An Animation System for Fracturing of Rigid Objects[⋆]

Ayşe Küçükyılmaz and Bülent Özgüç

Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey
{aysek, ozguc}@bilkent.edu.tr

Abstract. This paper describes a system for the animation of fracturing brittle objects. The system combines rigid body simulation methods with a constraint-based model to animate fracturing of arbitrary polyhedral shaped objects under impact. The objects are represented as sets of masses, where pairs of adjacent masses are connected via a distance-preserving linear constraint. Lagrange multipliers are used to compute the forces exerted by those constraints, where these forces determine how and where the object will break. However, a problem with existing systems is that the initial body models exhibit well-defined uniformity, which makes the generated animations unrealistic. This work introduces a method for generating more realistic cracks without any performance loss. This method is easy to implement and applicable on different models.

1 Introduction

Realistic animation of fracture is a difficult one, because in order to generate a convincing animation, we need to understand the physical properties of the objects in a scene, rather than considering them as merely geometric shapes. These bodies should be thought of as real objects that have mass, elasticity, momentum, etc., and they display certain material properties. Another difficulty of fracture animation is that the scenes change dynamically during the animation. The bodies are fragmented to create new bodies which are again subject to the same effects. Physically precise animations cannot be realized successfully by computation, not only because they are time consuming and hard, but also the real motions and the fragmentation of objects require an extensive amount of calculations. However, such great accuracy is not a requisite for animation purposes. By using physically based animation techniques, we can create realistic-looking shatters and breaks with much less effort, yet with as much visual precision as necessary.

In this paper we discuss a system for computer animation that is implemented for generating animations of rigid objects that involve fracturing. This implementation combines the methods for simulating the fracturing of brittle

[⋆] This work is supported by EC within FP6 under Grant 511568 with the acronym 3DTV.

objects with the rigid body simulation techniques in order to generate realistic-looking fracturing animations. Additionally, an improvement to this system is made through some techniques used for generating and modifying the object models more realistically.

The organization is as follows: In Sect. 2, we give a description of the object model that is proposed by Smith, Witkin and Baraff [14], which forms the basis for the simulation of the fracturing process. The process of generating each animation frame, which combines the rigid body and the fracture simulation techniques, are described in Sect. 3. Finally, some example animations generated by our system and conclusions are provided in Sect. 4 and 5, respectively.

2 Object Models

For the simulation of the fracturing of rigid objects, each object is modeled as a system of particles connected by distance-preserving constraints. The object models are constructed in tetrahedral mesh representation, using a tetrahedral mesh-generation software package such as NETGEN [10]. For each tetrahedron in the mesh, a particle is located at its center of mass, where the mass of each particle is a function of the volume of the tetrahedron it represents, and the density of the material at that point. For each pair of tetrahedra with a shared face, the corresponding particles are connected with a rigid constraint, which has strength proportional to the area of the shared face. The usage of distance-preserving constraints realistically models the inflexible nature of the objects that we are trying to animate by preventing the neighboring particles to change their positions relative to each other.

Although these constraints model the fracturing behavior successfully, the effects generated by using them alone are not satisfactorily realistic. This is due to the fact that the generated mesh, which characterizes the models' density, is uniform. Thus the cracks are developed deterministically.

In order to achieve user control on the fracturing behavior of the objects, some modification heuristics can be applied on their initial models. This might be useful for defining the overall crack pattern of the final models. Cleaving

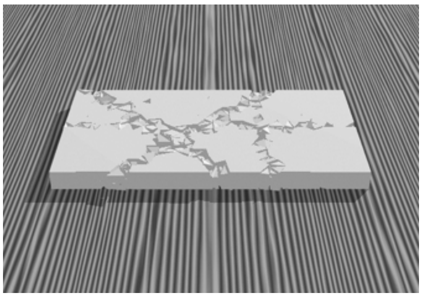

Fig. 1. The cleaving effect

planes are used for systematically modifying the connection strengths along a cross-section of the objects. Fig. 1 illustrates the cleaving effect on a sample animation.

In addition, three-dimensional noise functions provide a way to change the connection strength of the objects procedurally to achieve different fracturing behavior for the same object geometry.

However, these methods are done as preprocessing on the models. Further randomization can be achievable by modifying the connection strengths during the fracture process. The idea depends on the fact that, in real life, when an object shatters, its inner material properties change due to the cracks occurring on it. This technique will be explained in Sect. 3.

3 Generating the Animation

The animation frames are generated by calculating the motion paths of the objects in the scene and determining their updated positions and orientations for each frame by the bisection technique. In the case of a contact between two objects, the motion paths of the objects are updated accordingly and fracture calculations are performed. If these calculations result in the shattering of the object, the resulting shards are modeled as new objects and they are included in the animation calculations. This process is repeated until either all the desired frames of animation are generated or the system finally comes to rest.

3.1 Rigid Body Simulation

The states of the objects in the space can be represented by their positions, orientations, and angular and linear velocities. Therefore, given the initial states, finding the states at any given time is simply an initial value problem. An ODE integrator using the Fourth-Order Runge-Kutta method with adaptive step sizing is implemented to solve this initial value problem and calculate the motion of the objects (see [9]). By applying an adaptive step-sizing algorithm, an upper bound for error is maintained in motion calculations.

However, when there is collision between objects, these unconstrained motion calculations fail to give realistic results. Thus, by applying appropriate responses on the objects when they are in contact, the impenetrability constraints can be enforced throughout the animation, letting the objects continue their unconstrained motion paths.

A contact between two objects is defined by the contact normal and a contact point extracted from contacting features of the two objects. In the case of multiple contact points between two objects, each contact point is considered as a separate contact.

After the contact points and the corresponding contact normals of a contact are determined, the appropriate response for the contact is calculated by looking at the projection of the relative velocity of the objects at the contact points over the contact normal.

The vector E, which is the impulse that acts on an object that is in contact, can be defined as:

$$E = \int F(t)dt \tag{1}$$

where, $F(t)$ is the vector function that defines the contact force acting on the object during the course of the contact. The required change in the velocity of the object due to this collision can be achieved by applying the changes $\Delta P(t)$ and $\Delta L(t)$ to the linear and angular momentums of the object respectively. $\Delta P(t)$ and $\Delta L(t)$ are defined as:

$$\Delta P(t_c) = E, \tag{2}$$

$$\Delta L(t_c) = (p - x(t_c)) \times E. \tag{3}$$

Here, p is the contact point, $x(t)$ is the position of the center of mass of the object and t_c is the time of the collision.

Even though the contacting objects are neither moving towards each other nor moving apart at the contact point (i.e. a resting contact), the impenetrability constraint can still be violated if the contacting objects are accelerating towards each other. In this case, contact forces must be calculated and applied to the objects in order to prevent them from accelerating into each other.

Besides requiring the relative acceleration of the objects, a_{rel}, to be nonnegative, two other conditions must be satisfied while calculating the contact forces. Firstly, the contact forces should never be attractive; and secondly, the contact forces must remain as long as the corresponding contact remains and no more. By combining these conditions together, we can formulate the problem of finding the contact forces as a quadratic programming (QP) problem as follows:

$$\min f^T(Af + b) \text{ subject to } \begin{cases} (Af + b) \geq 0 \\ f \geq 0 \end{cases} \tag{4}$$

Here $(Af + b)$ is the concatenation of all the a_{rel} values for all of the resting contacts and f is the concatenated vector of contact forces that are required for enforcing the impenetrability constraints. The concatenated vector is separated into its force dependent and force independent parts in order to be able to formulate it as a QP problem.

3.2 Fracture Simulation

The simulation of the fracturing process makes use of the lattice model representation of the objects. The crack initialization is invoked due to some external force applied to a point on the outer surface or in the inner region of the object. Upon the application of such a force, in response, constraint forces are calculated for connections in the lattice model in order to preserve the distances on the lattice of particles. In case the constraint force for a connection is greater than the current connection strength, that connection is removed. Otherwise, the existing connection strength is weakened by the amount of the constraint force applied on it. Any connection for which the resulting connection strength is weaker than a predefined threshold is removed.The process of modifying the connection strengths will be explained in detail after providing the details on how to calculate the constraint forces.

For calculating the constraint forces that act on the system of particles of the object, the positions of the particles are placed in a vector named q, such that, for an n particle system, q is a $3n \times 1$ vector defined as:

$$q = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}. \tag{5}$$

A mass matrix M is defined in such a way that it holds the particles' masses on the main diagonal, and 0's elsewhere. So a mass matrix for n particles in 3D is a $3n \times 3n$ matrix with diagonal elements $\{m_1, m_1, m_1, m_2, m_2, m_2, ..., m_n, m_n, m_n\}$.

Finally, a global force vector Q is obtained by joining the forces on all particles, just as we did for the positions. From Newton's Second Law of Motion, the global equation on the particle system is as follows:

$$\ddot{q} = M^{-1}Q, \tag{6}$$

where M^{-1} is the inverse of the mass matrix, M.

A similar global notation will be used for the set of constraints: Concatenating all scalar constraint functions form the vector function $C(q)$. In 3D, for n particles subject to m constraints, this constraint function has an input of a $3n \times 1$ vector, and an output of an $m \times 1$ vector. In our case, this constraint function consists of the scalar distance-preserving constraints in the form:

$$C_i(p_a, p_b) = \|p_a - p_b\| - d_i, \tag{7}$$

where p_a and p_b are the positions of two particles connected to constraint i, and d_i is the distance between the particles that needs to be preserved.

Assuming initial positions and velocities are legal, we try to come up with a feasible constraint force vector $\hat{Q}$ such that the distance preserving constraints are held. In other words, for the initial conditions that satisfy $C(q) = \dot{C}(q) = 0$, we are trying to find the constraint force vector $\hat{Q}$, such that, when added to Q, guarantees $\ddot{C}(q) = 0$. Taking the derivative of $C(q)$, we get:

$$\dot{C} = \frac{\partial C}{\partial q}\dot{q}. \tag{8}$$

$\frac{\partial C}{\partial q}$ is called the Jacobian of C, and will be denoted by J. Differentiating the above formula once again with respect to time gives

$$\ddot{C} = \dot{J}\dot{q} + J\ddot{q}. \tag{9}$$

Replacing $\ddot{q}$ according to relation 6 and adding the constraint forces gives

$$\ddot{C} = \dot{J}\dot{q} + JM^{-1}(Q + \hat{Q}). \tag{10}$$

where $\hat{Q}$ is the unknown constraint force vector. Setting $\ddot{C}(q) = 0$ gives

$$JM^{-1}\hat{Q} = -\dot{J}\dot{q} - JM^{-1}Q. \tag{11}$$

In order not to break the balance of the system, it has to be assured that no work is done by the constraint forces in system, for all valid position vectors:

$$\hat{Q}.\dot{q} = 0, \quad \forall \dot{q} | J\dot{q} = 0. \tag{12}$$

All vectors that satisfy this requirement can be written as

$$\hat{Q} = J^T\lambda, \tag{13}$$

where λ is a vector with the same dimensions as C. The components of λ are known as Lagrange multipliers and they tell how much of each constraint gradient is mixed into the constraint force. From 11 and 13:

$$JM^{-1}J^T\lambda = -\dot{J}\dot{q} - JM^{-1}Q. \tag{14}$$

Note that the above formula is a system of linear equations of the form $Ax = b$ where A is a matrix and x and b are vectors. By calculating the λ vector from equation 14 and placing it in equation 13, the constraint force vector $\hat{Q}$, which satisfies the given rigidity constraints can be calculated.

Additionally, to ensure that the λ vector is a physically realizable solution for the system, the conjugate gradient method [11], which gives the minimum norm solution of the system, is used since the minimum norm solution of the system is also the physically realizable one.

Once a crack is invoked at some point of the model, due to some external or internal force, the connection strengths are modified procedurally. Obviously, the connections that are close to the crack region will be affected more than the connections that are far away from it. The strengths are modified gradually as given in the following algorithm. However, weakening the connection strengths uniformly produces cracks that are visually artificial. Hence, in order to introduce a randomness into the crack pattern, some connections are made weaker than the others. These connections, and the amount of weakening are selected randomly (lines 9-11). This operation introduces no performance loss, yet it is very successful in generating crack patterns. Moreover, even though two geometrically same objects are broken under the same conditions, the system produces distinct final cracks. In addition, with this modification, formation of longer cracks is achieved.

Fig. 2 compares the effect of modifying the connections with and without the given technique. The object in (a) is broken with the original algorithm, while (b), (c), and (d) are broken with our modified one. It is easily observable how the crack patterns change every time the algorithm is run. In addition, with the technique used here, not only a successful randomization in cracks is achieved, but also the cracks formed after the fracture are longer.

Algorithm: modifying the connection strengths

```
changeConnectionStrengths($latticeNode1, latticeNode2, -change$)
 1  decrease the connection strength between $latticeNode1$ and $latticeNode2$
        by $change$
 2  $change \leftarrow \alpha \times change$ where $0 < \alpha < 1$
 3  for $latticeNode \in \{latticeNode1, latticeNode2\}$
 4    select $selNeighborNode \in$ neighbors of $latticeNode$ randomly
 5    $str \leftarrow$ connection strength between $selNeighborNode$ and $latticeNode$
 6    $str \leftarrow str - \beta \times change$ where $0 < \beta < 1$ and $\beta > \alpha$
 7    connection strength between $selNeighborNode$ and $latticeNode \leftarrow str$
 8    for $(neighborNode \in$ neighbors of $latticeNode) \wedge$
          $(neighborNode \neq selNeighborNode)$
 9        if $change \geq \tau$ where $\tau$ is a predefined threshold
10            changeConnectionStrength($latticeNode, neighborNode,$
                $change$)
11        endif
12    end
12  end
```

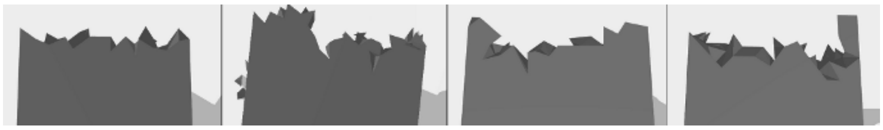

Fig. 2. A comparison of the crack patterns generated by modifying the connection strengths (a) uniformly (b-d) with the given algorithm

4 Results

This section presents the important results generated by our system. The movies for the animations given in the figures, are accessible through the Internet. For viewing these, please visit the web page:

`http:\\www.cs.bilkent.edu.tr\~aysek\research.php`.

Fig. 3 shows an adobe wall struck by a heavy weight. The wall, which consists of 4080 tetrahedra with 8897 shared faces, is fixed to the ground and experience collision only with the heavy ball. In addition, since it is the only breakable object in the scene, the fracture calculations are done only for it.

In Fig. 4, cleaving is used to make the connections passing through the middle of the table's surface weaker than the rest of the surface to give a more realistic look.

Three major steps take place during the creation of an animation. The first step, generation of the object models, is performed once for an animation scene, and the results are stored in a file. This is a very fast process, taking only a few

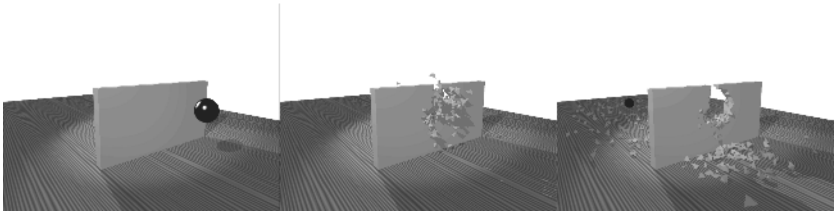

Fig. 3. An adobe wall breaking under the impact of a heavy ball

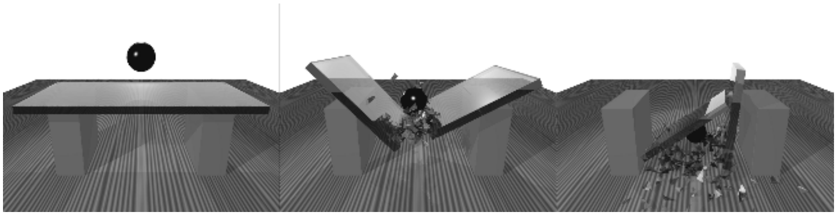

Fig. 4. Glass table breaking under the impact of a heavy ball

seconds even for very large numbers of tetrahedra. The second step, creation of the animation, is the most time consuming one. The time required in this step for the animation shown in Fig. 3 was 4437 seconds, giving an average calculation speed of 61.7 seconds/frame. As it can be seen from the performance graph in Fig. 5, the generation of the frames that are created just after the shattering occurs took significantly more time than the average. The main reason for this is the big number of contacts that occurred between the newly created fragments. After a few frames, the calculation durations stabilize near 55 seconds. The time required for the third step, visualization of the results, greatly depends on the material qualities of the object in the animation scene.

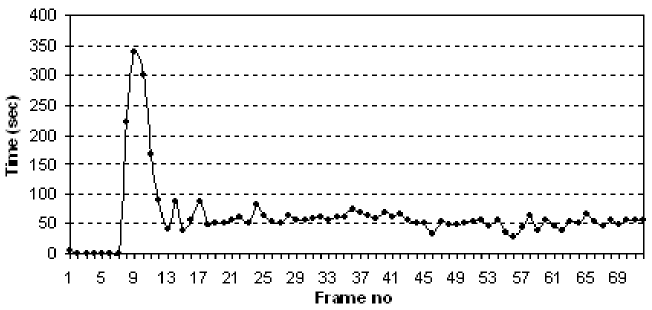

Fig. 5. Calculation times for the breaking wall animation on a 1.6 GHz Pentium4 machine

5 Conclusion

This study explores and implements a method for animating brittle fracture realistically. The implementation follows the method explained in [12]: The objects are represented as tetrahedral mesh and lattice models as explained in Sect. 2. In our implementation, as an improvement to the existing model, the constraint strengths are further modified by some heuristics in order to simulate the irregularity in the material properties.

Naturally, the number of the tetrahedra increases with respect to the complexity of the object geometry, and for generating visually improved animations. However, the time required for generating the animation increases as this number increases. Another limitation stems from the space requirements. The files describing the geometry of a high-resolution tetrahedral object are quite large. As a result of this drawback, the tetrahedron meshes for the samples illustrated in Sect. 4 were generated just as dense to illustrate the breaking behavior. However, higher-quality animations could have been generated.

The animation generated by the formulation presented in previous sections outputs a fracture effect where there are several fragments consisting of single tetrahedron. Although, Smith et. al. suggest in [12] that particles consisting of a single tetrahedron can be eliminated without loss of visual effects, this approach results with gaps around the cracks that seem to originate from nowhere. Therefore, in this study, single tetrahedron objects are left as is. However, this resulted with identical looking fragments, which can be seen in the results section.

As explained in previous sections, the constraint-based model is not sufficient on its own to mimic real world. Since the lattice construction assigns masses to tetrahedra according to their volumes, the density of objects stay uniform at every point of their bodies. This imposes that an object is never weak at some parts of its body, and this results in a uniform shattering effect, which seems dull. Since it is desirable to have irregularities in objects' mass distributions; some techniques are implemented in order to eliminate such uniformities by modifying the constraint strengths. The first class of these techniques, which are done as preprocessing, include applying noise function on an object that assigns different strengths to different parts of a body in a random manner, and using a cleaving function, which modifies the strengths at given regions. With cleaving, the animation can be controlled dynamically, by assigning strengths appropriately to regions that are desired to fall apart or stay intact. The other technique involves changing the strengths dynamically during the process. This presents good results, and require no user processing on models.

References

1. Baraff, D.: Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies. SIGGRAPH 89 Conference Proceedings (1989), 223-237
2. Baraff, D.: Fast Contact Force Computation for Non-penetrating Rigid Bodies. SIGGRAPH 94 Conference Proceedings (1994), 23-34

3. Baraff, D.: Non-Penetrating Rigid Body Simulation. Eurographics'93 State of the Art Repors (1993)
4. Baraff, D.: Physically Based Modeling: Principles and Practice, Chapter Rigid Body Simulation. SIGGRAPH 2001 Course Notes, ACM SIGGRAPH (2001)
5. Mirtich, B.: Impulse-based Dynamic Simulation of Rigid Body Systems, Ph.D. Thesis, University of California, Berkeley (1996)
6. Moore, M., Wilhelms, J.: Collision Detection and Response for Computer Animation. ACM Computer Graphics (1998), 22-4:289-298
7. O'Brien, J. F., Hodgins, J.: Animating Fracture. Communications of the ACM, Vol. 43 No. 7 (2000)
8. O'Brien, J. F., Hodgins, J.: Graphical Modeling and Animation of Brittle Fracture. SIGGRAPH 99 Conference Proceedings (1999), 33:287-296
9. Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T: Numerical Recipes in C; The Art of Scientific Computing, 1st edn. Cambridge University Press, Cambridge, NY, USA (1992)
10. Shcberl, J.: NETGEN - 4.3. www.sfb013.uni-linz.ac.at/ joachim/netgen/, (2003)
11. Shewchuk, J. R.: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Computer Science Tech. Report 94-125, Carnegie Mellon University, Pittsburgh, PA (1994) see also http://www.cs.cmu.edu/ quake/papers.html
12. Smith, J., Witkin, A., Baraff, D.: Fast and Controllable Simulation of the Shattering of Brittle Objects. Graphical Interface, Montreal, Canada (2000)
13. Terzopoulos, D., Fleischer, K.: Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. SIGGRAPH 88 Conference Proceedings (1988), 22:287-296
14. Witkin, A., Baraff, D.: Physically Based Modeling: Principles and Practice, Chapter Differential Equation Basics. SIGGRAPH 2001 Course Notes, ACM SIGGRAPH (2001)