

Bachelor-Thesis

zur Erlangung des akademischen Grades *Bachelor of Science (B.Sc.)* im Studiengang Informatik – Digitale Medien und Spiele mit Fachrichtung Spiele

**Implementierung der Ductile-Fracture-Bruchsimulation in der Unity Engine**

Prüfer: Prof. Dr. Christoph Lürig

Vorgelegt von: Paul Froelich

Nagelstraße 27

54290 Trier

Matrikelnummer: 965335

Bearbeitungszeitraum: März bis August 2020

Eingereicht am: ergänzen

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel

**„Implementierung der Ductile-Fracture-Bruchsimulation in der Unity Engine“**

selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Hilfsmittel verwendet und wörtliche sowie sinngemäße Zitate als solche gekennzeichnet habe.

Trier, den x.x.2020

Paul Froelich

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Vorwort & Danksagung

// TODO

Kurzfassung

// Überarbeiten

Die Simulation von Deformierungen und Frakturen von Körpern finden weitflächig Anwendung, beispielsweise in Analyseprogrammen zur Prognose von Bruchstellen bei Bestandsteilen von Motoren oder in 3D-Animationsfilmen. Doch durch den zunehmenden Realismus im Bereich der Videospiele steigt auch das Interesse an realistischen Zerstörungsanimationen und der Anwendung von realistischen Zerstörungen als zentrale Spielmechanik.

Das Ziel dieser Bachelor-Abschlussarbeit ist es, einen einfachen Algorithmus zur Berechnung von spröder und duktiler Frakturenpropagierung aus dem Jahre 1998 in der Spieleengine *Unity* zu implementieren und zu beurteilen, wie realistisch die Verwendung eines solchen Algorithmus in einem Videospiel ist. Hierbei wird besonders die Performanz des Algorithmus betrachtet, aber es werden auch Urteile zur Willkür der Frakturen gefällt und inwiefern dies zu einer wünschenswerten und sinnvollen Gameplay-Mechanik beiträgt. Die Arbeit richtet sich demnach vor allem an Entwickler von Softwaresystemen mit weichen Echtzeitanforderungen (zum Beispiel Videospiele).

1. Inhaltsverzeichnis

[Vorwort & Danksagung iii](#_Toc48957593)

[1 Einleitung 1](#_Toc48957594)

[1.1 Motivation 1](#_Toc48957595)

[1.2 Historischer Überblick 2](#_Toc48957596)

[1.2.1 Deformierungsmodelle 2](#_Toc48957597)

[1.3 Motivation der Entwicklung des GMABF-Algorithmus 3](#_Toc48957598)

[1.4 Aufbau des Verfahrens 3](#_Toc48957599)

[2 Der theoretische Hintergrund 3](#_Toc48957600)

[2.1 Das kontinuierliche Modell 4](#_Toc48957601)

[2.1.1 Die Begriffe “Kontinuum”, “spröde”, “duktil”, “plastisch“ und „elastisch“ 4](#_Toc48957602)

[2.1.2 Das Maß der Belastung 7](#_Toc48957603)

[2.1.3 Duktile Fraktur 9](#_Toc48957604)

[2.2 Das diskrete Modell 10](#_Toc48957605)

[3 Die Implementierung in Unity 10](#_Toc48957606)

[3.1 Planung 11](#_Toc48957607)

[3.1.1 Architektur 11](#_Toc48957608)

[3.1.2 Testsystem 12](#_Toc48957609)

[3.2 Ablauf der Implementierung 12](#_Toc48957610)

[3.3 Mögliche Erweiterungen 12](#_Toc48957611)

[3.4 Einpflegung in ein existierendes Spiel 12](#_Toc48957612)

[4 Diskussion der Ergebnisse 13](#_Toc48957613)

[5 Ausblick 13](#_Toc48957614)

[6 Abbildungsverzeichnis 14](#_Toc48957615)

[7 Tabellenverzeichnis 15](#_Toc48957616)

[8 Literaturverzeichnis 16](#_Toc48957617)

[9 Anhang 17](#_Toc48957618)

# Einleitung

// Die Einleitung:

// - definiert das Thema

// - motiviert die Thematik

// - Stellt die Bedeutung der Arbeit heraus

// - liefert relevante Hintergrundinformationen

// - umfasst Literaturrecherche

// - bindet das Thema in den breiteren Kontext ein

// - grenzt es ebenso sinnvoll von anderen Themen ab

// - Formuliert am Ende eine Hypothese

In dieser Bachelor-Abschlussarbeit geht es um die Implementierung eines Algorithmus zur Berechnung von Frakturen in Körpern aus sprödem oder duktilem Material in der Spieleengine *Unity*. Der erste Teil des Algorithmus wurde im Jahre 1999 von den Autoren James F. O’Brien und Jessica K. Hodgins veröffentlicht und behandelte die Frakturpropagation spröder Materialien. Der zweite Teil wurde im Jahre 2001 von denselben Autoren und Adam W. Bargteil präsentiert und ergänzte den Algorithmus um duktile Materialien.

Ziel der Implementierung ist die Analyse des Algorithmus für seine Tauglichkeit in weichen Echtzeitsystemen. Hierbei wird besonders auf die Performanz der Implementierung des Algorithmus geachtet und seine Komplexität festgestellt. Im Zuge dessen werden Verbesserungen des Algorithmus für den Einsatz in Videospielen vorgeschlagen. Ferner wird der Algorithmus in einen historischen Kontext gesetzt und es werden andere und neuere Algorithmen betrachtet.

## Motivation

Die Deformierung und Zerstörung von Objekten in Videospielen als eine zentrale Gameplay-Mechanik ist eine junge und kaum erschlossene Disziplin (is it though?). Doch die algorithmischen Möglichkeiten zur Berechnung von Frakturen existieren schon seit 1988 und die wissenschaftlichen Fachgebiete, welche die mathematische Grundlage dieser Algorithmen liefern, reichen bis ins Jahr 1958 zurück.

Diese Algorithmen waren vor allem für die Animation von Körpern in 3D-animierten Filmen zuständig. Sie waren also nicht dafür bestimmt, in einem Echtzeitsystem ausgeführt zu werden. Dies wird besonders klar bei der Betrachtung der Performanzangaben des Algorithmus: die Simulation von einer Sekunde einer zerbrechenden Glasscheibe dauerte auf einem SGI O2 (eine Low-End Unix-Workstation von 1996) im Durchschnitt 273 Minuten.

// folgender Paragraph eventuell fehl am Platz

Außerdem sind die Algorithmen definitiv von Softwaresystemen mit hohen Sicherheitsanforderungen (beispielsweise Frakturprognosen von Materialien in Motoren) abzugrenzen, denn die Ergebnisse der Simulationen sind nicht auf physische Genauigkeit ausgelegt, sondern darauf, zufriedenstellende Ergebnisse für Unterhaltungsfilme zu liefern.

Der theoretische Hintergrund

Weiterhin wird die Implementierung des Algorithmus in der Unity-Spieleengine beschrieben, die im Rahmen dieser Abschlussarbeit durchgeführt wurde. Hierbei betrachten wir die Planung, die Durchführung und die möglichen Erweiterungen und Verbesserungen der Implementierung. Abschließend werden die Ergebnisse des Experiments diskutiert und ein Ausblick auf die Zukunft der Simulation von Deformierung und Fraktur von Körpern, besonders im Rahmen der Unterhaltungsindustrie, geliefert.

## Historischer Überblick

Physikbasierte Modelle für computergenerierte Grafiken und Animationen gewannen Mitte der 1980er Jahre an Popularität aufgrund ihrer Vorteile gegenüber kinematischer Animation.

Konventionelle Animation ist kinematisch; Sie wird beschrieben durch die Bewegung ihrer geometrischen Einzelteile zu jedem Zeitschritt, möglicherweise ergänzt durch Interpolationen, um einen weicheren Übergang zu ermöglichen. Physikbasierte, dynamische Animation funktioniert durch Kraftanwendungen auf Körper, sodass die resultierende Position des Körpers aus numerischen Verfahren gewonnen werden kann.

Die Vorteile der dynamischen Animation gegenüber der kinematischen Animation bestehen darin, dass sie realitätsnäher ist und es dadurch für den Entwickler der Animationen einfacher ist, realistische und damit zufriedenstellende Animationen umzusetzen. Aufgrund dieser physikbasierten Modelle wurden daraufhin Techniken entwickelt, welche auch die Animation von nicht-starren Körpern ermöglichten.

### Deformierungsmodelle

1988 publizierten Demetri Terzepoulos und Kurt Fleischer die wissenschaftlichen Arbeiten *Deformable Models* und *Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture*, welche die Funktionsweise eines Modells zur Deformierung von elastischen, viskoelastischen und plastischen Körpern aufzeigen. Das Konzept des Modells folgt, genauso wie der in dieser Arbeit implementierte Algorithmus, der Unterteilung des Systems in ein kontinuierliches und ein diskretes Modell. Das kontinuierliche Modell basiert auf Energiefunktionen – Funktionen, die den Status eines Körpers als Eingabe bekommen und die resultierende Energie der Deformierung als Ausgabe formulieren. Die Diskretisierung des kontinuierlichen Modells geschieht über die Finite-Differenzen-Methode (kurz FDM). Dieser Algorithmus lieferte vielversprechende Ergebnisse und eine Basis für weitere Ansätze.

Alan Norton und seine Kolleginnen und Kollegen Greg Turk, Bob Bacon, John Gerth und Paula Sweeney lieferten 1991 eine Technik zur Simulation solider Objekte, welche unter größeren Belastungen zerbrechen. Sie realisierten dies durch ein sog. Spring-Mass-System (dt. Feder-Masse-System). Hierbei wurden stets die Distanzen zwischen zwei Massepunkten des zu simulierenden Objektes gemessen, wobei die Verbindung zwischen eben jenen Massepunkten gekappt wurde, sobald die Distanz einen Grenzwert überschritt. Um ungewollt zusammenhängende Einzelbruchteile zu vermeiden, wurden benachbarte Massepunktverbindungen ebenfalls gelöst.

## Motivation der Entwicklung des GMABF-Algorithmus

Die bereits beschriebenen Deformierungsmodelle lieferten eine sinnvolle Basis für die Weiterentwicklung von Modellen, welche verschiedenste Eigenschaften von Materialien berücksichtigen konnten. Allerdings waren die genannten Modelle dadurch limitiert, dass sie keinerlei Informationen liefern konnten, wo die sog. Frakturfläche einer Fraktur lag oder in welche Richtung ihre Normale zeigte. Des Weiteren stellten die Modelle keine Remeshing-Vorschriften der gebrochenen Körper zur Verfügung. Die zerbrechenden Körper konnten also nur entlang der Grenzen ihrer Bestandsteile zerbrechen, welche selbst unzerbrechlich waren.

Der GMABF-Algorithmus löst dies durch ein Finite-Elemente-Modell, Differenzgleichungen zur Beschreibung des zusammenhängenden Verhaltens und einer Remeshing-Vorschrift, die bei einem Bruch das zugrundeliegende Gitter kleiner aufteilt.

## Aufbau des Verfahrens

Der GMABF-Algorithmus basiert auf der Finite-Elemente-Methode (kurz FEM) und der Kontinuumsmechanik. Die **FEM** ist eine bekannte und bewährte Methode zur Diskretisierung von Volumen, die darauf basiert, das zu diskretisierende Volumen in geometrische Elemente, in diesem Fall Tetraeder, zu unterteilen. Die **Kontinuumsmechanik** ist ein Fachgebiet aus der Mechanik und beschäftigt sich mit der Bewegung von deformierbaren Körpern aufgrund von äußeren Belastungen. Sie betrachtet das Material eines Volumens nicht als eine Menge von Partikeln / Teilchen / Molekülen / Atomen, sondern als eine kontinuierliche Menge, die dann mit der FEM diskretisiert wird. In den folgenden Kapiteln werden die beiden Bestandteile gründlicher beleuchtet.

# Der theoretische Hintergrund

Das der Arbeit zugrundeliegende GMABF-Modell besteht aus 3 Teilen: dem kontinuierlichen Modell, welches auf der Kontinuumsmechanik basiert, dem diskreten Modell, welches die Finite-Elemente-Methode verwendet und zuletzt der Remeshing-Vorschrift.

Zuerst wird der Algorithmus zur Berechnung von Frakturpropagation in spröden Materialien beleuchtet und erklärt. Im Rahmen dessen werden das kontinuierliche und das diskrete Modell grundlegend erklärt und die Remeshing-Vorschrift vollständig erläutert.

Darauf folgt die Erklärung der Erweiterung des Algorithmus um duktile Materialien, welches ausschließlich das kontinuierliche Modell des vorher beschriebenen Algorithmus für spröde Materialien erweitert und das diskrete Modell, sowie die Remeshing-Vorschrift nicht beeinflusst.

Zuletzt wird die Remeshing-Vorschrift erläutert.

// …

## 2.1 Das kontinuierliche Modell

Das Konzept des kontinuierlichen Modells ist es, an jedem räumlichen Punkt eines Körpers Daten über die Belastung an dieser Stelle zu liefern. Genauer bedeutet dies, dass die internen Belastungen des Körpers in Form von tensilen (umgangssprachlich äquivalent zu „spannenden“ oder „ziehenden“) und kompressiven Kräften durch das Modell zur Verfügung stehen. Man kann so an jedem Punkt innerhalb des Körpervolumens die vorliegenden kompressiven und tensilen Kräfte bestimmen.

Die folgenden Abschnitte stellen die Informationen bereit, die für die Umsetzung eines solchen Modells benötigt werden. Zuerst wird ein grundlegendes Verständnis für die Begriffe *Kontinuum*, *duktil*, *spröde*, *plastisch* und *elastisch* geschaffen, daraufhin wird das Maß für Belastung (engl. „strain“) und Belastungsänderung (engl. „strain rate“) definiert und erläutert.

Des Weiteren wird der Bezug zu den physikalischen Eigenschaften eines Materials geschaffen und in das Modell eingepflegt, sodass ein Maß für die gesamten inneren Kräfte definiert wird. Zuletzt wird dann die Erweiterung um duktile Materialien beschrieben.

### 2.1.1 Die Begriffe “Kontinuum”, “spröde”, “duktil”, “plastisch“ und „elastisch“

Ein Kontinuum ist beispielsweise die Menge der reellen Zahlen; Es gibt unendlich viele von ihnen und zwischen zwei verschiedenen reellen Zahlen existiert immer eine andere reelle Zahl. Bei Betrachtung des mathematischen Konzepts eines dreidimensionalen Raumes, repräsentiert durch ein kartesisches Koordinatensystem, so ist ein solcher Raum auch ein Kontinuum, denn zwischen zwei verschiedenen Orten in diesem Raum gibt es immer einen weiteren Ort. Die Definition eines Kontinuums kann auf das Material eines Volumens erweitert werden. Um es mathematisch genau zu definieren (Fung, 1994):

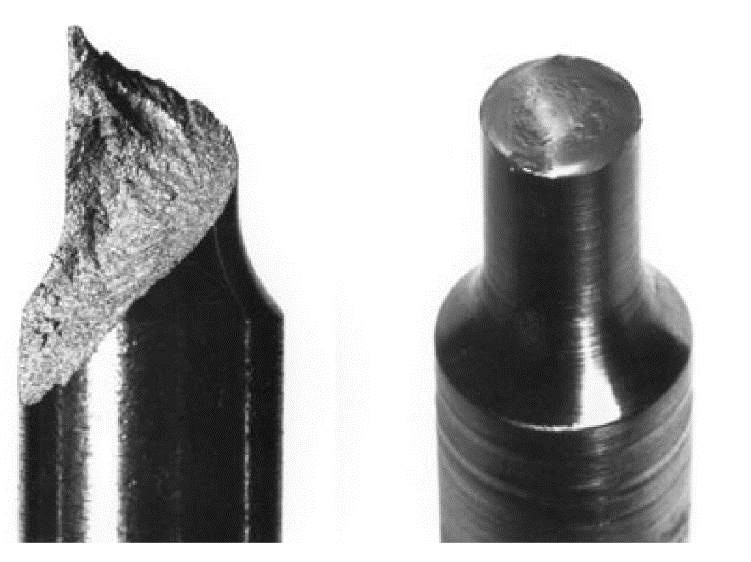
Es existiere ein Volumen V0. Wenn die Dichte, der Schwung, die Energie und die Belastung an jedem Ort im Volumen V0 definiert ist und wenn jede dieser Eigenschaften durch eine kontinuierliche Funktion mit räumlichen Koordinaten in V0 definiert werden kann, dann ist das Material im Volumen V0 kontinuierlich (Fung, 1994).

Für die folgenden Erklärungen reicht es aus, sich einen geometrischen Körper mit einem kontinuierlichen Material so vorzustellen, dass an jedem Punkt innerhalb des Körpervolumens die Belastung des Körpers messbar ist.

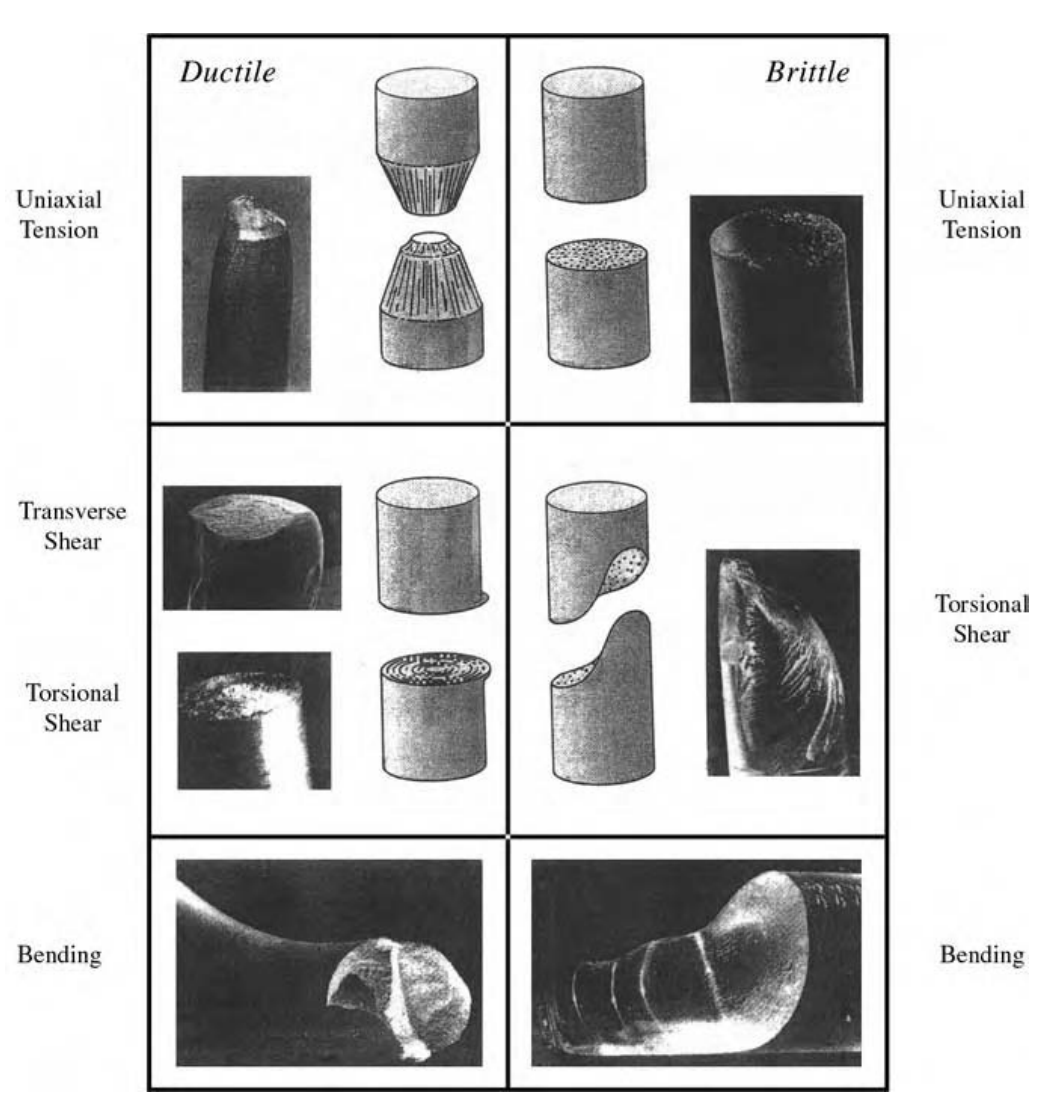
Die Notwendigkeit, die folgenden Begriffe in ihrem physischen Kontext erneut zu erläutern, liegt an ihrer Diskrepanz zur umgangssprachlichen Nutzung. Um ihre Bedeutung zu verstehen, muss zuerst verstanden werden, wie Deformierungen und Brüche auf einer grundlegenden physikalischen Ebene funktionieren.

Ein Material verformt sich unter Belastung, bis es bricht. Diese Verformung ist erst elastischer Natur und hat dann eine plastische Phase. Wenn die Belastung des Materials in der elastischen Verformungsphase aufhört, dann kehrt das Material in seine Ursprungsform zurück, als wäre die Verformung nicht aufgetreten. Bricht die Belastung in der plastischen Phase der Verformung ab, dann kommt es allerdings zu bleibenden, irreversiblen Verformungen des Materials. Der Übergang zwischen elastischer und plastischer Phase wird Elastizitätsgrenze oder Streckgrenze genannt (Macherauch, 1992).

Ein sprödes Material ist ein Material, was sehr direkt nach der Elastizitätsgrenze bricht, also keine oder nur kaum plastische Verformung aufweist. Ein duktiles Material dagegen zeigt eine deutliche plastische Verformung, bevor es bricht.



**Abbildung 3.1** (links) ein spröder Bruch, (rechts) ein duktiler Bruch (Macherauch, 1992)



**Abbildung 3.2** (links) duktile Brüche und (rechts) spröde Brüche unter verschiedenen Belastungsarten (Lewis, Reynolds, & Gagg, 2004)

### Das Maß der Belastung

Bevor Belastung gemessen werden kann, müssen wir festlegen, welche Daten wir von dem zu messenden Objekt lesen können. Wir können die Belastung eines Objektes an einem bestimmten Punkt feststellen, wenn wir die **globale** **Position**, die **Materialposition** und die **globale** **Geschwindigkeit** des Punktes kennen.

* Die globale Position ist durch einen dreidimensionalen Ortsvektor beschrieben, der im globalen Koordinatensystem auf den Punkt zeigt.
* Die Materialposition ist ein dreidimensionaler Ortsvektor, welcher allerdings vom Ursprung des sog. Model-Koordinatensystems auf den Punkt zeigt, *nicht* vom globalen Koordinatensystem. Das Model-Koordinatensystem ist ein lokales Koordinatensystem, repräsentiert durch die Position eines Ursprungs (im Folgenden gekennzeichnet mit dem Vektor , der vom globalen Koordinatenursprung zum Ursprung des Model-Koordinatensystems zeigt) und die Rotation der Achsen eben jenes Koordinatensystems. In der Regel existiert ein Model-Koordinatensystem pro Objekt in der Simulation. Es dient dazu, die Position und die Rotation des Objektes zu beschreiben.
* Die globale Geschwindigkeit des Punktes ist die Änderung der globalen Position des Punktes relativ zur Zeit.

Definiert sei eine Funktion , wobei die Position des zu messenden Objekts im globalen Koordinatensystem bezeichnet und ein beliebiger Vektor in Materialkoordinaten des zu messenden Objektes ist. bezeichnet also den Ortsvektor des Model-Koordinatensystems im globalen Koordinatensystem. Damit ist eine Funktion, die von Materialkoordinaten in globale Koordinaten umrechnet.

Um nun die Belastung des Objektes an einem bestimmten Punkt zu berechnen, benötigen wir den Greenschen Verzerrungstensor, der im Folgenden mit bezeichnet wird. ist eine dreidimensionale, quadratische und symmetrische Matrix. In Tensornotation wird er folgendermaßen beschrieben:

Wobei das Kronecker-Delta bezeichnet. Es ist definiert als

Die Tensornotation liefert eine Beschreibung der Matrix-Komponente in der Zeile und der Spalte . In Matrixnotation sähe der Verzerrungstensor dann also folgendermaßen aus:

Der Greensche Verzerrungstensor reagiert nur auf Deformierungen. Wird das zu messende Objekt lediglich durch Starrkörpertransformationen verschoben oder gedreht, so schlägt das Maß nicht an.

Weiterhin wird der Verzerrungsratentensor benötigt, welcher ebenfalls eine dreidimensionale, quadratische und symmetrische Matrix ist. Er beschreibt die Veränderung der Deformierung über die Zeit. Er ist in Tensornotation definiert als:

bezeichnet hierbei die Ableitung von über die Zeit. Die Matrixnotation des Verzerrungsratentensors ist umfangreich und wird an dieser Stelle aus Platzgründen vermieden.

Das Modell stellt dem Entwickler damit ein Maß der Deformierung und ein Maß der Deformierungsrate eines beliebigen Punktes innerhalb des zu bemessenden Körpers zur Verfügung. Das Ziel des theoretischen Modells ist allerdings nicht nur die Deformierung und die dadurch entstehenden internen Kräfte zu messen, sondern auch zu erkennen, wann eine Deformierung zu einem Bruch führt und wie dieser Bruch durch das Volumen des Objektes propagiert. Diese Eigenschaften eines Volumens sind abhängig von seinem Material. Ist ein zu simulierendes Objekt aus Glas, so wird es sich kaum deformieren, bevor es bricht. Diese Eigenschaft wird als spröde (engl. „brittle“) bezeichnet und ist zurückzuführen auf einige physikalische Eigenschaften von Glas, zum Beispiel seine Tenazität / Zähigkeit, seine Dichte, seine Lamé-Konstanten und seine Dämpfungsparameter.

Um also Brüche zu simulieren und nicht nur die Deformierung, müssen die physikalischen Eigenschaften des Materials, aus das das zu simulierende Objekt besteht, berücksichtigt werden. Dies geschieht folgendermaßen:

Es wird ein Tensor der Belastung definiert. Dieser Tensor kombiniert die oben definierten Tensoren zur Messung von Deformierung und Deformierungsrate mit den Eigenschaften des Materials des zu simulieren Körpers. Das Ergebnis ist ein Maß für die gesamte innere Belastung eines Körpers. Der Tensor besteht aus zwei Einzelteilen, der elastischen Belastung durch Deformierung () und der viskosen Belastung durch die Deformierungsrate ().

Die genaue mathematische Definition von den beiden Teilkomponenten der gesamten inneren Belastung folgen. Die elastische Belastung () bezieht den Greenschen Verzerrungstensor ein und verrechnet die sogenannten Lamé-Konstanten des Materials mit dem Tensor. Die zwei Lamé-Konstanten repräsentieren die Elastizitätsmaße von Festkörpern und werden dargestellt durch einen physikalischen Druck (Kraft pro Fläche, N/m2). Ihre konventionelle Beschreibung ist gegeben durch und . Mit diesen Informationen kann nun in Tensornotation definiert werden:

Auch hier liefert die Tensornotation wieder eine Beschreibung der Matrix in der Zeile und in der Spalte . Analog wird die viskose Belastung durch die Deformationsrate () definiert, welche, anstatt der Lamé-Konstanten, welche ein Maß für Elastizität ist, zwei andere namenlose Konstanten und verwendet, die dämpfende Eigenschaften des Materials repräsentieren. Sie werden ebenfalls als ein physikalischer Druck (N/m2) dargestellt. Die Tensornotation von ist somit:

Nun fehlt nur noch die Definition eines einzigen Maßes, um das kontinuierliche Modell bezüglich spröder Frakturpropagation abzuschließen. Die Tenazität / Zähigkeit eines Materials wird durch eine Druckkonstante (N/m2) repräsentiert. Sie wird in den folgenden Beschreibungen aufgegriffen und ihre Relevanz wird deutlich, denn sie ist essenziell, um zu erkennen, ab welchem Ausmaß innerer Kräfte sich ein Bruch auftut.

### Duktile Fraktur

Um das kontinuierliche Modell auf duktile Frakturen zu erweitern, ist nur eine kleine Änderung notwendig. Bei der Berechnung der elastischen Belastung wird der Greensche Verzerrungstensor durch die elastische Verzerrung ersetzt, welche im Folgenden definiert wird. Des Weiteren wird eine Routine eingebaut, welche die plastische Verzerrung aktualisiert. Dazu im Folgenden ebenfalls mehr.

Es wird mit der Neudefinition von Deformierung und Verzerrung begonnen. Betrachtet man die Greensche Verzerrung, so kann man sie aufteilen zwei Komponenten: elastische und plastische Verzerrung.

Diese Aufteilung ermöglicht dem Entwickler die Separation der beiden verschiedenen Deformierungsstadien eines duktilen Materials und somit die Simulation eben jener. Die Berechnung der gesamten Verzerrung ist bereits durch den Teil des spröden Modells gegeben. Im Folgenden wird definiert, wie die plastische Verzerrung zu berechnen ist. Durch die Subtraktion der plastischen Verzerrung von der gesamten Verzerrung ergibt sich dann schließlich die elastische Verzerrung .

Im Rahmen der Berechnungen werden zwei Materialkonstanten definiert, und . bezeichnet die elastische Grenze des Materials, also ab welchem Punkt plastische Deformierung auftritt. bezeichnet die plastische Grenze des Materials, also ab welchem Punkt Frakturen auftreten. Ein sprödes Material hätte somit die Materialkonstanten und .

Es ist wichtig zu erkennen, dass die plastische Verzerrung zu Anfang der Simulation nicht existent ist (es sei denn sie wird explizit gesetzt). Somit wird im ersten Durchgang der Kerninteraktionsschleife (zum Zeitpunkt t0) mit der Null-Matrix als plastische Verzerrung begonnen. Der nächste Durchgang (zum Zeitpunkt t1) berechnet die Differenz der plastischen Verzerrung vom ersten Zeitpunkt zum zweiten Zeitpunkt der Simulation, genannt . Die neue plastische Verzerrung ergibt sich dann aus folgender Berechnung:

Die Differenz der plastischen Verzerrung wird aus der elastischen Verzerrung berechnet. Dafür wird die Ableitung der elastischen Verzerrung , genannt , benötigt:

Wobei die Einheitsmatrix bezeichnet und die Spur der elastischen Verzerrung beschreibt, also die Summe der Elemente der Hauptdiagonale. Ausformuliert geschieht Folgendes: Der Durchschnitt der Hauptdiagonale von wird gebildet und ersetzt die Elemente der Hauptdiagonale einer Einheitsmatrix. Dann wird diese Matrix von subtrahiert. Dadurch werden durch die Deformierungen gegebenenfalls entstehende Vergrößerungen des Volumens verhindert. Im Folgenden definieren wir das von-Mises-Kriterium, welches, falls es zutrifft, angibt, dass plastische Verformungen auftreten:

Schließlich definieren wir die Berechnung von , damit berechnet werden kann:

Somit können nun sowohl die gesamte Verzerrung , als auch die plastische Verzerrung berechnet werden. Und durch Umformulierung der Zusammensetzungsformel kann auch berechnet werden. Damit sind alle notwendigen mathematischen Maße und Formeln zur Berechnung dieser Maße gegeben, um elastische und plastische Verformung zu simulieren.

## 2.2 Das diskrete Modell

Das oben beschriebene kontinuierliche Modell liefert alle mathematischen Funktionen, um die verschiedenen Eigenschaften eines geometrischen Körpers aus einem bestimmten Material zu berechnen. Doch um diese Berechnungen sinnvoll auf einen geometrischen Körper in einer Computer-Simulation durchzuführen, bedarf es einer sog. Diskretisierung. Hierbei wird der zu simulierende Körper auf eine gewisse Art und Weise in Teile unterteilt oder mit einem Gitter bedeckt, welches die Grenzen des Materials absteckt, jenachdem welche Diskretisierungsmethode gewählt wurde.

In diesem Projekt wurde die sog. Finite-Elemente-Methode, kurz FEM, gewählt. Sie besagt, dass der zu simulierende Bereich des Materials in kleinere Subbereiche, Elemente genannt, unterteilt werden soll. Das Material innerhalb des Volumens dieser Elemente wird durch sogenannte Shape- oder Base-Functions (dt. Form- oder Basisfunktionen) beschrieben. Diese Funktionen hängen jeweils von genau einem Knotenpunkt dieses Elementes ab. Die FEM kann mit verschiedenen Arten von Elementformen umgesetzt werden, zum Beispiel mit Quadern, Würfeln, oder – wie im Falle dieses Projektes – mit Tetraedern.

Um nun einen dreidimensionalen Körper simulieren zu können, muss er also diskretisiert werden. Das bedeutet, dass der Raum, den der Körper einnimmt, durch Tetraeder definiert wird. All diese Tetraeder sind über ihre Eckpunkte mit ihren Nachbartetraedern verbunden. Existiert also zum Beispiel einen Würfel als zu simulierender Körper, so kann dieser Würfel auf verschiedene Arten in Tetraeder unterteilt werden. Eine Beispielunterteilung wird in Abbildung (?) gezeigt. Wenn es sich um komplexere dreidimensionale Körper handelt als die handelsüblichen Primitive (Sphären, Würfel, Zylinder, usw.), so wird die Unterteilung in Tetraeder komplizierter.

Hier kommt die Natur dreidimensionalen Modelle in Videospielen der Diskretisierung allerdings zugute, denn sie sind meistens mit Oberflächen aus Dreiecken realisiert. So können sich bei der Erstellung der Tetraeder diese Dreiecke zunutze gemacht werden, denn Tetraeder haben als Flächen ausschließlich Dreiecke.

Eine wichtige Eigenschaft muss das entstandene Netz aus Tetraedern allerdings besitzen: Es muss konsistent sein. Das bedeutet konkret, dass alle Kanten, die gemeinsam Verlaufen, auch dieselbe Länge besitzen. Dies ist noch einmal grafisch in Abbildung (?) festgehalten. Dies ist wichtig für die Remeshing-Vorschrift des Algorithmus, denn es vereinfacht die Behandlung der Tetraederverbindungen zueinander ungemein, wenn von dieser Eigenschaft ausgegangen werden kann.

Im Folgenden wird anschaulich das diskrete Modell erläutert, insbesondere, wie es dem Entwickler mithilfe der Diskretisierung durch die Tetraeder und durch die Formeln, die aus dem kontinuierlichen Modell stammen, gelingen kann, interne Belastungen von dreidimensionalen Körpern approximieren zu können und so Frakturen simulieren zu können.

# Die Implementierung in Unity

Die Implementierung des in dieser Arbeit beschriebenen Algorithmus zur Frakturerkennung und -propagation von sowohl spröden als auch duktilen Materialien geschah in der Unity3D-Spieleengine. Diese Engine ist weitläufig bekannt und weist alle erdenklichen Bestandteile einer Spieleengine auf, die einem die Implementierung des Algorithmus erleichtern. So entstand zum Beispiel kein Mehraufwand, weil zuerst ein Renderer entwickelt werden musste. Des Weiteren bietet die Engine dem Entity-Component-System (kurz ECS) konforme Game-Objects, welche in ihrer minimalen Form nur einen Punkt im Raum beschreiben, oder in einer komplexeren Ausführung eines der Tetraeder, die im Kapitel xyz besprochen wurden. Letztlich bietet die Unity-Engine eine äußerst nützliche Funktionalität zum schnellen Testen: das Starten des Spiels im Editor. Dadurch wird der Algorithmus im selben Fenster aktiv, in welchem gerade noch die Game-Objects gesetzt und manipuliert wurden.

Die Implementierung des Algorithmus folgt streng dem „Entity-Component-System“-Muster insofern, dass alle Objekte oder komplexeren Entitäten des Algorithmus als Game-Objects realisiert sind. Der Algorithmus setzt beispielsweise voraus, dass das zu simulierende Objekt aus Tetraedern besteht, die über Knotenpunkte miteinander verbunden sind. So sind die Tetraeder selbst als Game-Objects umgesetzt und die Knotenpunkte, die die Tetraeder verbinden, ebenfalls. Diese intuitive Herangehensweise ermöglichte eine schnelle Testbarkeit, denn Game-Objects in Unity sind zur Laufzeit eins Spiels recht einfach zu manipulieren. So kann ein Nutzer des Algorithmus etwa ein Mesh an Tetraedern und Knotenpunkten erstellen, das Spiel innerhalb der Unity-Engine starten und zur Laufzeit des Spiels die Knotenpunkte mit der Maus verschieben und somit Deformierungen des Meshes erschaffen.

Die Algorithmus-Funktionalität wurde folgendermaßen an die Game-Objects angehangen: In Unity existiert das Konzept von Skripten. Ein Skript in Unity ist eine Komponente eines Game-Objects, implementiert durch eine C#-Klasse. Diese Quellcodedatei enthält mehrere Callback-Funktionen, die bei Start des Spiels (zum Beispiel innerhalb des Editors) oder während das Spiel läuft von der Engine aufgerufen werden können (insofern sie implementiert sind). So existiert die Start()-Methode, welche bei Beginn des Spiels einmal aufgerufen wird und die Update()-Methode, die einmal pro Durchlauf der Kerninteraktionsschleife des Spiels aufgerufen wird. Da das Skript ansonsten eine reguläre Klasse im objektorientierten Sinne ist, lassen sich einfach weitere Methoden und Attribute definieren.

Im Folgenden wird der Vorgang der Implementierung beschrieben. Es wird beleuchtet, in welcher Form die Diskretisierung erreicht wurde, also inwiefern das diskrete Modell umgesetzt wurde. Daraufhin wird die Berechnung der verschiedenen Maße von Deformierung, Belastung und interner Kräfte beschrieben. Nicht zuletzt werden Inhalte aufgelistet, die weiterhin implementiert werden könnten. Abschließend wird kurz erläutert, wie das System in ein existierendes Unity-Spieleprojekt eingeplegt werden kann.

## Planung

Der Beginn der Bearbeitung dieser Abschlussarbeit bestand aus einer intensiven Recherche an Materialien, die den Einstieg in die sehr physikalische Thematik erleichtern konnten. Hierbei wurde vor allem auf die vom Algorithmus referenzierte Literatur zurückgegriffen, um ein generelles Verständnis für die physischen Abläufe zu erlangen. Die im Algorithmus-Paper verwendete Notation, die sogenannte Tensor-Notation, musste nachvollzogen und verstanden werden, äußerte sich dann aber als sehr hilfreich bei der Formulierung der verschiedenen Daten. Nachdem der Algorithmus verstanden und in einen physikalischen Kontext eingebunden war, konnte mit der Planung der Software begonnen werden. Es stand bereits zu Anfang fest, dass die Implementierung in der Unity-Engine stattfinden soll und so umgesetzt werden sollte, dass sie von verschiedensten Unity-Spieleprojekten eingebunden werden konnte.

### Architektur

Die Idee der Architektur war folgende: Das zu simulierende Objekt würde als ein Game-Object realisiert sein. Es solle ein Kind-Game-Object besitzen, welches den expliziten Namen "FEM\_Mesh" tragen müsse. Dieses Kind-Game-Object wäre das Wurzelelement für die Datenhaltung und Funktionalität des Algorithmus. Dieses "FEM\_Mesh"-Game-Object besäße eine C#-Skript-Komponente, die den Algorithmus umsetze. Die Daten des Algorithmus - die Tetraeder und Knotenpunkte - wären als Kind-Game-Objects des "FEM\_Mesh"-Game-Objects realisiert und würden ihre Daten und Funktionen innerhalb einer C#-Klasse kapseln, die sie als Skript-Komponente angehangen bekämen. Hierbei wird kurz das Konzept eines Unity-Prefabs erläutert. Prefabs erlauben in Unity, ein Game-Object mit all seinen Komponenten und Einstellungen als eine Datei abzuspeichern. Dies ermöglicht, dieses Game-Object zu einem beliebigen Zeitpunkt im Spiel in beliebiger Anzahl, mit beliebiger Position und Ausrichtung zu instanziieren. Es solle ein Prefab für einen Knotenpunkt existieren, denn solch einer solle im Quellcode instanziiert werden können. Diese Architektur sollte dem Nutzer des Algorithmus ermöglichen, den Algorithmus einfach in ein eventuell bereits existierendes Spieleprojekt einzubinden, indem man dem zu simulierenden Game-Object ein weiteres Kind-Game-Object anhängen würde, welches die beschriebene Funktionalität implementierte.

### Testsystem

Damit war eine Architektur gegeben. Nun fehlte noch das Konzept eines Workflows, welches das sinnvolle und breite Testen der zu entwickelnden Funktionalität ermöglichen würde. Hierfür würde ein Editorskript reichen, dass die spontane Erstellung von Tetraeder-Meshes mit ihren Knotenpunkten ermöglichen sollte. Des Weiteren sollte das Tetraeder-Mesh nicht einfach nur eine im Quellcode vorhandene Datenstruktur sein, sondern sichtbar gezeichnet werden, damit eventuelle Inkonsistenzen, Fehler oder Artefakte sofort erkannt würden. Damit war die Grundlage für die Implementierung des Algorithmus in Unity gegeben.

Anhand dessen wurden im Laufe der Implementierung dem Entwickler sinnvoll erscheinende Tests umgesetzt. Für jeden zu testenden Aspekt des Projekts wurde eine neue Unity-Szene angelegt. Eine Unity-Szene ist ein Konzept aus der Unity-Engine, welches (unter anderem) alle Positionen, Rotationen und Skalierungen jedes Game-Objects in der Spielewelt in einer Datei speichert. Ein Projekt kann mehrere Szenen haben. Die eigentliche Verwendung von Szenen basiert auf dem Konzept von Levels in Videospielen, aber kann in diesem Kontext auch hervorragend zum Testen verwendet werden.

## Ablauf der Implementierung

## Mögliche Erweiterungen

## Einpflegung in ein existierendes Spiel

# Diskussion der Ergebnisse

// In diesem Kapitel sollen deutlich die eigenen Ergebnisse und Erkenntnisse herausgestellt werden.

# Ausblick

# Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Tabellenverzeichnis

# Literaturverzeichnis

O’Brien, J. F., Bargteil, A. W., & Hodgins, J. K. (2002). Graphical Modeling and Animation of Ductile Fracture.

O'Brien, J. F., & Hodgins, J. K. (1999). Graphical Modeling and Animation of Brittle Fracture. *ACM SIGGRAPH 99.* Los Angeles, Kalifornien, USA.

# Anhang