An Efficient Algorithm for Brittle Fracture Simulation

Miles Barr*
Student at the University of Victoria

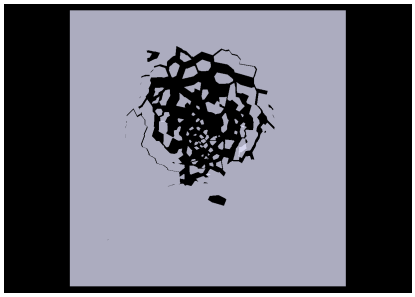

Figure 1: *Glass shattering*

Abstract

In the this paper, I propose an algorithm for brittle fracture simulation based on Voronoi diagrams. The algorithm allows for dynamic holes and recursive fractures of a mesh with high computational efficiency. These results are attained by precomputing all the possible fracture paths and then running a simple crack propagation model at runtime. The algorithm can be run in real-time because it avoids complex runtime operations such as mesh slicing and re-meshing. Additionally, configurable parameters allow a variety of materials to be simulated. I demonstrate the algorithm by shattering a flat plane using several materials.

Keywords: brittle fracture, shatter, rigid body, cracking

Concepts: •**Computing methodologies** → **Physical simulation;**

1 Introduction

Interested in brittle fracture simulation, I began researching existing papers on the topic. Many papers have been written variations of brittle fracture simulation, but most can be placed into two categories. The first is those with complex tension models that try to simulate real-world forces. These usually produce nice results, but are slow, confusing, and difficult to implement. The second is those which are overly simplified, using patterns to cut a mesh. These algorithms are typically easier to understand, but can still hardly be called efficient.

I think it is important to take a step back and look at the uses of brittle fracture simulation. Few cases necessitate true physical simulation. Industrial applications might require modeling of physical stresses, but in these cases running time is not a concern. On the other end of the spectrum, brittle fracture simulation is more common in computer animation and video games. In computer animation and video games true physical accuracy is not the highest concern. In both disciplines, having a realistic result is more important than how that result is achieved. Often, an artificial approach can produce results which are visually indistinguishable from a physically based one. In video games, computational efficiency is extremely important, and is the main reason true dynamic destruction is rarely seen.

In this paper, I propose a method for brittle fracture simulation which solves these problems. The method is intuitive, produces nice results, and is very efficient—enough to be run in real-time, at a large scale. The algorithm allows for holes and chunks to be broken off a mesh, all while avoiding both the problems of real-time mesh slicing and re-meshing. The algorithm is also generalizable to both 2D and 3D models.

Author's note: In this paper, the terms 'volume,' 'faces,' and 'mesh' can be used interchangeably with the terms 'area,' 'lines,' and 'graph' to translate between a 3D and 2D versions of the algorithm.

2 Literature Review

The most cited paper on brittle fracture simulations is *Graphical Modeling and Animation of Brittle Fracture* by O'Brien and Hodgins [O'Brien and Hodgins 1999]. Their approach models the stresses in an object based on impact forces and numerous material parameters. Their algorithm produces very realistic results, which are demonstrated using several models and materials. However, the drawback of their approach is that the algorithm is far too slow for real-time simulations.

3 Phases and Running Times

The proposed algorithm is split into two independent phases: precomputation of a shatter diagram, and surface propagation on impact. The precomputation phase only needs to be run once, regardless of the future subdivisions of a model. The surface propagation phase is run at every impact on a model or its residual fragments from other impacts.

The big-O running times of both phases are independent to the number of vertices in a mesh. Instead, the running times depend on a *shatter resolution* value H which specifies the number of cells that make up the original mesh. The precomputation stage runs in $O(H \log H)$ time, and the surface propagation stage runs in $O(H)$

*e-mail:milesbarr2@gmail.com

SIGGRAPH 2016 Posters, July 24-28, 2016, Anaheim, CA
ISBN: 978-1-4503-ABCD-E/16/07
DOI: http://doi.acm.org/10.1145/9999997.9999999

time. It is important to keep in mind that the shatter resolution can be set as small or as large as one wants, and for most purposes small values are effective. The shatter resolution value can also be defined dynamically to account for a particular computer's performance.

4 Precomputation

The precomputation stage constructs a *shatter diagram* to be used for all future subdivisions of a mesh. The stage is called *precomputation* because the shatter diagram does not need to be computed on an impact or even at runtime. The shatter diagram can be generated separately and stored alongside a mesh.

The first step for computing the shatter diagram is to place a number of points equivalent to the shatter resolution within a mesh's volume. These points can be placed randomly, or by other methods, but should generally be uniformly distributed.

A Voronoi diagram bounded by the mesh is generated for the list of points. The Voronoi diagram algorithm extracts a list of faces which correspond to each Voronoi site. Voronoi cells which intersect the mesh's surface will partially contain the internal Voronoi faces, and partially contains the intersected faces of the mesh. These pairs of Voronoi sites with the surrounding faces will be called *cells* and are the smallest unit which the final object can be broken into. Cells should also store a list of neighbouring cells, which are cells that shares a face.

The precomputation stage can be summarized as follows:

1. Place H random points within a mesh's volume.
2. Generate a Voronoi diagram bounded by the mesh for the points.
3. Extract a list Voronoi sites paired with their surrounding faces and neighbours.

5 Surface Propagation

The surface propagation stage involves shattering a mesh into fragments, or more accurately, fragments into smaller fragments. Fragments consist of a list of cells, a material, and properties needed for rendering. Additionally, fragments are rigid bodies with physical properties such as position, velocity, and mass. The initial model should be represented as a fragment which contains all the shatter diagram's cells.

5.1 Impact Properties

Fragments are broken into smaller fragments by an impact force applied at an impact point. The impact force is a dimensionless quantity which is scaled by the shatter durability of the impacted fragment. The impact point is a point within the impacted fragment's volume which receives the full impact force. The distribution of the impact force across a fragment is determined by its material.

5.2 Materials

A material consists of two properties: shatter durability and shatter locality. The shatter durability represents an object's resistance to crack propagation in relation to local impact forces. The shatter locality represents the focused distribution of shatter forces towards an impact point. Combined, these can represent the brittle fracture properties of many compounds. For example, standard glass has a moderate shatter durability and a moderate shatter locality; cracks propagate but do not shatter the entire object. Tempered glass however, has a high shatter durability and low shatter locality; the material is resistant to shattering but entirely shatters once enough force is applied.

5.3 Impact Distance

To calculate local impact forces we need the distance between the impact point and the center of every cell. There are several ways to define the center of a cell, but for simplicity we will use the site which defined the Voronoi cell.

5.4 Impact Factor

Upon impact, an *impact factor* is computed for every cell in the impacted fragment. The impact factor is the relation between the impact force, distance, and the shatter locality of the material.

We will define the impact factor for a given cell i as follows:

$$X_i = F/(d_i^L + 1)$$

X_i is the impact factor for a given cell i.

F is the impact force.

d_i is the distance between the impact point and the center of a given cell i.

L is the shatter locality constant of a material.

The equation dissipates force with distance at a rate defined by the material's shatter locality constant. One is added to the denominator to avoid division by zero.

5.5 Shatter Durability

The shatter durability is a threshold for the difference in impact factors between neighbouring cells. A crack appears between two cells when the difference in impact factors is greater than their material's shatter durability. However, cells will stay connected without cracks if they are connected by a chain of other cells after the impact.

The following equation determines if crack will appear between two neighbouring cells i and j.

$$C = \begin{cases} 0, & \text{if } |X_i - X_j| \leq U \\ 1, & \text{if } |X_i - X_j| > U \end{cases}$$

6 Data Structures

The algorithm uses two main data structures: fragments and cells. Fragments consist of a list of cells, a material, rigidbody properties, and rendering properties. Cells consist of the site which defined its Voronoi cell, a list of faces, a list of neighbours, and a Boolean visited value. Additionally, a material data structure acts as a container for the shatter durability and shatter locality properties.

7 Pseudocode

Algorithm 1 Shatters a fragment into smaller fragments

Input: Impacted fragment S, impact point p, impact force F
Output: List of fragments

```
1:  function SHATTER–FRAGMENT(S, p, F)
2:      for all cells s in S.cells do
3:          s.visited ← false
4:          for all neighbours n of s do
5:              if n.fragment ≠ S then
6:                  Remove n from s.neighbours
7:              end if
8:          end for
9:      end for
10:     L ← empty list of fragments
11:     for all cells s in S.cells do
12:         if not s.visited then
13:             R ← Extract–Subfragment(s, p, F, S.material)
14:             Copy any additional properties from S to R
15:             L.add(R)
16:         end if
17:     end for
18:     return L
19: end function
```

Algorithm 2 Extracts a separated sub-fragment from an impacted fragment

Input: Cell s, impact point p, impact force F, material M
Output: Separated fragment

```
1:  function EXTRACT–SUBFRAGMENT(s, p, F, M)
2:      s.visited ← true
3:      R ← new fragment
4:      R.material ← M
5:      R.cells.add(s)
6:      s.fragment ← R
7:      for all neighbours n of s do
8:          X_s ← Shatter–Factor(s, p, F, M)
9:          X_n ← Shatter–Factor(n, p, F, M)
10:         diff ← |X_s − X_n|
11:         if diff > M.durability then
12:             R ← R.merge(Extract–Subfragment(n, p, F, M))
13:         end if
14:     end for
15:     return R
16: end function
```

8 Implementation Details

8.1 Memory Allocation

Cells can be efficiently allocated on the heap because they do not need to be reallocated on subsequent impacts. Only a single fragment can contain a given cell so cells pointers can be safely passed down to sub-fragments.

8.2 Rendering

For efficient rendering, fragments should store vertex arrays. After an impact, a fragments can build a vertex array by looping over its cells and adding faces which do not have a neighbour.

9 Implementation Results

Due to time constraints, I only managed to implement the algorithm for a flat glass pane. The program allows a user to apply an impact force to any portion of the glass by clicking on the screen. Additionally, a settings window allows the user to adjust the shatter resolution, shatter durability, shatter locality, and impact force. Presets are provided for materials which resemble standard glass, tempered glass, and clay.

The algorithm was straightforward to implement and is very efficient. In its present state it could be used for dynamic glass shattering in animations or video games.

9.1 Shatter Diagrams

The images below shows a Voronoi diagram for different shatter resolutions.

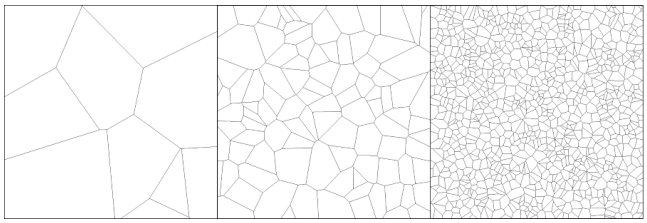

Figure 2: *Shatter resolution 10, 100, and 1000*

9.2 Materials

Only the shatter durability and shatter locality constants were changed to attain the results below. A shatter resolution of 1000 was used for both the examples.

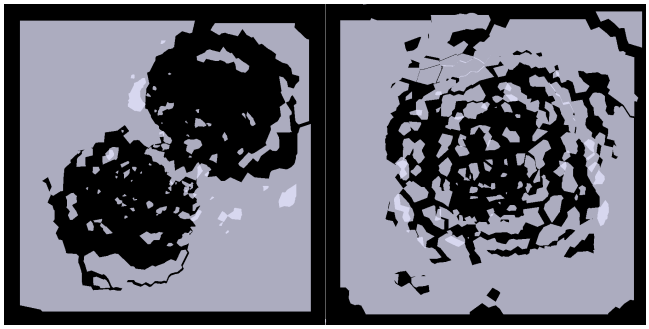

Figure 3: *Standard glass (left) and tempered glass (right)*

10 Future Work

The algorithm is implemented for a 2D plane, but should also be implemented for 3D models. Implementing the algorithm for 3D models is more complicated because it requires 3D Voronoi diagrams, mesh algorithms, and a rigidbody physics system. These components are common, but are typically more difficult to use and combine than their 2D counterparts.

There is room to experiment with different equations for the impact factor which might lead to more accurate representations of certain materials. Another possibility is to introduce some randomness into the equation to make results less predictable.

References

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 137–146.