*Authors: Bailey Shirtliff (7722391) and Mark Cortilet (7600856)*

## Introduction

For our term project, we decided to implement the method for fracturing brittle objects that was outlined in the paper written by Hahn and Wojtan for SIGGRAPH 2015. This method dynamically computes crack initiation and propagation in a given object based on the spatial differences in toughness and strength in that object's material. It then interpolates those cracks using a Lagrange reference frame and a boundary element method (BEM) system across a coarse triangle mesh of the original object to produce hi-resolution cracks in the original object both quickly and efficiently. The analysis that the authors of the paper ran on their own simulations showed that they could produce realistic fractured surfaces, without being too many complicated mesh manipulations.

As an interesting side-note, this method is also the first known approach in fracture simulation to use the boundary element method to calculate fracture propagation in computer graphics. It is also the first approach to simulating brittle body fractures that takes into account high resolution spatial differences in material toughness throughout an object.

## In-depth analysis of technique steps

In this section, we outline the steps taken by the Hahn-Wojtan method. Please note that while Hahn and Wojtan only outline three (general) steps in their paper, we will be including converting the hi-resolution mesh into its implicit surface and coarser triangle BEM mesh as a fourth step, since we consider it a core factor of the method that causes it to be as efficient and as accurate as it is. We have also split crack initiation and crack propagation into separate steps, in order to better differentiate between the two.

The first step that a simulation using the Hahn-Wojtan method takes after reading in the initial hi-resolution surface mesh is to convert that mesh into its implicit surface equivalent, and from there generate a low-resolution triangle mesh that can be used in BEM calculations. If the hi-resolution mesh extends too far outside of the BEM mesh, the BEM mesh can be extended in the direction of the normal. Hahn and Wojtan accomplish this by passing their hi-resolution mesh into OpenVDB, which creates and stores the implicit surface for that mesh, and then use VCGlib to create the corresponding BEM mesh.

Once the pre-processing has been completed, the simulation begins and continues to run until either the maximum number of steps (specified by the user) has been reached, or no new elements have been added to the list of fractured elements. Each iteration of the loop performs the calculations for deformations and surface stresses, then executes the crack initiation and crack propagation functions, then finally updates hi-resolution mesh with any newly added or modified cracks. At the end of each iteration, they write the hi-resolution mesh to OpenVDB, which renders that mesh as a frame in the simulation's animation.

The first step in loop is the calculation of surface stresses on the displacements and stresses being exerted on the surface of the mesh. To do this the simulation employs the boundary element method (BEM), utilizing a matrix that is assembled of various derivatives for equations of linear elastostatics, written as boundary integral equations. Fractures are represented as a single sheet of triangles, so that the simulation can employ a simplified version of the Symmetric Galerkin Boundary Element Method, as traction conditions no longer need to be accounted for. Each element currently being evaluated has an entry added into the matrix, and the results for each evaluation is stored in the matrix, meaning that while the matrix grows throughout the duration of the simulation, the only elements that have to be computed are for the new elements of the current time-step. Through this, the simulation can calculate the stress intensities along the crack front. In order to do this, Hahn and Wojtan use the hyENA library, a library for the solution of partial differential equations using BEM.

Next, the simulation calculates whether any new cracks are created in the current time-step. If a crack is initiated, it has two vectors associated with it; the surface normal relative to the fracture surface, and the direction that the crack with propagate, which is usually the inward normal of the triangle. A crack is created if the following four conditions are met.

First, the principal stress being exerted on an element exceed the material strength at that elements location. This is the primary condition for crack initiation, and can be evaluated in two ways; use the maximum principal stress to account for tensile fractures(where the stress being exerted is in the outwards direction), or use the minimum principal stress for compression fractures (where the element has some force acting on it in an inwards direction). Elements that have the greatest difference between their local principal stress and material strength will fracture first. Since the BEM mesh is composed of triangles, each triangle is evaluated using the co-rotations FEM method for computing principal stress, where each triangle is extended in a different dimension to make a tetrahedron. The centroid of the tetrahedron is where strength is evaluated.

The next condition that must be met is that the element being evaluated must not have been fractured in a previous time-step. In order to accomplish this, a list of fractured elements must be maintained. This condition prevents elements from overlapping one another, which cause issues in the integral solutions for BEM elements.

The third condition, that the element must be farther than the average BEM edge-length than any other crack-front, increases the efficiency of the simulation. If a crack is initiated too close to another crack front, it will very quickly become redundant. Finally, the number of cracks already in the simulation has not exceeded some user-defined value. This also increases the efficiency of the simulation, while also allowing the user to have some control over the simulation.

Once all cracks have been initiated for the current time-step, the simulation calculates how all the new and existing cracks propagate through the object. As noted above, when a new crack is added, new elements are added to the BEM matrix. Each new crack also has an associated crack front, which is subdivided into nodes (Hahn and Wojtan subdivide into 50 equally spaced nodes). This subdivision is what allows for hi-resolution fracture surfaces, especially temporal resolution, as each node is evaluated and propagated separately from the others. For each node, the simulation calculates the principal stress (which is expressed in a 3-dimensional local coordinate system) for that node and propagates it if the effective stress intensity on that node exceeds the material toughness. Hahn and Wojtan derive these values from Anderson's equation for strain energy release rate. If the propagation criteria is met, the simulation calculates the speed and direction of the propagation. The speed can be calculated using a linear approximation for the ratio of static and dynamic stress intensity factor. The direction is calculated using the principal stress intensities, where the first and third element drive the crack forward, and the second causes the crack to turn if and only if it is a non-zero value. The direction calculation is further expanded to account for spatial differences in material toughness by projecting material toughness onto a plane that is orthogonal to the crack-front and converted to polar coordinates. Every time a crack is propagated, new triangles are added to the BEM mesh that connect the new crack-fronts to ones that already existed.

Finally, once all crack calculations have been completed, the simulation updates the BEM coarse mesh with the cracks, then translates those into the implicit surface. The simulation performs a breadth-first search on the implicit method in order to find all separated fragments, so that it can extract the corresponding hi-resolution from the hi-resolution mesh for those fragments. Hahn and Wojtan do this by iterating over the different segments of a fractured mesh and passing each first back into VDB, then passing that result into VCG. They then write the final result into OpenVDB for display.

**Advantages of this method:**

In the following section, we will explore the advantages in using this method for simulating fractures in brittle rigid bodies.

First, it shows a notable increase in efficiency over the more commonly used methods. One reason for this is that every time a new element is evaluated during the crack initiation and propagation step, its entry in the Lagrange system is calculated at most one time, since the BEM matrix is expanded with all other previously computed element values. Another reason for this is the conversion of the hi-resolution surface mesh to corresponding coarse triangle mesh. Doing this reduces the resolution of the BEM grid, and so improves the efficiency of the deformation calculation, which is a integral part of the entire system. Using a coarse mesh also makes cutting into the surface model easier.

Next, since crack initiation and propagation depend on only two variables, they can be separated into two individual functions. The use of only a single variable to control the behaviour of these two functions reduces the complexity of each, and allows the use of seeded cracks in the geometry. At the same time, the use of BEM in calculating deformations and crack propagation reduces the required number of degrees of freedom from those required by other continuum mechanical models such as FEM, which further reduces the complexity of those calculations.

The final advantage of using this method is that the results appear to be quite accurate, and have been shown to reflect how objects actually fracture in nature. In particular, crack initiation tend to start at points of low strength in the object material, and cracks propagate from areas of high toughness in the material towards areas of lower toughness, which is exactly what would be seen in the real world. Additionally, since the evaluation of crack propagation at each time-step is treated as a Lagrange reference frame, and the result is interpolated for each point along the crack front, the crack surfaces are more detailed and accurate than what would be seen in other approaches. Indeed, fracture surfaces show patterns that are also seen in nature, such as the “rivers” in the 3-point bending example, and the “chevrons” shown in the granular toughness fields test. An added benefit of the interpolation of the crack front at each time-step is that this allows the accurate simulation of granular materials.

**Disadvantages of this method:**

Now that we have explored the advantages of the Hahn-Wojtan method, we will now discuss the disadvantages of this method.

First, as noted in the paper, the heavy reliance on quasi-static linear elasticity means that this method is ill-suited for use in larger displacements or ductile fracture. This obviously limits its use to only simulating small-scale brittle fractures.

Next, using larger time-steps during the crack propagation calculations can cause some serious problems in the simulation. One problem that may occur is that extremely large time-steps may mean that elements intersect one another in the BEM mesh, which would cause quasi-singular integrals during the assembly of the BEM matrix. The other problem that could happen is that if the BEM mesh does not represent the hi-resolution mesh accurately enough, "incomplete" cracks may be formed, meaning that components of the implicit surface that would normally be separated as fragments will stay connected and form a larger piece of geometry.

The final problem of this method is one that we personally experienced, in that it makes extensive use of advanced mathematics and physics formulae. This means that in order to fully implement this method, specifically the interpolation using BEM and a Lagrange system, one requires a thorough understanding of those topics, or heavily rely on external libraries without understanding what they're doing under the hood.

**Alternatives for deformations:**

While the Hahn-Wojtan method for simulating brittle body fractures utilizes BEM for calculating deformation and crack propagation, most other approaches to simulating fractures in rigid bodies use other methods. In the following section, we explore some of these alternatives in order to better contrast our chosen method against the more common ones being used today.

*Mass-spring system*

Systems that model solid materials or objects, such as ropes, cloth or jello, often use a mass-spring system, where objects are decomposed into mass points or particles, and this is the primary method of determining deformations that is still used in games today. These particles are connected to one another with massless elastic springs, where each spring consists of a resting length and a spring constant variable that denotes its stiffness. There are two kinds of springs in a mass-spring system. One type of spring controls how much the object is allowed to stretch and shear, and each particle is connected to all adjacent particles. The other type allows the object to bend without collapsing in on itself. In the latter case, each particle is connected to alternating particles (i.e. the first particle is connected to the third, and the second particle is connected to the fourth).

The advantages of using this kind of system for deformations is that it is fairly easy to implement, and as such it is reasonably fast. However, there are several problems with use a mass-spring system. For example, bending an object and stretching an object cannot be treated separately, since doing one of these directly impacts the ability to do the other. Another problem is that it is very difficult to set spring values that correspond to real-world values of a particular material, and often a lot of experimentation is required to find the ideal values.

*Continuum mechanical models*

Nearly every other system used for deformations can be grouped together, as they are all different models of continuum mechanics. Continuum mechanical models are often used to split an infinite, continuous problem into discrete, concrete problems that lie within a finite subspace. The three most common of these are variations of this, and are finite element mapping (FEM), extended finite element mapping (X-FEM), and material point method (MPM).

These systems were initially used in physics to model the behaviour of physical phenomena and the behaviour of various substances in different environments. As such, they can be used in deformation calculations to accurately determine what happens when a given material fractures under stress. However, they require a thorough understanding of the physics calculations being used in order to implement any of these models from scratch. As such, it is often necessary to rely on external libraries or frameworks that have already implemented these models for you.

**Alternative approaches in brittle fracture simulations:**

Although our method provides a new and more efficient method for generating fractures during simulation over the more conventional approaches being used today, fracture simulation is something that has been extensively explored in computer graphics for quite a while. The following section will discuss both the two of the more common methods used to generate fracture in simulations in computer graphics.

*Practical approach*

The practical approach to generating and simulating fractures in brittle rigid bodies is still the most common approach used in movies and games today. In this approach, the artist uses some CAD software to introduce fractures into the specified geometry, breaking it down into pieces that they have explicit control over, essentially “cutting it apart”. They then re-assemble those pieces back together to form the completed geometry, and set the amount of force required to break the geometry apart into its components. During the simulation itself, should the stress on a surface exceed the values set by the artist, the object breaks apart along the seams, and those pieces are displaced according to some physics calculation.

There are a couple benefits to this approach. First, it gives artists complete control over how a given object fractures and breaks apart, which allows them to explicitly create realistic or aesthetically pleasing fractures. It also simplifies the simulation itself, since the only calculations required are the ones that determine whether fractures occur, and the displacement of the pieces should the object fracture.

However, there are downsides to this approach as well. In large scenes comprised of complicated geometry that should fracture into many smaller fragments, it can be extremely tedious for an artist to actually go through that scene and edit each object individually. The other drawback of this approach is that the fracturing of the objects is only as realistic or pleasing as the artist makes them, which means that while excellent fracture animations can be created, animations featuring unrealistic or poor fractures can be created as well.

### *Geometry-Based Approach*

While the practical approach is most often used (effectively) in animations and games, scientific simulations tend to approach brittle body fracture simulations using more geometry-based approaches. These approaches apply cracks dynamically on the application of some stress on the object, and update the surface mesh accordingly. In order to do this they utilize some predefined crack pattern that is either defined on the creation of the geometry (similar to the practical approach), or the patterns are generated through the geometric decomposition of 3D space via techniques such as Voronoi diagrams. In contrast to the Hahn-Wojtan method, these methods use FEM, X-FEM, or MPM methods in order to calculate any deformations that take place throughout the simulation.

The main advantage of using geometry-based approaches is that they provide increased workflow flexibility, in that the simulation itself will handle calculating any cracks that appear in all objects in the scene, rather than the practical approach of a human being having to spend time on each object.

There are of course a couple drawbacks to using this sort of approach as well. First among these is that the realism of the simulation and the fractures that appear entirely depends on the choice of fracture pattern and how it deals with deformations. The other main drawback of these approaches is that the complexity of the simulation is increased over the practical approach. These simulations have to calculate the initiation of cracks relative to where stress is being applied on the surface mesh, how those cracks propagate through the object over time, and deformation calculations. These calculations are in addition to the basic computation of the magnitude of the stress being applied on the mesh, and the displacement of any fragments that are created as a result. In addition, many systems treat objects as rigid bodies during their dynamic motion, and as such, the deformation calculations require an extra step that converts the impulse exerted on that object into the equivalent force on the deformable surface. This extra step adds extra complexity to the deformation calculation, and requires either extra input from the user, or some form of integration in order to mitigate the effects of certain variables on that conversion.

## Our implementation

We decided to use python as our language of choice. We chose python because it is great at handling matrix maths along with some supported libraries that handle the geometry math, reading our object data, and displaying our object. The libraries chosen are PyMesh, PyAssimp, and Pygame. PyMesh is a geometry processing library which handles most of our geometry and object math using NumPy and other such libraries that handle matrix math. PyAssimp is an open asset import library which handles reading and interpreting our data types, such as the one we're using which is .obj. Pygame is a library that helps us display and animate our objects using OpenGL. For our implementation, we follow Hahn and Wojtan's example of setting markers along the front of the edge, moving each independently of one another, and connecting those new points to the existing crack mesh. However, we deviate from what they did in that we are not using the BEM method, as we could not find any suitable libraries, and we don't understand the method well enough that we could implement it on our own.

To get the animation of the output, we run the entire simulation first. This will output a mesh at the end of each iteration of the simulation's main loop as a .obj file using PyMesh. Then that .obj file is read in with PyAssimp, which creates its own mesh for it, and stores that mesh in a list. Once the simulation has completed, either because all markers have intersected a non-parallel surface that is part of the object's mesh, or the max number of iterations has been reached, we use Pygame to render each mesh that was output as a separate frame in order to animate the simulation. The user can toggle a wireframe view in order to observe how the crack front is being constructed and how the object is being updated after each iteration.

For each iteration of the simulation, we move each one of the markers that we placed at evenly spaced intervals along the crack front. When moving the markers, we determine whether the marker should move in the direction denoted by its normal vector or not by determining the distance from that marker to an area of lower toughness and weighting that marker's movement appropriately. Once that weighting has been determined, we divide the total speed of the marker, which is set when we initialize that marker, into two components; the forward speed, which moves the marker in the forward direction, and the vertical speed, which is determined by the markers normal vector. Each marker is moved independently of one another, resulting in a less realistic propagation that what is seen in Hahn and Wojtan's paper, since adjacent markers can move in amounts that don't correspond to what would be seen in nature or even move in opposite directions. We also do not update each marker's normal vector or its direction vector after moving it, since in homogenous materials the crack front would loop over onto itself. This affected what we could actually do with the simulation, since we could only specify toughness values along the y-axis, although we can still specify multiple areas.

Once we finish moving all the markers along the crack front, we run an intersection test on each of them to determine whether they intersected with a non-parallel surface that is part of the surface mesh of the object. Neither PyMesh and PyAssimp have a line segment-triangle test implemented, so we had to write one ourselves. If a marker does intersect the surface mesh, we mark it as such and set its location to the point that intersects the mesh. This helps filter out which markers we propagate every iteration, as only markers that haven't been marked intersecting the mesh are propagated.

After we finish updating all the markers, we use them to construct a triangle mesh that corresponds to how the crack propagated since the last iteration. During the construction of this mesh, we use all the markers, regardless of whether they have intersected the object mesh or not. Once the mesh has been finished, we use PyMesh to merge it with the existing crack mesh that was constructed as part of previous iterations. Then we remove any duplicate vertices or faces, and any degenerate triangles, with PyMesh's provided methods.

We had planned to use the Constructive Solid Geometry (CSG) portion of the PyMesh library to update the initial mesh with the crack mesh as it propagates through the object. The purpose of CSG is to add simple geometry together over each iteration of the render in order to create more complex objects. In our case, everytime we add to the crack in the object, we expand the crack and add the previous geometry to the current crack iteration. However, we ran into a couple serious issues that prevented us from using it, which are described below.

We had to use PyAssimp to pass meshes into OpenGL, since most libraries that we could find that manage triangle meshes either required an external program to actually render out the mesh, or could only render out a single frame at one time. In order to actually use PyAssimp, we needed to write the mesh being stored by PyMesh to an .obj file, then use PyAssimp to read it back in. We do this at the end of each iteration of the simulation, and append each PyAssimp mesh to the end of a list of meshes.

Once the simulation completes, we iterate over that list of meshes and render each mesh as a single frame. The last mesh is rendered for an extra 6 seconds, then the program resets back to the first mesh. We also allow the user to toggle between a shaded view and a wireframe view, so that they can see exactly how each marker is moved as the simulation progresses.

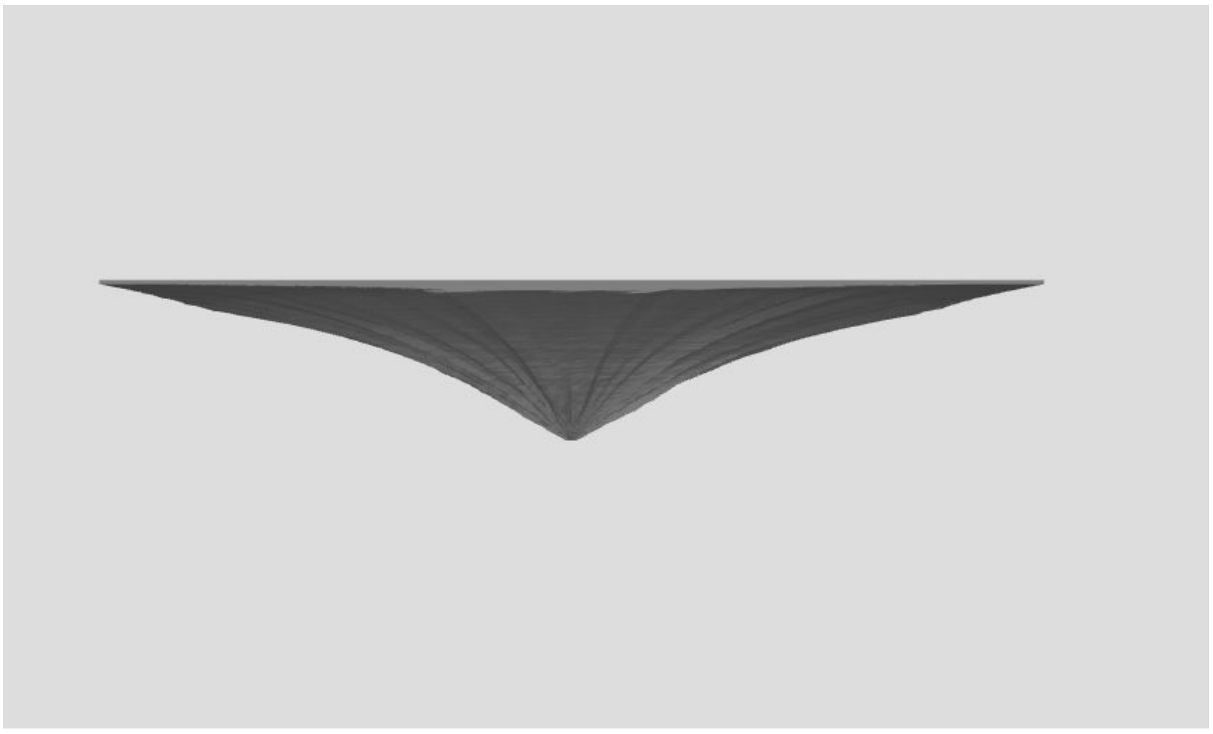
**Our implementation - Issues**

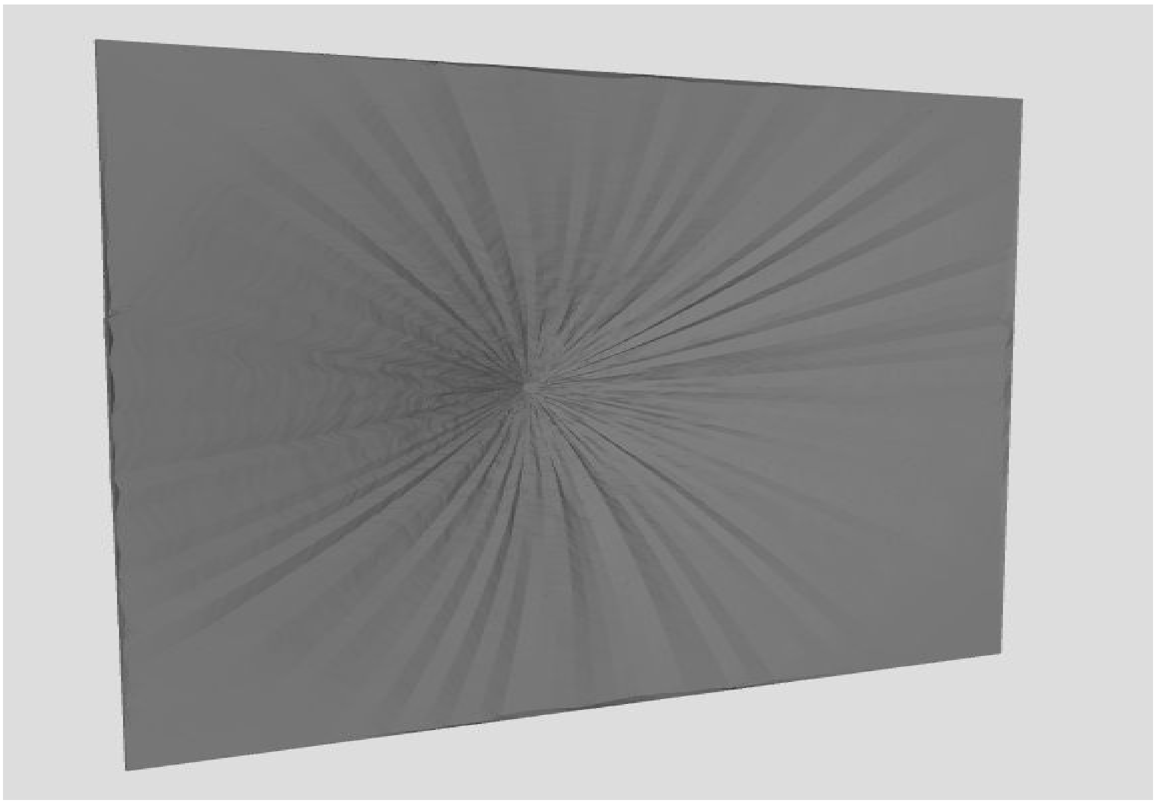
The following section lists the issues we encountered while implementing our project:

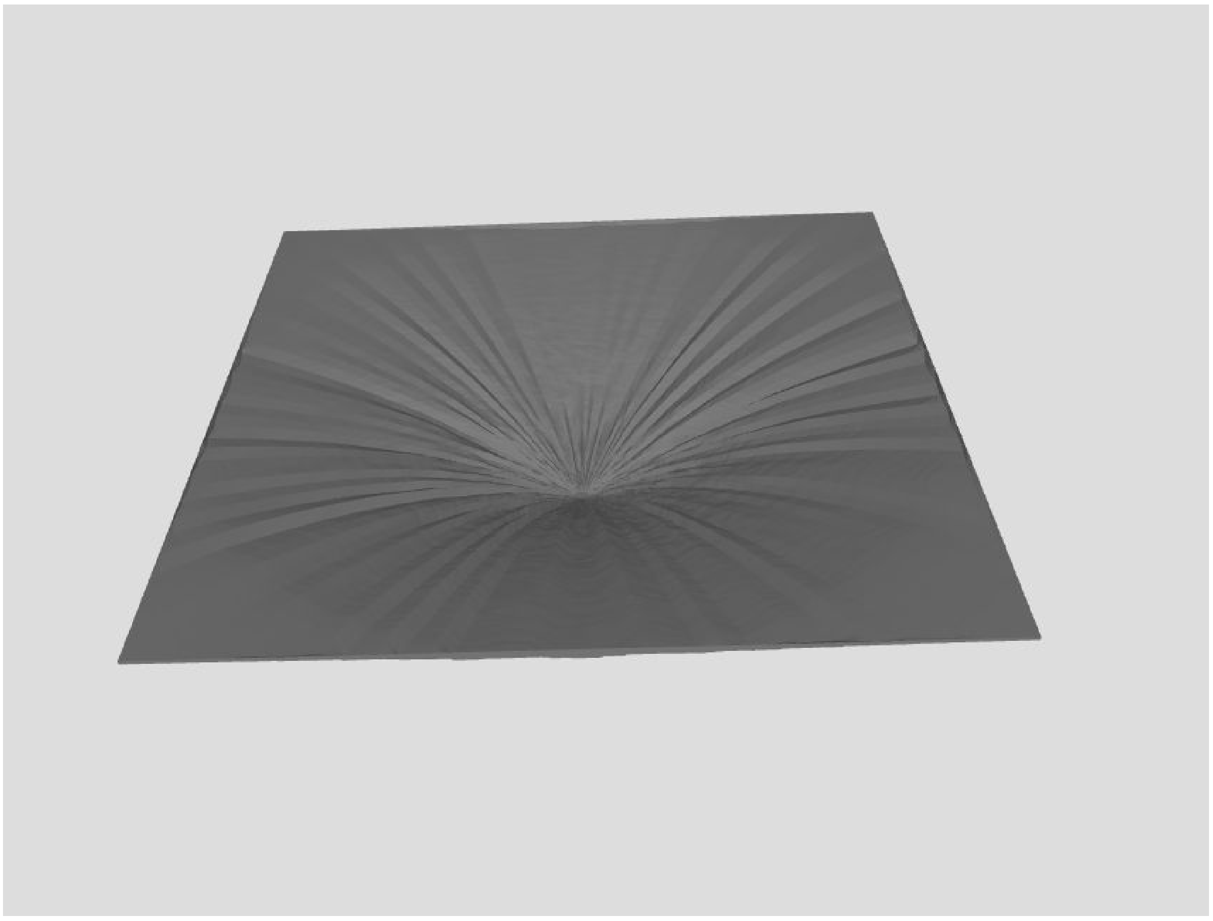
1. There is a major performance bottleneck caused by the CSG operation, which is exacerbated when we attempt to run a CSG operation on a mesh (upwards of 1 minute per iteration). Due to this, we decided to try only updating the original mesh, without taking into account how the crack has propagated. This would have affected how the simulation works, since cracks won’t be able to detect intersections with one another, and will only terminate upon reaching the surface mesh. However, the CSG libraries still would not work properly, and so we decided to render only the crack mesh, since that would be sufficient to demonstrate how it propagates through the object.
    1.1. Since we’re using external libraries for the CSG, it’s unlikely that this issue can be fixed on our end. The current setup for the simulation only has one crack (since adding more cracks would also incur a major performance hit), so this shouldn’t impact the current simulation.
2. Rendering the final fractured mesh doesn’t output the correct image. With CSG enabled, the crack front is partially visible through the surface mesh in places it shouldn’t be. In addition, the crack mesh is incorrectly shaded when rendered. We have verified that the crack mesh is formed correctly and is smooth using external programs, so this appears to be an issue with PyAssimp and how it passes information to OpenGL.

3. Currently we are keeping the same number of markers throughout the entire simulation. This means resolution of the crack front decreases as it gets farther away from its origin, since the length of the crack front increases, and the distance between the markers increases as well. The solution to this is to dynamically allocate markers based on the length of the crack front and then placing them evenly along the boundary, instead of using static markers.
4. When moving markers, we don’t take into account how the adjacent markers have been moved, which means that markers can move in ways that wouldn’t be seen in nature. Unfortunately, we can’t really do this without BEM, since we have to process each marker in serialized order so at most only one adjacent marker will have been moved. We managed to somewhat mitigate this by processing all the markers after they have been moved, and averaging the movement of adjacent markers.

## Screenshots







**References**

Hahn, D. and Wojtan, C. (August 2015). High-Resolution Brittle Fracture Simulation with Boundary Elements. Retrieved from http://pub.ist.ac.at/group_wojtan/projects/2015_Hahn_HRBFwBE/

Hahn, D. and Wojtan, C. (July 2016). Fast approximations for boundary element based brittle fracture simulation. Retrieved from http://pub.ist.ac.at/group_wojtan/projects/2016_Hahn_FastFracture/

Zhu, Y., Bridson, R. and Greif, C. (August 2015). Simulating Rigid Body Fracture with Surface Meshes. Retrieved from https://www.cs.ubc.ca/~rbridson/docs/zhu-siggraph2015-surfacefracture.pdf

O'Brien, J. and Hodgins, J. (1999). Graphical Modeling and Animation of Brittle Fracture. Retrieved from http://graphics.berkeley.edu/papers/Obrien-GMA-1999-08/Obrien-GMA-1999-08.pdf

Batty, C. (2014). University of Waterloo CS888 - Advanced Topics in Computer Graphics: Physics-based animation (Lectures 1 + 2). Retrieved from https://cs.uwaterloo.ca/~c2batty/courses/CS888_2014/Lecture1.pdf and https://cs.uwaterloo.ca/~c2batty/courses/CS888_2014/Lecture2.pdf

Dingliana, J. (2015). Trinity College Dublin CS7057 - Realtime Physics: Mass-spring Systems and Cloth. Retrieved from https://www.scss.tcd.ie/Michael.Manzke/CS7057/cs7057-1516-14-MassSpringSystems-mm.pdf

Baker, M. (August 2011). Authoring destruction with the Dynamica Bullet Maya plugin. Retrieved from http://bulletphysics.org/siggraph2011/michael_siggraph2011.pdf

Unknown author (August 2013). Control how objects shatter in Blender 2.6. Retrieved from https://www.creativebloq.com/control-how-objects-shatter-blender-26-8134076

Hahn, D. and Wojtan, C. (August 2015). Source code. Retrieved from https://github.com/david-hahn/FractureBEM

**Libraries Used**

PyGame: PyGame is a free, open source python programming language library for making and displaying multimedia applications. It is built on top of the SDL library, and is easily portable to any platform.
Link: https://www.pygame.org/wiki/about

PyMesh: PyMesh is a geometry processing library built by Qingnan Zhou from New York University. It handles geometry processing in python and C++. It takes advantage of python to create minimalistic and easy to use interfaces, and C++ to handle the computationally intense functionalities.
Link: https://github.com/qnzhou/PyMesh

PyAssimp: PyAssimp is apart of an open asset import library, Assimp, which handles reading and interpreting data in many different formats such as OBJ, BLEND, and much more.
Link: https://github.com/assimp/assimp/tree/master/port/PyAssimp