Procedural Modelling of Destructible Materials

J. van Gestel[1] and R. Bidarra[2]

[1] Cannibal Game Studios, The Netherlands
[2] Computer Graphics Group, Delft University of Technology, The Netherlands

ABSTRACT

Traditional content creation for computer games is a costly process. In particular, current techniques for authoring destructible behaviour are often limited to a single object basis. In this paper, we build on previous research results to develop a novel method for designing reusable destructible behaviour which can be applied in real-time to a variety of objects. To separate the destructible behaviour from particular objects, we introduce the concept of destructible materials: where the material of an object usually defines the way an object looks, a destructible material will determine how it breaks. Destructible materials provide a reusable definition and intuitive way of designing and tweaking destructible behaviour *of objects in game development, which can then be applied in real-time.*

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Geometric algorithms, languages, and systems; Additional Key Words and Phrases: procedural destruction, games, fractures, cracks.

1. Introduction

With the large increase of computing power available in the latest generations of gaming consoles and personal computers, comes an equally large increase in the demand for higher quality graphics and game play. To satisfy this demand, more and higher-detailed content is required as larger game worlds need to be created in higher fidelity. The traditional approaches of creating content manually do not scale well, resulting in large cost increases for productions. This is particularly so in the area of destruction, where normally several variations of an object have to be produced in order to create a convincing result.

Procedural approaches for creating content are therefore becoming increasingly attractive, as they reduce the amount of manual labour required to create a larger amount of content. Application of procedural approaches for creating e.g. terrain, buildings and trees is becoming more common. However, procedural destruction has not yet found its way into mainstream game development.

Several reasons can be pointed out for this. Maybe the most important is that for designers the ease of use of a procedural approach, and particularly, the degree of control it provides them is determinant for its adoption, in addition to the obvious aspects of performance and graphical quality. In contrast, whether the approach actually produces a (physically) realistic result is not much of an issue, as long as the result is convincing.

In this paper, we present a reusable method for authoring specific destructible behaviour, which can then be applied to a variety of objects in real-time. This destructible behaviour was designed building on the basic approach presented by Martinet et al [MGD*04], in which a designer can specify the definition of cracking and fracturing patterns using graphs. From these definitions, volumes are derived and combined with the original model using Boolean operations to create cracks or fractures. This approach provides a visual and intuitive way of designing destructible behaviour. A destructible material then consists of a collection of specific crack and fracture patterns.

Section 2 presents a short overview of existing techniques for simulating object destruction. Section 3 presents the main elements of the cracking and fracturing approach. Sections 4 and 5 present our method for authoring destructible materials and briefly describe our prototype system implementation. Section 6 discusses some results produced by the prototype, and Section 7 draws some conclusions.

2. Related work

The simulation of the physical process of cracking and fracture of objects has been a widely studied area in computer graphics. O'Brien and Hodgins [OH99] propose a method for modelling and animating brittle fractures by analysing the stress tensors computed over a finite element model. Müller et al [MHT*05] extend upon the work of O'Brien and Hodgins, reducing the complexity of the fine element model by using cubical elements instead of tetrahedral ones. They further improve the performance by using a single set of global material properties instead of per element. Galoppo et al [GOM*06] proposes a method that also uses a finite element model, however, they solve the simulation entirely on the GPU. Utilizing the enormous processing power of the GPU their simulation is able to easily simulate deformations for objects. The finite element

model provides realistic results, but methods using this technique are often too computationally complex to be used in real-time applications, lack intuitive ways of editing or do not offer enough control for designers.

Instead of simulating a physical process it is also possible to simply replicate its effects, without trying to replicate the process that leads to the effect. Müller et al [MHT*04] present a method for mesh-less deformations of objects, replacing the internal energies of the finite element models with geometric constraints and forces by distances of current positions to goal positions. Objects are considered as point clouds, where each point is moved under influence of external forces. Clusters of points are then moved a certain distance back to their original position, providing a fast and mathematically stable method for simulating even extreme deformations. Rivers and James [RJ07] extend upon the work of Müller et al, improving both the quality of the deformations and the computational complexity of the algorithms. By removing points from the simulation that become too far removed from their neighbours, they extend the approach with fracturing of objects. Shape matching methods are more suited for real-time applications; however, like the finite element-based methods, they lack appropriate editing control for designers.

Smith et al [SWB01] try to combine the advantages of the accuracy of the finite element model approaches with the lower computational complexity of a more geometric approach. They generate a finite element model but instead of propagating forces over the elements in the model, they use rigid springs to form distance constraints between model elements. This method is very similar to the finite element-based methods and, therefore, suffers from the same drawbacks.

In short, despite valuable research progress, current methods are still far from widely applied in a gaming context and, in particular, they still fail regarding performance, reusability, and/or designer-friendliness.

3. Basic approach

Contrary to the approaches surveyed in the previous section, the approach presented by Martinet et al. [MGD*04] [DGA05] offers designers more control and interactive editing capabilities. Their method is therefore an excellent basis for the destructible material approach presented in this paper, and therefore it will be first discussed here in some more detail.

The method presented by Martinet et al. allows for the procedural cracking and fracturing of objects. Cracks are created by performing Boolean difference operations between carving volumes and an input model. A carving volume is generated from a graph representing that crack, which has been projected onto the surface of the target object. Fractures are created by performing Boolean intersection and difference operations between fracture masks and an input model. As modelling many fragments by hand is not feasible they propose a technique that automatically creates a user-defined amount of fragments with a specified distribution in shape and size by iteratively applying fracture masks to the input object(s).

Martinet et al. use a Hybrid Tree [AGC*08] model as their main data structure. A Hybrid Tree is a tree-like data structure that holds implicit primitives or triangle meshes at its leaves, which are combined by Boolean, blending and warping operators located at the intermediate nodes of the tree.

The graphs representing *cracks* model the branching structure of the crack. Each node holds the profile information of the crack, i.e. the width and depth of the crack. When applying a crack the graph is projected onto the surface of an object to form a geometric skeleton. Either a profile curve is then swept along this skeleton, or implicit skeletal primitives such as tetrahedrons are used, incorporating the information stored in the nodes to create a carving volume. The carving volume is characterized as a Hybrid Tree and combined with the input object into another Hybrid Tree, where it is cracked using a Boolean difference operation.

Fractures, on the other hand, are defined by a fracture mask and two simple parameters. The fracture mask represents the crack between two fragments, and is implemented using primitives with level of details. This allows for both smooth and noisy cut patterns. Fracturing an object into many pieces would require the placement of many of these fracture masks, which would be a tedious process if done by hand. Here, in contrast, a designer can control the procedure using two simple parameters, which allow him to somehow change the shape and size of the fragments. The

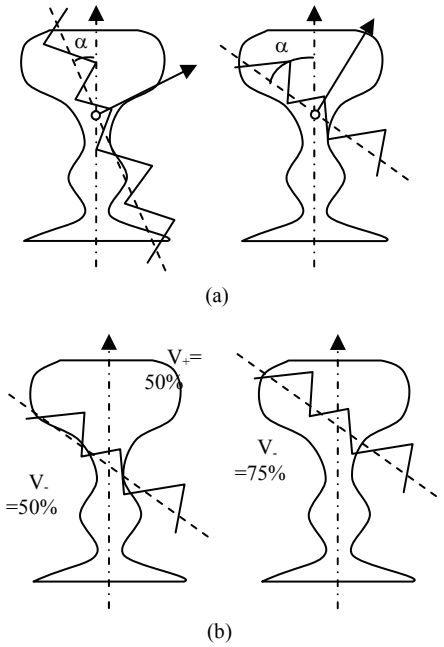

Figure 1 *The shape of fragments can be controlled by adjusting (a) the angle α of the fracture pattern relatively to the principal axis of the object, and (b) the relative volume ratio determining the overall size of fragments. From [MGD*04].*

first parameter, α, determines a rotation of the fracture mask around the principal axis of the object, see Figure 1.a. This allows for creating either long, thin shards or roughly 'round' fragments. The second parameter, V, determines the volume ratio between the two fragments, see Figure 1.b. This allows for creating fragments that can range from all being roughly the same size to having a mix of large and small pieces. By recursively applying the fracture to the fragments many interlocking pieces can be created.

4. Application in game development

The basic method described in Section 3 offers control over the shape and application of cracks and fractures, and allows for some interactive editing capabilities. However, it was not conceived to be applied in game development, and it therefore still lacks some essential features for that purpose, namely *reusability* and *real-time* processing.

To increase the reusability we introduce the concept of *destructible materials*. Analogously to how a material defines how something looks, a destructible material determine how something breaks apart. We define a destructible material as a collection of several crack and/or fracture patterns. These patterns are reusable by several materials, and materials are reusable by several objects. Figure 3 illustrates this hierarchy, structuring the different elements that define the destructible behaviour. The lower level contains the most elementary definition of crack or fracture patterns, either a graph or the definition of the cross section of a fracture. These elements can be created and edited individually, after which they can be combined into actual cracks or fractures, at the second level. The cracks and fractures contain further information like the size of the crack and the fracture parameters.

Because all components are reusable, once a library of cracks, fractures and materials has been created, making any object destructible will be as simple as selecting and assigning the desired destructible material, significantly reducing the amount of time required for creating content.

Combining the intuitive method described by Martinet et al. with a real-time implementation, allows designers to receive real-time feedback on the designs they are creating.

How a destructible material is actually used will depend on the specifics of the game development project it is used in. That, in turn, determines which selection can be made of which crack or fracture patterns, for example based on various kinds of events like impact type or force, or the remaining structural integrity of the object.

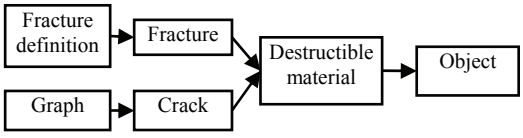

Figure 2 *Destructible materials definition spans several levels, each level deploying instances of the previous one*

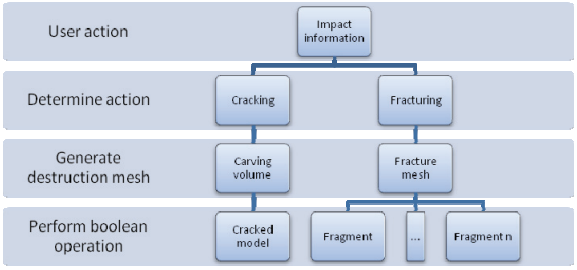

Figure 3 *High level overview of the entire destruction procedure*

5. Prototype system

We implemented this approach to destructive materials in our own prototype system. The system includes a real-time implementation of the method described by Martinet et al., as well as an editor for the interactive specification of destructive materials, a module for quick visualization of the results of destruction operations, and a destructive material library manager. In this section we briefly describe several implementation aspects of the system, with a special focus on the cracking and fracture algorithms. The reader is referred to [Ges11] for a more comprehensive discussion of the whole approach.

A destruction operation consists of four main steps: (i) user input, (ii) action determination, (iii) destruction mesh generation and (iv) Boolean operation. In the first step, the user triggers the start of the operation. This provides information like a ray specifying the location and direction of an impact, and the strength of that impact. In the second step, depending on the impact information, a decision is made to whether cracking or fracturing of the object should take place. In the third step, either a carving volume (for crack) or fracture mesh mask (for fracture) is generated. Finally, in the Boolean operation step, the input model is then combined with the generated mesh, to yield the resulting object(s); see Figure 3.

5.1 Cracking

A cracking operation is initiated by casting the input ray onto the target object, to determine an impact polygon. This ray could for instance represent a projectile striking the object. To efficiently retrieve the exact polygon impacted by the ray, an octree decomposition of the polygons forming the model is pre-computed.

A surface marching algorithm is then applied to wrap the crack graph around the model, starting from the found impact polygon. Boundary representation information is pre-computed for the model, in order to efficiently retrieve neighbouring polygons during the surface marching,

The information stored in each node of the graph, i.e. the width and depth of the crack, is directly used to generate a triangular carving volume, so as to minimize the number of polygons that will be involved in the Boolean operation. To avoid generating self intersecting geometry, the geometric skeleton generated by the surface marching algo-

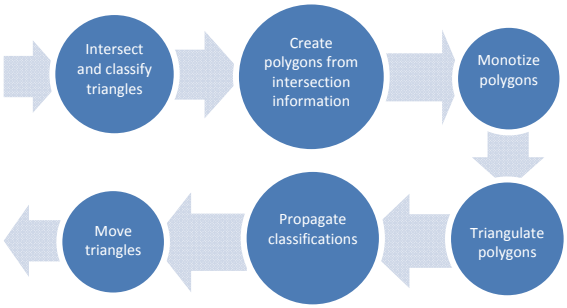

Figure 5 *High level overview of the Boolean operation algorithm*

rithm is first filtered to resize or remove any nodes that lie too close together.

For the Boolean operation implementation, we followed the method presented in [Hub90], see diagram in Figure 5. Using the pre-computed octree, triangles can quickly be retrieved, dramatically reducing the amount of triangle-triangle intersection calculations. Line segments resulting from the triangle-triangle intersections are efficiently calculated using the technique described in [Mol97]. All line segments are turned into polygons using a custom solving algorithm, after which they are turned monotone using [Rou98] in O(n log n) time. Once all polygons are monotone, they are triangulated in linear time using [BKO*00].

Finally, the classifications determined during the intersection phase are propagated to all polygons that were not involved in any intersections. Any polygons classified as being outside of the target object are removed from the model, and triangles from the carving volume classified as being inside the target object are copied into the model to close the gap. By replacing polygons in place in the model, the Boolean operations become independent of the polygon count of the model, and depends only on the surface area covered by the carving volume.

5.2 Fracturing

A fracturing operation requires fewer steps than a cracking operation, as no graph has to be wrapped on the target object. From the fracture pattern we are able to directly generate a mesh, by extruding the cross-section definition in a direction perpendicular to its plane; see Figure 6. This mesh is then transformed using a transformation matrix that contains the two parameters for the rotation, α, and relative volume ratio, V, (see Figure 1, in Section 3), as

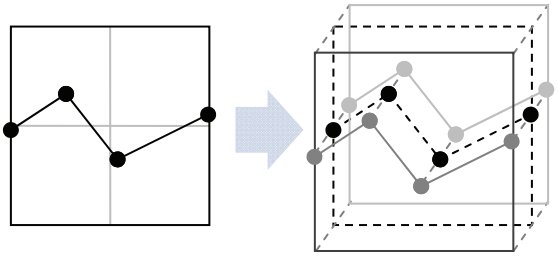

Figure 6 *Fracture pattern extruded to a three dimensional fracture mesh*

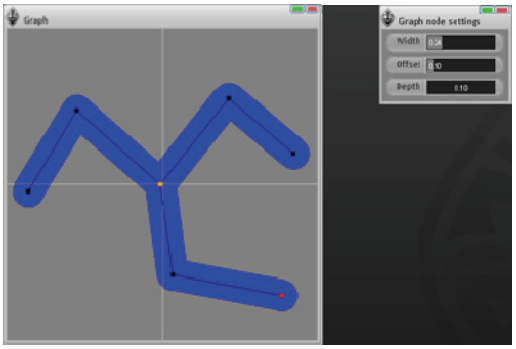

Figure 4 *Interactive node adjustment of a crack pattern*

well as a scaling factor that is used to assure the mesh fully intersects the target object. To determine the principal axis of the target object, a dispersion matrix is calculated from all vertices that form the target object. Using a Jacobian decomposition, the three eigenvectors are calculated for the dispersion matrix, which are directly used to form the rotation basis of the transformation matrix.

6. Results

The prototype editor enables a designer to interactively design crack and fracture patterns. Visualizing the patterns as they are made allows a designer to tweak any input values on the fly, as e.g. the width or depth of a crack node; see Figure 4. The interactive click-and-drag nature of the editor allows a designer to create simple patterns in about a minute, while more complex patterns only take a few minutes.

The real-time nature of the presented approach enables the designer to preview the destructible material being created at any phase. This allows for quickly verifying whether the destructible material designed produces the desired results, or making further changes if required. A simple destructible material can be created in as little as ten minutes.

We now show several examples worked out in our prototype system. For this, we will use three base models (see Figure 7): a door, a massive (solid) vase, and a hollow (thin) vase. Figure 8, for example, shows how these three objects are cracked using the 3-pronged crack pattern from Figure 4 for their destructive material. Figure 9, finally, depicts the same objects being fractured, using the fracture pattern of Figure 6 for their destructive material. In each pass, the algorithm is applied on the result of the previous iteration.

We made several performance analysis of our implementation and found out that the fracturing algorithm is by far much less expensive than the cracking algorithm. For the latter, performance profiling determined that only the Boolean operation significantly impacts the performance, as all other steps combined only require two milliseconds or less on average to complete. Table 1 reports the duration for the cracking operations on the models showed in Figure 8. Timings were gathered on a standard desktop PC with an Intel core Duo 6600 @ 2.4 GHz.

Figure 7 *Base models for the examples in this section: a door (208 triangles), a solid vase (288 triangles) and a thin vase (576 triangles)*

Figure 8 *Cracking of different object using the same destructible material*

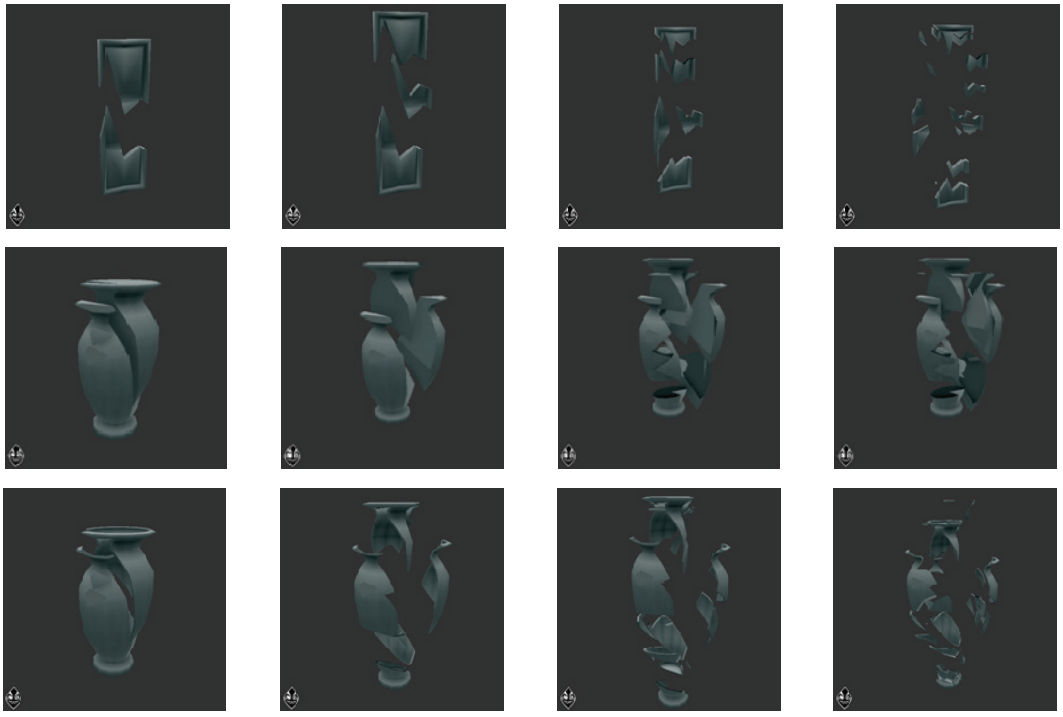

Figure 9 *Fracturing of different object using the same destructible material: successive operations are performed on the result of previous ones*

Model	Triangles	ms
Door	208	42
Solid vase	288	17
Thin vase	576	40

Table 1 *Triangle counts and duration times for the cracking operations show in Figure 8.*

We further observed that the triangle intersection method takes on average half of the time of the whole Boolean operation; see diagram in Figure 5. Twenty five percent of the time is needed to move triangles between the models; and the remaining twenty five percent is taken by the rest of the steps, like propagating classifications and updating the boundary representation. Please refer to [Ges11] for a more thorough treatment and analysis of all performance measurements.

7. Conclusion

In this paper we have introduced the novel concept of *destructible materials*, which builds on an existing intuitive approach for procedural destruction, and extends it with pattern reusability and high performance, making it suitable for use in game development settings. Our prototype implementation shows that destructible materials allows for easy creation and testing of reusable destructible behaviour, which can be applied to a variety of objects.

Our current implementation can solve only simple cases in real-time, i.e. under $1/60^{th}$ of a second. However, most computational demands arise from the Boolean operation algorithm, which is an area of the prototype where many optimizations can be implemented. As fracturing an object consists of multiple applications of the same pattern, these applications can be distributed along several frames, allowing for a smooth user experience.

In future work, we plan to further investigate these and other performance constraints in order to actually apply the destructive material approach in our game development projects. In particular, we are considering approaching the triangle-triangle intersection calculations, by far the most computationally demanding, with multithreaded, or even hyper-threaded methods, as for example GPU-based solutions. Because the individual intersection calculations are disjoint a large speed-up should be attainable.

References

[AGC*08] ALLÈGRE R, GALIN E, CHAINE R, AKKOUCHE S: The Hybrid Tree: A Hybrid Constructive Shape Representation for Free-Form Modelling. *Heterogeneous Objects Modelling and Applications* 2008: 60-89

[BKO*00] BERG M DE, KREVELD M VAN, OVERMARS M, SCHWARZKOPF O: *Computational Geometry*. 2nd edition. Springer-Verlag, 2000, pp. 45-61

[DGA05] DESBENOIT B, GALIN E, AKKOUCHE S: Modeling cracks and fractures. *Visual Comput.* 21:717-726 (2005)

[Ges11] GESTEL VAN J: Procedural Destruction of Objects for Computer Games. MSc thesis, Delft University of Technology, 2011

[GOM*06] GALOPPO N, OTADUY MA, MECKLENBURG P, GROSS M, LIN MC: Fast simulation of deformable models in contact using dynamic deformation textures. *Symposium on Computer Animation* 2006: 73-82

[Hub90] HUBBARD PM: Constructive Solid Geometry for Triangulated Polyhedra. 1990

[MGD*04] MARTINET A, GALIN E, DESBENOIT B, AKKOUCHE S: Procedural Modeling of Cracks and Fractures. SMI 2004: 346-349

[MHT*04] MÜLLER M, TESCHNER M, GROSS M: Physically-Based Simulation of Objects Represented by Surface Meshes. *Computer Graphics International* 2004: 26-33

[MHT*05] MÜLLER M, HEIDELBERGER B, TESCHNER M, GROSS M: Meshless deformations based on shape matching. *ACM Trans. Graph*. 24(3): 471-478 (2005)

[Mol97] MOLLER T: A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 1997: 25-30

[OH99] O'BRIEN JF, HODGINS JK: Graphical Modeling and Animation of Brittle Fracture. SIGGRAPH 1999: 137-146

[RJ07] RIVERS AR, JAMES DL: FastLSM: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph*. 26(3): 82 (2007)

[Rou98] O'ROURKE J: *Computational Geometry in C*. Cambridge University Press, 1998

[SWB01] SMITH J, WITKIN AP, BARAFF D: Fast and Controllable Simulation of the Shattering of Brittle Objects. *Comput. Graph. Forum* 20(2): 81-90 (2001)