

Procedural Modeling of Cracks and Fractures

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

†Artis-GRAVIR INRIA Rhône Alpes

‡LIRIS CNRS UCB Lyon 1

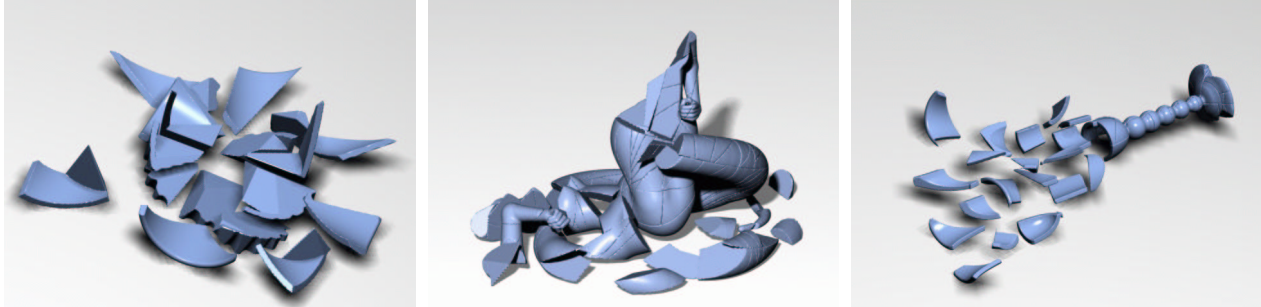


Figure 1. A scotch glass, a statue and a flute glass broken into 18, 128 and 48 fragments respectively

Abstract

This paper presents 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 fragments of different shapes.

Keywords: Procedural modeling, 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 those 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 crack propagation and fragments [10, 6, 9, 2]. Such methods are indispensable for correct and accurate simulation of shattering and breaking and have produced images

of striking realism. Specific techniques have been developed for simulating objects shattering induced by explosions [5, 6]. Other methods model static crack patterns on dry mud [4], and ceramics [3].

Physically based simulations often require the discretization of objects into voxels or tetrahedral meshes to compute internal forces. The discretization often leads to some artefacts in the crack patterns which makes fragments look not very realistic. Artefacts are the more visible as fractures are propagated along the boundaries of the initial mesh or voxel grid. Simulations are difficult to control, therefore their usage may be cumbersome for some applications and more simplistic, albeit less accurate, approaches may be useful.

This paper proposes an original procedural method for creating cracks and breaking objects into fragments. Our approach enables the designer to control the regions where cracks or fractures propagate and the shape of generated fragments. Our crack and fracture modeling system relies on the Hybrid Tree model [1] that combines skeletal implicit surfaces and triangle meshes in a constructive tree. Cracks and fragments are defined using incremental Boolean operations between the original model and carving volumes or fracture masks. Because the objects need not be voxelized or tetrahedralized as for physically based techniques, we are not limited in resolution when creating fragments and our method can create small thin shards easily. Our algorithm is simple and efficient, allowing the designer to break an object interactively.

2 Modeling cracks

The creation of cracks on the surface of an object is performed in three steps (Figure 2). First, the designer defines a two dimensional crack pattern in a specific graph editor. Then, the graph is mapped onto the surface of the original object to create a geometric skeleton. The carving volume is defined by sweeping a profile curve along this skeleton, and cracks are created by using Boolean difference operations between the original input model and carving volume.

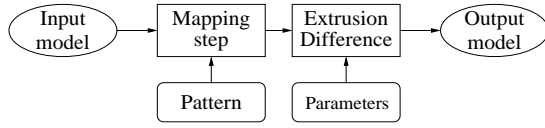


Figure 2. Simulation cycle of the cracking process

Crack pattern In our system, cracks are designed after images using a specific editor to obtain realistic patterns. 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. Therefore, the graph implements the paths followed by a turtle in a plane.

Mapping The designer simply needs to project one point of the graph onto the surface of the original object. The whole graph is then automatically transformed into a three dimensional skeleton derived from the turtle geometry representation [7] by applying a surface marching algorithm, using the relative directions and length from the data stored at the nodes and edges of the graph.

The turtle may generate a self-intersecting skeleton. 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.

Volume generation The 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. In our modeling system, carving volume are characterized as the union of prism, tetrahedral and pyramidal implicit primitives (Figure 3).

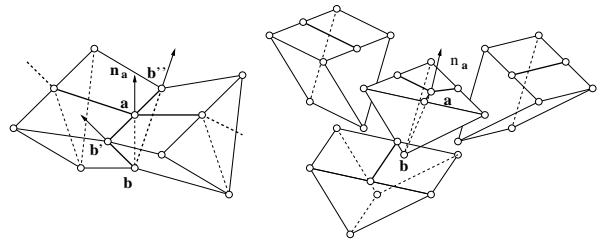


Figure 3. Carving volume generated by a line segment of the crack skeleton

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 $\mathbf{a} \pm w\mathbf{t}$. Eventually, vertices \mathbf{b}' and \mathbf{b}'' are raised above the surface by a user controlled offset distance to avoid artefacts in convex regions of high curvature.

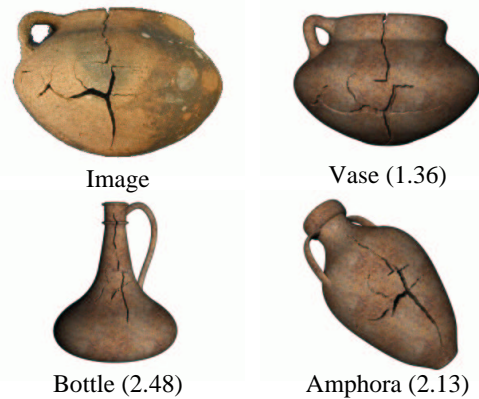


Figure 4. A real clay vase (upper left) and some synthetic models created after the original image

Results Figure 4 shows a comparison between a real broken vase made of clay, and some synthetic model created with our method. The paths of the cracks on the synthetic models were created after the original image. The depth of the cracks were adapted to the thickness of the original model automatically so that cracks should pierce the model and not only create surface scratches. Reported timings (in seconds) include the mapping of the crack pattern and the volume generation process.

3 Modeling fractures

A simulation starts with the selection of a fracturing tool that is characterized by a fracture mask defining the cutting profile. The fracturing tool T is applied to the object A to break it into two fragments by computing the Boolean intersection $A \cap T$ and difference $A - T$ between the original object and fracture mask. The fragments are expressed as Hybrid Trees and can be further broken into pieces by repetitively applying the selected tool (Figure 5).

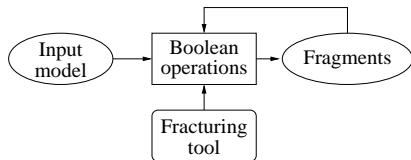


Figure 5. Simulation cycle of our fracturing system

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.

Fracture masks Fracture masks are solids that define the profile of the fracture between two fragments. The fracture masks are positioned relatively to the original objects so that it should embed part of it and cut it into parts. Fracture masks not 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. Simple skeletal implicit primitives such as ellipsoids, boxes or half spheres create fragments with straight and smooth cut patterns. Our system also implements height field primitives to create more complex fracture profiles. Those primitives enable the designer to reproduce realistic surface characteristics in terms of profile and roughness after real world examples.

Fracture regions In our system, the designer controls the regions where fractures will occur by constraining fractures to a limited volume. 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.

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.

Controlling the shape of fragments An original object may be broken interactively by editing the position and orientation of the fracture masks in space to control the shape of the fragments. While this approach provides the designer with a tight, although low level, control over the shape of the final broken object, it is cumbersome and inefficient for modeling an object shattering into hundreds of pieces.

To overcome this problem, we have developed an algorithm that automatically breaks an object into a set of fragments whose size and shape are controlled by the designer. The simulation is controlled by two main parameters. The volume ratio between two fragments, denoted as ΔV , characterizes the distribution of the size of the fragments. The relative shape of fragments is defined by a parameter denoted as α that defines whether fragments should be long thin shards or roughly round pieces. Indeed, α represents the angle between the main direction of an object and the normal of the main direction of the cutting tool. Therefore, the designer can select the orientation of the fracture masks relatively to the principal axes of the original object, which enables him to control the global shape of the generated fragments (Figure 6).

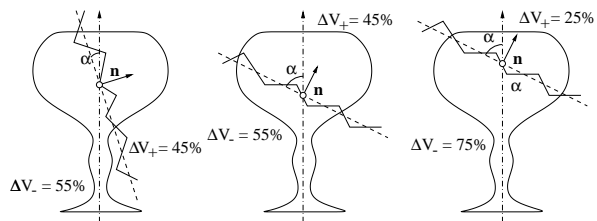


Figure 6. Controlling the shape of fragments by selecting the orientation of the fracture mask relatively to the principal axis of the shape

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 T , we automatically adjust the position and orientation of the mask so that the size and shape of the generated fragments $A \cap T$ and $A - T$ should conform to the parameters ΔV and α prescribed by the designer (Figure 6).

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 a point cloud representation. This process is performed once and for all as a pre-processing step. 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 cre-

ated depending on the level of the octree. Straddling cells are further subdivided until the maximum octree depth is reached.

The volume of the object is proportional to the total number of points, denoted as n . We simply classify points inside or outside the mask to compute the volume of the fragments. Let n^+ denote the number of points detected inside, the 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.

The point cloud representation is also used 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^{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.

Results The images in Figure 1 show a scotch glass, a statue and a champagne flute glass broken into 18, 128 and 48 pieces respectively. Table 1 reports the time needed to generate point cloud representation and the Hybrid Tree models of the fragments, as well as the volume ratio ΔV and the angle α parameters.

Model	Pieces	Cloud	Break	ΔV	α
Flute	48	2.08	5.96	0.55	$\pi/2$
Glass	18	3.06	5.11	0.50	$[0, \pi/2]$
Statue	128	8.96	56.37	0.75	$[\pi/4, \pi/2]$

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

The volume ratio between fragments of the scotch glass was constrained to 0.5 so as to get pieces of the same volume. The orientation of the cut was computed randomly in $[0, \pi/2]$ to produce some long thin fragments. The champagne flute glass was broken using a volume ratio of 0.55 to produce fragments of roughly the same size, and the principal cutting direction was automatically set orthogonal to the principal direction of the fragments so as to avoid long thin pieces.

4 Conclusion

We have presented an efficient procedural approach for modeling cracks and fractures in solid materials. The designer can control the pattern of the cracks and the size and shape of fragments easily. Objects shattering into many interlocking fragments may be generated automatically.

In the near future, we plan to investigate the creation of fracture masks and crack patterns with different levels of detail to generate fractured models at different resolutions. We also 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.

References

- [1] R. Allègre, A. Barbier, S. Akkouche and E. Galin. A Hybrid Shape Representation for Freeform Modeling. *Shape Modeling International*, 2004.
- [2] B. Cutler, J. Dorsey, L. McMillan, M. Müller 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. Simulation of three-dimensional cracks. *The Visual Computer*, **16**(7), 371–378, 2000.
- [5] O. Mazarak, C. Martins and J. Amanatides. Animating Exploding Objects. *Graphics Interface Proceedings*, 211–218, 1999.
- [6] M. Müller, L. McMillan, J. Dorsey and R. Jagnow. Real-Time Simulation of Deformation and Fracture of Stiff Materials. *Eurographics Workshop on Animation and Simulation*, 113–124, 2001.
- [7] L. Mundermann, P. MacMurchy, J. Pivovarov and P. Prusinkiewicz. Modeling Lobed Leaves. *Computer Graphics International Proceedings*, 2003.
- [8] M. Neff and E. Fiume. A visual model for blast waves and fracture. *Graphics Interface Proceedings*, 193–202, 1999.
- [9] 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.
- [10] J. Smith, A. Witkin and D. Baraff. Fast and Controllable Simulation of the Shattering of Brittle Objects. *Graphics Interface Proceedings*, 27–34, 2000.