Procedural Modeling of Cracks and Fractures

Aurélien Martinet † Eric Galin ‡ Brett Desbenoit ‡ Samir Akkouche ‡

†Artis-GRAVIR
INRIA Rhône Alpes
655 Avenue de l'Europe, Montbonnot
38334 St Ismier Cedex, France
Aurelien.Martinet@imag.fr

‡LIRIS
Université Claude Bernard Lyon 1
43 Boulevard du 11 Novembre 1918
69622 Villeurbanne Cedex, France
Eric.Galin|Brett.Desbenoit|Samir.Akkouche@liris.cnrs.fr

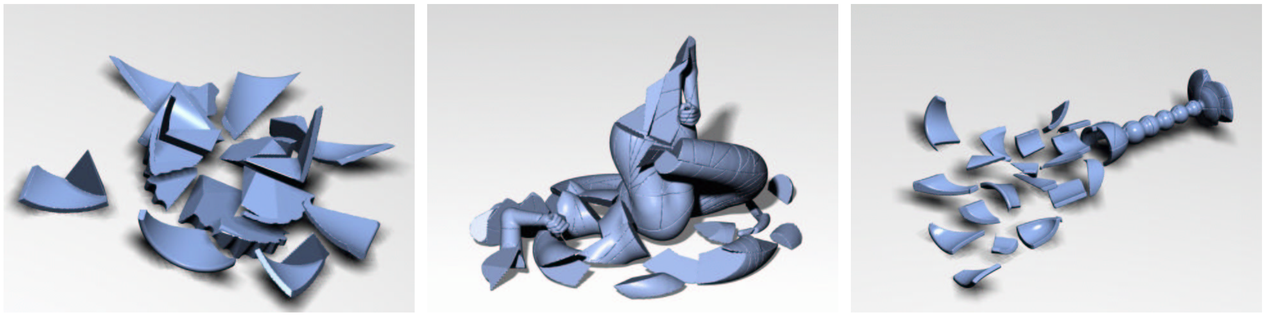

Figure 1. The left and right images show two broken glasses with 18 and 48 fragments respectively. The center image shows a statue shattered into 128 fragments

Abstract

In this paper, we present a procedural method for modeling cracks and fractures in solid materials such as glass, metal and stone. Existing physically based techniques are computationally demanding and lack control over crack and fracture propagation. Our procedural approach provides the designer with simple tools to control the pattern of the cracks and the size and shape of fragments. Given a few parameters, our method automatically creates a vast range of types of cracks and generates fragments of different shapes.

Keywords: Procedural modeling, implicit surfaces, fractures, cracks.

1. Introduction

Realistic animation of breaking objects is a challenging task in computer animation. Breaking an object often creates many small and interlocking pieces. The complexity of these fragments makes modeling by hand impossible. Consequently, the simulation of cracking, breaking and shattering has received some attention in the computer graphics community.

Most existing techniques rely on involved and computationally demanding physically based simulations to compute both crack propagation and create fragments [4, 12, 9, 13]. Such methods are indispensable for correct and accurate simulation of shattering and breaking and have produced images of striking realism.

Physically based simulations often require the discretization of objects into voxels or tetrahedral meshes to compute internal forces. This discretization often leads to some artifacts in the crack pattern which makes fragments look not very realisitic. Those artifacts are the more visible as fractures are propagated along the boundaries of the initial mesh or voxel grid. Moreover, it is difficult to control a simulation so that breaking should occur only in a given region and so that some fragments should have a specific user defined shape. Therefore, their usage may cumbersome for some applications and more simplistic, albeit less accurate, approaches may be useful.

In this paper, we propose an original procedural method for creating cracks and breaking objects into fragments. Our approach is phenomenological and closely related in spirit to the procedural surface aging technique proposed by Paquette et *al.* [14]. By observing real world fractures and crack, we identify some simple patterns and parameters that will guide our procedural techniques and will provide an interactive control to the designer.

Our crack and fracture modeling system relies on a hybrid representation of shapes that combines skeletal implicit surfaces and triangle meshes. In our implementation, we use the Hybrid Tree [1] model that combines skeletal implicit primitives and triangle meshes in a tree structure whose nodes hold Boolean, blending and warping operators. The Hybrid Tree provides us with a number of advantages.

Cracks are defined using Boolean difference operations between the original input model and carving volumes. The carving volume is defined as the union of complex implicit skeletal primitives such as tetrahedra and swept volumes obtained by sweeping a profile curve along the edges of a crack pattern graph that has been projected onto the surface of the original object.

Fractures are created using Boolean intersection and difference operations between the original input model and fracture masks. Fractures masks are defined at different levels of detail as skeletal implicit primitives or complex meshes that define the small details of appearance of the cuts. Since modeling every fragment by hand is impossible, we propose a technique that automatically creates a user defined number of fragments with a specified distribution in shape and size by repetitively applying intersection and difference operators between the original object and parameterized fracture masks.

Combining skeletal implicit surface and mesh models enables us to deal with a variety of input models. Cracks and fractures are handled in a coherent a consistent modeling framework. The Hybrid Tree structure enables us to incrementally create small and interlocking fragments. Because the objects need not be voxelized or tetrahedralized as for physically based techniques, we are not limited in precision or resolution when creating fragments. Therefore, we avoids shape artifacts resulting from the discretisation preprocessing step required by physically based techniques and we can easily create very small shards.

The overall simulation algorithm is simple and efficient, allowing the designer to intuitively break an object and interactively see the results.

The remainder of this paper is organized as follows. Section 2 present a short overview of existing crack and fracture simulation techniques. Section 3 recalls the fundamental properties of the Hybrid Tree. Section 4 and 5 present our method for cracking and breaking objects. Eventually, Section **??** present some examples produced by our system and compares the results with some real world images.

2 Related work

Animating and simulating fractures has received a lot of attention in the computer graphics community. Terzopoulous and Fleischer [18] first presented a general technique for modeling visco-elastic and plastic deformations. Norton et *al.* [11] described a technique for modeling the breaking of volume objects that were discretized into a set of cubes attached to one another by springs.

More recently, O'Brien and Hodgins [12] proposed a method for modeling and animating brittle fracture by analysing the stress tensors computed over a finite element model. This model was extended in [13] to model ductile fractures for a wider range of materials. Smith et *al.* [16] presented a technique for shattering brittle objects using a set of point masses connected by distance preserving linear constraints. Mûller et *al.* [9] used a hybrid implicit-explicit integration scheme to compute deformations and fracture of stiff materials in real time.

Some work address the fragmentation of solid objects induced by explosions. Mazarak et *al.* [8] used a voxel based approach to model solid objects that break apart when they encounter a blast wave. Nef and Fiume [10] proposed a recursive pattern generator to divide a planar region into polygon shards.

Other relevant work in the computer graphics literature includes techniques for modeling static crack patterns on dry mud, ceramics or even paint. Hirota et *al.* [4] developed a mass-spring system for simulating the static crack and fracture patterns created by drying mud. Gobron et *al.* [3] described a method for modeling the propagation of cracks on the surface of objects using a cellular automata. Recently, Paquette et *al.* [15] presented a method for simulating the crack propagation, loss of adhesion and curling effects of paint peeling.

3 Background

The Hybrid Tree model [1] relies on a tree data structure whose leaves hold either implicit primitives or triangle meshes. These are combined by Boolean, blending and warping operators located at the nodes of the tree. The internal representation of a model is either an skeletal implicit surface or a triangle mesh, and automatically adapts to the operations performed on the model. The conversions between the two representations are completely transparent for the user.

The evaluation of the Hybrid Tree is achieved in an incremental way by recursively traversing the tree structure. There are two different kinds of queries. Potential field function queries at a given point in space are performed whenever the implicit formulation is required or lends itself for some operations. The implicit surface representation is used for performing the blending and Boolean operations needed to carve cracks and generate the fragments using fracture masks. It is also used to compute the relative volume of fragments using an octree decomposition of space with Lipschitz conditions and interval arithmetic.

Polygonization queries are invoked if a surface representation is needed. Triangle meshes are produced for fast visualization crack pattern mapping and fast surface marching.

4 Modeling cracks

The creation of cracks on the surface of an object is performed in three steps as represented in Figure 2. First, the designer defines a crack pattern, denoted as P, which is implemented as a graph whose topology corresponds to the connectivity of the cracks and whose edges and the nodes contain the profile information of the crack, i.e., the width and the depth of the crack. This graph may be either created by hand, by procedural techniques or computed from real world images.

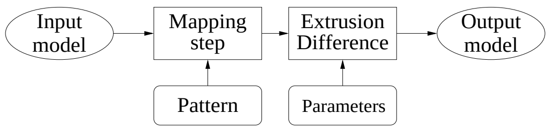

Figure 2. Simulation cycle of the cracking process

The second step of the algorithm maps the graph pattern onto the original object A to create the geometric skeleton of the crack, which will be denoted as S. Starting from a vertex on the object, our algorithm marches along the surface and progressively creates and propagates the crack pattern by creating a set of connected edges on the surface.

Eventually, we create a volume representation of the crack, denoted as V, by sweeping a profile curve of the crack along the edges of the skeleton. The final object is defined by removing the volume V from the original shape using a difference Boolean operation.

4.1 Designing cracks

Cracks are characterized by a graph that defines both its branching features and its geometry. The nodes of the graph hold information about the width and the depth of the crack at the corresponding vertex, as well as the directions and the angles between junctions. The edges of the graph include information about the length of the crack. The width and depth along an edge of the graph are defined as a linear interpolation of the values at the corresponding nodes so as to create cracks of varying thickness and depth.

Crack patterns may be easily created using a specific crack-graph editor, which enables the designer to control all the parameters freely. Different cracks may be created an saved into a collection of patterns, or implemented as procedural cracks, much like textures. In our implentation, cracks are defined from real world images so as to create realistic crack patterns. The varying depth of the crack cannot be retrieved easily in the general case. Therefore, the designer may simply edit the depth parameters at the nodes of the graph, or specify that the cracks should always cut through the object.

4.2 Mapping

Once the crack specifications have been completed, the designer maps the crack pattern onto the surface of the original object. The edges of the graph are projected as line segments onto the surface by applying a surface marching algorithm, which creates the geometric skeleton of the crack. This marching algorithm is recursively called at the nodes of the tree to create crack junctions.

It is worth noticing that the marching algorithm used to map the crack pattern may generate self-intersecting skeletal elements. In practice, this occurs only if large crack patterns are applied in regions of high curvature. Nevertheless, the carving volume generation process described in the next paragraphs still creates consistent objects.

4.3 Volume generation

Once the skeleton of the crack has been created on the surface, we generate a carving volume that will create the crack in the object using a difference Boolean operator. This carving volume is defined by sweeping a vee-shaped profile curve parameterized by the width and depth of the crack along the line segments of the skeleton S as depicted in Figure 3. In our modeling system, this carving volume will be characterized as Hybrid Tree defined as the union of prisms, tetrahedra and pyramids implicit primitives.

The vertices of the carving volume for every line segment of the skeleton are computed as follows. Let $\mathbf{a}$ denote a vertex of the skeleton and w and d denote the width and depth of the crack at vertex $\mathbf{a}$ respectively. We first compute the normal of the implicit surface at vertex $\mathbf{a}$ as well as the tangent vector to the skeleton, denoted as $\mathbf{n}$, and $\mathbf{t}$ respectively. The bottom vertex is defined as $\mathbf{b} = \mathbf{a} - d\,\mathbf{n}$, whereas the border, denoted as $\mathbf{b}'$ and $\mathbf{b}''$ are computed as $\mathbf{a} \pm w\,\mathbf{t}$.

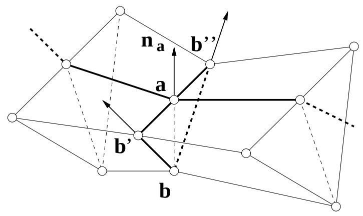

Figure 3. Carving volume generated by a line segment of the crack skeleton

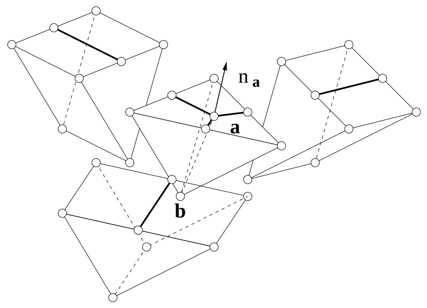

Figure 4. Tetrahedral carving volume generated at a junction

In practice, those vertices may be slightly inside or outside of the original object depending on the local curvature of the surface, although they should be above the surface so as to create a hole without artefacts. Therefore, vertices $\mathbf{b}'$ and $\mathbf{b}''$ are first cast onto the surface and then raised above it by a user controlled offset distance. Junctions are handled in a more specific way so that carving polyhedra should join in a consistent fashion. We use a tetrahedral volume to join three edges of the skeleton as depicted in Figure 4. In the general case, we create a n-sided pyramid to join n edges. Eventually, the crack is carved in the object by creating the final Hybrid Tree defined as the difference $A - V$.

4.4 Results

Figure 5 shows a comparison between a real broken vase made of clay, and the corresponding synthetic model created with our method. The paths of the cracks on the synthetic model were created after the original image. The depth of the cracks adapted to the thickness of the original model automatically during the crack volume generation process so that cracks should pierce the model and not only create surface scratches.

Figure 6 shows how the crack pattern created after the original image of the clay vase was applied to a bottle and an amphora model. As for the vase model, the depth of the cracks was computed automatically.

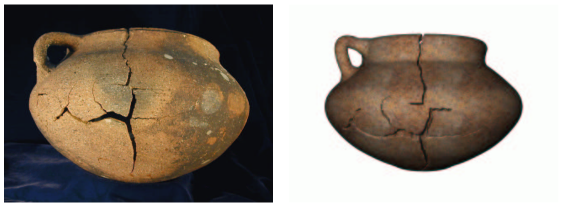

Figure 5. A real clay vase (left) and a synthetic model (right) with the fracture geometry created after the original image

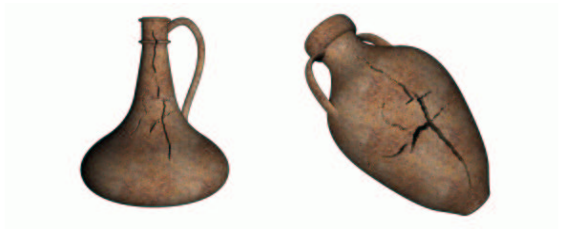

Figure 6. A cracked clay bottle and an ancient amphora

Table 1 reports the timings for generating the final cracked Hybrid Tree representations of the vase, bottle and amphora models. Timings include the mapping of the crack pattern and the volume generation process. The final triangle meshes were generated using a modified Marching Cubes algorithm as presented in [7] on a 256^3 voxel grid. The number of triangles of the final model is also reported.

Model	Cracking	Triangles
Vase	1:36	415 192
Amphora	2:13	249 940
Bottle	2:48	213 036

Table 1. Timings (in seconds) for generating the cracked models in Figure 6 and 5

The cracked snake woman model in Figure 7 was produced by spawning 32 cracks randomly across the surface.

5 Modeling fractures

A simulation starts with the selection of a fracturing tool and the definition of its properties and parameters taken either from user specified values or statistical distribution. Let A denote the original object and F the fracture mask. The fracturing tool is applied to the object to break it into

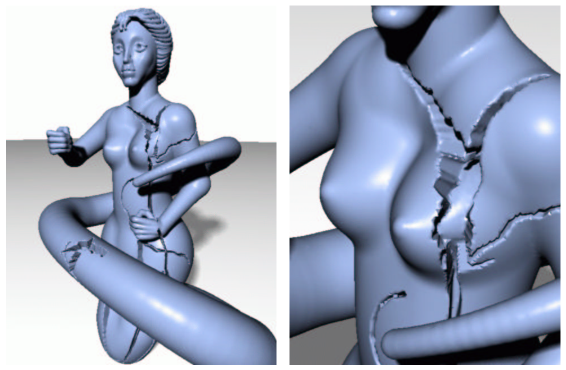

Figure 7. A cracked statue model

fragments by computing the Boolean intersection $A \cap F$ and difference $A - F$ between the original object and fracture mask. Those fragments are expressed as Hybrid Trees and can be further broken into pieces by repetitively applying the selected tool. Our algorithm automatically locates the fracture masks in space so that the volume and shape of fragments should have a user defined distribution. The whole process in repeated as shown in Figure 8 until the number of pieces specified by the designer is reached.

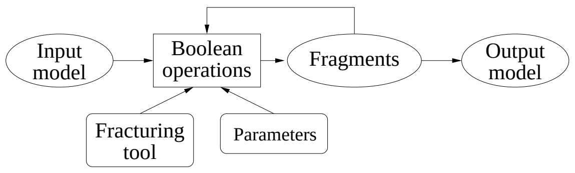

Figure 8. Simulation cycle of our system

When the simulation is completed, all the fragments are fully characterized by a Hybrid Tree which may be polygonized for fast visualization. Mechanical characteristics such as their mass, volume, inertia tensors may be computed easily to create physically based animation.

5.1 Fracture masks

In our system, a fracture mask is defined as a parameterized volumetric shape that defines the pattern of the fracture that will be generated between two fragments. In practice, the fracture masks are positioned relatively to the original objects so that it should embed part of the object and cut it into parts.

Fracture masks no only characterize the overall large scale pattern of the crack between two fragments, but also the small details which make the crack surface rough or smooth. Therefore, our system implements fracture masks as primitives with levels of detail. At a lower resolution, fracture masks defined as simple skeletal implicit primitives such as ellipsoids, boxes or half spheres which create fragments with straight and smooth cut patterns. At a higher resolution, those otherwise simple shapes are represented by complex meshes whose vertices are perturbed by a noise function so as to generate more realistic looking fracture cuts.

Our system also allows the designer to create very complex fracture patterns by using a height field which may be edited as an image. This technique enables the designer to reproduce very realistic surface characteristics in terms of profile and rugosity after real world examples.

5.2 Controlling the shape of fragments

An original object may be broken in two ways. First, the user may interactively edit the position and orientation of the fracture masks in space to control the shape of the fragments. This approach provides the designer with a tight, although low level, control over the shape of the final broken object. In practice, this technique is cumbersome and inefficient for modeling a object shattering into hundreds of pieces.

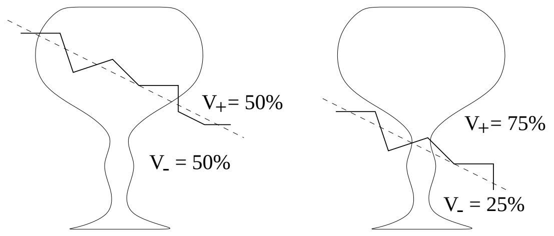

Figure 9. A glass broken into two pieces with different volume statistics

To overcome this problem, we have developed an automatic algorithm. The simulation is parameterized by both the distribution of the size of the fragments that will be created, and the global shape of the fragments. The designer controls whether the fragments should be long thin shards or roughly round pieces. The designer also defines the relative volume of the fragments produced by the cutting process.

At every step of the algorithm, we select the location and orientation of the fracture mask randomly. Then, given an initial object A and a fracture mask F, we automatically adjust the position and orientation of the mask so that the size and shape of the generated fragments $A \cap F$ and $A - F$ should conform to the size distribution and shape specifications provided by the designer (Figure 9).

This algorithm requires the evaluation of the volume of the fragments as well as the computation of their main directions. The original object is first converted into point cloud representation. This process is performed once and for all as a pre-processing step. The cloud point representation enables us to compute the volume and evaluate the shape characteristics of the fragments efficiently.

The point cloud representation is created as follows. We adaptively sample the object using an octree decomposition of space. Cells that are detected outside the object are skipped. If a cell is detected inside the object, as many points as needed are created depending on the level of the octree. Straddling cells are further subdivided until the maximum octree depth is reached. A point is created for a straddling cells at maximum depth if the field function value at the center $f(\mathbf{p})$ of the cell is positive. In our implementation, we used a combination of the Lipschitz techniques described in [6] and interval arithmetic [17] to characterize cells.

Volume computation The overall volume of the object is proportional to the overall number of points, denoted as n. When applying fracture masks F to create fragments, we simply classify points inside or outside the mask to compute the relative volume of the fragments. Let n^+ denote the number of points detected inside, the relative volumes are $\mathcal{V}_{A\cap F} = n^+/n$ and $\mathcal{V}_{A-F} = 1 - n^+/n$. This classification is performed efficiently by evaluating the field function value for all the points in the point cloud representation.

Fragment shape control The cloud point data-structure enables us to compute the principal axes of the object using the Karhunen-Loeve transformation. The principal axes of the point cloud are found by choosing the origin at the centre of gravity and forming the dispersion matrix computed as follows:

$$\sigma_{ij} = \frac{1}{n}\sum_{i=1}^{i=n}(\mathbf{p}_i - \bar{\mathbf{p}}_i)(\mathbf{p}_j - \bar{\mathbf{p}}_j)$$

The sum is over the n points of the sample and the $\mathbf{p}_i$ are the i$^{\text{th}}$ components of the point coordinates. $\bar{\mathbf{p}}_i$ stands for averaging. The principal axes and the variance along each of them are then given by the eigenvectors and associated eigen-values of the dispersion matrix.

Therefore, the designer can select the orientation of the fracture masks relatively to the principal axes of the original object, which enables him to intuitively control the global shape of the generated fragments (Figure 10).

5.3 Fracture regions

The designer may simply control the region where fractures will occur. Given a volumetric region denoted as R, fractures will be performed on the Hybrid Tree defined as the intersection $A \cap R$ whereas the difference $A - R$ will be preserved and kept crack-free.

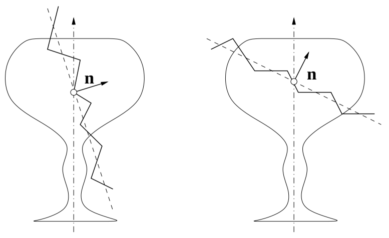

Figure 10. Controlling the shape of fragments by selecting the orientation of the fracture mask relatively to the principal axis of the shape

As for fracture masks, simple smooth and regular volumes such as spheres or boxes create smooth and straight cut patterns that do not look very realistic. Therefore, we have implemented a variety of template bumped and noisy regions that create more realistic fracture patterns. In our system, those volumes are defined by randomly perturbing the locations of the vertices mesh of spheres and ellipsoids along their vertex normal using a noise function.

5.4 Results

The images in Figure 1 show different models broken into pieces. The scotch glass (Figure 1, left image) was broken into 18 pieces. The volume ratio between fragments was constrained to 0.5 so as to get pieces of the same volume whereas the orientation of the cut was computed randomly so as to produce some long sharp thin fragments. The champagne glass (Figure 1, right image) was broken into 48 pieces using a volume ratio of 0.55, which produced fragments of roughly the same size. The principal cutting direction was automatically set orthogonal to the principal direction of the fragments so as to avoid long thin pieces. The statue model (Figure 1, center image) was shattered into 128 pieces using a volume ratio between fragments of 0.75.

Model	Pieces	Cloud	Fracturing	Triangles
Champagne	48	2:08	5:96	387 641
Glass	18	3:06	5:11	335 947
Statue	128	8:96	56:37	467 824

Table 2. Timings (in seconds) for generating the broken models in Figure 1

Table 2 reports the time needed to generate the Hybrid Tree models of the fragments. The timings for generating the point cloud generation is also reported. The total number of triangles of the fragments is also reported.

The images in Figure 11 show a sphere falling on the floor and breaking into 24 pieces. The fragments were computed automatically, and constrained to have almost the same volume and avoid long thin shapes. Fragments were first polygonized for fast rendering. The mechanical characteristics such as their mass, volume, inertia tensors were computed easily from the implicit surface representation using an octree decomposition of space. The animation was produced with a physically based animation system incorporating collision detection.

The images in Figure 12 show a wine flask hit by a solid sphere and breaking into 44 pieces. This simulation involved the same parameters as for the falling sphere.

6 Conclusion

We have presented some efficient procedural techniques for modeling cracks and fractures in solid materials such as glass, metal and stone. Our procedural approach provides the designer with simple parameterized tools to control the pattern of the cracks and the size and shape of fragments. Objects shattering into many interlocking fragments may be generated automatically. Both the shape and the size of synthetic fragments may be easily controlled by using a variety of fracture masks and crack patterns created after real world images.

In a future work, we plan to further investigate the design fracture masks and crack patterns with different levels of detail and generate fractured models at different resolutions. In particular, we plan to automatically generate textures from the geometry of the cracks to create realistic textures that will be used for display at a low level of details.

Acknowledgments

The authors would like to thank Olivier Galizzi for providing the animation and collision engine that produced the animations presented in this paper.

References

[1] R. Allègre, A. Barbier, S. Akkouche and E. Galin. A Hybrid Shape Representation for Freeform Modeling. *Submitted to Shape Modeling International 2003*.

[2] B. Cutler, J. Dorsey, L. McMillan, M. Mûller and R. Jagnow. A Procedural Approach to Authoring Solid Models. *SIGGRAPH 2002 Proceedings*, 302–311, 2002.

[3] S. Gobron and N. Chiba. Crack pattern simulation based on 3d surface cellular automata. *The Visual Computer*, **17**(5), 287–309, 2001.

[4] K. Hirota, Y. Tanoue and T. Kaneko. Generation of crack patterns with a physical model. *The Visual Computer*, **14**(3), 126–137, 1998.

[5] K. Hirota, Y. Tanoue and T. Kaneko. Simulation of three-dimensional cracks. *The Visual Computer*, **16**(7), 371–378, 2000.

[6] D. Kalra and A. Barr. Guaranteed ray intersections with implicit surfaces. *SIGGRAPH 1989 Proceedings*, 297–306, 1989.

[7] L. Kobbelt, M. Botsch, U. Schwanecke and H.-P. Seidel. Feature Sensitive Surface Extraction from Volume Data *SIGGRAPH 2001 Proceedings*, 57–66, 2001.

[8] O. Mazarak, C. Martins and J. Amanatides. Animating Exploding Objects. *Graphics Interface Proceedings*, 211–218, 1999.

[9] M. Mûller and L. McMillan and J. Dorsey and R. Jagnow. Real-Time Simulation of Deformation and Fracture of Stiff Materials. *Eurographics Workshop on Animation and Simulation*, 2001.

[10] M. Neff and E. Fiume. A visual model for blast waves and fracture. *Graphics Interface Proceedings*, 193–202, 1999.

[11] A. Norton, G. Turk, B. Bacon, J. Gerth and P. Sweeney. Animation of Fracture by Physical Modeling. *The Visual Computer*, **7**, 210–219, 1991.

[12] J. O'Brien and J. Hodgins. Graphical modeling and animation of brittle fracture. *SIGGRAPH 99 Proceedings*, 137–146, 1999.

[13] J. O'Brien, A. Bargteil and J. Hodgins. Graphical modeling and animation of ductile fracture. *ACM Transactions on Graphics*, **21**(3), 291–294, July 2002.

[14] E. Paquette, P. Poulin and G. Drettakis. Surface Aging by Impacts. *Graphics Interface Proceedings*, 2001.

[15] E. Paquette, P. Poulin and G. Drettakis. The simulation of paint cracking and peeling. *Graphics Interface Proceedings*, 59–68, 2002.

[16] J. Smith, A. Witkin and D. Baraff. Fast and Controllable Simulation of the Shattering of Brittle Objects. *Graphics Interface Proceedings*, 27–34, 2000.

[17] J. Snyder. Interval arithmetic for computer graphics. *SIGGRAPH 1992 Proceedings*, 121–130, 1992.

[18] D. Terzopoulos and K. Fleischer. Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. *SIGGRAPH 88 Proceedings*, 269–278, 1988.

[19] B. Wyvill, A. Guy and E. Galin. Extending the CSG Tree (Warping, Blending, and Boolean Operations in an Implicit Surface Modeling System). *Computer Graphics Forum*, **18**(2), 149–158, 1999.

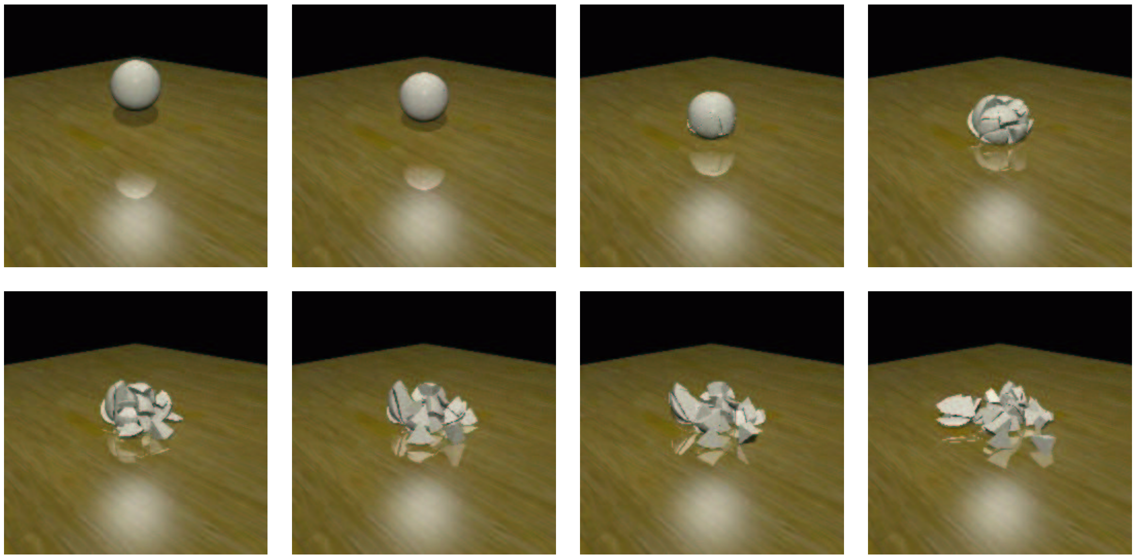

Figure 11. A ball breaking into pieces

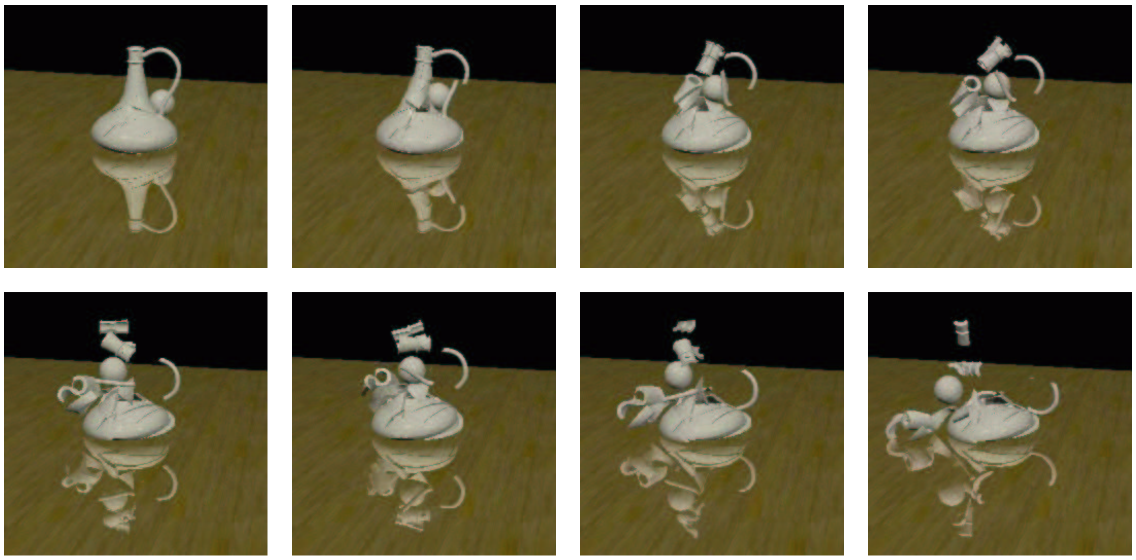

Figure 12. A flask breaking into pieces