REAL-TIME, PLASTICITY-BASED FRACTURE USING THE FINITE ELEMENT METHOD AND RIGID BODY PROXIES

BY

Garrett O'Malley

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
awarded by DigiPen Institute of Technology
Redmond, Washington
United States of America

May
2015

Thesis Advisor: Erik Mohrmann

DIGIPEN INSTITUTE OF TECHNOLOGY

GRADUATE STUDIES PROGRAM

DEFENSE OF THESIS

THE UNDERSIGNED VERIFY THAT THE FINAL ORAL DEFENSE OF THE MASTER OF SCIENCE THESIS TITLED

Real-time, plasticity-based fracture using the finite element method and rigid body proxies

BY

Garrett O'Malley

HAS BEEN SUCCESSFULLY COMPLETED ON May 19th, 2015.

MAJOR FIELD OF STUDY: COMPUTER SCIENCE.

APPROVED:

Dmitri Volper date Graduate Program Director	Xin Li date Dean of Faculty
Dmitri Volper date Department Chair, Computer Science	Claude Comair date President

DIGIPEN INSTITUTE OF TECHNOLOGY

GRADUATE STUDIES PROGRAM

THESIS APPROVAL

DATE: May 19th, 2015

BASED ON THE CANDIDATE'S SUCCESSFUL ORAL DEFENSE, IT IS RECOMMENDED THAT THE THESIS PREPARED BY

Garrett O'Malley

ENTITLED

Real-time, plasticity-based fracture using the finite element method and rigid body proxies

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE AT DIGIPEN INSTITUTE OF TECHNOLOGY.

Erik Mohrmann date	Xin Li date
Thesis Committee Chair	Thesis Committee Member
Jason Hanson date	Gary Herron date
Thesis Committee Member	Thesis Committee Member

ABSTRACT

This thesis presents a survey of real-time finite element method based deformable fracture and presents new techniques in the field. Techniques for simulating fracture, including physically-based and geometrically-based techniques, are described. Considerations are given to real-time vs. non-real-time applications as well as appearance, believability, precision, and accuracy. The focus in this research has been to reduce the computational expense of fracture, the aesthetic appeal of low-resolution fracture meshes, and interaction with rigid-bodies.

ACKNOWLEDGMENTS

For Louisa and her endless patience, support, and encouragement.

For Dr. Erik Mohrmann and his countless explanations to help me build a better simulation.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

List of Algorithms

CHAPTER 1

Introduction

Fracture simulation is of interest in many fields. Simulations for Condensed matter physics and material science require accuracy because of the focus of these fields on understanding material properties [AS88, CRHG96, DM95]. Procedural fracture has been an area of interest to the computer graphics community for over twenty years [TF88]. Fracture simulation is used in offline animation [BHTF07, SSF09], and computing power has progressed to the point where real-time fracture simulation has been demonstrated in interactive and game environments [PEO09, MCK13].

This thesis examines the field of fracture simulation with a primary interest in real-time simulations. Offline animation is discussed primarily in the context of inspiration for real-time simulations. The core of this paper focuses on the use of the finite element method as a means to simulate deformation and fracture in real-time through linear approximations.

In the area of real-time fracture simulation, speed and aesthetic appeal is of utmost concern. The finite element method is a computationally expensive technique and long-standing fracture techniques add expense and another rate limiting step. This thesis presents a new approach to fracture, “Plasticity-Based Fracture,” that is computationally inexpensive. This thesis also presents an algorithm, “Aesthetic Joint Pruning,” to allow low-resolution fracture to look more appealing, and it presents a technique for interacting with rigid-bodies by generating rigid-body proxies at run time.

CHAPTER 2

Physics and Math Primer

1. Physics background

Many modern simulators use physics techniques for creating realism and believability. This chapter introduces the physics of deformation and fracture.

1.1. Hooke's law. Hooke's law is a first order approximation for restoring forces that depend on distance, in one dimension [Gol50, Fre65, TM08]. Hooke's law describes the force a spring would exert on a weight that is either stretching or compressing the spring. As the spring is stretched or compressed, the force increases proportionally. The constant of proportionality, k, depends on the stiffness of the spring. Hooke's law has solutions which are oscillations. It can be written as:

$$\vec{F} = -k\vec{x}, \tag{2.1}$$

where $\vec{F}$ is the force and $\vec{x}$ is the displacement.

Hooke's law can be considered as an elastic stress-strain model [LRK10]. Stress is a measurement of force per area while strain is the ratio of deformation of the initial length. For a spring, the stress is the applied force on the spring, and $k\vec{x}$

is the response behavior of the spring causing strain as long as the spring is not stretched past its elastic region. It is important to note that the strain response is proportional to the applied stress. When generalizing the stress-strain relationship to a continuous medium which allows shear, rather than a simple spring system, the equation becomes:

$$\sigma = \mathbf{c}\epsilon, \tag{2.2}$$

where σ is the stress tensor of the applied forces and shears, $\mathbf{c}$ is the stiffness tensor which is analogous to the previous spring constant, k, and ϵ is the resultant strain.

The off-diagonal elements of σ represent shear stresses. Shear stress is the component of the force which is coplanar to the cross section of a body as opposed to normal stress which is force component perpendicular the material cross section.

For example, compressing marshmallow peep in the x direction causes the peep to expand in the y and z directions. The ratio of this compression to expansion is indicated by Poisson's Ratio, ν.

Poisson's ratio is used to model multi-dimensional interactions. Poisson's ratio is defined as the amount of contraction when strained in a particular direction. For example, an object strained in the x-direction would have a ratio, ν given by

$$\nu = -\frac{\epsilon_y}{\epsilon_x}, \tag{2.3}$$

where ϵ_y is the contraction strain in the y-direction, ϵ_x is the strain in the x-direction.

For a homogenous material, where $\nu_{yx} = \nu_{zw} = \nu_{yz}$, Poisson's ratio applies equally in all perpendicular dimensions, $\epsilon_y = \epsilon_z = -\nu\epsilon_x$. In three-dimensions, the total strain in a particular dimension, e.g. in the x-direction, is the sum of the strain in all the components, e.g.:

$$\epsilon_x = \epsilon_{x \text{ from } \sigma_x} + \epsilon_{x \text{ from } \sigma_y} + \epsilon_{x \text{ from } \sigma_z} = (\sigma_x - \nu\sigma_y - \nu\sigma_z)/E, \tag{2.4}$$

where E is Young's Modulus of the material and with similar equations for the y and z directions.

Young's Modulus, or elastic modulus, is a measurement of the stiffness of a material. It is defined as the ratio of stress along an axis to the strain along that axis.

1.2. Brittle versus ductile fracture. When it comes to deformation, there are two types of behavior: plastic and elastic, [LRK10]. When an elastic object is deformed, the deformations are removed when the applied stress is removed. When a plastic object is deformed some of the deformation becomes permanent. Materials are generally elastic below a material dependent plastic threshold and plastic above it. Both behaviors are common. A piece of rubber that is compressed or stretched will elastically return to its original shape. Aluminum that is bent will remain bent after the bending force is removed.

Fracture is the process of a body splitting into multiple bodies due to applied stress. Elasticity and plasticity create different types of fracture. When a plastic body fractures, extensive plastic deformation is visible after the fracture. This can be described as "pulling apart" rather than "shattering." This can be seen when twisting a plastic spoon. During the twisting process, the rod will begin to tighten around the fracture point and eventually pull apart.

Elastic materials will fracture in a brittle manner by not deforming before breaking, e.g. dropping a piece of glass or a holiday ornament onto the ground. In these cases, the material will shatter rather than pulling apart.

2. Numerical techniques

In order to simulate the physics of deformable and fracturable bodies, solving differential equations through a continuous medium is required. There are many common techniques for solving such systems.

2.1. Finite difference method. Finite difference methods utilize the difference in a function between two discrete evaluation points to approximate derivatives and solve differential equations throughout a volume [MM05]. Typically the derivatives are turned into differences and the equations are solved algebraically, e.g.:

$$\Delta f(x) = f(x + b) - f(x + a), \tag{2.5}$$

where f is a function, x is a location in the function, and a and b are the offset values, typically $a = -b$.

Finite difference methods suffer from the stiff problem [HW04]. A stiff equation is numerically unstable and can require extremely small time steps. This is problematic for real-time applications as a smaller step size requires higher iteration count per graphical frame. In a continuous volume, this difference can evaluate the strain in the local volume by evaluating the displacement of two neighboring nodes.

2.2. Boundary element method. The boundary element method integrates the differential equation over the volume to solve for the function and makes use of Green's theorem to turn a volume integral into an integral over the surface that encloses the volume [Ban94]. The advantage of such a method is the speed and efficiency of the calculations on the boundary. This integral requires fewer evaluations to solve than the volume integral, thus leading to improvements in speed. However, a boundary element method needs to perform offline calculations in order to define the surface of the object for solving. Boundary element methods are inefficient for fracture simulation, due to the prohibitive cost of recalculating the surface.

2.3. Meshless method. Unlike other methods discussed, meshless methods do not work on a system of particles with a defined connectivity. Instead, [LL07], each particle samples the contiuum via the other particles in a local neighborhood, often weighted by distance, or other criteria. It is often used in simulations that allow particles to move independent of each other, or that need variable precision, such as fluid simulations. The first meshless method developed was Smoothed-Particle Hydrodynamic (SPH), which was created for astrophysical applications before being generalized for other uses [GM77, Kel06].

Meshless methods typically scale poorly and are difficult to implement for a deformable body because of the constant recalculation of boundary conditions of the deformable. The disadvantages outweigh the advantages for simulating deformables using meshless methods.

2.4. Finite element method. The finite element method (FEM) has been used in the physical engineering fields for various purposes such as structural analysis [Red05]. The technique is discussed at length in [Say08]. The method discretizes a differential equation over a continuum to a finite number of linear systems evaluated at nodes on the boundary of the continuum, so long as certain boundary conditions are met. In the case of deformation and fracture the differential equation is Newton's second law, $\Sigma\vec{F} = m\vec{a}$, with Hooke's law (equation 2.2) indicating forces related to any strain in the material.

Solving a large number of linear systems leads to the linear complementary problem (LCP), discussed in Appendix A. One technique for solving LCPs is the conjugate gradient method, see Appendix B.

The FEM is advantageous because its representation of complex geometry via discretization is similar to representations commonly used in computer graphics. It is efficient for simulating fracture, as well as being robust and stable, i.e. it doesn't suffer from the stiffness problem. This combination makes it efficient for fracture simulations. Using the FEM to simulate deformation and fracture is the primary focus of this work.

CHAPTER 3

Modeling Deformation

There are many techniques for modeling deformation and fracture. These techniques have different advantages, and therefore have different applications. While fracture is the primary focus of this work, deformation is important for the simulation of ductile fracture.

1. Spring-based deformables

Cloth is one of the earliest deformables to simulate in real time. The work of [Pro96] made use of a mass-spring network to simulate cloth behavior. This implementation makes use of particles connected by three types of springs (see Eqn 2.1). The first type is the structural spring, which connects adjacent particles, i.e. in the center of the cloth, each particle will be attached to each of its four nearest neighbors through structural springs. The structural springs resist stretching and compression of the cloth and maintain its area. Of the three spring types, this typically has the highest stiffness, k.

The second type of spring used in Provot's cloth implementation is the shear spring. The shear springs are connected between diagonal nearest neighbors. Similar to the structural springs, away from the edges of the cloth, each particle will be attached to four diagonal nearest neighbors using shear springs. These help the cloth maintain its shape and aspect ratio. If it's a square piece of cloth, the shear springs will help ensure the cloth remains square and not a diamond shape.

The final type of spring is the bend spring. These springs are connected between next-nearest neighbors in the horizontal and vertical directions. Similar to the previous two types of springs, central particles will be connected to four second nearest neighbors. The bend springs resists bending of the cloth. It helps ensure the cloth will not fold in on itself. Of the three springs, this typically has the lowest stiffness, k.

Another implementation of deformable simulation using springs is the pressure model soft body, [MO03]. A pressure-model soft-body differs from a cloth simulation in that it is a spring-mass deformable model that maintains an interior volume. In a pressure-model soft body, each particle is connect to nearest neighbors on the surface of the body via springs. In order to prevent such a model from collapsing, Matyka et al., simulated a pressure force inside the deformable which pushed against the faces of the body. The pressure force is proportional to the inverse volume, $\frac{1}{V}$, and is applied to each face of the soft-body in the outward normal direction proportional to the area of the face.

Spring-mass models have their limitations. Notably, spring forces are subject to the stiff problem (see Ch. 2.1). For real-time applications this is a hindrance as more complex spring based models require smaller time steps. This is also a limitation on the types of deformables springs model well. It is difficult to extend spring-mass deformables to other types of materials and bodies, such as steel plates or rubber balls, because of the stiff problem. To model such materials, the stiffness values approach infinity which requires prohibitively small time steps.

2. Finite element method deformables

Models using the finite element method (FEM) are the focus of this work. As discussed in chapter 2.4, the FEM breaks a large continuous object into smaller pieces. For graphical applications, it is typically broken into tetrahedral elements via tetrahedralization because of the natural fit with triangle mesh representations commonly used for rendering. Tetrahedralization is outside the scope of this work; there are many open-source projects for performing tetrahedralization on meshes, [dt03].

To clarify terminology which will be used to describe the FEM simulation technique, the discussion of [ESHD05] is followed throughout this section. A “node” is a vertex of the tetrahedron on the FEM mesh and an “element” is a tetrahedral volume element in the deformable. The update loop is a fairly straightforward process (see Alg. 1).

The FEM discretizes the equations of motion for each element. Elements interact with each other via boundary conditions which are used to assemble the global dynamics equation. This leads to a Linear Complementarity Problem [LCP,

Algorithm 1 Update loop for FEM objects

```
function UPDATE(deltaTime)
    UpdateOrientations()
    CalculateStiffnessMatrix()
    AddPlasticityForce(deltaTime)
    DynamicsAssembly(deltaTime)
    ConjugateGradientSolver()
    IntegrateNodePositions(deltaTime)
end function
```

Appendix A] which may be solved using Conjugate Gradient Solver [Appendix B]. Solving the LCP results in a velocity which is used to update the positions of the nodes within the deformable. The updated positions and velocities lead to changes in the strain which are used to update the deformable on the next time-step.

2.1. Time integration. The fundamental equation we are trying to solve is Newton's second law, $\Sigma\vec{F} = m\vec{a}$. In the case of an elastic solid, we are considering several forces, including damping, internal stress, and any external forces, $\mathbf{f}_{ext}$ (e.g. gravity, collisions, etc). For an elastic solid, our stress term is given by (see Sect. 1),

$$\mathbf{Ku} = \mathbf{f}, \tag{3.1}$$

where $\mathbf{K}$ is the stiffness matrix, $\mathbf{u}$ is a vector of nodal displacements, and $\mathbf{f}$ is a vector of node forces (see Sect. 2.4). The nodal displacements are given by:

$$\mathbf{u} = \mathbf{x} - \mathbf{x}_u, \tag{3.2}$$

where $\mathbf{x}$ is the current position of the node, $\mathbf{x}_u$ is the position of the node in the undeformed object.

The nodal displacements can be used to calculate the strain of a body using the deformation gradient which is the change in world coordinates of a point with respect to the material coordinates. The deformation gradient,F_{ij} is given by:

$$F_{ij} = \frac{\partial u_i}{\partial x_j}, \tag{3.3}$$

where i and j are indices of the nodes within a given element.

The linear Cauchy strain matrix, ϵ, is an approximation of Green's strain tensor, and is used to measure the deformation of a body. The Cauchy strain matrix is given by,

$$\epsilon_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}). \tag{3.4}$$

The damping term in our equation is a velocity damping term given by, $\mathbf{C}\dot{\mathbf{x}}$ where $\mathbf{C}$ is our damping matrix, and $\dot{\mathbf{x}}$ is the time-derivative of our position (i.e. velocity). Putting all of these terms into Newton's second law, we get:

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}(\mathbf{x} - \mathbf{x}_u) - \mathbf{f}_{ext} = 0, \tag{3.5}$$

where $\mathbf{M}$ is our mass matrix. We want to integrate equation 3.5 using implicit discretization which means we evaluate both $\mathbf{x}$ and $\mathbf{v}$ at time $m+1$, and substitute $\ddot{\mathbf{x}} = \frac{\mathbf{v}^{m+1} - \mathbf{v}}{\Delta t}$, we get

$$\mathbf{M}\frac{\mathbf{v}^{m+1} - \mathbf{v}}{\Delta t} + \mathbf{C}\mathbf{v}^{m+1} + \mathbf{K}(\mathbf{x}^{m+1} - \mathbf{x}_u) - \mathbf{f}_{ext} = 0, \tag{3.6}$$

and further substituting, $\mathbf{x}^{m+1} = \mathbf{x}^m + \mathbf{v}^{i+1}\Delta t$, yields,

$$\mathbf{M}\frac{\mathbf{v}^{m+1} - \mathbf{v}}{\Delta t} + \mathbf{C}\mathbf{v}^{m+1} + \mathbf{K}(\mathbf{x}^m + \mathbf{v}^{m+1}\Delta t - \mathbf{x}_u) - \mathbf{f}_{ext} = 0, \tag{3.7}$$

after algebraic manipulation and substituting, $\mathbf{f}_u = -\mathbf{K}\mathbf{x}_u$, results in,

$$(\mathbf{M} + \Delta t\mathbf{C} + \Delta t^2\mathbf{K})\mathbf{v}^{m+1} = \mathbf{M}\mathbf{v}^m - \Delta t(\mathbf{K}\mathbf{x}^m + \mathbf{f} - \mathbf{f}_{ext}), \tag{3.8}$$

where we finally have an equation of the form $\mathbf{A}\mathbf{v}^{m+1} = \mathbf{b}$ where,

$$\mathbf{A} = \mathbf{M} + \Delta t\mathbf{C} + \Delta t^2\mathbf{K}, \tag{3.9}$$

and

$$\mathbf{b} = \mathbf{M}\mathbf{v} - \Delta t(\mathbf{f}_u - \mathbf{f}_{ext} + \mathbf{K}\mathbf{x}). \tag{3.10}$$

This linear complementarity problem (App. A) may be solved with a variety of methods, such as the conjugate gradient method (App. B). The remainder of section 2 discusses the assembly of the component in Eq 3.9 and Eq 3.10, as well as methods to handle numerical issues that arise in their solution.

2.2. Stiffness matrix. As described by [ESHD05], the stiffness matrix is the concatenation of several other matrices, namely the matrices describing the stress and strain of the object. There are two important "versions" of the stiffness matrix: the local stiffness matrix and the global stiffness matrix. The local stiffness matrix represents the stiffness between nodes within a volume element. The global stiffness matrix is the concatenation of all the local stiffness matrices (see Alg 2), where Re is the orientation matrix of element e, Ke is the local stiffness matrix of element e, and K is the global stiffness matrix.

The local stiffness matrices are stored as $4x4$ matrices in order to reduce memory footprint. This is possible because the off-diagonal $4x4$ matrices are transposes of each other, [ESHD05].

Algorithm 2 Assembling the global stiffness matrix and global force vector

```
function CalculateStiffnessMatrix
    for all element $e$ do
        for all node $i$ of $e$ do
            for all node $j$ of $e$ do
                if $j \geq i$ then
                    $tmp = Re * Ke_{ij}*$Transpose($Re$)
                    $K'_{ij} = K'_{ij} + tmp$
                    if $j > i$ then
                        $K'_{ji} = K'_{ji}+$Transpose($tmp$)
                    end if
                end if
            end for
            $f_i = f_i + f^e_m$
        end for
    end for
end function
```

In the most physically accurate simulation, both of the global and local stiffness matrices would be updated each frame. In order to increase efficiency the local stiffness matrices are usually calculated once, at the time of the deformable object's creation. Updating the local stiffness matrix adds little accuracy for the cost.

2.3. Plasticity forces. Plasticity is the permanent deformation of a deformable [ESHD05]. To simulate this, a plastic force is calculated when the object deforms. This internal force leads to a permanent deformation of the object.

Three parameters control plasticity: plastic yield, plastic creep, and plastic max. The plastic yield sets the minimum deformation threshold for accumulating plastic force. If the total stress exceeds the plastic yield, then some of the excess is added to the plastic force. The plastic creep determines what fraction of the excess is added to the net plastic force. Lastly, the plastic max is the largest possible net plastic force.

2.4. Dynamics assembly and lumped mass matrix. The dynamics matrix (Eqn 3.9) and the dynamics vector (Eqn 3.10) are assembled in a method similar to that discussed earlier for assembling the global stiffness matrix and force vector (see Sect 2.2). The pseudocode for calculating the force offsets can be seen in Alg. 3, where Ke is the local stiffness matrix of element e and Re is the orientation matrix of element e.

Algorithm 3 Calculating offset force

```
function CalculateOffsetForce
    for all element $e$ do
        for all node $i$ of $e$ do
            $tmp = 0$
            for all node $j$ of $e$ do
                $tmp = tmp + Ke_{ij} * xu_j$
            end for
            $fu'_i = fu'_i - Re * tmp$
        end for
    end for
end function
```

The mass matrix, M (see Eqns 3.9 and 3.10), storage method has a huge impact on performance. Either the mass is stored per node or per element. Dynamics assembly is the only piece of the simulation that is changed by this choice. The choice has no simulation accuracy consequences but has a performance effect in terms of both storage and CPU calculations. As shown in [ESHD05], pgs 356 to 358, the per node mass matrix is a fully filled out matrix with no zeroes. The volume-element-based mass matrix is dubbed the “lumped mass matrix” because it is a diagonal matrix.

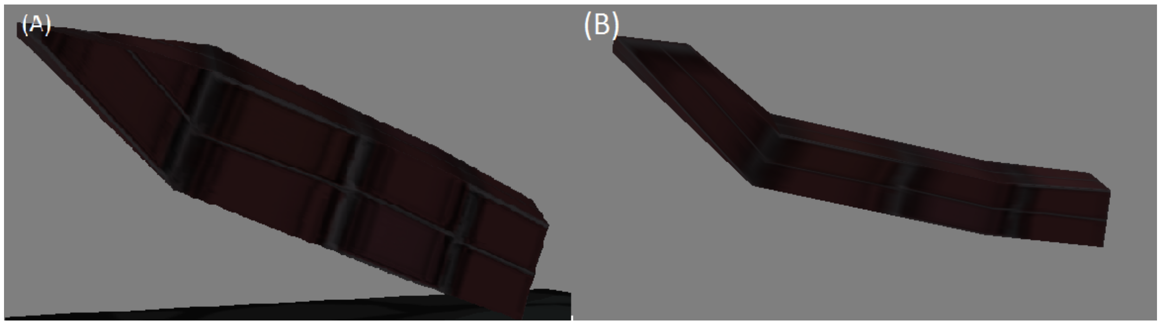

Figure 1. Comparison of stiffness warping.

(A) Large deformation with no stiffness warping. (B) Stiffness warping turned on. This artifact occurs due to linearization of the strain tensor.

The lumped mass matrix results in faster calculations, less memory use, and simpler algorithms. There is also another convenient consequence of this choice. The damping matrix is defined by:

$$\mathbf{C} = \alpha\mathbf{M} + \beta\mathbf{K}, \tag{3.11}$$

where α is the mass damping parameter and β is the stiffness damping parameter.

According to [ESHD05], it is common practice to set $\beta = 0$ which means the damping matrix is proportional to the mass matrix. Because the simulation uses the lumped mass matrix, this means the damping matrix is also a diagonal matrix. This results in even more computational and storage savings.

2.5. Stiffness warping. As discussed earlier (see Sect 2.1), the linear approximation to the elasticity results in large volume growth of elements when objects are subjected to large rotational deformations. This is not accurate or believable behavior. Müller et al. describe a method for dealing with this [MG04].

In order to correct this error, the stiffness forces are calculated in the undeformed frame of the FEM. This is a three step process: rotate the current position back to the rest frame, calculate the stiffness force, then rotate the stiffness force back to the world frame. The orientation matrix must be calculated. Two methods are described in [ESHD05]: polar decomposition or Gram-Schmidt orthonormalization (see App C). An example the effects of stiffness warping can be seen in Fig. 1.

2.6. Element inversion. Element inversion is when one node of an element can be forced onto the other side of the triangle formed by the other three nodes, thus inverting the tetrahedron. Element inversion can happen when a strong force interacts with the FEM deformable. This leads to incorrect windings (and normals) for the element. Without correcting this, the FEM deformable will typically explode.

Irving et al. presented a method for dealing with this [ITF06]. Before calculating the stresses and forces in the FEM deformable, they perform all calculations using the deformation gradient. The deformation gradient measures the change in the deformation of the body from one point to another. An element within the deformation gradient, $\mathbf{F}$ (see Eqn 3.3) where x is the current, deformed configuration, and u is the undeformed reference configuration. While this method is generally more robust than other methods, it is more expensive and slower.

Teschner et al. described a method which adds a volume preservation term into the FEM calculations [THMG04]. This volume preservation term is zero unless the element is inverted in which case it adds a force working against the inversion.

A third method is described by Nesme et al. [NPF05]. When calculating the rotation matrix, they check that the last vertex is on the correct side of the triangle formed by the other three vertices in the tetrahedron. If not, they flip the last basis axis to make the basis right-handed. The methods presented by Teschner et al. and Nesme et al. are both efficient. The method of Nesme et al. fits well with the stiffness warping correction presented by Müller et al. [MG04].

3. Multi-physics: interacting with rigid bodies

There are several methods for producing interactions between deformables and rigid bodies. Among those methods are one-way coupling, two-way coupling, and embedding. Davis performed comparisons between these methods, [Dav13].

One-way coupling is typically performed with one primary system and one passive system. The primary system can interact with the passive system, but the passive system cannot affect the primary system. While not physically-realistic, it can be used to great effect for visual purposes.

For example, a passive cloth simulation coupled to a rigid-body engine as the primary system. In such a scenario, the cloth would be pushed around by the rigid bodies, but the rigid-bodies would be unaffected by the cloth. In systems with a large mass asymmetry, this can produce believable results.

Two-way coupling is more complicated because the "passive" system can affect the "primary" system which means it is no longer passive. One important consequence of two-way coupling is that both systems must run at the same speed. For example, if cloth simulation system runs at 120 fps and a rigid body simulation runs at 60 fps, then the interactions between the two systems must run at 120 fps. Hence the computation cost of the rigid-body system is doubled by the two-way coupling.

Fishman described the embedding method [Fis12]. Similar to one-way coupling, embedding has one primary system and other secondary systems. Secondary systems that require two-way interaction with the primary system are embedded into proxies for the primary system. The primary system does not define interactions with secondary systems.

For example, consider interaction between rigid bodies and FEM deformables. In such an interaction, the rigid bodies could be the primary system. When an FEM is created, it is embedded into a rigid-body proxy. The rigid-body proxy then interacts within the rigid-body system, i.e., the rigid-body system sees the proxy as just another rigid body. During the rigid-body solving step, the rigid-body system applies resolution to the proxy the same way it would to any other rigid body. After this resolution step, the proxy reports the results to the embedded FEM. At this point, the FEM update step is run to calculate the resultant behavior of the deformable. This is the approach taken by [Fis12], because it is fast and efficient. The opposite approach of using rigid bodies as the secondary system and embedding them into an FEM was performed by Parker et al. [PEO09]. They only generated proxies for rigid bodies upon collision with an FEM deformable. This approach is slower, however it was required to implement proxies within a third-party rigid-body engine.

CHAPTER 4

Modeling Fracture

There are a wide range of applications for simulating fracture. The techniques typically fall into one of two categories: physically-believable simulations and physically-accurate simulations.

Fracture simulation is an area of interest for the physical science and engineering fields, notably condensed matter and material science [AS88, CRHG96, DM95]. In the physical sciences there is typically no user interaction nor graphical display of the fracture process. The goal is most often to understand properties of materials. These simulations are a prime example of simulation for accuracy, with little to no concern over the visual appeal of the fracture. The materials are often modeled using a lattice network. Each node in the lattice is a point mass connected to neighboring nodes via a spring-like force. The node motion is then simulated based on the spring forces affecting each node. These simulations are usually run with a time-step on the order of $1/c$ where c is the speed of sound in the material being simulated. This step size is far too small for the graphics and animation community hence the tendency to shy away from techniques such as this.

Medical surgery simulations are often found in a middle ground between physical-accuracy and physical-believability. The surgery simulations are typically run in an interactive fashion [BG00, BGTG03]. The goal of these simulations is to provide a surgery training tool and a planning tool. Biesler et al. addressed the issues of collision detection between the surgical tool and tissue, tissue relaxation after fracture, and optimization for interactivity.

One of the earliest fracture simulations for graphical purposes is the seminal work of [TF88]. Animation for movies is one of the primary applications of non-real-time physically-believable fracture simulation. Often the primary goal is convincing the viewer of the resultant animation behavior. Irving et al. animate human muscles during a key-frame motion of a skeleton in order to create a more plausible and believable animation [ITF06]. The results of Bao et al. include fracturing thin shell ornaments dropping on the ground [BHTF07]. Su et al. expand on previous work to allow faster fracture for an interactive frame-rate simulation [SSF09].

Real-time animation is almost entirely focused on speed and believability. These simulations are typically brief and bear less scrutiny as non-real-time animation, thus they sacrifice accuracy for performance. Parker et al. demonstrate fracture techniques in a game, *Star Wars: The Force Unleashed* [PEO09]. They implemented a FEM deformable based fracture technique (see section 2). Müller et al. demonstrate a real-time fracture simulation involving meteors destroying a coliseum in a convincing fashion, [MCK13] (see section 1).

1. Geometrically-based fracture

One technique for fracture is using geometry to determine fracture planes after a collision. This avoids the need for expensive dynamics calculations at run time. This technique is usually applied to brittle fracture, as a deformation model would still be required for ductile fracture. Geometrically-based techniques rely on applying a prescored fracture pattern onto an impact point. This results in objects not fracturing under their own internal stresses.

One issue with prescored fracture patterns is the constraint on artist's control. If the fracture pattern is applied near the edge of the object, then part of the fracture pattern will go off the edge of the object. Su et al. address this problem by allowing the prescored pattern to be scaled in addition to translated, [SSF09]. In this method all of the pattern is guaranteed to be used.

Another issue with prescored fracture techniques is making them appear different enough each time to be visually appealing. Müller et al. address this by performing "volumetric approximate convex decompositions" (VACD) on the fracturing body, [MCK13]. They set out three rules for meshes:

- "They are composed of convex pieces."
- "The pieces do not overlap."
- "Two pieces are physically connected if and only if one piece has at least one face that (partially) overlaps a face on the other piece and the two faces lie in the same plane with opposite normals." [MCK13]

The meshes they create via VACD fit these requirements, and they refer to said meshes as “compounds.” The algorithm proceeds as outlined in Algorithm 4. The fracture that develops from this algorithm is more dynamic than other geometrically-based algorithms because it depends on both the fracture pattern and the composition of the compound.

Algorithm 4 VACD Fracture

```
function FractureVACD
    Align fracture pattern to impact point
    Compute intersection of all cells with all convexes
    Fuse connected pieces into one object
    Form compounds from convex pieces in a cell
    Detect disjoint islands and separate them
end function
```

Geometric-based fracture methods involving Voronoi regions are presented in [BHTF07] and [SO14]. These methods are a hybrid between geometrically-based and physically-based fracture (see section 2). Using this method a fracture pattern is applied at the point of impact and the propagation is initiated based on the energy in the Voronoi cells of the entire body being fractured. In other geometrically-based fracture, the entire body is often not taken into consideration.

2. Physically-based fracture using FEM deformables

There are a number of ways physically-based fracture has been simulated. Here we discuss physically-based fracture using FEM deformables.

O’Brien et al. used a very similar method to simulate brittle fracture in [OH99] and ductile fracture in [OBH02]. Their basic deformable model is based on the finite element method in a similar fashion as outlined in section 2. After each time step, they resolve internal forces into tensile and compressive components. At each node, those forces are used to calculate a tensor describing how internal forces are acting to separate that node. Their algorithm proceeds as follows,

- Resolve internal forces into tensile and compressive forces, discarding unbalanced portions
- Form tensor describing internal forces that are acting to separate that node
- If forces are large enough, split node and calculate splitting plane
- All elements attached to node are split along the plane

The eigenvalue is calculated in order to separate tensile and compressive forces out of the stress tensor of an element. The tensile component, σ^+, corresponds to the positive eigenvalues while the compressive component, σ^+ correspond to the negative eigenvalues, which are calculated by,

$$\sigma^+ = \sum_{i=1}^{3} \max(0, v^i(\sigma))\mathbf{m}(\hat{\mathbf{n}}^i(\sigma)) \tag{4.1}$$

$$\sigma^+ = \sum_{i=1}^{3} \min(0, v^i(\sigma))\mathbf{m}(\hat{\mathbf{n}}^i(\sigma)) \tag{4.2}$$

where $v^i(\sigma)$ is the ith eigenvalue of the stress tensor, σ, $\hat{\mathbf{n}}^i(\sigma)$ is the corresponding unit length eigenvector, and $\mathbf{m}(\mathbf{a})$ is a symmetric matrix with $|\mathbf{a}|$ as an eigenvector, defined by,

$$\mathbf{m}(\mathbf{a}) = \begin{cases} \mathbf{a}\mathbf{a}^T/|\mathbf{a}| & : \mathbf{a} \neq 0 \\ 0 & : \mathbf{a} = 0 \end{cases} \tag{4.3}$$

The tensile and compressive forces are concatenated into a separation tensor. If the largest positive eigenvalue, v^+ of this tensor is greater than the material toughness, τ, then the material fails at the node. The corresponding eigenvector of this eigenvalue is used as the normal of the splitting plane. Once the splitting plane is calculated, all elements associated with the splitting node are examined to determine if they need to be remeshed. The remeshing algorithm, as outlined by O'Brien et al. [OH99] proceeds as follows,

- The splitting node is replicated into two nodes, n^+ and n^- on the positive and negative sides of the splitting plane, respectively
- All elements attached to the original node are examined
- If an element is not intersected by the splitting plane, then the element is assigned to either n^+ or n^- depending on which side of the splitting plane the element lies.
- If the plane does intersect an element, then that element is split along the plane. For each edge intersecting the plane, a new node is created. The intersected element will be split into three tetrahedrons because all elements must be tetrahedrons.

One important distinction in [OH99] is the purposeful neglect of plasticity. This neglect of plasticity results in brittle fracture. Their later work includes the plasticity term which allows for the resultant behavior to include ductile fracture. The inclusion of plasticity has only minor affects on the fracture algorithm itself however this change results in ductile fracture because the deformable will result in permanent deformation before the fracture.

Similar to O'Brien et al, Müller et al. perform brittle fracture on FEM deformables, [MMDJ01]. Depending on the maximum eigenvalue in the stress tensor and the material type of the body being fractured, a radius of fracture, r_{frac}, is calculated. All tetrahedron within r_{frac} are marked as either positive or negative depending on which side of the fracture plane they lie on. The tetrahedrons with opposite signs are disconnected. Müller et al. note that the fracture plane can be used to split the tetrahedron at the origin of the fracture if that tetrahedron is particularly large. They also note that their method is advantageous because crack growth is independent of both time step and granularity of the FEM mesh.

In [IO09], only surface cracks are simulated, rather than full fracture. As a follow up to the work discussed in [OH99] and [OBH02], the researchers used the finite element method to discretize their deformables. A triangle mesh is used for discretization instead of tetrahedrons, as they are only concerned with the surface. Additionally, the stress field is initialized with heuristics instead of a full FEM simulation. They use a four-part algorithm to generate and display the surface cracks as outlined in algorithm 5.

Surface crack generation algorithm in [IO09]

```
function SurfaceCrackGeneration
    Initialize stress field according to heuristics
    For each node compute failure criteria and store in a priority queue
    Crack the surfaced via user input and propagate via the stress field
    Display by moving the nodes or rendering cracks directly
end function
```

Parker et al. simulated FEM deformables fracture in a real-time game engine, [PEO09]. In previous works, the fracture was allowed to split individual tetrahedral elements. In the work by Parker et al. this splitting is removed as an optimization. Instead, the FEM deformables are only allowed to split along tetrahedral boundaries. Additional differences include the idea of embedding the rigid-body system in an FEM proxy (see section 3. There were other optimizations they implemented as well. Another major optimization is the use of islands in the mesh. The islands are formed at runtime based on nodal adjacency and are put to sleep. Sleeping islands are not actively updated, unless they are awoken by a collision. Further optimization includes not running the fracture algorithm every frame as well as parallelizing certain calculations (e.g. stress tensor updates, eigenvalue calculation, etc).

Parker et al. used rather coarse FEM meshes. In order to enhance the visual appeal of the fracture, they separated the graphical and FEM meshes. In order to couple the graphical mesh to the FEM mesh, they assign each point, in a graphical polygon to a tetrahedron in the deformable mesh. For a given point, a, the barycentric coordinates within the tetrahedron are calculated. As the tetrahedron is deformed, the position of a is updated based on the barycentric weights.

3. Other techniques in fracture

3.1. Fracture on the GPU. The expensive computations of fracture simulation are highly parallelizable. This lends itself to doing the calculations on the GPU. Real-time simulations are increasingly relying on the GPU, [PEO09], [MCK13]. A modular framework for simulating deformation and fracture on the GPU is presented in [MAP12]. Parker et al. [PEO09] use the GPU to calculate values such as stress tensors in their fracture algorithm (see sections 2 and 2).

3.2. Other fracture models. Pauly et al. modeled a deformable using a meshless method, [PKA$^+$05]. Their fracture technique is easier for nodes to split as they are not connected via a mesh compared to an FEM deformable fracture model. One downside is the expensive meshless method simulation instead of an FEM simulation for modeling the deformable itself (see section 2.3).

Glondu et al. simulate brittle fracture using modal analysis, [GMD13]. They break down the fracture algorithm into three parts. The first part is fracture initiation. For each fracturable body, an offline modal analysis estimates contact properties and deformation properties. Modal analysis is a measurement of vibrational modes of a material. In a material, certain frequencies are more likely to resonate. Resonance and vibrational modes are an enormous topic outside the scope of this work. Such topics are covered quite extensively in classical mechanics textbooks, such as [Gol50], [Fre65], or [TM08]. These properties are used to determine when and where fracture will be initiated on the body. Propagation of the fracture through the body is determined from a calculation of the stress state of the material, which is to say, once a crack tip is formed, it propagates in the direction of maximal stress. Generation of fracture fragments happens as they separate from the body. Glondu et al. stated the intent to extend this model to ductile fracture.

Smith et al. simulated brittle fracture through the use of distance-preserving linear constraints, [SWB00]. A fracturable body is a collection of particles connected via distance constraints. When colliding with another object, the distance constraints are broken at the point of impact. If the force applied to a particle is strong enough to break a constraint, it then weakens constraints around it. These weaken constraints are more likely to break, thereby propagating cracks.

3.3. Post-processing techniques. Another approach to visually appealing fracture in real-time is the use of post-processing effects. Busaryev et al. [BDW13] used a FEM deformable approach to brittle fracture, similar to [OH99]. In addition, there is post-processing adaptive remeshing around the fractured area. After the fracture, the newly exposed area is remeshed with a finer mesh to represent a torn area. We note that in the work presented by Busaryev et al, they focused on multi-layered thin plates. In follow-up work presented in [CYFW14], they demonstrate a similar technique as applied to a 3D FEM deformable fracture.

CHAPTER 5

Fracture Contribution

The fracture technique of this work uses an elastostatic calculation of stress and strain and calculates a separation tensor at each node within a finite element method (FEM) deformable similar to that of Parker et al. [PEO09] and O'Brien et al. [OH99]. Fracture determination is made by analyzing this separation tensor. This work only fractures along element boundaries and does not split elements, as does Parker et al. [PEO09].

This work has several key differences from previous work. First, the work uses a custom algorithm for determining when to perform fracture based purely on aesthetics dubbed Aesthetic Joint Pruning (see Sect. 4.1). Second, this work uses rigid-body-proxies as a means for fracturable objects to interact with a rigid-body physics engine (see Sect. 5). Finally, this work presents a new "Plasticity-based fracture" technique for calculating fracture at nodes (see Sect. 6) that is compared to the separation tensor based model.

The fracture algorithm (see Alg. 6) is performed independently on each island with each sub-step outlined below.

Algorithm 6 Fracture algorithm

```
function FractureCalculation
    DecomposeInternalForces
    CheckNodesForFracture
    PerformFracture
end function
```

1. Simulation architecture

Fracture simulation requires a thoughtful approach to general architecture for simulation purposes. For a deformable that cannot fracture, a single deformable entity is common practice, e.g., a finite element method (FEM) based deformable might have an engine entity called "FEMDeformable." This entity would contain all relevant data for simulating the deformable as well as all the calculations to perform the simulation.

Upon introduction of FEM-based fracture simulation, the simulation architecture must be flexible enough to support splitting of a deformable into two or more objects in a robust, efficient manner. Hypothetically, all volume elements and FEM-nodes could be used in a global CGM solver. However, such an architectural choice would eliminate the possibility for parallelization of the fracture simulation. This work utilizes a fracture architecture that allows islanding [PEO09].

1.1. Data storage. The storage method of the node and element data has a significant impact on the performance of an island-based FEM solver. This work stores the elements and nodes in manager classes, one manager for volume elements and one for nodes. The managers are structured in the same manner, using an arraylinked-list to track elements for efficient insertion and removal. Each manager has an underlying array data structure. Each entity in the array contains indices referring to the previous and next entities in the list, i.e. a doubly-linked list. Unused entities are stored in a free list. Should an entity be destroyed, its data is cleared and it is returned to the free list.

The node data includes:

- Matrix container for global stiffness matrices associated with that node
- Matrix container for global dynamics matrices associated with that node
- Dynamics vector
- Physical data: position, velocity, etc
- A list of which elements are currently using this node

The matrix containers are an STL map where the key is the index of a node interacting with the current node and the value is a 3x3 matrix.

The element data includes:

- Orientation matrices
- Elasticity data: Young's modulus, Poisson's ratio, etc
- Physical data: mass, volume, etc
- A list of nodes used by this element

Because the elements and nodes are managed by a global system, there is no need for an FEM-based deformable entity to manage them. In order to render the simulation, a global list of exposed faces is tracked. Upon fracturing, newly exposed faces are added to this list. The list of faces is used to update a list of vertices, normals, and texture coordinates to be drawn.

1.2. Islanding and solving. Because the elements contain which nodes compose this element, and the nodes keep track of which elements are using that node, a "connectivity graph" is formed that is used to form islands. At each simulation time step, nodes and elements are traversed in an exhaustive breadth-first manner with elements representing the "graph nodes" and the FEM-nodes representing the "graph edges." The elements and nodes are set on islands during this traversal. This is very similar to a contact graph used to form islands in a rigid-body engine [Cat07].

Because individual islands have no connection to one another, they can be solved independently. As an additional optimization, the simulation caches islands if no fracture event occurred on that island. Island formation is handled by the "FEM-solver" subsystem.

1.3. Rendering. The approach for rendering is a fairly simple one. The FEM-solver system maintains a list of "Graphical-Elements" to be rendered. A graphical-element is a key-value pair with a key of the FEM Element ID and a value which stores the local node indices which make up the face to be rendered. Separately, FEM-solver sends data to the GPU for rendering, including vertex position, normal, and texture coordinates. This GPU data is accessible to systems outside of the simulation. After each island has been solved, FEM-solver uses the list of graphical-elements to update the GPU data including the position and normal.

1.4. Creation of rigid-body proxies at run-time. This work uses rigid-body proxies (RBP) as the mechanic for interactions between the rigid-body system and the FEM system (see Sect. 3). Because of the overhead associated with collision detection and updating the proxy data, the number of RBPs should be minimized. For this reason, not all FEM elements are given a RBP. Creation of RBPs cannot be determined at compile time because fracture events can expose internal elements. Instead, only volume elements with exposed faces have a RBP. Using the aforementioned graphical-element list, the FEM elements are informed whenever a new graphical face is exposed. When an element is informed of its first exposed face, the element will create a proxy. This reduces the number of proxies and provides some boost to performance.

2. Decomposing internal forces

Decomposing the internal forces in an element is the process of breaking the stress into tensile and compressive forces at each node in the element.

Recall that the stiffness matrix, **K**, is given by:

$$\mathbf{K} = V_e \mathbf{B}^T \mathbf{D} \mathbf{B}, \tag{5.1}$$

Algorithm 7 Decomposing internal forces

```
function DecomposeInernalForces
    for all element $e$ do
        Compute stress tensor
        Decompose stress tensor into tensile and compressive components
        for all node $i$ of $e$ do
            Calculate and apply tensile and compressive forces
        end for
    end for
end function
```

where $\mathbf{B}$ is gradient of the element's shape functions, V_e is the element's volume, and $\mathbf{D}$ is the elasticity matrix.

Since the force on a node is given by $\vec{f} = \mathbf{K}\vec{u}$, the stress tensor is given by

$$\sigma = \mathbf{D}\mathbf{B}\vec{u}. \tag{5.2}$$

where both $\mathbf{D}$ and $\mathbf{B}$ are both static per element and their product could be computed at initialization.

O'Brien et al. present a method to decompose the stress tensor into tensile and compressive components [OH99]. Given a vector $\vec{a}$ in $\mathbb{R}^3$, a symmetric 3x3 matrix, $\mathbf{m}(\vec{a})$ with $\vec{a}$ as an eigenvector can be constructed using the outer product,

$$\mathbf{m}(\vec{a}) = \begin{cases} \vec{a}\vec{a}^T/|\vec{a}| & : \vec{a} \neq \vec{0} \\ \mathbf{0} & : \vec{a} = \vec{0}. \end{cases} \tag{5.3}$$

Let $\mathrm{v}^i(\sigma)$ be the ith eigenvalue of σ with a corresponding unit length eigenvector $\hat{\mathbf{n}}^i(\sigma)$. The tensile, σ^+, and compressive, σ^-, components of the stress tensor can be calculated by

$$\sigma^+ = \sum_{i=1}^{3} \max(0, \mathrm{v}^i(\sigma))\mathbf{m}(\hat{\mathbf{n}}^i(\sigma)) \tag{5.4}$$

$$\sigma^- = \sum_{i=1}^{3} \min(0, \mathrm{v}^i(\sigma))\mathbf{m}(\hat{\mathbf{n}}^i(\sigma)). \tag{5.5}$$

Using these, the tensile,$\vec{f}^+$, and compressive,$\vec{f}^-$ forces from an element onto each node can be computed using

$$\vec{f}^+ = \mathbf{B}^T \sigma^+. \tag{5.6}$$

The compressive force can be calculated in a similar fashion using the compressive stress tensor. Alternatively, it can be calculated using $\vec{f} = \vec{f}^+ + \vec{f}^-$. These forces are stored on each node as they are computed for later use in the fracture algorithm.

3. Checking nodes for fracture

Each node must be checked for fracture conditions after all elements using a particular node have finished calculating their tensile and compressive forces on that node. Checking nodes for fracture (see Alg. 8) is fairly straight forward. There are two main parts to the algorithm: computing the separation tensor and calculating eigenpairs. The separation tensor is formed from the tensile and compressive forces acting on a node. The eigenvalues and corresponding eigenvectors (i.e. eigenpairs) are used to determine if and how the fracture occurs.

The simulation calculates the separation tensor similarly to O'Brien et al. [OH99]. Let $\{\vec{f}_i^+\}$ and $\{\vec{f}_i^-\}$ represent the set of compressive and tensile forces on the ith node, respectively. The set of forces on the node are denoted by $\{\vec{f}_i\}$.

Algorithm 8 Checking nodes for fracture

```
function CheckNodesForFracture
    for all node $n$ do
        Compute separation tensor
        Compute eigenvalues $\lambda$ of separation tensor
        if $\lambda >$ material toughness then
            Fracture occurs at that node
            Fracture plane defined by corresponding eigenvector
        end if
    end for
end function
```

The corresponding sums of the compressive, tensile, and total forces on a node are represented by $\vec{f}_i^+$, $\vec{f}_i^-$, and $\vec{f}_i$. Using the $\mathbf{m}(\vec{a})$ notation (see Sect. 2), the separation tensor can be calculated as,

$$\zeta = \frac{1}{2}(-\mathbf{m}(\vec{f}_i^+) + \sum_{\vec{f}\in\{\vec{f}_i^+\}} \mathbf{m}(\vec{f}) + \mathbf{m}(\vec{f}_i^-) - \sum_{\vec{f}\in\{\vec{f}_i^i\}} \mathbf{m}(\vec{f})). \tag{5.7}$$

For calculating the eigenpairs of this tensor, the simulation uses an open-source solution, Eigen [GJ+10]. When checking for fracture, the simulation uses only the largest, positive eigenvalue to check for fracture.

4. Perform fracture

Once a fracture plane has been found, fracture must be performed. The simulation uses an approach similar to that of Parker et al. [PEO09]. The general algorithm is outlined in Alg. 9.

There are several subtleties in the fracture algorithm.

- Instead of performing all fracture events each frame, the simulation only performs the largest fracture event in a given time-step.

Algorithm 9 Performing fracture

```
function PerformFracture
    for all Fracture events do
        Replicate fracture node
        for all Element $e$ on original node do
            if $e$ is on positive side of fracture place then
                Replace original node with replicated node
                Attach $e$ to replicated node
            else
                $e$ retains relation to original node
            end if
        end for
        Recalculate cached data
        Expose new element faces
        Aesthetic Joint Pruning
    end for
end function
```

- For determining which side of the fracture plane an element resides on, a distance from plane to element center calculation is used.
- Fracture calculations are skipped on islands with 3 or less elements for aesthetic reasons. Fracture events on such islands produce single-tetrahedral islands.
- Node mass needs to be recalculated on fracture as node mass is determined by the elements that are sharing that node.
- Sometimes a fracture event creates a node with no elements attached. In these cases the replicated node is destroyed, and the fracture event is skipped.
- In order to expose element faces, the simulation checks each element for a face-face connection based on pre-fracture node list.

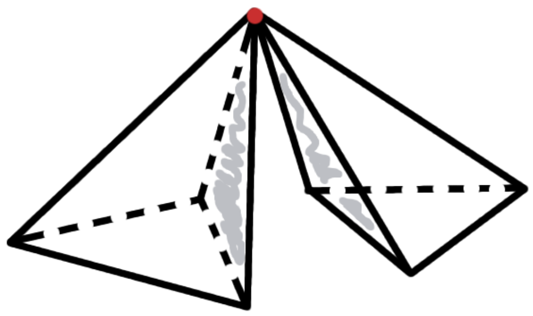

Figure 2. Ball joint.

A ball joint where two tetrahedrons share a node.

4.1. Aesthetic Joint Pruning. The simulation's post-fracture visual clean up, or Aesthetic Joint Pruning, is an algorithm to determine when additional fractures should be performed. This is the first presentation of such an algorithm.

During a fracture event, hinge and ball joints can form. A ball joint is when two elements are connected by a single node (see Fig. 2). A hinge joing occurs when two elements are connected by a single edge (see Fig. 3). This is an artifact of elements only being connected at shared nodes and is not visually appealing. The simulation checks for these joints and breaks them. If an element has three exposed faces that share a single fracture node, then it must have a ball joint. This implies that if all faces attached to a node are exposed, there is a ball joint at that node (see Fig. 2). In this case, the simulation replicates the node and attaches the element with the exposed faces to the new copy of the node, eliminating the ball joint.

Certain cases are not caught by this simple algorithm, such as the case where a hinge forms on a double-tetrahedron. A more robust algorithm might look at all exposed faces attached to a node and count them in order to determine if a ball

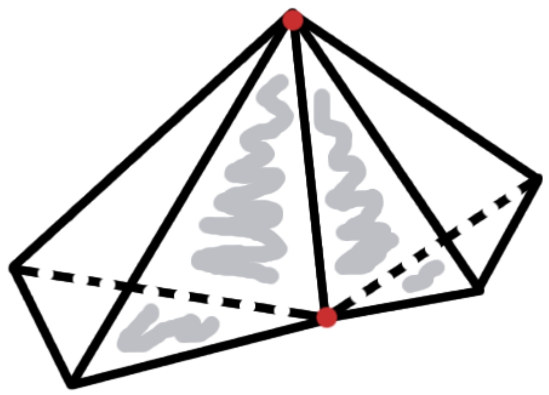

Figure 3. Hinge joint.

A hinge joint where two tetrahedrons share an edge.

joint has occurred. Alternatively a robust graph traversal using face-face connections between elements could also determine if this case exists. These cases are rare and the computational overhead of dealing with them was deemed unnecessary.

For a hinge joint (Fig. 3), there must be two exposed faces on either side of an edge between two nodes on an element. In this case, each node on the edge must be used by multiple elements. This joint is broken in a similar fashion as the ball joint case. The node replication is repeated for both nodes rather than the single node of a ball joint.

5. Rigid-body proxies

The simulation uses a Lagrangian velocity-constraint-based rigid-body engine [Cat05]. Collision detection is handled through Gilbert-Johnson-Keerthi Expanding-Polytope-Algorithm (GJK-EPA), [GJK88, Ber00]. The work of this section expands upon previous work by Fishman and Davis, [Fis12, Dav13] (see Sect. 3). This is the first simulation to use rigid-body proxies to interact with fractureable FEM deformables.

5.1. Proxy data. In order to simulate the interaction between FEM-based deformables and rigid bodies, the proxy needs certain data including velocity, mass, inertia tensor, and angular velocity. There are several ways which this information could be calculated and presented to the rigid-body proxy. The most notable issues are surrounding the mass and inertia tensor.

Given that an FEM deformable is broken into nodes, volume elements (tetrahedrons), and islands, there are several things to consider. The proxies and colliders are attached to the individual tetrahedrons (with GJK-EPA being run on the tetrahedron). One might consider using an entire island with a proxy. Two major issues arise from this alternate solution when dealing with fracture: the FEM-based deformable can quickly become non-convex so a non-convex collision detection scheme would be required and updating of proxies every frame with changes in the deformed shape. Secondly, due to the breaking and reforming of islands, the proxies would require regular destruction and recreation.

Determining how the mass and inertia tensor are calculated for each proxy is another major concern. From the perspective of the FEM-solver, the tetrahedrons are of uniform density, and the mass in the model is distributed at the nodes (see Sect. 2). That is to say that the mass of a tetrahedron is not equal to the sum of the masses of the nodes comprising that tetrahedron as the nodes are shared across multiple tetrahedrons.

Three possibilities for calculating mass of the proxy are the mass of the tetrahedron, the sum of the masses of the nodes on the tetrahedron, and the mass of the whole island. For this simulation, all cases were tested and the best stability was found when using the mass of the whole island for each tetrahedron. While this is physically unintuitive, stability and visual appeal are more important in a real-time environment than physical accuracy. In the cases of using the sum of the node masses or the mass of the tetrahedron, the interaction is only strong enough to effect the motion of the element instead of the motion of the center of mass of the whole island. This leads to strong local effects which require very small time steps to stabilize and propagate believably to the rest of the deformable body. It should also be noted that whenever an island is fractured, the mass of the proxies must be updated to reflect the new islands they reside on.

The momentum of the proxies are calculated using,

$$M_p\vec{v}_p = \sum_{i=0}^{3} m_i\vec{v}_i, \tag{5.8}$$

where M_p and v_p are the proxy mass and velocity, respectively, and m_i and v_i are the ith node mass and velocity, respectively.

Similar considerations must be made when calculating the inertia tensor of the tetrahedron. The simulation uses a simple point-mass model for calculating the inertia tensor, [Gol50, Fre65]. For the sake of stability and consistency with the mass calculations, the inertia tensor is calculated by evenly distributing the mass of the proxy to the nodes. In other words, each node uses one-quarter of the proxy mass for inertia tensor calculations. While this overestimates the inertia tensor, it was not found to create stability issues. An alternative approach could approximate the tetrahedral inertia tensor [BB04]. The angular momentum of the proxy is calculated using,

$$\mathbf{I}_p \vec{\omega}_p = \sum_{i=0}^{3} (\vec{p}_i - \vec{C}_e) \times m_i \vec{v}_i, \tag{5.9}$$

where $\mathbf{I}_p$ and $\vec{\omega}_p$ are the inertia tensor and angular velocity of the proxy, $\vec{C}_e$ is the center of the tetrahedral element, and $\vec{p}_i$ is the position of the ith node.

5.2. Proxy-proxy collisions. When considering proxy-proxy collisions its important to allow self collisions as a highly fractured deformable can easily collide with itself. Not all elements within an island should collide with each other, e.g. two neighbor elements that are interacting via the FEM-solver shouldn't collide as their interaction is defined by the FEM-solver.

The rigid-body engine uses a jump table of function pointers to determine what collision detection function to use based on the collider's shape. As an example, the rigid-body engine uses a separate sphere-sphere collision function instead of running GJK-EPA on sphere-sphere interactions. In this jump table, the proxy-proxy function performs a check on the two proxies to determine if they are collidable or not. The determination is made based on shared nodes. If the two elements share any nodes, then they cannot collide.

Alternative solutions could increase the number to one allowable shared node or perform a check to see how the two elements are connected (e.g. hinge or ball joint). This simulation uses aesthetic joint pruning which would eliminate cases where elements might collide if they are connected by 1 or 2 nodes (see Sect. 4.1).

5.3. Single tetrahedron islands. After fracture events, it is possible for a single tetrahedron to be separated into its own island. This can be an issue for stability and aesthetic reasons. A single tetrahedron is not a valid FEM-based object as the FEM technique works on a continuous volume broken into discrete pieces. Depending on the Young's modulus of a material, a single tetrahedron can appear unstable while resting on the ground. If the Young's modulus is too low, it will compress in unappealing ways. If it is too high, it will appear to hop on the ground. Even the most ideal values have some minor appears of jittering.

One attempted solution to this problem is to choose not to run certain parts of the FEM solving algorithm on a single tetrahedron (e.g. conjugate gradient method). The idea being that this single tetrahedron would behave as if it were a simple rigid body. This resulted in worse stability. Other proposed solutions include removing a single tetrahedron when detected or changing the rigid-body proxy into a true rigid-body if simply removing objects is not desired.

6. Plasticity-based fracture

This simulation has the option of running an entirely new fracture technique, referred to as plasticity-based fracture. The fracture determination is made based on excess plastic force.

Previously, any plastic force over the maximum allowable is ignored. In this technique, that excess force is stored on each node. When checking for fracture, the magnitude of the excess, p_{excess}, is calculated using,

$$p_{excess} = \sum_i |\vec{f}_{i,p_{excess}}|, \tag{5.10}$$

where $\vec{f}_{i,p_{excess}}$ is the excess plastic forces from each element on that node.

The fracture plane is calculated using,

$$\hat{n} = \text{normalize}(\sum_i \vec{f}_{i,p_{excess}}), \tag{5.11}$$

where $\hat{n}$ is the fracture plane normal.

After performing a fracture event, all elements attached to the fractured node have their plasticity reduced by an arbitrary factor. Additionally, the fracture node has its excess plastic force cleared. These two measures help reduce chance of catastrophic fracture. If these measures are not in place, then the nodes fracture repeatedly in successive frames.

One disadvantage of this technique is its non-physical basis for computing $\hat{n}$. The more traditional fracture calculation discussed previously is more physically accurate in this regard. However, this new technique is a computationally inexpensive method for calculating fracture in an FEM-based deformable.

7. Computational expense comparison

Analysis comparing plasticity-based fracture to separation tensor-based fracture was performed with regards to computational expense.

7.1. Floating point calculation analysis. A floating point operation (FLOPS) estimation was performed between the two algorithms, separation tensor-based fracture and plasticity-based fracture. The analysis was done on a per node and per element basis.

For separation tensor-based fracture, the following assumptions were made: 1) an eigenvalue calculation takes about 90 floating point operations. 2) For this calculation, each node has 4 tensile and 4 compressive components. 3) Per element, it is estimated that the stress tensor computation takes 216 FLOPS. 4) Decomposing the stress tensor takes about 315 FLOPS.

This leads to a total of 531 FLOPS per element per fracture calculation. On a per node basis, computing the separation tensor takes about 213 FLOPS and decomposing the separation tensor takes about 90 FLOPS for a total of 303 FLOPS per node per fracture calculation.

For plasticity-based fracture, one assumption was made: similar to the second assumptions for separation tensor-based fracture, there are about 4 plastic excess forces per node. On a per element basis, it is estimated there are roughly 48 FLOPS per element per fracture step. On a per node basis, it is estimated to take 51 FLOPS per node per fracture step.

7.2. Profiling comparison. A comparison between the two techniques was made using a custom system profiler. The profiling was run on objects with varying number of elements and nodes. The results are compared by amount of time spent in fracture calculation versus number of elements plus the number of nodes (see Fig. 4).

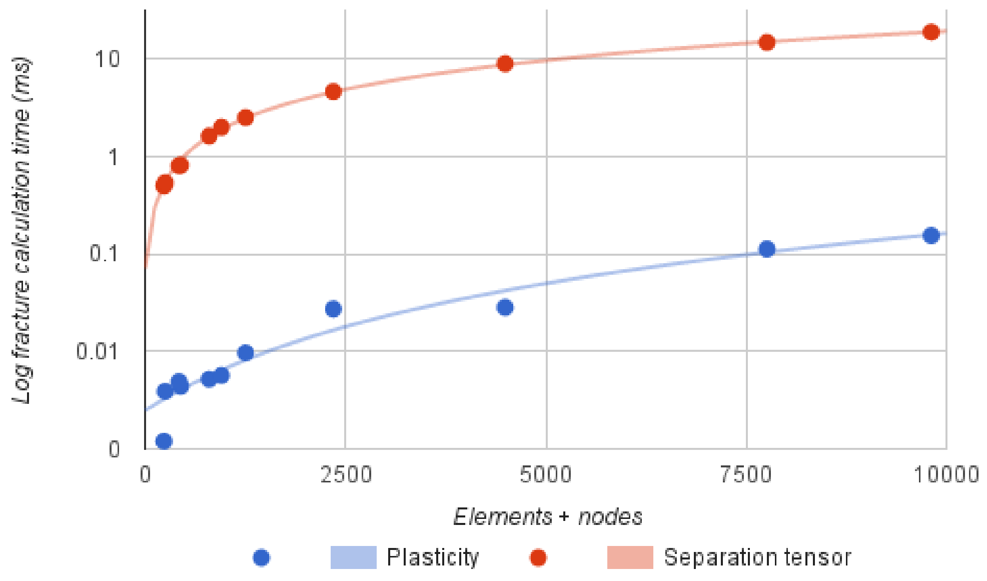

Figure 4. Comparison of fracture calculation time by technique..

Lines were added to guide the eye.

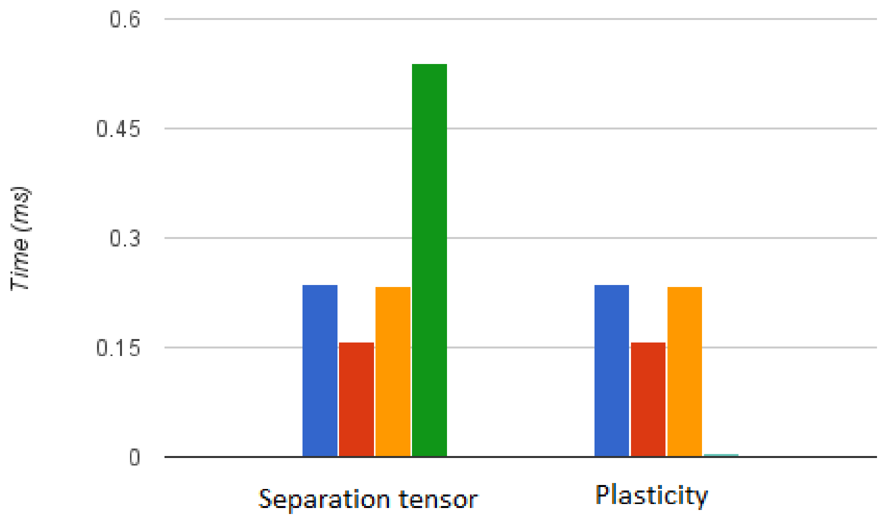

Figure 5. Sub-step comparison for separation tensor.

The data columns are stiffness assembly (blue), dynamics assembly (red), conjugate gradient solver (orange), and separation tensor-based fracture calculation (green). The plasticity-based fracture calculation column (teal) is indistinguishable from the x-axis. The data was taken on an island with 250 nodes + elements.

A log fit to the data was performed to guide the eye. In the chart, the blue, bottom line is for plasticity-based fracture and the red, top line is for separation tensor-based fracture. There are several parts of the FEM-based deformable that contribute to the overall computation expense. A comparison of how the fracture techniques stand up against the other contributions can be seen in figure 5. On the chart, the y-axis is a measure of time in milliseconds, and along the x-axis is the sub-step contributions to the total computation time. With plasticity-based fracture, the computational bottle-neck is no longer the fracture calculations, but rather the other parts of the deformable calculation.

CHAPTER 6

Conclusions

Fracture simulation is a broad field. It has increasing applications in real-time environments [PEO09, MCK13], and offline animations [BHTF07, SSF09]. Growing computational power is increasing the number of viable options for fracture simulation in real-time.

In the area of real-time fracture simulation, speed and aesthetic appeal are of utmost concern. Finite-element-method-based deformation and fracture is computationally expensive. This thesis presents a new approach to fracture, "Plasticity-based fracture," that is less computationally expensive than other techniques. Plasticity-based fracture is efficient enough that the computational bottle neck for fracture simulation is no longer the fracture calculations. This thesis also presents an algorithm to allow low-resolution fracture to look more appealing through joint pruning ("Aesthetic Joint Pruning"), and it presents a technique for interacting with rigid-bodies by creating rigid-body proxies at run-time.

The techniques presented here should help to push the field of real-time, dynamic fracture forward by presenting faster algorithms and allowing for the use of lower resolution models especially if combined with mesh coupling [PEO09]. The rigid-body proxy system provides a way to add physically-based ductile fracture to an existing rigid-body engine.

Future work in this area includes finding a better technique for calculating the fracture plane in plasticity-based fracture, better friction handling for rigid-body proxies, multi-threading and GPU implementations, and dealing with collision detection, including an effective broad phase for highly tesselated, deformable objects.

APPENDIX A

LINEAR COMPLEMENTARITY PROBLEM

The linear complementarity problem (LCP) was originally presented in [CD68]. The general formulation, as explained in [Erl13], is given by the equation:

$$\vec{y} = \mathbf{A}\vec{x} + \vec{b}, \tag{A.1}$$

where the vectors are in $\mathbb{R}^n$ and $\mathbf{A}$ is in $\mathbb{R}^{n\times n}$. The solution is further restricted by:

- $\vec{x} \geq 0$,
- $(\mathbf{A}\vec{x} + \vec{b}) \geq 0$,
- $\vec{x}^T(\mathbf{A}\vec{x} + \vec{b}) = 0$.

These restrictions must be satisfied in order to have a solution for values of $\vec{y}$ and $\vec{x}$. Note that the restrictions are defined without the use of $\vec{y}$, and that (the first two items in the aforementioned list) the conditions hold on an element-wise case, i.e. if $\vec{x} \geq 0$ then $x_i \geq 0$ for all i. The LCP shows up in physics simulation when solving a system of bodies interacting by Newton's second law, $\vec{F} = m\vec{a}$

For a detailed example, we turn to the one dimensional case. The conditions on the equation $y = ax + b$, become:

- $x \geq 0$,
- $ax + b \geq 0$,

	$b < 0$	$b = 0$	$b > 0$
$a < 0$	0	1	2
$a = 0$	0	∞	1
$a > 0$	0	1	1

Table 1: Solution space for 1D LCP for x, y as discussed in [Erl13]. Shows the number of possible solutions for x, y in the 1D case for each combination of a and b.

- $x(ax + b) = 0$,

where a, x, b, and y are all members of $\mathbb{R}$. The third condition can be rewritten as $xy = 0$. Due to this condition, the solutions are restricted such that either $x = 0$ or $y = 0$. We can consider our solution space by the table 1. This table contains the number of solution sets which are possible for the various combinations of x and y. As an example, if we consider the case where $b = -1$ and $a = 1$, then the only possible solution (as the table says there is only one possible solution) is that $x = 1$ and $y = 0$. Considering the case where $b = 1$ and $a = -1$, then there must be two solutions as outlined in the table: $(0, 1)$ and $(1, 0)$ (in the notation (x, y)).

If the matrix $\mathbf{A}$ is invertible, then it is possible to solve LCP through matrix inversion. However, it is often not the most efficient method for solving the problem. Notably, if the matrix is singular (i.e. non-invertible) or if it is a large, sparse matrix, then numerical solvers are the best option. There are a number of numerical solvers that exist for LCPs including Projected Gauss Seidel (see [Erl13]) and Conjugate Gradient Method (see appendix B).

APPENDIX B

CONJUGATE GRADIENT METHOD

A very thorough discussion of the Conjugate Gradient Method (CGM) is presented in [She94]. Here we present only a brief outline. CGM was developed in order to solve large systems of linear equations of the form:

$$\mathbf{A}\vec{x} = \vec{b}, \tag{B.1}$$

where $\mathbf{A}$ is a known, symmetric, positive-definite matrix in $\mathbb{R}^{n\times n}$, $\vec{b}$ is a known vector in $\mathbb{R}^n$, and $\vec{x}$ is a vector in $\mathbb{R}^n$ which we are solving for.

CGM works iteratively and is efficient for solving large, sparse matrices, such as the stiffness matrix discussed in section 2.4. This linear system of equations is a special case of the LCP, presented in appendix A), with $\vec{y} = 0$ from equation A.1.

The solution to equation B.1 will minimize the quadratic form,

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T\mathbf{A}\vec{x} - \vec{b}^T\vec{x} + c \tag{B.2}$$

where $\vec{x}$, $\mathbf{A}$, and $\vec{b}$ are the same as in equation B.1, and c is a scalar constant.

The quadratic form can be minimized by setting the gradient of $f(\vec{x})$ to 0 and solving for $\vec{x}$. The gradient of $f(\vec{x})$ is given by

$$\nabla f(\vec{x}) = \frac{1}{2}\mathbf{A}^T\vec{x} + \frac{1}{2}\mathbf{A}\vec{x} - \vec{b} = \mathbf{A}\vec{x} - \vec{b} \tag{B.3}$$

where the simplification in the final form is the result of $\mathbf{A}$ being a symmetric matrix.

In order to solve this system of equations, iteratively choose orthogonal search directions $d_0, d_1, ..., d_{n-1}$ in the phase space of $\mathbf{A}$. The initial direction is given by, $\vec{d_0} = \vec{r_0} = \vec{b} - \mathbf{A}\vec{x_0}$, where $\vec{x_0}$ is the initial guess of $\vec{x}$, and $i+1$ direction is given by,

$$\vec{d}_{i+1} = \vec{r}_{i+1} + \beta_{i+1}\vec{d}_i, \tag{B.4}$$

where β is given by

$$\beta_{i+1} = \frac{\vec{r}_{i+1}^T \vec{r}_{i+1}}{\vec{r}_i^T \vec{r}_i}. \tag{B.5}$$

The residual, r_i, is updated via

$$\vec{r}_{i+1} = \vec{r}_i - \alpha_i \mathbf{A}\vec{d}_i, \tag{B.6}$$

where α is given by

$$\alpha_i = \frac{\vec{r}_i^T \vec{r}_i}{\vec{d}_i^T \mathbf{A}\vec{d}_i}. \tag{B.7}$$

The indices i range from 0 to n.

The residual is related to the error term by transforming the error, $\vec{e}$ by $\mathbf{A}$, i.e. $\vec{r_i} = \mathbf{A}\vec{e}$. The search directions are conjugate with respect to A, i.e. $\vec{d}_i^T \mathbf{A}\vec{d}_j = 0$ for $i \neq j$. Algorithmically, CGM has four primary steps:

- Update $\vec{x}_{i+1}$ based on search direction, $\vec{d}_i$
- Update residual, $\vec{r}_{i+1}$, based on search direction, $\vec{d}_i$
- If the residual is small enough, exit
- Update search direction, $\vec{d}_{i+1}$, based on residual, $\vec{r}_{i+1}$

The search direction is updated via the residual because of the aforementioned relation between residual and error. Updating the search direction in this manner also provides a new search direction which is orthogonal, or conjugate, in A [She94].

APPENDIX C

GRAM-SCHMIDT ORTHONORMALIZATION AND POLAR DECOMPOSITION

Gram-Schmidt orthonormalization and polar decomposition are two techniques used to construct rotation matrices for finite element method based deformables (see section 2).

1. Gram-Schmidt orthonormalization

Gram-Schmidt orthonormalization is a technique for transforming a set of linearly independent functions into an orthogonal basis [AW05]. Consider a set of k linearly independent vectors, $\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_k}$. We can define the projection of vector $\mathbf{a}$ onto vector $\mathbf{e}$ using the inner product (for vectors in $\mathbb{R}^N$, simply the dot product):

$$\mathrm{proj}_{\mathbf{e}}\mathbf{a} = \frac{\langle \mathbf{e}, \mathbf{a} \rangle}{\langle \mathbf{e}, \mathbf{e} \rangle}. \tag{C.1}$$

We can calculate a set of orthogonal vectors, $\mathbf{u_1}, \mathbf{u_2}, ..., \mathbf{u_k}$, using:

$$\mathbf{u}_1 = \mathbf{a}_1, \tag{C.2}$$

$$\mathbf{u}_2 = \mathbf{a}_2 - \mathrm{proj}_{\mathbf{e}_1}\mathbf{a}_2, \tag{C.3}$$

$$\mathbf{u}_k = \mathbf{a}_k - \sum_{i=j}^{k} \mathrm{proj}_{\mathbf{e}_j}\mathbf{a}_k, \tag{C.4}$$

where $\mathbf{e}_i = \frac{\mathbf{u}_i}{|\mathbf{u}_i|}$. We can now construct an orthonormal matrix from the vectors $\mathbf{e}_i$.

2. Polar decomposition

In polar decomposition, a square, invertible matrix, $\mathbf{A}$, can be uniquely decomposed into

$$\mathbf{A} = \mathbf{U}\mathbf{P} \tag{C.5}$$

where $\mathbf{P}$ is the orthonormal basis and $\mathbf{U}$ is the transformation matrix between the basis and $\mathbf{A}$. The matrix $\mathbf{P}$ is calculated by

$$\mathbf{P} = (\mathbf{A}^*\mathbf{A})^{(\frac{1}{2})}, \tag{C.6}$$

where $\mathbf{A}^*$ is the conjugate transpose of the matrix, and $\mathbf{U}$ is:

$$\mathbf{U} = \mathbf{A}\mathbf{P}^{-1}. \tag{C.7}$$

The matrix $\mathbf{P}$ is the orthonormal matrix used for rotations.

REFERENCES

[AS88] Sepehr Arbabi and Muhammad Sahimi. Elastic properties of three-dimensional percolation networks with stretching and bond-bending forces. *Phys. Rev. B*, 38:7173–7176, Oct 1988.

[AW05] G.B. Arfken and H.J. Weber. *Mathematical Methods for Physicists*. Mathematical Methods for Physicists. Elsevier, 2005.

[Ban94] Prasanta Kumar Banerjee. *The Boundary Element Methods in Engineering*. McGraw-Hill, 2nd edition, 1994.

[BB04] Jonathan Blow and Atman J. Binstock. How to find the inertia tensor (or other mass properties) of a 3d solid body represented by a triangle mesh. 2004.

[BDW13] Oleksiy Busaryev, Tamal K. Dey, and Huamin Wang. Adaptive fracture simulation of multi-layered thin plates. *ACM Trans. Graph.*, 32(4):52, 2013.

[Ber00] Gino Van Den Bergen. Abstract proximity queries and penetration depth computation on 3d game objects, 2000.

[BG00] Daniel Bielser and Markus H. Gross. Interactive simulation of surgical cuts, 2000.

[BGTG03] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus Gross. A state machine for real-time cutting of tetrahedral meshes. In *Pacific Graph*, pages 377–386. IEEE Computer Society, 2003.

[BHTF07] Zhaosheng Bao, Jeong-Mo Hong, Joseph Teran, and Ronald Fedkiw. Fracturing rigid materials. *IEEE Trans. Vis. Comput. Graph.*, 13(2):370–378, 2007.

[Cat05] Erin Catto. Iterative dynamics with temporal coherence. 2005.

[Cat07] Erin Catto. Box2d, 2007.

[CD68] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

[CRHG96] J. W. Chung, A. Roos, J. Th. M. De Hosson, and E. van der Giessen. Fracture of disordered three-dimensional spring networks: A computer simulation methodology. *Phys. Rev. B*, 54:15094–15100, Dec 1996.

[CYFW14] Zhili Chen, Miaojun Yao, Renguo Feng, and Huamin Wang. Physics-inspired adaptive fracture refinement. *ACM Trans. Graph.*, 33(4):113, 2014.

[Dav13] Joshua Davis. Survey of multi-physics: A comparison between coupling and embedding. Master's thesis, DigiPen Institute of Technology, 2013.

[DM95] Frdric Donz and Sophie-Adlaide Magnier. Formulation of a 3-d numerical model of brittle behaviour. *Geophysical Journal International*, 122(3):790–802, 1995.

[dt03] NETGEN development team. Netgen. `http://www.hpfem.jku.at/netgen/`, 2003.

[Erl13] Kenny Erleben. Numerical methods for linear complementarity problems in physics-based animation. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, pages 8:1–8:42, New York, NY, USA, 2013. ACM.

[ESHD05] Kenny Erleben, Jon Sporring, Knud Henriksen, and Kenrik Dohlman. *Physics-based Animation (Graphics Series)*. Charles River Media, Inc., 2005.

[Fis12] Theodore Fishman. Real-time deformation with the finite element method and embedding using rigidbody proxies. Master's thesis, DigiPen Institute of Technology, 2012.

[Fre65] A.P. French. *Newtonian Mechanics*. M.I.T. introductory physics series. Nelson, 1965.

[GJ$^+$10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2):193–203, Apr 1988.

[GM77] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, November 1977.

[GMD13] Loeiz Glondu, Maud Marchal, and Georges Dumont. Real-time simulation of brittle fracture using modal analysis. *IEEE Trans. Vis. Comput. Graph.*, 19(2):201–209, 2013.

[Gol50] Herbert Goldstein. *Classical Mechanics.* Addison-Wesley, 3rd edition, 1950.

[HW04] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.* Springer, 2nd edition, 2004.

[IO09] Hayley N. Iben and James F. O'Brien. Generating surface crack patterns. *Graphical Models*, 71(6):198–208, 2009.

[ITF06] G. Irving, J. Teran, and R. Fedkiw. Tetrahedral and hexahedral invertible finite elements. *GRAPH. MODELS*, 68:66–89, 2006.

[Kel06] Micky Kelager. Lagrangian fluid dynamics using smoothed particle hydrodynamics, 2006.

[LL07] Shaofan Li and Wing Kam Liu. *Meshfree Particle Methods*. Springer, 1st edition, 2007.

[LRK10] W Michael Lai, David Rubin, and Erhard Krempl. *Introduction to Continuum Mechanics*. Elsevier, 4th edition, 2010.

[MAP12] Derek John Morris, Eike Falk Anderson, and Christopher Peters. A modular framework for deformation and fracture using gpu shaders. In *VSMM*, pages 267–274. IEEE, 2012.

[MCK13] Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Real time dynamic fracture with volumetric approximate convex decompositions. *ACM Trans. Graph.*, 32(4):115, 2013.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, GI '04, pages 239–246, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[MM05] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, 2nd edition, 2005.

[MMDJ01] Matthias Müller, Leonard McMillan, Julie Dorsey, and Robert Jagnow.

Real-time simulation of deformation and fracture of stiff materials. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, page 113–124, New York, NY, USA, 2001. Springer-Verlag New York, Inc., Springer-Verlag New York, Inc.

[MO03] Maciej Matyka and Mark Ollila. Pressure model of soft body simulation. In *Proc. of Sigrad, UMEA, 2003*, 2003.

[NPF05] Matthieu Nesme, Yohan Payan, and François Faure. Efficient, physically plausible finite elements. In *Eurographics*, August 2005.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. In Tom Appolloni, editor, *SIGGRAPH*, pages 291–294. ACM, 2002.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH*, pages 137–146, 1999.

[PEO09] Eric G. Parker, Pixelux Entertainment, and James F. Obrien. Real-time deformation and fracture in a game environment. In *Proceedings of the 2009 Symposium on Computer Animation*, pages 156–166, 2009.

[PKA$^+$05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus H. Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. *ACM Trans. Graph.*, 24(3):957–964, 2005.

[Pro96] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–154, 1996.

[Red05] J. N. Reddy. *An Introduction to the Finite Element Method*. McGraw-Hill, 3rd edition, 2005.

[Say08] Francisco-Javier Sayas. A gentle introduction to the finite element method, 2008.

[She94] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[SO14] Sara C. Schvartzman and Miguel A. Otaduy. Fracture animation based on high-dimensional voronoi diagrams. In *Symposium on Interactive 3D Graphics and Games, I3D '14, San Francisco, CA, USA - March 14 - 16, 2014*, pages 15–22, 2014.

[SSF09] Jonathan Su, Craig A. Schroeder, and Ronald Fedkiw. Energy stability and fracture for frame rate rigid body simulations. In Dieter W. Fellner and Stephen N. Spencer, editors, *Symposium on Computer Animation*, pages 155–164. ACM, 2009.

[SWB00] Jeffrey Smith, Andrew Witkin, and David Baraff. Fast and controllable simulation of the shattering of brittle objects. In *Graphics Interface*, pages 27–34. Blackwell Publishing, 2000.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation:

Viscolelasticity, plasticity, fracture. *SIGGRAPH Comput. Graph.*, 22(4):269–278, June 1988.

[THMG04] Matthias Teschner, Bruno Heidelberger, Matthias Müller, and Markus H. Gross. A versatile and robust model for geometrically complex deformable solids. In *Computer Graphics International*, pages 312–319. IEEE Computer Society, 2004.

[TM08] P.A. Tipler and G. Mosca. *Physics for Scientists and Engineers: With Modern Physics*. W. H. Freeman, 2008.