically equivalent to either a solid cube or a hollow cylinder with thick walls. Two useful modeling shapes that are variants of a solid cube are a rectilinear slab of arbitrary dimensions and the solid formed by creating a lattice between two parallel Bezier patches, which we call a thickened Bezier patch. A wide range of variations on a cylinder can be created by specifying an inside and an outside surface of revolution that can be joined to create a solid region. The thickened cylinder of Fig. 3 is a simple example of such a solid.

To model more complex objects, we define a gluing operation that allows two or more simple shapes to be joined. Shape A is glued to shape B by first positioning the two shapes so that the regions to be glued together are adjacent or coincident with one another. Then those nodes from shape A that are to be joined to shape B are specified; call them a_i. Each of these nodes a_i is paired with another node b_i in shape B that is nearest to it. Gluing is completed by having each bond containing one of the nodes a_i altered by having a_i replaced by the node b_i from shape B. Each of these bonds is called a glue bond. If a bond from shape A joins two of the specified gluing nodes a_i and a_j, then the spring constant and damping constant of the bond are both set to zero so as to not duplicate bonds already in shape B. All of the nodes a_i are then dropped from the model. Now shapes A and B have been joined to form a larger dynamic solid model that will act as one object.

The Utah teapot model was made by gluing together six separately modeled shapes. The main body of the teapot was obtained by revolving two profile curves (inside and outside profile curves made from Bezier splines) about a vertical axis. A cylindrical rim (needed to keep the lid from falling into the teapot) was glued to the inside of the upper opening. The teapot handle was constructed from two curved solid cylindrical shapes that were glued end-to-end, then both ends glued to the main teapot body. The teapot spout was constructed from two curved slabs forming each side of the spout. The two slabs were glued together along their edge, then the base of the spout was glued to the main teapot body. The teapot lid is the volume of revolution between two revolved Bezier curves.

Rendering

A dynamic solid model is composed solely of nodes and bonds, and many possible rendering techniques can be used to create images of such a structure. To date we have chosen to create a polygonal description that matches the node positions with vertices. Some alternatives to this would be to render each node as a sphere or to create a level surface of a potential field where each node contributes to the value of the field.

To create a polygonal image at a given time step, each face of each unbroken cell of a model is examined to see if it lies on the surface or in the interior of the model. If it is on the surface, a matching polygon is created. Normal vectors are calculated for the vertices of each surface face and are used for Phong shading. At each step in a simulation, all nodes in a model are identified as either isolated or as being connected to other nodes within a structure or fragment. Once all bonds that link a point to its neighbors have been severed, the node is isolated and will remain so throughout the duration of the simulation. Isolated nodes are the smallest particles resulting from breakage. Currently, we render such nodes as simple tetrahedra, although clearly more cosmetic choices are available.

We render images from the polygonal description using a ray-tracing renderer written by Kay (Kay and Kajiya 1986) that uses a hierarchy of bounding volumes to speed up the intersection calculations.

Discussion

The model described above was applied to animate a teapot breaking in the video *Tipsy Turvy* (Figs. 4–7). Other actions of dishware (e.g., hiccups and sneezing!) were animated using the flexibility of dynamic solid models. There are many ways in which the model can be improved to make it a more powerful animation technique. We discuss some of

them below so that they can be used as topics for future research.

Differential equation solution methods

It is an immense computational task to solve the equations of motion for complex flexible objects. Therefore, an important research problem is to find more efficient methods for such simulation. A problem in simulating ceramics as flexible objects is that the models (and the resulting differential equations) must be so stiff that the computation time becomes prohibitive. The speedups to flexible body simulation proposed by Pentland and Williams (1989) and Witkin and Welch (1990) do not readily apply to the problem of animating fracture. For the *Tipsy Turvy* animation, we chose to make the teapot softer than real ceramics in order to complete the simulation within a two-month period.

We used Euler's method to solve the equations of motion, even though Euler's method is known to be inferior to other techniques (e.g., Runge-Kutta or implicit methods) for accuracy and stability in solving stiff differential equations (Recipes 1988). However, we chose not to apply such methods, because the system of equations would potentially be restructured at each time step as a consequence of the fracture model we used. In retrospect, it is likely that with some effort Runge-Kutta or other techniques could be successfully applied, if, for example, the integration method whenever a fracture occurs was modified.

Breakage tuning

The breaking threshold is a single parameter to control a complex phenomenon. The appropriate value is affected by stiffness and the resolution of the model. We tuned the breakage threshold by first selecting the stiffness and damping constants appropriate for the behavior of the material. Then experiments were performed by colliding small cylinders of material at moderate velocity with an immovable surface. The desired breakage threshold will cause a cylinder to break in response to the internal distortion of the material *after* the collision, rather than crumbling upon initial impact with the immovable surface. Once an appropriate breakage threshold is found for the cylindrical sample, some additional adjustment is necessary to cause realistic breakage of a larger structure. A small number of trials was usually sufficient to obtain the desired value.

In order to have better control over the breakage processm it is desirable to express the relationship between stiffness and breaking threshold scales with model resolution. Another problem is that uniform material properties cannot be easily approximated with nonuniform grids. Understanding the scaling aspects of fracture and stiffness should help to construct models that do not show the effects of nonuniform grid approximations.

Breakage appearance

Two undesirable artifacts of the current implementation are the staircasing of the fractured edges and the large number of isolated shards (rendered as tetrahedra) that emerge from the breakage. The former problem has been dealt with effectively by using splines to smooth the broken edge (Hart and Norton 1990). The latter problem is more complex and may require changing the behavior of cells in the breaking process.

Friction and damping

The current models for friction and damping are linear and do not sufficiently dissipate the energy of slow vibrations or gradual sliding. An improved model should provide more damping and friction with slower motion

Collision detection

Our methods of collision detection were potentially problematic in two respects: our use of the penalty method and the node-based approach. Either of these leads to imprecision in locating collisions, and potentially can result in visible problems. The penalty method did not appear to cause a problem with any of the animation sequences. However, the node-based approach did lead to problems when grid spacing was not uniform. For example, particles can pass undetected through the center of long, narrow grid cells if they stay far from the end vertices. We conclude that our approach is a practical technique for generating realistic animations with

many colliding pieces; however, it is very important to construct uniform or nearly uniform grids during the modeling process.

We were pleased with the efficiency of the collision-detection search algorithm: the cost of collision detection appears to grow linearly. This efficiency was a requirement for animating the products of fracture, and indicates that the inclusion of collision detection is not an intrinsic obstacle to very complex animation problems. We discovered an important principle for using collision and breakage in physically based models: it is necessary to ensure that the collision-response forces are no greater than the internal binding forces; otherwise explosion can occur.

Control of flexible models

We were able to manipulate the teapot and other models by varying the physically defining parameters and introducing invisible characters. The teapot appeared to hiccup when gravity was temporaily increased to 5 *g*, then returned to 1 *g* (Fig. 5). A sneeze was simulated (see Fig. 6) by softening the constituent springs of the teapot (i.e., reducing the spring constants by two orders of magnitude), allowing the teapot to sag under the influence of gravity, the returning the springs to their original stiffness. The vases in the scene were made to wobble (an apparent response to the teapot) by causing invisible spheres to collide with them.

All the above techniques required some trial and error. This process would be greatly facilitated by more interactivity (i.e., faster simulation). However, regardless of the speed of computers there will be a continuing need to cause physically based models to behave in a desired way. This experience causes us to see much value in physically based modeling methods that direct simulation toward motion or positional goals. (Barr (Barzel and Barr 1987) refers to this general approach as "teleological modeling.")

Unfortunately most methods of goal-directed physically based modeling are difficult to apply to flexible/breaking object simulation. For example, the complexity of our models and interactions would prohibit the use of constraint methods (Witkin and Kass 1988; Platt and Barr 1988) except on the simplest of the animated actions (e.g., wobbling vases). The methods of "dynamic constraints" (Barzel and Barr 1987) could be used to move shapes to desired positions, but such motion would not show the realistic flexing and fragmentation we desired. Based on our experience, a more appropriate goal-directed approach is the methodology of control theory (Raibert et al. 1984). Control forces could be applied during the course of the simulation in response to the current state of the system, with the magnitude of such forces chosen to push the system toward position or motion objectives. Such an approach is tractable computationally and capable of compensating for the instability of the simulation process, although not necessarily capable of achieving precise space-time constraints.

Conclusion

We have presented a model for animating breakage of materials and applied it to the breaking of a ceramic teapot. We believe that our general approach to breakage modeling is appropriate and have found the model to be successful in meeting its original objectives. This further confirms the value of physically based modeling for animation of physical phenomena. The realism of the model should be improved in varions ways. Our experience with the teapot animation has suggested several refinements and areas for further research.

Acknowledgements. Charles Bennett helped formulate the physical model for breakage. Jakub Wejchert read this manuscript thoroughly and offered valuable scientific and literary criticism.

References

Barzel R, Barr A (1988) A modeling system based on dynamic constraints. ACM Comput Graph SIGGRAPH 22(4)

Bentley JL, Friedman JH (1979) Data structures for range searching. Comput Surveys 11(4)

Crandall SH, Dahl NC, Lardner TJ (1978) An introduction to the mechanics of solids, 2nd Ed. McGraw-Hill, New York

Feynman RP, Leighton RB, Sands M (1965) The Feynman lectures on physics, Vol. 2. Addison-Wesley, Reading, Mass (esp chapters 38 and 39)

Hahn JK (1988) Realistic animation of rigid bodies. Comput Graph 22(4) 299–308

Hart J, Norton A (1990) Use of curves in rendering fractures. Proc Curves and Surfaces, Comput Graph

Kay T, Kajiya JT (1986) Ray tracing complex scenes. Comput Graph 20(4) 269–278

Miller GSP (1988) The motion dynamics of snakes and worms. Comput Graph 22(4) 169–178

Moore M, Wilhelms J (1988) Collision detection and response for computer animation. Comput Graph 22(4) 289–298

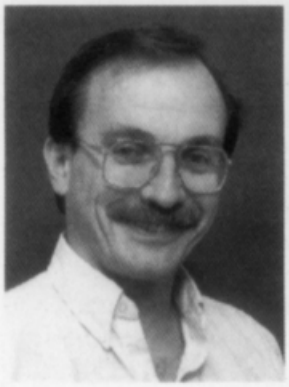
Pentland A, Williams J (1989) Good vibrations: modal dynamics for graphics and animation. ACM Comput Graph SIGGRAPH 23(3) 215–222
Platt J, Barr A (1988) Constraint methods for flexible models. Comput Graph 22(4) 219–288
Press WH, Flannery BP, Teukolsky SA, Vetterling WT (1988) Numerical recipes in C: the art of scientific computing. Cambridge University Press, Cambridge
Raibert M, Brown HB Jr, Chepponis M (1984) Experiments in balance with a 3D one-legged hopping machine. Int J Robotics Research 3(2)
Terzopoulos D, Platt J, Barr AH, Fleischer K (1987) Elastically deformable models. Comput Graph 21(4) 205–214
Terzopoulos D, Fleischer K (1988) Modeling inelastic deformation: viscoelasticity, plasticity, fracture. Comput Graph 22(4) 269–278
Terzopoulos D, Witkin A (1988) Physically based models with rigid and deformable components. IEEE Comput Graph Appl 8(6)
Witkin A, Kass M (1988) Spacetime constraints. Comput Graph 22(4) 159–168
Witkin A, Welch W (1990) Fast animation and control of nonrigid structures. ACM Comput Graph SIGGRAPH 24(4) 243–252

ALAN NORTON. Alan Norton was born in Salt Lake City, UT on August 20, 1947. He received the B.A. degree from the University of Utah in 1968, and the Ph.D. from Princeton University in 1976, both in mathematics. He was instructor at the University of Utah, 1976–79, and Assistant Professor at Hamilton College 1979–80, before joining IBM Research. He first worked with B. Mandelbrot, developing algorithms for generating fractals and making images of them. Then, from 1982 to 1987 he worked on the RP3 project, doing research on parallel algorithms, architectures and performance analysis.
Currently at IBM Research, Hawthorne, New York, he manages a project in computer animation and rendering. His research interests include computer graphics and animation, parallel architectures, and fractals. He is a member of the IEEE computer society, ACM and the American Math Society.

PAULA SWEENEY has been working at IBM since 1984. She has worked on the design and implementation of operating systems and animation systems. Currently her interest is in realistic animation using physics and control theory. Paula has a B.A. in Mathematics from Manhattanville College and an M.S. in computer science from New York University. She is a member of the ACM and SIGGRAPH.

GREG TURK is a graduate student in the Computer Science Department at the University of North Carolina at Chapel Hill. At UNC he has been involved in the development of rendering algorithms for the Pixel-Planes graphics engine. He worked in the computer graphics group at the IBM T.J. Watson Research Center during the summer of 1988. He is currently supported by an IBM Graduate Fellowship. His interests include physically-based modelling, collision detection, synthetic texture generation and constructive solid geometry.
Turk received a BA in mathematics from UCLA in 1984 and an MS in computer science from UNC in 1989.

ROBERT BACON only recently discovered that he has been working on visualization throughout most of his professional career. Following graduate studies at the University of Chicago, Mr. Bacon began working with computers in such diverse endeavors as process control, machine tool control, computer typesetting, image synthesis, and computer generated animation. He has been a frequent contributor to animation exhibits since 1985. He is a member of IEEE Computer Society and SID.



JOHN GERTH graduated from MIT in 1970 but left science to help found an elementary school as its kindergarten teacher. From 1974–78 he was an instructor in architecture at Virginia Tech. Exposure to APL led him to IBM, working first as a developer and later joining the design team. Moving to the Watson Research Center in 1986, he has studied computer graphics since 1988.
His research interests include high performance rendering and the qualitative display of quantitative information.