Experiments on a level set method for ductile fracture

Small Project GMT

Thomas Breekveldt
4037324

January 31, 2014

1 Introduction

The game development industry has seen a large focus on realistic graphics. While there is continuous development, other aspects of the game development should be capable of contributing more. One of these areas is game physics.

Often when something is destroyed in a game environment, an artist already defined how it should look destroyed long before you picked up the game and decided how you wanted to destroy said object. Making the player's options, in my opinion, feel rather limited. Some games do have actual physics for destructable objects, but are then limited to specific objects and regions. Perhaps my personal observation came from a limited dataset, but nevertheless this observation became the spark of my interest in game physics. It made me wonder what the state of the art for physics simulation is, how far we can go with this. And where else should I look than recent research developments.

Continuing from the drive for realism, simulations running in real time were dropped in favor of research on slower physically accurate simulations. Along with the emphasis on deformation and destruction the paper A Level Set Method for Ductile Fracture by Hegemann et al.[1] caught my attention.

This report contains the implementation details and the results of an experimentation with the implementation according to the paper of Jan Hegemann et al. To be more precise, experimentation with the tearing of objects under stress. A number of simulations have been ran on the same object but with varying material properties to evaluate what the method is capable of. To decrease the complexity of the implementation several aspects have been simplified. The mesh object is simplified and the simulation uses forward Euler instead of backward Euler to run the simulation. The reasons and implications for these will be explained in the relevant chapters.

Chapter 2 will elaborate on the method described by Hegemann et al. and highlight the core mechanics of the method. In Chapter 3 I will give an indepth description of my implementation of the method and the differences between the original method. Then in chapter 4 I'll present the setup of the experiment and its results. Where I'll be discussing them along with the state of the implementation in chapter 5. Finally a few guide-lines for future work on this code-base in chapter 6.

2 A Level Set Method for Ductile Fracture

The method described in the original paper [1] focusses on the *ductile fracturing* of *hyper-elastic* materials. This means that the simulated object can tear in a slow manner, not instantly snapping in two like a match stick or crumbling like a wall. Hyper-elasticity means that the stress inside an object is the same regardless of how its current position has been reached. This simplifies calculations as only the current state of the object matters and no part of its previous deformation has an effect on the stress. But the surface has to be continuously updated while the fracturing is in process. Before I explain the contribution by

Hegemann et al. I will first explain the basis which the method relies on. Here is a quick summary:

- Tetrahedal mesh coupled with levelset to visualize the object
- Backwards Euler FEM simulation on the material nodes to update the positions
- Griffiths energy evolution to alter levelset for the fracturing

The section about (self)collision will be omitted because my implementation does not implement (self)collision and the experiment is constructed that (self)collision does not affect anything.

2.1 Core Mechanics

The mesh object, called a *tetrahedal mesh*, is a collection of tetrahedra created from a uniform point grid. Where a triangle mesh is created with two triangles using four nodes in a 2d point grid, six tetrahedra are created from eight points in a 3d point grid. Figure shows an example of a 2d grid. This mesh object works together with a *levelset* to increase the level of detail and be more dynamic. This levelset consists of a uniform point grid, which in this case is a 3d point grid because our object is 3 dimensional, where for each point in the grid stores the distance to the closest surface of the object. Here a positive value means the point is outside the object and a negative value means it is inside the object. From these point values a more accurate surface can be constructed ontop of the tetrahedal mesh. Tetrahedra outside of the levelset are empty, tetrahedra inside the object are complete, and tetrahedra intersecting the levelset surface are cut by a plane of the level set. With this coupling of the two data structures the deformation (ie. compression or stretching) of the object is encoded in the tetrahedra, and the alteration of the surface is encoded by the levelset and the corresponding cut-planes of the tetrahedra.

Finite Element Method (FEM) simulation will be ran on the object which was just described. This is a simulation where the influence of every point connected to another point is calculated when the simulation makes another time step. The rest of this section describes the computations required for the FEM simulation.

The calculation of current stress inside the object becomes straightforward when using the tetrahedra. Sifakis et al. [2] present in their paper in chapter 4 the pseudo code for their algorithm 1. In this procedure the difference in position of each tetrahedra compared to its rest-state is used to compute the amount of deformation for that tetrahedra, called the *deformation gradient* $\mathbf{F}$. To compute meaningful stress forces from $\mathbf{F}$, we first convert the deformation gradient to the *first Piola-Kirchoff stress* value $\mathbf{P}$. This conversion depends on the *strain measure*, a mathematical relation between deformation and strain energy, used. Chapter 3 of Sifakis et al.[2] describe different strain measures

and their benefits and downsides. Hegemann et al. use for this simulation the *corotational energy density* defined as:

$$\mathbf{P}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda tr(\mathbf{R}^T\mathbf{F} - \mathbf{I}) \tag{1}$$

The matrix $\mathbf{R}$ comes from the matrix polar decomposition of $\mathbf{F} = \mathbf{RS}$. Where $\mathbf{R}$ is the rotational component and $\mathbf{S}$ is the symmetric component of the decomposition. Internal stress forces can be computed from the combination of the strain measure and the deformation gradient.

When computing new positions for the mesh points the movement of a single point has influence on the movement of other points in the same time step. And those altered movements have a different influence on the single point mentioned before, altering its movement and its influence on the other points. In other words, the accurate computation of the point's movement is affected by itself, making the result of the equation dependent on itself as input. To solve this the backwards Euler method by Sifakis et al.[2] is used to construct an iterative process that computes these new point locations. This does complicate a number of things. Where we used the deformation gradient and the first Piola-Kirchoff stress value we now need both of their derivatives. The derivative of the deformation gradient $\delta\mathbf{F}$ is constructed in algorithm 2 presented by Sifakis et al.[2] in a similar fashion as $\mathbf{F}$. The derivative of the corotational strain measure $\delta\mathbf{P}$ is presented by McAdams et al.[3].

The backwards euler method uses an iterative process to converge to the correct answer of the current time-step. To compute one time step a matrix equation of the form $Ax = b$ has to be solved for x. This is done using a Krylov subspace method such as Conjugate Gradients. The Conjugate gradient method numerically solves the $Ax = b$ equation. In our case the matrix A is the strain matrix $\mathbf{K}$ which is n x n where n is the number of nodes inside the uniform grid. We can avoid constructing this large matrix because the Conjugate Gradients method only relies on a matrix-vector multiplication with A and we compute this multiplication directly with Algorithm 2 given by Sifakis et al. [2] where $\delta\mathbf{P}(\mathbf{F})$ is given by McAdams et al.[3]:

$$\delta\mathbf{P} = 2\mu\delta\mathbf{F} + \lambda tr(\mathbf{R}^T\delta\mathbf{F})\mathbf{R} + \{\lambda tr(\mathbf{S} - \mathbf{I}) - 2\mu\}\delta\mathbf{R} \tag{2}$$

2.2 Innovation

The main contribution stated by the paper is their usage of Griffiths energy minimization. But first it is needed to compute the stress energy values for the nodes. Instead of computing the strain measure from the deformation gradient we compute the energy density.

$$\tilde{\Psi}(\mathbf{F}) = \mu\|\mathbf{F} - \mathbf{R}\|_F + \frac{\lambda}{2}(tr(\mathbf{S} - \mathbf{I}))^2 \tag{3}$$

The nodal energy values are then computed as defined by Hegemann et al.[1]. This is a summation over the energies of the tetrahedra T_k node X_p is part of,

normalized by their respective volume.

$$\Psi(\mathbf{X}_p) = \frac{\Sigma_{k,\mathbf{X}_p \in T_k} \Psi(T_k) \int_{T_k} N_p(\mathbf{X}) d\mathbf{X}}{\Sigma_{k,\mathbf{X}_p \in T_k} \int_{T_k} N_p(\mathbf{X}) d\mathbf{X}} \tag{4}$$

This Ψ value can then be entered in the Griffith energy formula along with the ϕ value which is the node's levelset value, the distance of the node to the closest surface. This creates a new $\hat{\phi}$ value which will be used to update the level set.

$$\hat{\phi} = \phi + \Delta s \delta_\epsilon(\phi)(\Psi - \kappa) \tag{5}$$

The nodal energy Ψ is compared against the fracture resistance κ. This only generates a positive value when the nodal energy exceeds the fracture resistance. The resulting value is then scaled by the Dirac delta function δ_ϵ. This delta function is special because it creates an infinitely large number when the input is zero, and results in zero otherwise. But since we're dealing with a function approximate $\delta_\epsilon(\phi) = \frac{1}{\pi} \frac{\epsilon}{\epsilon^2 + \phi^2}$ with ϵ to be machine precision, input values close to zero will also result in values larger than zero. This is then scaled yet again by the time step size Δs. This time step size is under constraint of Courant-Friedrichs-Lewy condition. The CFL restriction reduces the time step size in relation to the nodal velocities, higher nodal velocities result in smaller time steps. This is to ensure numeric stability. Finally the value is added to the current ϕ.

If $\hat{\phi} < \phi$ this would effectively mean the volume got increased. A negative value is inside the surface and decreasing this value would effectively push the surface further away from the point. Because we only want to lose volume when fracturing or damaging objects we ignore the new $\hat{\phi}$ value when $\hat{\phi} < \phi$, only when the volume shrinks is when the levelset gets updated.

With this you have a function that notices high stress values at the surface of objects and reacts by moving the surface closer to the stressed point. Effectively creating a tear in the surface. That is exactly what we want when a material is affected forces that are large enough.

3 Implementation

This chapter elaborates on the implementation decisions made when recreating the method. Among these are the library choice and concessions made to simplify the implementation.

3.1 The Starting Point

When the choice had been made to recreate the method by Hegemann et al. the authors had no source-code available online. And with the scope of the method an entire implementation from scratch was not feasible. Therefore preliminary research had to be conducted to find potential starting points for this project.

From an online search a number physics libraries and Finite Element Method simulators are freely available.

There are a number of software packages such as FEBio that can perform FEM simulation straight out of the box. It is as simple as installing a program, give it an input object and it generates an output for you. FEBio is a software package mainly being used for bio medical simulations[4], but it is a FEM simulator nontheless. Unfortunately all of these packages I found were closed source software packages. That meant that I would be unable to directly implement the method because it requires modification of the input object during the simulation.

PhysBAM is a physics engine from the Stanford University and their website[5] has a couple examples listed. These examples vary between images and videos, about fluid and fracturing simulations among others. One of the authors of the paper I'm trying to implement worked on these examples, or has previously cooperated with some of the authors of the examples. Noticable being M. Teran working on both the method i'm trying to implement and being a co-author of McAdams et al. and Craig Schroeder's work on deformable models and rigid body simulation. The PhysBAM library has been made partially available for download along with source-code of a fluid simulation implemented using a level set method and a smoke simulation.

OOFEM is an object oriented finite element method simulation library[6]. Has an active community behind it using the available forum and tutorials. But seemed to lack some datastructures required for the method. The available footage showcasing work with the libarary were disappointing when compared to PhysBAM, both in quality as in numbers and diversity.

Neither of the choices were matching my needs perfectly: On one end PhysBAM a high end physics library without having built-in FEM simulation and on the other end the OOFEM library where I would have to implement data structures myself. This would include the math for the generation and coupling between a level set and a tetrahedal mesh.

Having read the paper by Sifakis et al.[2] I had a grasp on the basics of FEM simulation. But having read the paper by Vese et al.[7], I became more daunted by implementing a level set. This swayed my opinion in the direction of PhysBAM. I was more daunted by constructing the data structure than figuring out the exact workings of the pseudo code.

Some initial problems were the lack of documentation which was mediated by a third party program constructing the bare minimum of documentation using the source code directly. This documentation consists of inheritance overview, and a structured collection of class- and method-names. The other problem was the PhysBAM website only having instructions to deploy the library for linux. Being unable to compile large parts of the library due to differences between MSVS C++ compiler and linux gcc the choice was made to switch the entire project to linux.

3.2 Implementation

To reconstruct the datastructure individual components were used rather than the combination of level set and tetrahedal mesh into one. Mainly because of the individual components were readily available within the library. Because of the lack of coupling decribed in chapter 2 between the level set and the tetrahedal mesh it was required to reconstruct the tetrahedal mesh when the level set got modified. The same counted for reinitializing the levelset to start the next iteration of the simulation. However, the function to create a level set from a mesh looked at same world space the tetrahedal mesh was simulated in but required the object's rest-state to describe the surface in the level set. If the tetrahedal mesh was modified in world space (ie. moved, compressed, etc) the parts of the level set would be wrong. This is because the grid the level set worked on was not modified, creating an offset between the two. This was remedied by creating a second tetrahedal mesh which would only be used to reinitialize the level set. The order of execution would be:

1. FEM simulation time-step.
2. Update level set using Griffiths energy evolution.
3. Update both tetrahedal meshes using new level set.
4. Update level set using rest-state mesh.

The tetrahedal mesh is not using cut-planes to represent the surface but just the triangles of the surface tetrahedrons. The implications of not having this appeared to be minimal and was thus left as a could-have feature.

The FEM simulation is done using forward Euler. Calculations of the deformation gradient $\mathbf{F}$ and the first Piola-Kirchoff stress $\mathbf{P}$ are done normally but the time-stepping is naively done by looking at the internal and external forces and disregarding the influence a point should have on the points its connected with. With a plain computation per node of $f = m \times a$, $v = v + a \times \Delta t$, and $x = x + v \times \Delta t$. This is because the implementation for backwards Euler never converged to a solution value despite frequent debugging. This was causing the material to explode rather than stabilizing. Forward Euler was considered to be acceptible enough to produce experiment results.

3.3 Compromises

With the recomputation of the mesh comes a significant memory burden. Instead of currently allocated memory being altered, a new block of memory is requested to rebuild the mesh tetrahedal relations from scratch. Not once, but twice, because of the two mesh objects being used. So when recomputing the mesh we reallocate two large blocks of memory which costs a significant amount of time each iteration.

The lack of cut-planes for the surface only removes a degree of detail from the simulation. Where in the original method the cutplane would move as the

levelset moves creating a smooth tearing. In this implementation the mesh stays the same until a node is no longer contained by the levelset, making the tetrahedra pop loose. An analogy for for this difference is similar to comparing the tearing of fabric or tearing open a blouse where the buttons pop open one-by-one. How much this affects accuracy is unknown.

Using forward Euler has a large downside: forces and stress distribution behave as ripples now. If implemented correctly, the displacement of one node has effect on another node, and displaces it. But that second displacement has effect on the initial displacement, meaning the result of the computation is dependent on itself as input. With forward Euler the effect of node displacement will only be accounted for the next frame. By increasing the amount of sub-frames for calculation between each drawn frame this effect is reduced.

The amount of subframes has therefore been increased to a static number e.g. 20x the frame rate, instead of a scaling factor dependent on nodal velocities as the orignial implementation by Hegemann et al. has, with the CFL restriction.

4 Experiment

The experiment is a crucial part of this project for two reasons. One is to validate the implementation, to see if the results created by the original author can be repeated. The other is to push this implementation of the method to its limits.

4.1 Setup

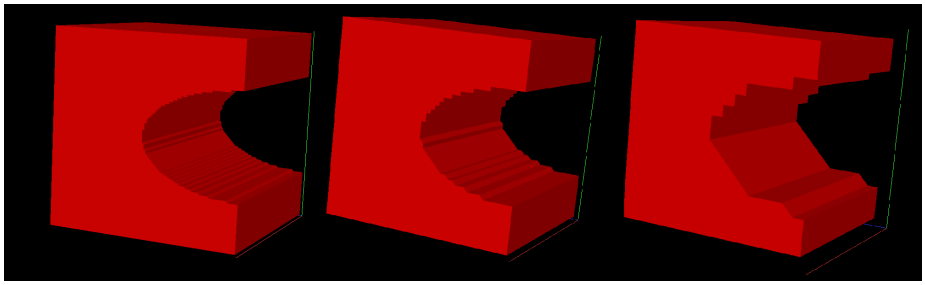

Figure 1: Object used to compare different material properties at different resolutions of grid points. From left to right: $100^3, 50^3, 20^3$

The setup of the experiment is a cube with a cylinder shape cut out from its side as shown in Figure 1. This object is being pulled both up and down at the same time to create a stretching stress force. This setup mimics one of the scenarios in the example video. In that video three similar object with different fracturing resistances were put under the same stress tests for a simple and direct comparison. However, in this experiment there is no gravity force

being simulated. The object shape was chosen because it guarantees tearing in specific regions and the stress test was chosen because of its simplicity and the lack of requirement for (self)collision.

Initial experiments were to give an indication to what each variable does to the simulation. Being new to the subject I was not fully aware of the properties. The first experiment was therefore varying the Poisson ratio to see how it affects the simulation. Second came the varying of the Young's modulus. After those initial tests came the experiment of lowering the fracture resistance to see how the simulation behaves, and increasing the resolution of the simulation. However, higher resolution simulations suffer from the exponential increase in run-time and therefore less of those simulations have been ran.

4.2 Results

For the first experiments the varying of the Poisson ratio was chosen. This variable is only meaningful when set between 0 and 0.5. Three relatively arbitrary numbers were chosen to represent a low, medium and high Poisson ratio. These values are relatively 0.10, 0.23 and 0.49. Other variables were kept the same with the fracture resistance $\kappa = 100$, Young's modulus $= 50$ and the resolution at 20^3 points. The results of these simulations are shown in Figure 2.

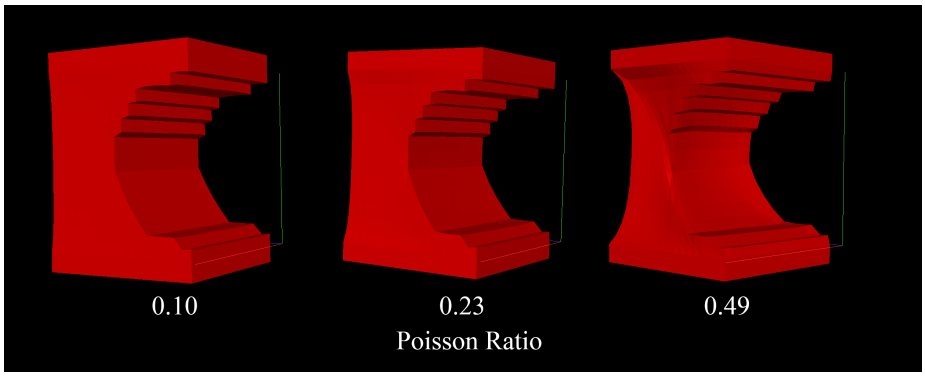

Figure 2: The behaviour of different Poisson ratio's at the same time step in the simulation.

From these results we can deduct that an object with a higher Poisson ratio tries harder to maintain its volume. This would be done by creating stress forces which pull the points closer together. With a poisson ratio of 0.49 the object is thinned in the center to compensate for the stretching of the volume. When the Poisson ratio is lower the inward forces will be lower making the effect become less apparent.

The experiments with the Poisson ratio have been repeated a couple of times with a varying Young's modulus, as shown in Figure 3 still having the same fracture resistance $\kappa = 100$ and resolution $= 20^3$ points. When looking at the

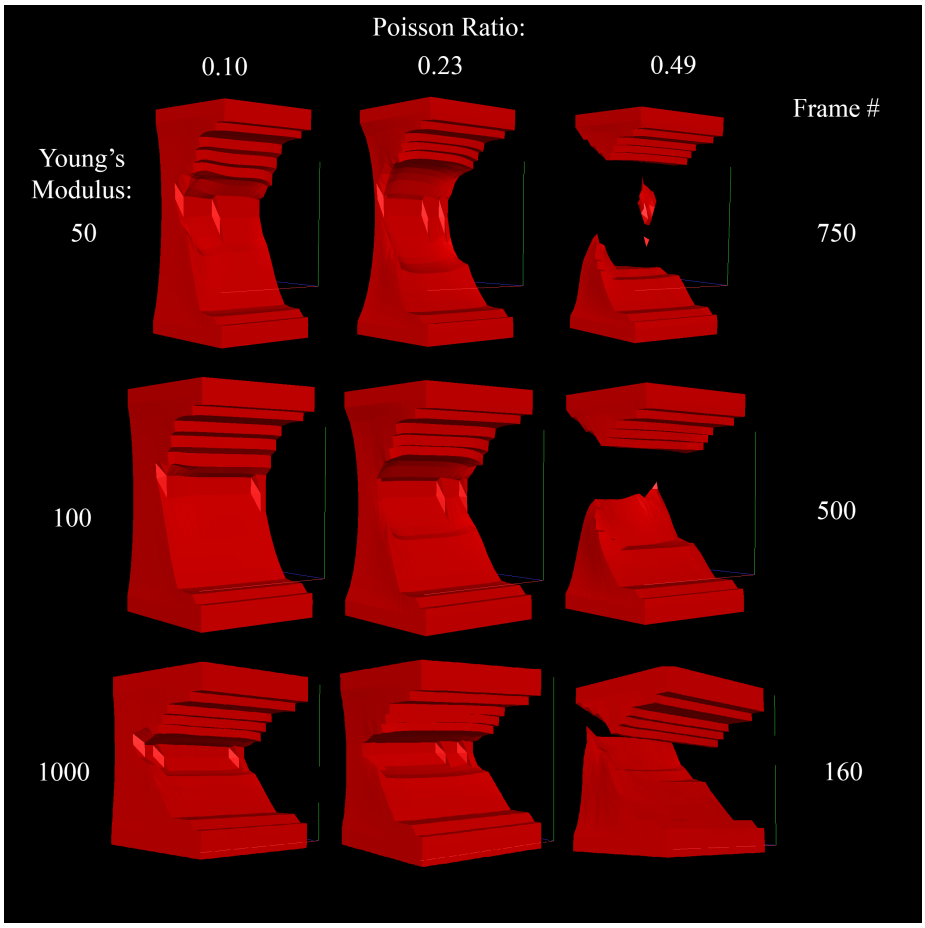

Figure 3: Different combinations of Young's modulus and Poisson ratio. All entries on the same horizontal row are from the same time step in the simulation.

fractures in the simulation it appears that with the chosen values the Young's modulus has the biggest influence in when a fracture occurs, and the Poisson ratio affects more where the fracture occurs. With a higher or lower Young's modulus the fracturing occurs, respectively, faster or slower. Having a high Poisson ratio increases the stress in the corners resulting a diagonal fracture starting from the center and going to the top corner, where with a lower Poisson ratio the fracture occurs horizontal through the thinnest part of the object. The increased influence of the Young's modulus could be explained by the opposite stretching forces generated by the experiment affecting the Young's modulus the most. Where perhaps a twisting motion could affect the Poisson ratio more.

As a side note: an interesting aspect of the method is the inherent ability to

generate fragment pieces when fracturing occurs as visible in the top right case of Figure 3.

Armed with this knowledge of the two material properties some experiments with the fracture resistance have been made. When increasing the fracture resistance it generally takes longer for the fracturing occurs until either the Young's modulus or the Poisson ratio contributes enough to create a fracture, given this still occurs within the simulation time. What was more interesting to examine was the effects occuring when the fracture resistance gradually got lowered. An example is shown in Figure 4 where the Young's modulus = 1000, the Poisson ratio = 0.49 and the resolution is still at 20^3 points.

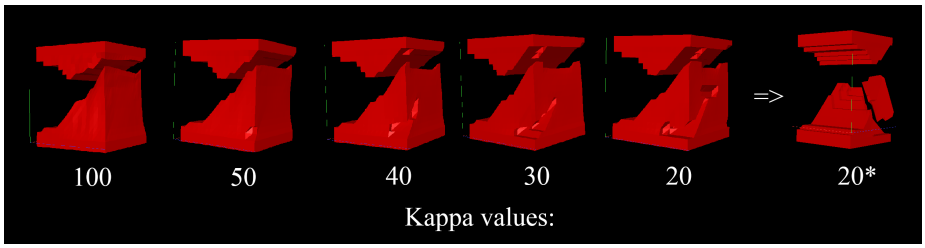

Figure 4: The effect of lowering the fracture resistance value κ. With $\kappa = 20$ a piece breaks off seconds after the initial fracture occurred.

This rotated view compared to the previous examples is to put emphasis on the crack appearing in the backside. When the diagonal fracture occurs from the normal fracturing with κ set normally the Poisson ratio creates stress values large enough to cause fractures in other corners aside from the top ones. When κ is sufficiently low the following happens. The extra fracture causes a large piece of the object to remain attached to the main object through a relatively small contact area. With the gravity-free simulation environment and the damping force, all points want to eventually stop moving. But the top and bottom of the initial object will continue to get moved regardless, creating a stress on the smaller connecting piece to pull all remaining points with it. What happened in the case of $\kappa = 20$ is that the stress on that smaller part exceeded the κ-value, resulting in more fractures and eventually releasing the large chunk from the pulling forces on the bottom. In other words, the fracture resistance was so low that the object fractures under its own movement. This behaviour could potentially be used to simulate extremely fragile foam-materials which breaks into chunks when the object is moved, eg. foam in a bathtub is barely able to remain intact when creating a non-convex shape and moving it around.

After exploring the three material properties let's scale up a simulation to a resolution of 50^3 points. Taking a previously used experiment of a lower resolution, it is scaled up directly as shown in the second example in Figure 5. This created artefacts similar to the previous experiment when lowerin the κ value. By increasing the κ value we get a similar simulation again. However, these snapshots were not taken at the same time in the simulation. There

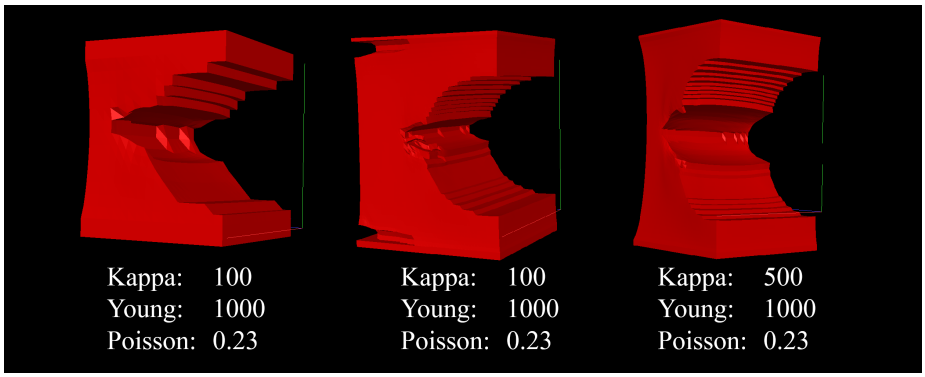

Figure 5: The effect of lowering the fracture resistance value κ. With $\kappa = 20$ a piece breaks off seconds after the initial fracture occurred.

appears to be something off in the implementation with regards to scaling, or the lack of regards to scaling, so to get a perfect scaling in the simulation the Young's modulus and perhaps even the Poisson ration have to be scaled to match.

5 Discussion

Due to scaling not working properly there is a large possibility that the exact values used in the simulation can not be taken for face value. When for instance testing a bridge with this simulation and it can withstand "200", the exact unit is unknown due to the scaling. What it can do is being used to compare two instances with equal properties but different shapes because the comparison is invarient under this problem of unit scale. Being better still remains better, regardless of the exact quantity.

From the simulation it also never became clear if the method could be used for materials more often simulated as rigid bodies. Such as metal, wood, or concrete objects. It seems rather ideal to have one simulation capable of simulating many different materials. Though, it is quite possible the lack of this observation can be blamed on the implementation of the method as well as the experiment. The implementation's downsides have already been explained in Section 3.3. The experiment could be considered flawed because the pushing and pulling on the top and bottom of the objects was a fixed movement each timestep, instead of a force being applied on the points. Having a (slowly) applied force could simulate objects only capable of minute movements to have them build stress in the appropriate areas rather than snapping immediately at the border where the object is being pulled. This effect was visible in the second example of Figure 5 where the fracturing occurs on the top and bottom exactly at the border of points being moved a fixed step and the points reacting

purely to internal forces.

6 Future Work

Having shortcomings in regards to the original method, it is fairly easy to suggest improvements. Starting with the most obvious but probably biggest task is the actual implementation of a backwards Euler method. This should remove the ripple effect from the simulation resulting in potentially enabling the simulation of solid materials instead of jelly-like substances. At the least the reaction to forces and locations of fracturing will become far more accurate.

Enable the CFL condition to regulate the timestep size rather than a fixed time step size. This would either speed up the simulation or make it more accurate depending on the situation. If the simulation is light, the CFL condition will enable larger time steps. And if the simulation is intense, the CFL condition will reduce the time step size to increase the accuracy.

Coupling the tetrahedal mesh data structure with the level set and add the cut-planes. This will affect quite a number of things. While adressing the main memory issue, it will significantly increase the performance because the complete reconstruction of two mesh objects and a level set is quite intensive, and is by far the most intensive part of an iteration. At the same time it will increase the amount of detail with the cut-planes, allow for smoother fracture development and potentially more fragments.

A thorough check of the math to find the scaling issue. My assumption is that I forgot to scale the mass with the density over the amount of points, but was unable to pinpoint the right location. If the math is checked completely, a simulation could then accurately state that an object can resist x-amount of Newton force.

Implementing the collision model written by the authors of the paper to allow the simulation of interaction between different objects and fragments.

Aside from fixing the implementation flaws or shortcommings I could see the simulation of extremely brittle and foam like materials working. But it would probably require some tweaking to not destroy the entire object at the slightest movement.

References

[1] J. Hegemann, C. Jiang, C. Schroeder, and J.M Teran. A level set method for ductile fracture, 2013.

[2] E. Sifakis and J. Barbic. Fem simulation of 3d deformables solids: a practitioner's guide to theory, discretization and model reduction., 2012.

[3] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. Efficient elasticity for character skinning with contact and collisions., 2011.

[4] University of Utah. Febio, http://www.febio.org.

[5] Stanford Univeristy. Physbam, http://physbam.stanford.edu/.

[6] Borek Patzak. Oofem, http://www.oofem.org/en/oofem.html.

[7] L. Vese and T. Chan. A multiphase level set framework for image segmentation using the mumford and shah model., 2002.