

Anwendung von Softbody-Techniken an Spielobjekten in der Unreal Engine

*

18. Dezember 2021

*,

Kurzfassung

In Computersimulationen werden elastische Objekte verwendet. Diese werden bewusst weggelassen oder geplant eingesetzt. Das Ziel in der vorliegenden Arbeit ist es zu beantworten, welche Techniken in der Unreal Engine angewendet werden können, um das Spielerlebnis realistischer zu gestalten. Dazu wird die Forschungsfrage gestellt: Welche verfügbaren Weichkörpertechniken sind in der Unreal Engine umsetzbar und mit welchen Frameworks lassen sich diese erweitern? Um diese Forschungsfrage beantworten zu können, wird der aktuelle Markt auf Erweiterungen analysiert und basierend auf diesen Erkenntnissen werden Beispiele für die virtuelle Testumgebung angefertigt. Es stellt sich heraus, dass einige Methoden bereits vorhanden sind, einige jedoch einen Mehraufwand beinhalten. Weiterhin fehlt es an einem ausgereiften Plugin für die Unreal Engine. Daraus lässt sich ableiten, dass in diesem Segment ein Aufholbedarf besteht.

Keywords— Weichkörperdynamik, Game Engine, Shader-Programmierung, Nvidia FleX

Abstract

Elastic objects are used in computer simulations. These are deliberately omitted or used in a planned way. The aim in this thesis is to answer which techniques can be used in the Unreal Engine to make the game experience more realistic. To this end, the research question is posed: Which available soft body techniques are available in the Unreal Engine and which frameworks can be used to extend them? In order to be able to answer this research question, the current market is analysed for extensions and based on these findings, examples are made for the virtual test environment. It turns out that some methods are already available, but some involve additional work. Furthermore, there is a lack of a mature plug-in for the Unreal Engine. From this it can be deduced that there is a need to catch up in this segment.

Keywords— softbody, game engine, shader-programming, nvidia flex

Inhaltsverzeichnis

Abbildungsverzeichnis	VI
Tabellenverzeichnis	VII
Abkürzungs-/Fremdwortverzeichnis	VIII
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel der Ausarbeitung	1
1.3 Status Quo	1
2 Grundlagen	3
2.1 Game Engine	3
2.2 Deformierbare Objekte	4
2.2.1 Typische Objekte	5
2.2.2 Physikalische Grundlagen	6
2.2.3 Das Model Spring/Mass Model	7
2.2.4 Particle Based Simulation	7
2.2.5 Finite Elemente-Methode	8
3 Aktuelle Frameworks	9
3.1 Nvidia FleX	9
3.2 AMD OpenGPU FEMFX	10
3.3 VICODynamics	11
3.4 PhysX 5.0	12
3.5 Entscheidungsfindung	12
4 Konzeption und Implementierung	14
4.1 Überlegung	14
4.2 Native Unreal Implementation	14
4.2.1 Erstellung eines Seiles	14
4.2.2 Erstellung von Kleidung	15
4.2.3 Erstellung von Haaren	15
4.2.4 Erstellung eines vorgetäuschten Weichkörpers	15
4.3 Nvidia FleX Implementation	17
4.3.1 Erstellung eines Seiles	17
4.3.2 Erstellung von Kleidung	17
4.3.3 Erstellung von Haaren	17
4.3.4 Erstellung eines Weichkörpers	17
4.4 Anwendung an einem Charakter	18
4.4.1 Vorgehensweise bei einem Charakter	18
4.4.2 Implementierung eines Charakters nur mit Techniken aus der Unreal Engine	18
4.4.3 Implementierung mit Hilfe des FleX Plugins	21
4.4.4 Separierung der kosmetischen Objekte	24
4.4.5 Separierung der Muskeln	25

4.5	Anwendung an Vegetationen	26
4.5.1	Nativ Unreal-Implementation	26
4.5.2	Weiterführung der Implementierung	28
4.5.3	Nvidia FleX-Implementation	28
5	Ergebnisse und Bewertung	29
5.0.1	Seil/Kette	29
5.0.2	Kleidung	29
5.0.3	Haare	30
5.0.4	Weichkörper	30
5.1	Visueller Vergleich der umgesetzten Ergebnisse	30
5.1.1	Testen der Charaktere	30
5.1.2	Testen der Vegetation	31
5.2	Problematiken des Nvidia FleX-Frameworks	31
5.2.1	Umsetzung der Charaters	31
5.2.2	Umsetzung der kosmetischen Objekte	31
5.2.3	Sonstige Probleme	32
6	Fazit und Ausblick	33
6.1	Fazit	33
6.2	Ausblick	33
7	Weitere Abbildungen	35
Literaturverzeichnis		40

Abbildungsverzeichnis

4.1	Visualcode zum Deformieren von Objekten	15
4.2	Material Visualcode zum erstellen des Tessellation-Effektes	16
4.3	Charakter "Rampage" mit markierten Arealen	18
4.4	Der Charakter im Physik-Editor	19
4.5	Aktivieren der simulierten Knochen im Code	20
4.6	Konfiguriertes Kleidungsstück des Charakters "Rampage"	21
4.7	Charakter "Rampage" als FleX Static Mesh-Objekt	22
4.8	Charakter mit Knochen im und um den simulierten Weichkörper	23
4.9	Code zum beheben auftretender Probleme beim FleX-Objekt	24
4.10	Code für den Material Shader	27
4.11	Code für den Material Shader mit Fuß Erkennung	28
7.1	Testareal für die Basis-Komponenten	35
7.2	Testareal für die umgesetzten Beispiele	35
7.3	Übersicht der FleX-Objekte	36
7.4	Übersicht beider Haar-Umsetzungen	36
7.5	Übersicht aller Seil-Implementationen	37
7.6	Übersicht aller Weichkörper	38
7.7	Übersicht aller Kleidungs-Implementierungen	38
7.8	Übersicht aller Charakter-Implementierungen	39
7.9	Übersicht aller Gras-Implementierungen	39

Tabellenverzeichnis

3.1 Übersicht Frameworks	13
4.1 FleX-Einstellungen des Charakters	22

Abkürzungs-/Fremdwortverzeichnis

Framework: Eine Softwarebibliothek mit Ansammlungen von Funktionen

scripten: Programmieren (Englisch: to script)

CPU: Central Processing Unit (Deutsch: Prozessor)

Middleware: Eine Software, die als Brücke zwischen Zwei Ebenen verwendet wird

Mesh: Englisch für Masche

SDK: Software Development Kit. Ansammlung von Softwareprogrammierwerkzeugen

VisualFX: Visuale Effekte

Shader: Effekte, die auf der Grafikeinheit berechnet werden.

LODs: Level of Detail von Objekten, die entfernter haben weniger Details.

Blueprint Visual Script: Eine visuelle Programmierprache, die in der Unreal Engine verwendet wird.

Static Mesh: Eine Statische Struktur aus Texturen

Tessellation-Shader: Ist ein Teil der 3D Grafikpipeline

Realtime: Englisch für Echtzeit

Toolset: Satz Werkzeuge

Rigidbody: Starrer Körper

Cyberpunk 2077: Ein Spiel des polnischen Herstellers CD Project Red

Triple A-Spiele: Spiele die von umsatzstarken Unternehmen produziert werden.

Plastizität: Fähigkeit von Feststoffen nach Überschreitung einer Krafteinwirkung sich irreversibel zu verformen.

Elastizität: Fähigkeit von Feststoffen durch äußere Einwirkung hervorgerufene Formänderung aus eigener Kraft wieder rückgängig zu machen.

Keyframe: Eine Schlüsselmarkierung

Kollisions-Trigger: Ein Areal welches als Auslöser für die Kollision dient.

Plastic Deformation: Die Simulation von Plastizität

Nvidia GameWorks: Eine Ansammlung von spezifischen Frameworks von Nvidia für die Spielebranche

DirectX 11: Ansammlung von Softwarekomponenten für die Darstellung von 3D und 2D Grafik

Github: Ein Dienst zur Versionsverwaltung von Software im Internet

FBX: Filmbox Format

fork: Abspaltung eines Projektes

Trigger-Event: Auslöser

Raycast: Sehstrahl

Ports: Portierung eines Programmes

Lerp-Funktion: Lineare Interpolation

FPS: Frames per Second

1 Einleitung

1.1 Motivation

Bei der Betrachtung von realen Gegenständen und deren Interaktion untereinander, fällt auf, dass die Elastizität und Plastizität in der Unterhaltungsindustrie, wie zum Beispiel in Computerspielen, vernachlässigt werden. Jedes Objekt besitzt einen Innenwiderstand und verformt sich bei genügend Krafteinwirkung. Bisher ist die Verformung von Objekten jedoch nur in Cinematic-Szenen oder in Echtzeit gescriptet umgesetzt worden. Aufgrund der aktuellen Entwicklung im Hardwarebereich, vor allem im Forschungssegment Simulation mit dem Beispiel [1], wird eine erneute Betrachtung der Thematik interessant.

1.2 Ziel der Ausarbeitung

Das Hauptziel dieser Ausarbeitung ist, eine grundlegende Evaluation des aktuellen Stands der Technik aufzuzeigen. Resultierend aus dieser soll eine Anwendung erstellt werden, mit der es möglich sein soll zu ermitteln, in welchem Umfang die vorhandenen Methoden genutzt werden können. Dabei sollen verschiedene Spielobjekte mit physikalischer Deformierbarkeit dargestellt werden. Basierend darauf wird ein realistisches Spielerlebnis für den Nutzer geschaffen werden. Die Unreal Engine soll dabei genutzt werden, um einen besseren Einblick in die Thematik zu erhalten. Es soll aufgezeigt werden, wie die Technik grundsätzlich funktioniert und wie sie in der Engine realisiert werden kann. Außerdem sollen alle aktuellen Frameworks auf dem Markt analysiert und mit deren Hilfe soll der Funktionsumfang der Engine mit möglichen Beispieltechniken erweitert werden. Zugleich sollen die Frameworks quelloffen sein. Die ausgearbeiteten Erkenntnisse werden in den nachfolgenden Kapiteln in der Engine umgesetzt. Dabei werden typische Szenarien erprobt und bewertet. Es ist zu beachten, dass die gesamte Simulation in Echtzeit berechnet wird.

1.3 Status Quo

Die Spielebranche setzt vermehrt auf Fotorealismus in Spielen [2]. Bei den aktuellen Spielen ist auffällig, dass Reflexionen, Grafiken mit hohen Auflösungen und virtuelle Effekte großzügig verwendet werden. Daher ableitend stellt sich die Behauptung auf, dass die führenden Produzenten der Spielebranche mehr Wert auf ein Realtime-errechnetes, grafisches Erlebnis setzen, statt vermehrt die physikalischen Komponenten zu nutzen und auszubauen. Es zeigt sich, dass die realistische Simulation von Haut, Haaren, Stoffen, Flüssigkeiten etc. nur in geringfügigem Maße vorhanden ist. Grund hierfür kann ein zu kleines Toolset von Frameworks oder Spiele Engines, die solche Funktionen nicht anbieten, sein. Ein weiterer Punkt ist die Komplexität der zu simulierenden Eigenschaften. Wenn die Eigenschaften von Flüssigkeiten bspw. näher betrachtet werden, besteht die Herausforderung darin, dass alle möglichen Formen angenommen werden müssen und Rücksicht auf die Dichte genommen werden muss. Außerdem besteht die Option Spiele als Multiplattformtitel zu entwickeln, was zur Folge hat, dass Entwickler durch ein limitiertes Budget nicht den vollen Umfang der bereitgestellten Tools verwenden können. Es wäre günstiger ein Spiel so zu entwickeln, dass es auf vielen Plattformen laufen kann, statt mehrfache Ausführungen zu haben, welche plattformspezifisch sind. Bei exklusiven Konsolenspielen fällt auf, dass einige Techniken seltener

verwendet werden, da diese nur auf einer älteren Hardware laufen und dadurch Leistungseinbußen zu Lasten des Spielergebnisses hätten. Als Musterbeispiel lässt sich der Skandal um Cyberpunk 2077 aufführen. Dieses Spiel ist primär für die neue Generation von Hardware entwickelt worden und Techniken, wie Reflexionen etc., haben massiv die ältere Konsolen-generation überfordert. Aufgrund dessen wurden extreme Einschnitte in die Grafik und KI gemacht, damit der Titel mit konstanter Bildrate abgespielt werden kann. Folglich ist die Entscheidung nachzuvollziehen, auf realistische physikalische Komponenten zu verzichten, wenn die Berechnung der Grafik ohnehin nicht bewerkstelligt werden kann [3]. Im Gegenzug dazu gibt es eine überschaubare Menge an Spielen, die vermehrt auf den Realismus in der Physik Wert legen. Zum Zeitpunkt der Recherche zu dieser Arbeit sind dies Simulationen mit dem Hauptaugenmerk auf der Zerstörung von Fahrzeugen. Dies wurde auch bei dem Spiel Cyberpunk 2077 berücksichtigt. Zu den bekanntesten Spielen in diesem Bereich zählen BeamNG.drive[4] oder Wreckfest[5]. Im historischen Kontext gesehen, ist die Leistung von Computern begrenzt gewesen, sodass es nicht möglich war aufwendige Simulationen zu berechnen. Erst mit der Parallelisierung der Prozesse und dem Ausbau der Hardware wurden die aktuellen Mittel geschaffen, um anspruchsvollere Darstellungen und Physikberechnungen umzusetzen.

2 Grundlagen

2.1 Game Engine

Game Engines oder zu Deutsch Spiele Engines beinhalten, die dem Programmierer für die Erstellung eines Spiels zur Verfügung gestellten Werkzeuge. Im Software-Fachjargon wird dies als spezifische Framework-Sammlung bezeichnet. Dabei werden Funktionen zum Erstellen von Spielen, möglichst benutzerfreundlich, in einer Anwendung für Künstler und Entwickler bereitgestellt. Eines der Hauptmerkmale ist die Darstellung der Grafik. Dabei werden Grafik-Frameworks wie DirectX, Vulkan oder OpenGL als Grundlage verwendet und in das Konzept der Engine mit eingebaut. Zusätzlich werden alle Grafiken in ein physikalisches System eingebettet, damit sie miteinander agieren können. Hierbei werden alle im Spiel vorhandenen Objekte als Vektoren dargestellt und als Akteure gehandhabt. Weitere Bestandteile sind Audioquellen, die anhand der Entfernung im physikalischen System berechnet werden. Ein zusätzliches Modul erweitert alle Akteure hinsichtlich der Netzwerkebene, damit diese in einem Mehrspielermodus alle synchronisiert werden können. Um die genannten Komponenten zu verwalten, hat eine Spiel Engine in der Regel ein eigenes Datenverwaltungssystem. Dieses bewerkstelligt die Verknüpfung aller Elemente sowie die Sicherung in einer strukturierten Art. Die Ausführung von Code, einer oftmals verwendeten Programmiersprache, ist ein weiteres Merkmal in diesem Konstrukt. Dabei werden visuelle Programmiersprachen, wie Blueprints oder klassische Code-Dateien, zum Programmieren verwendet.

Die Engine, die in der folgenden Ausarbeitung verwendet werden soll, ist die Unreal Engine von Epic Games Inc [6]. Diese beinhaltet die bereits genannten Bausteine und zählt zu den bekanntesten Engines auf dem Markt [7]. Die erste Unreal-Version wurde 1998 auf den Markt gebracht und hat das Spiel Unreal Tournament mit benannt. Seitdem wird sie kontinuierlich weiterentwickelt. Sie hat auch eine Menge bekannter Triple A-Spiele hervorgebracht. Darunter fallen Spiele aus der Borderlands-Reihe oder das bekannteste Beispiel für die Eigenentwicklung ist Fortnite.

Im Vergleich zu ihren Konkurrenzprodukten ist die Unreal Engine in der Grafik besser aufgestellt und bietet eine native Implementation zum virtuellen Programmieren. Zum aktuellen Zeitpunkt ist mit Version 5 ein Nachfolger angekündigt worden. Das Update hat unter anderem ein eigenes Physiksystem und weitere Features erhalten. Diese konnten bereits bei etablierten Redaktionen auf positive Resonanz stoßen [8].

Im Rahmen einer anfänglichen Analyse, die herausfinden sollte, welche Optionen für das Erarbeiten des Themas in Betracht gezogen werden können, war bei den meisten Frameworks die Unreal Engine als verfügbare Entwicklungsumgebung vorhanden. Weiterhin hat sich bei der Auswahl positiv gezeigt, dass die Engine seitens Epic Games einen neuen Standard auf den Markt etablieren soll. Im Vergleich wird die Unity Engine zumindest im Bereich der Softbody-Technologien eher vernachlässigt behandelt. Währenddessen AMD oder Nvidia mit Epic Games medienwirksam im Bereich der Weichkörperdynamik zusammenarbeiten. Aus diesem Grund und der Tatsache, dass die aktuellen Implementierungen des Physiksystems namens PhysX [9] sowie das Framework Apex Destruction [10] von Epic Games bewilligt worden sind, ist die Unreal Engine für die nähere Betrachtung gewählt worden.

2.2 Deformierbare Objekte

Das Konzept der deformierbaren Objekte lässt sich aus der Physik ableiten. Festkörper haben eine bestimmte mikroskopische Struktur, um ihren Aggregatzustand zu erhalten. Es handelt sich dabei um Bindungsenergie [11]. Wie im Absatz 2.1 aufgeführt, wird eine vereinfachte Version der Realität angenommen, um diese möglichst genau, ohne hohe Rechenlast, darstellen zu können. Dabei wird die Bindungsenergie der einzelnen Körper nicht berücksichtigt, sodass keine Leistungseinbußen durch Mehraufwand beim Berechnen dieser entstehen. Aufgrund der Banalisierung bedient man sich dem Konzept des schlichten Festkörpers in der Simulation. Diese Festkörper werden in der Theorie als Punkte in einem Koordinatensystem der Engine dargestellt. Sie werden als festes Mesh simuliert und wiederum mit einer Textur ins Koordinatensystem projiziert. Um die Struktur wird ein Feld erzeugt, welches als Kollisionserkennung dient. Hierbei handelt es sich um eine Komponente, welche die grundlegenden physikalischen Gesetze simuliert. Die simulierten Körper behalten ihre Form und geben die eingehenden Kräfte unverändert weiter. Deshalb werden sie Rigidbodys, zu Deutsch starre Körper, genannt.

Das gleiche Konzept lässt sich auf die deformierbaren Objekte anwenden. So können Objekte nicht nur als starr simuliert werden, sondern erfahren an den jeweiligen Flächen, an denen Kraft einwirkt, eine Veränderung. Diese Kraft wird vom Körper aufgenommen und verhält sich nach einem Muster, welches zuvor von dem Programmierer festgelegt worden ist. Das heißt, der relative Abstand zwischen zu simulierenden Punkten hat keine fixe Größe und ändert sich dynamisch anhand eines Algorithmus [12]. Dabei bleibt das Volumen des Körpers beim Simulieren erhalten. Der Fachterminus hierfür lautet Softbody, zu Deutsch Weichkörperdynamik.

Bei den Softbodys kann nach dem Grad der Implementation unterschieden werden: Entweder nach der angewandten mathematischen Methode oder nach dem Grad der Simulation (komplett oder Teilsimulation). Bei der Teilsimulation wird meistens nicht auf die „echte“ Simulation zurückgegriffen, sondern es wird lediglich der Anschein erweckt, das Material hätte sich nach der Krafteinwirkung verformt. Aktuelle Entwicklerstudios bedienen sich meist Realtime- oder Teilsimulationen, die wie nachfolgend beschrieben, funktionieren. Dazu zählen:

1. Morph Targets

Morph Targets oder auch Blendshapes genannt, sind Übergänge von Zuständen eines Objektes basierend auf festen Ausgangsformen. Dabei wird das zu simulierende Objekt in seinem Start- und Ausgangszustand mit Keyframes verknüpft. Der Übergang wird von der Engine simuliert. Diese Technik wird vermehrt zur Simulation von Gesichtszügen verwendet. Ein Nachteil dieser Technik ist ein immer gleich aussehender Effekt, der sich nicht dynamisch an die jeweilige Situation anpasst und in jedem Fall gleich verhält. Dadurch werden äußere Einflüsse, die das Objekt anders verformen, ausgeschlossen.

2. Skeletal Meshes

Mit Skeletal Meshes werden hauptsächlich Charaktere oder sich bewegende Formen animiert. Diese erhalten in einer 3D-Grafik-Software ein Skelett. Dieses wird durch einen Artist animiert, wodurch es so aussieht, als ob es sich realistisch verhält. An diesem Skelett werden in einer Game Engine Kollisionsobjekte angebracht, die die umliegenden Texturen bei einer eingehenden Veränderung beeinflussen. Umgangssprachlich werden Knochen, die diese Eigenschaften vorweisen, „Spring“ oder „Bouncy Bones“ genannt. Klassische Beispiele hierfür sind große Muskelmassen oder Fettgewebe.

3. Procedural Meshes

Procedural Meshes nutzen die Kräfte vom Eingabewert bei der Interaktion von Objekten, um damit neue Teilobjekte zu erstellen oder diese zu beeinflussen. Aus dem statischen Objekt wird dadurch ein dynamisches erzeugt. Bei der Berechnung können weitere Meshes hinzugefügt oder die aktuellen angepasst werden.

4. Material Vertex Displacement

Bei dieser Technik wird ein Objekt basierend auf seinem Material bzw. seiner Textur verändert. Mit Hilfe eines Shaders werden die Flächen, die das Objekt besitzt, verschoben. Es sollte beachtet werden, dass sich nur jene Meshes des Objektes verschieben. Weitere Anpassungen zur maßgetreuen Interaktion müssen z. B. durch die Kollisions-Trigger-Verschiebung vorgenommen werden.

2.2.1 Typische Objekte

Da der Begriff Softbody sehr umfassend sein kann, sollen hier nun einige Beispiele aufgezeigt werden, welche genauer beschreiben in welchem Umfang diese in Spielen oder Simulationen verwendet werden können:

1. Kabel, Schläuche, Seile, Ketten

Diese Objekte haben die natürliche Eigenschaft sich zu verbiegen, dabei aber ihre Form weitestgehend beizubehalten. Deshalb können sie nicht als Festkörper modelliert werden. Außer die Objekte sollen als Dekoration dienen, dann können sie jedoch nicht zur Interaktion genutzt werden. Aufgrund dessen bedient man sich anderen Methoden, die im weiteren Verlauf der Bachelorthesis näher beleuchtet werden.

2. Haare

Jedes Haar kann sich beliebig oft verformen, behält aber seine Form eines sehr langen Seiles bei. Haare werden oftmals nur als Haarbündel vereinfacht dargestellt, da die Darstellung einzelner Haare einen sehr hohen Rechenaufwand in der Computersimulation benötigt.

3. Kleidung

Kleidung wird ähnlich wie Haare oft von der alltäglichen Physik beeinflusst. Im Gegensatz zu der einfachen Form eines langen Seiles ist das Simulieren eines Stoffes komplizierter, da durch Querverstrebungen von Fasern und Material die Komplexität der Simulation zunimmt.

4. Organische Materialien

Unter organische Materialien fallen Muskeln, Haut, Fettgewebe oder auch Pflanzen. Diese haben die Eigenschaft bei einer Krafteinwirkung zurück in ihre Ursprungsform zu gelangen. Sie sind im Vergleich zu Kleidung noch komplexer, da nicht nur die Verflechtungen des Stoffes, sondern auch die Problematik mehrerer Schichten oder vielfältiger Formen besteht.

5. Volumenkörper

Jeder Körper, der in der nicht simulierten Welt existiert, hat eine physikalische Elastizität. Beispiele dafür sind Objekte, die z. B. einen dickflüssigen Aggregatzustand haben oder spezifische Objekte wie z. B. ein Luftballon. Für weitere Körper dieses Materials, wie bspw. Kunststoff, gilt, wenn ein gewisser Schwellenwert überschritten wird, tritt Plastizität ein. Ab diesem Punkt ist die Veränderung nicht reversibel. Für diese

gibt es den Fachterminus “Plastic Deformation”. Jener lässt sich durch Abwandlungen auch auf andere Materialien, wie Metall oder andere offensichtlich verformbare Stoffe, übertragen. Die Plastizität umfasst jedoch nur zu einem Bruchteil die Weichkörperdynamik.

2.2.2 Physikalische Grundlagen

Die Grundlage für elastische Körper bildet ein Teilgebiet der Physik. Dieses wird unter dem Namen Kontinuumsmechanik untersucht. In einer Simulation wird, wie im Absatz Grundlagen Games Engines erwähnt, ein vereinfachtes Modell der Welt angenommen, welches durch die Newtonschen Gesetze erweitert wird. Für die Betrachtung eines Punktes in der Simulation werden folgende Eigenschaften, ableitend aus dem zweiten Newtonschen Gesetz, verwendet: Position, Geschwindigkeit sowie Kraft als Vektor und Masse. Dies wird im Code wie folgend realisiert:

```
struct Punkt
{
    x = Postion;
    v = Geschwindigkeit;
    F = Kraft;
    m = Masse;
};
```

Das zweite Newtonsche Gesetz besagt:

$$F = m * a$$

Dabei wird in der Engine bei jedem Frame folgende Funktion aufgerufen, um die Position aller Punkte in allen Körpern berechnen zu können:

$$F = 0$$

$$F+ = F_{\text{GEngine}} + F_{\text{Kolisionskraft}}$$

$$v+ = F * \Delta t / m$$

$$x+ = v * \Delta t$$

Auf dieser Basis aufbauend kann nun eine elastische Verknüpfung der Objekte erfolgen.

2.2.3 Das Model Spring/Mass Model

Hierbei wird das vorhandene Modell der Engine mit der Eigenschaft einer Feder erweitert, um die Weichkörperdynamik zu implementieren. Dabei werden die Punkte im selben Mesh mit denen einer Feder verbunden, die folgende Eigenschaften aufweist:

```
struct Feder
{
    A, B = Punkte in der Welt;
    ks = Steifigkeit;
    L0 = Restlaenge;
    kd = Daempfungsfaktor;
};
```

Folglich werden alle Punkte, die logisch beieinander liegen mit Federn verbunden, damit die gewünschte Struktur und daraus die gewünschte Form erstellt werden kann. Darauf wird nun das Hookesche Gesetz angewendet. Eine Problematik, die aus der Berechnung entsteht, ist die harmonische Schwingung der Feder. Diese besagt, dass die Feder ohne eine Gegenkraft endlos weiterschwingt. Durch Zugabe eines Dämpfungsfaktors wird das Schwingen mit der Zeit ausgebremst und die Feder kommt zum Stillstand. Vorteilhaft bei dieser Darstellung ist die Steifigkeit der Feder, wodurch verschiedene Materialien dargestellt werden können. Dennoch ist diese Methode nicht für die Simulation aller Materialien geeignet, denn damit können nur Teile oder gewisse Umstände visualisiert werden. Beispielsweise wird hier die Plastizität der Objekte vernachlässigt. Im Rahmen einer Seil- oder Haar-Simulation führt es dazu, dass die Objekte beim Vorgang des Verbiegens in sich zusammenfallen, wenn zu wenig Punkte verwendet werden. Ein Vorteil ist jedoch die Vereinfachung der Problemkomplexität. Dementsprechend ist die Berechnung auf leistungsschwächeren CPUs auch möglich [13].

2.2.4 Particle Based Simulation

Jedes Objekt wird im Rahmen dieser Simulationsmethode mit Partikeln dargestellt und darin wird die Position, die Schnelligkeit, die inverse Masse, in denen sich das Partikel-Objekt befindet, gespeichert. Die Partikel liegen wie in der realen Welt aneinander und simulieren dadurch eine atomähnliche Struktur. Allerdings sind sie nicht explizit miteinander verbunden. Die Partikel werden wie folgt dargestellt:

```
struct Particle
{
    float pos[3];
    float vel[3];
    float invMas;
};
```

Durch die Abstraktion auf Partikel können verschiedene Areale in den Strukturen markiert werden und dadurch können andere Algorithmen für die zu simulierenden Eigenschaften verwendet werden. Das erweitert den Partikel um Nebenbedingungen wie Maximal- und Minimalwerte der Abstände für die Kleidungssimulation, eine Form für Festkörper oder aber auch die Dichte für die Simulation von Flüssigkeiten. Ebenso können diese auch global für das gesamte Objekt definiert werden und werden dann in dem zur Simulation verwendeten Algorithmus berücksichtigt [14]. Durch die Wahl einer solchen Repräsentation lässt sich die Berechnung einfacher auf eine GPU auslagern. Eine Methode, die basierend auf dieser Vorlage mehrere Eigenschaften simulieren kann, ist Position Based Dynamics. Diese Methode wird unter anderem bei dem Framework Nvidia FleX als Grundlage verwendet [15].

2.2.5 Finite Elemente-Methode

Die FE-Methode ist eine oft verwendete Simulation, um komplexe Objekte auf ihre Festigkeit und ihre Verformung hin zu berechnen. Sie findet vor allem im Bereich des Ingenieurwesens ihre Anwendung. Dabei wird das zu berechnende Objekt in eine endliche Anzahl von Teilkörpern zerlegt, in die finiten Elemente. Diese sind im zweidimensionalen Bereich ein Viereck oder Dreieck. In der dritten Dimension werden Tetraeder, Pentaeder, Pyramiden oder Hexaeder verwendet. Nachdem diese bestimmt worden sind, werden sie in Elementmatrizen bzw. Gleichungen überführt und beschreiben so die Unbekannten, die berechnet werden sollen. Hierbei handelt es sich um die Position und Geschwindigkeit der Gegenstände. Damit die Gleichungen gelöst werden können, werden alle Punkte der Teilkörper betrachtet und berechnet. Zur Berechnung des Körpers wird das Ergebnis aller Gleichungen zu einer Gesamtmatrix zusammengefasst. Wichtig hierbei ist zu bedenken, dass es sich um ein Näherungsverfahren handelt und dieses keine exakte Lösung liefern kann. Jedoch ist das Verfahren exakt genug, um es in der Kontinuumsmechanik als Hauptmethode zur Lösung dehnbarer Körper zu verwenden [16].

3 Aktuelle Frameworks

Im folgenden Abschnitt werden die aktuellen Erweiterungen, die ohne zusätzlichen Entwicklungsaufwand in der Unreal Engine verwendet werden können, näher beleuchtet. Dabei wird auf die Entwicklung dieser, die genaue Umsetzung sowie die Dokumentation der Implementation eingegangen.

3.1 Nvidia FleX

Nvidia FleX ist ein Framework, welches vom gleichnamigen Hersteller Nvidia Corporation [17] in der Software-Sammlung GameWorks [18] entwickelt wird. Kategorisiert wird es in VisualFX. Das letzte Update war Version 1.2 [19]. Basierend darauf wird ein Fork für die Integration gewartet, welche als Plugin die Unreal Engine erweitert. Die Teilnahme am Epic Games-Entwicklerprogramm ist ausreichend, um Zugriff auf das Plugin zu erhalten. Das Framework basiert auf einer, im Absatz Particle Based Simulation genannten, technischen Umsetzung und implementiert die Position Based Dynamics-Methode. Jene wurde im Rahmen der Forschung erweitert, um verschiedene Weichkörperdynamiken zu simulieren [15]. Im Zusatzmodul der Unreal Engine wird die Technik in standardisierte Spielobjekte eingebettet und durch Kopplungen an die Treibersoftware der verbauten Grafikkarten auf den Grafikeinheiten berechnet. Dadurch wird die Rechenlast von der CPU auf die GPU umgelegt. Um diese Berechnung möglich zu machen, wird auf die herstellerspezifischen Treiber zugegriffen. Dabei besteht eine Kompatibilität zwischen Nvidia, Intel und AMD-Grafikeinheiten. Grundvoraussetzung ist eine Grafikeinheit, die DirectX 11 unterstützt. Folglich ist diese in der Regel nicht für Cross Plattform-Entwicklungen geeignet.

Aktualität

Das Plugin wurde bis zur Unreal Engine 4.19 offiziell von Nvidia aktualisiert [20]. Auf Github sind inoffizielle Updates von Nutzern bis zur Version 4.21 zu finden[21]. Das Plugin wurde zum Stand der Analyse 36.000-fach geforkt. Das Verkünden des PhysX 5.0 Frameworks und die Aufnahme der Partikelsimulation lässt den Schluss nahe, dass die FleX-Erweiterung nicht mehr weiterentwickelt wird. Dies lässt sich auch mit dem letzten Commit auf Github (Stand: 2017) bestätigen.

Dokumentation

Die Dokumentation von Nvidia ist für das Plugin ausführlich und verständlich gestaltet. Dazu gibt es eine Ausführung für die Entwickler [22] und die Technical Artists [23]. Für beide Anwenderseiten sind jeweils Beispiele vorhanden. Als weitere Information liefert Nvidia eine Präsentation zu dem jeweiligen Plugin und stellt diese öffentlich zur Verfügung [14]. Beim Analysieren der Dokumentation ist folgendes für das Plugin der Unreal Engine aufgefallen: Dieses beinhaltet nicht die, wie in der Techdemo vorhandene, “Plastic Deformation” für Festkörper [24]. Außerdem wird darauf hingewiesen, dass das FleX-Plugin nicht als Hauptelement für das Gameplay verwendet werden sollte, sondern nur für sekundäre Effekte. Dies lässt sich auch in den Spielen, die diese Technologie beinhalten, feststellen. Hierbei ist das Framework im Allgemeinen gemeint, da dort Trigger-Events, Raycastings und weitere fehlen [25]. Stattdessen sollte auf das Physik-Framework mit der Rigidbody-Implementierung PhysX, welche auch in der Unreal Engine verwendet wird, gesetzt werden.

Verwendung in Spielen

Nvidia FleX hat die Besonderheit bereits in Spielen eine Verwendung gefunden zu haben. Die zwei bekanntesten Titel sind Killing Floor 2 und Fallout 4. Beide Titel bedienen sich verschiedener Techniken des Frameworks und binden diese in die Spielwelt mit ein. Killing Floor 2 ist ein Ego-Shooter, der im Survival Horror Genre angesiedelt ist. Daher ist es naheliegend, dass mit der Technologie entsprechende Effekte, wie Flüssigkeiten oder Objekte mit elastischen Eigenschaften, gewünscht sind. Da zum Zeitpunkt der Veröffentlichung des Spieles nur die Nvidia Grafikkarten-Generation unterstützt wurde, war diese Option nur für Besitzer dieser Karte vorbehalten. Ein entsprechender Test der Spiele-Redaktion "Gamestar" belegt die Leistungseinbußen des Frameworks. Im Test mit der tagesaktuellen Hardware wurden bis 30 FPS weniger gemessen [26]. Diese Erkenntnis bestätigt die Behauptung, dass die Ansätze für realistische Physikberechnungen mit älterer Hardware noch in der Entwicklungsphase stehen. Ähnliche Tests von Spielern bestätigen, dass sich die Leistungseinbußen in keiner Relation zum Spielerlebnis einordnen lassen. Im Gegensatz dazu hat Fallout 4 den Patch 1.3 FleX nur für einen bestimmten Partikeleffekt verwendet [27]. Dabei werden Partikel beim Aufprall von Projektilen in das Zielobjekt emittiert. Diese Partikel sind Objekte, die mit der virtuellen Welt interagieren und die Eigenschaften des getroffenen Materials besitzen. Dabei sind keine signifikanten Leistungseinbußen zu verzeichnen. Weitere Beispiele zur Verwendung des Frameworks wurden durch Videos auf den gängigen Plattformen dokumentiert.

3.2 AMD OpenGPU FEMFX

Der Hersteller Advanced Micro Devices Inc.[28] hat mit GPUOpen[29] ein Konkurrenzprodukt zu dem Framework von Nvidia veröffentlicht. Unter dem Namen GPUOpen veröffentlicht AMD Open Source Software, die der Erweiterung von Spiele Engines dienen soll. Das Framework FEMFX ist drei Jahre nach der Veröffentlichung von Nvidia FleX bekannt gegeben worden. Jedoch wurde im Gegensatz zu FleX nur die Version 0.1.0 veröffentlicht und weitere Updates blieben bis zur Analyse aus. Die FEMFX-Bibliothek beinhaltet die technische Umsetzung der FE-Methode, welche in Absatz 2.2.5 näher erläutert wurde. AMD setzt hier die Voraussetzung, dass die Berechnungen nur auf der neuen Ryzon-Technologie laufen sollen. Zumindest sollten hierbei leistungsstarke CPUs vorhanden sein, die auf dem Niveau des AMD Ryzen™ 7 2700X sind. Damit wird bewusst nur auf das Marktsegment der Computer gesetzt und Teile der Konsolen werden ausgeschlossen. Außerdem sind im Code eindeutig Marker für den Präprozessor im Source-Code gesetzt, sodass es nur auf einem Windows-Betriebssystem laufen kann. Gleichzeitig ist es Voraussetzung mit der Visual Studio IDE zu bauen. Von der Standalone-Version wurde, wie beim FleX-Framework, ein Fork verwendet, um daraus das Plugin für die Unreal Engine zu erstellen. Beim Plugin werden, ähnlich wie beim FleX-Plugin, Objekte in der Engine als FEM-Objekte gekennzeichnet und diese direkt auf der CPU berechnet. Damit dies bewerkstelligt werden kann, muss vorher das Objekt in dem 3D-Grafikbearbeitungsprogramm Houdini bearbeitet werden [30].

Aktualität

Zum aktuellen Zeitpunkt wurde das Plugin offiziell von AMD bis zur Unreal Version 4.18 herausgebracht [31]. Inoffizielle Ports sind bis Version 4.24 vorhanden [32]. Da zum Zeitpunkt der Analyse keine weiteren Updates gemacht worden sind und ein Großteil der für das Framework Verantwortlichen das Unternehmen verlassen hat, lässt es den Schluss nahe, dass es die letzte Version bleibt.

Dokumentation

Das Plugin besitzt eine ausreichende Dokumentation. Weitere Anwendungsbeispiele, z. B. wie die Objekte in der Unreal Engine angelegt werden müssen, werden nicht weiter erläutert und wurden nicht dokumentiert. Die Herausgeber des Plugins stufen das Verwenden der Erweiterung als schwierig ein [33]. Beispielprojekte, die in den Ankündigungen gezeigt worden sind, werden nicht mit Einstellungen in der Engine erläutert. Es wird nur ein Demo-Projekt zu Verfügung gestellt, welches jedoch lediglich einige Aspekte des gesamten Plugins abbildet.

Verwendung in Spielen

Außer den zwei Demo-Projekten wurden keine Spiele damit ausgestattet.

3.3 VICODynamics

Hierbei handelt es sich um ein Plugin des gleichnamigen Entwicklerstudios VICO [34]. Die Erweiterung ist wie das FEMFX-Framework nur für die Ausführung auf der CPU implementiert. In der Beschreibung wird bestätigt, dass es auf Konsolen laufen kann. Dies macht es im Vergleich zu den anderen zwei Erweiterungen einsatzfähiger und somit hebt sich die technische Hürde der Plattform auf. Das Plugin basiert auf der bereits in Absatz 2.2.4 erwähnten Particel Based Simulation. Ob die Implementierung gleich ist, steht offen, da der Code nur für Käufer des Plugins zugänglich ist. Die Anwendung der Erweiterung wird ähnlich gehandhabt wie beim Nvidia FleX-Framework. Einem Objekt in der Spiele Engine kann eine Komponente als Erweiterung hinzugefügt werden. Dadurch wird es gesondert auf der CPU berechnet. Im Gegensatz zu den Frameworks gibt es keine Standalone-Version des Plugins, daher ist es nur zusammen mit der Unreal Engine verwendbar.

Aktualität

Das Plugin kann im Epic Marketplace von der Version 4.12 bis zur Version 4.26 erworben werden. Das dahinter verborgene Studio antwortet auf dem Marketplace regelmäßig auf Fragen und ein aktiver Support der Käufer ist auch erkennbar [35].

Dokumentation

Das Hauptaugenmerk der Dokumentation liegt auf Videos. Vom Installieren des Plugins bis zum Erstellen des ersten Projektes ist es in Video- und teilweise in Textform vorhanden. Dies ist ausbaufähig. Dennoch ist es ausreichend, um damit arbeiten zu können. Zusätzlich gibt es ein Forum zum Austauschen, welches aktiv von Nutzern bespielt wird. Ein umfangreiches Demo-Projekt steht hier auch zu Verfügung.

Verwendung in Spielen

Zum Zeitpunkt der Recherche wurde noch kein Spiel damit veröffentlicht. Es ist jedoch anzumerken, dass in dem Forum Nutzer Videobeiträge veröffentlichen, bei denen das Plugin zum Einsatz kommt. Videospiel-Journalisten wurden auf ein Video aufmerksam, welches einen realistischen Sturm darstellt. Der Entwickler der Simulation gibt auch an dieses Plugin zu verwenden. Welche Komponenten genau verwendet werden, lässt sich jedoch nur vermuten. Im Video fliegen offensichtlich Kabel und ein Stück Stoff, welches vom Wind beeinflusst wird, herum [36].

3.4 PhysX 5.0

PhysX ist eine Physik-Engine, die vom Hersteller Nvidia entwickelt wurde. Sie wird von der Unreal Engine genutzt. Ab der Version 4.23 kündigte Nvidia PhysX 5.0 [37] an und Epic Games kündigte ebenfalls ein eigenes Physik-Framework für die Engine an. PhysX 5.0 ist jedoch nur in der Applikation Nvidia Omniverse verfügbar. Somit ist zu vermuten, dass ein Update in der Unreal Engine von beiden Seiten nicht mehr gewünscht ist. PhysX hat ab der Version 5.0 die FE-Methode, wie im Absatz 2.2.5 erwähnt, implementiert. Die Physik-Engine wird exklusiv in die Nvidia Omniverse App "Isaac Sim" eingebaut. Dies ist eine Nvidia exklusive Simulationssoftware. Sie lässt sich mit der Unreal Engine synchronisieren. Jedoch ist das Plugin, welches zur Verfügung gestellt wird, nur mit den bereits vorhandenen Systemen der Unreal Engine kompatibel. Damit wird ergo keine neuere PhysX-Komponente importiert. Das Hauptaugenmerk liegt hierbei auf dem kollaborativem Arbeiten [38].

3.5 Entscheidungsfindung

Wie der Tabelle 3.1 zu entnehmen ist, ist unter Berücksichtigung aller Rahmenbedingungen das Nvidia FleX-Plugin die einzige Alternative zu Erweiterung der Unreal Engine. Das Plugin von AMD hat keine ausreichende Dokumentation und ist bei einigen Testversuchen mit der Unreal Engine abgestürzt. Außerdem ist auch beim Testen aufgefallen, dass Objekte, die einen komplexen Aufbau besitzen, in dem Programm Houdini noch vorbereitet werden müssen. Dieses Programm ist nur in einer begrenzten Version für Artists frei erhältlich. Da die Objekte in Houdini erstellt werden müssen, können diese nicht in der Unreal Engine per Menü zu einem FEM-Objekt konvertiert werden. Dadurch ist das Framework nur eingeschränkt im Rahmen dieser Arbeit verwendbar bzw. seine Nutzung ist mit einem großen Mehraufwand verbunden. Dieses Framework entfällt deshalb als mögliches Plugin zum Umsetzen der geplanten Komponenten. Das Plugin von VICODynamics ist in der Dokumentation etwas besser aufgestellt als die Erweiterung von AMD. Jedoch entspricht eine Kaufbedingung nicht dem Ziel dieser Ausarbeitung und deshalb fungiert sie hierbei als Ausschlusskriterium. Das Nvidia-Framework hat durch seine Implementation und durch die mehrfache Verwendung in Spielen gezeigt, dass es auf den ersten Blick eine handliche Erweiterung der Unreal Engine darstellt. Ein weiterer Aspekt bei der Auswahl ist die Möglichkeit, die Objekte auf der GPU berechnen zu können, gewesen. Es ist außerdem zu ergänzen, dass die Dokumentation die meisten Informationen bereitgestellt hat, was das Arbeiten mit ihr am einfachsten und verständlichsten gestaltet hat.

Framework	Unterstützung Unreal Version	Umsetzung	Dokumentation
Nvidia FleX	4.19	Seile, Kleidung, Fest-/Weichkörper, Flüssigkeiten, Gase	Artist und Entwickler Dokumentation, Source Code einsehbar, Beispiel Projekt
AMD FEMFX	4.18	Kleidung, Frakturen, Weichkörper und Plastic Deformation	Erste Schritte-Dokumentation vorhanden, Source Code einsehbar, Beispiel Projekt
VICODynamics	4.12 - 4.26	Kleidung, Seile, Weichkörper	Dokumentation anhand Beispiele Projektes, Source Code, Videos
Nvidia PhysX 5.0	nicht in Unreal verwendbar	Textilien, Gase, Weichkörper, Kleidung	Videos, umfangreiche Dokumentation

Tabelle 3.1: Übersicht Frameworks

4 Konzeption und Implementierung

4.1 Überlegung

Um einen aussagekräftigen Vergleich ziehen zu können, werden die oben aufgeführten Möglichkeiten nun in der Unreal Engine sowie dem FleX-Plugin evaluiert. Danach sollen diese in einem Anwendungsfall demonstriert werden und im Anschluss bewertet werden. Im Rahmen dessen sollen alle klassischen Fälle abgedeckt werden, um auch einen realistischen Vergleich ermöglichen zu können.

4.2 Native Unreal Implementation

Im folgenden Abschnitt werden die einzelnen Komponenten der Unreal Engine aufgeführt und analysiert. Dabei soll versucht werden Beispielobjekte möglichst realitätsgerecht abzubilden. Das erste Beispiel soll die Erstellung eines Seiles sein.

4.2.1 Erstellung eines Seiles

Wird ein Seil genauer betrachtet, weist es die Eigenschaft auf sich aufgrund seiner Dicke verschieden stark zu verbiegen. Je dicker das Material ist desto schwerer lässt es sich verformen. Weitere Aspekte, die zu bedenken sind, sind das Material, woraus die Reißfestigkeit resultiert, sowie dessen Aufbauschema. Für die Simulation eines Seiles können zwei Wege in der Unreal Engine verwendet werden. Für die erste Methode muss die Erweiterung der Kabel-Komponente aktiviert sein. Diese lässt sich in den Plugins der Engine aktivieren. Ein Seil besitzt einen Start- und Endpunkt. Dieser kann jeweils an einen Actor gekoppelt oder manuell auf einen Punkt gesetzt werden. Unter den Standardeinstellungen reagiert das Seil mit keinen anderen Spieleobjekten, außer mit den genannten Fixpunkten. Um das realistische Verhalten eines Seils zu simulieren, wird auf die Eigenschaft der Kollision gesetzt. Genauer betrachtet werden zwischen den zwei Fixpunkten Partikel platziert, welche mit der Verlet-Integration berechnet werden und zusammenhängend eine Kette bilden. Die Positionen der Partikel bilden eine Art Bindeglieder, an denen das Mesh herumgeschlagen wird. Je weniger der Partikel vorhanden sind, desto ungenauer werden die Verformungen. Da aus designtechnischer Sicht bewusst auf das Simulieren von physikalischen Abhängigkeiten verzichtet wurde, können Objekte nur als Kind-Objekt ohne eine tatsächliche Masse an das Seil angehängt werden. Daraus resultiert die ungenügende Eigenständigkeit der Komponente. Schließlich interagiert jedes Objekt, welches damit gekoppelt wurde, mit dem Seil, kann es aber nicht beeinflussen. Also ergibt sich der Schluss, dass bei Kollision mit dem Kind-Objekt auch das Seil nicht beeinflusst wird. Aus diesem Grund ist es als reines Dekorationsobjekt zu betrachten. Die daraus resultierende Problematik kann durch das eigenständige Ergänzen durch Code kaschiert werden. Eine weitere Möglichkeit besteht darin, die Kabelkomponente als dekorative Sache zu verwenden und die eigentliche Physik durch einen “Physikal Constraint Actor” zu erzeugen. Ein “Physikal Constraint Actor” ist ein Modul, mit Hilfe dessen sich die physikalischen Abhängigkeiten zweier Akteure genauer bestimmen lassen. Darunter fallen Optionen wie die Schwingungen jeder Koordinatenachse und wie sich diese mit einer Weichkörperdynamik verhalten sollen [39] [40]. (Ergebnis: 7.5b)

4.2.2 Erstellung von Kleidung

In der Unreal Engine ist Kleidung nur als Skeletal Mesh-Objekt verwendbar. Hierzu wird der "NVIDIA's NvCloth Solver" genutzt, der in die Engine implementiert worden ist. Nach der standardisierten Vorgehensweise wird ein Objekt in die Unreal Engine importiert, welches Textilien als Mesh aufweist. Dieses muss als Skeletal Mesh beim Import erkannt werden oder als ein solches markiert sein. Im Mesh-Editor kann für die Kleidung dann ein Verhalten definiert und ein Stoff zugewiesen werden. Jedoch interagiert die Kleidung nicht mit anderen Meshes, sondern benötigt Kollisionsboxen, welche separat für das importierte Objekt oder für die Umgebung gesetzt werden müssen [41]. (Ergebnis: 7.7a)

4.2.3 Erstellung von Haaren

Seit dem Update 4.26 kann auf die Groom-Erweiterung in der Unreal Engine zurückgegriffen werden. Dabei werden in einem externen 3D-Grafikprogramm, die Haare erstellt, bearbeitet und danach in die Engine importiert. Die notwendige Physik wird automatisch durch die Engine berechnet. Aufgrund des hohen Bedarfs an Rechenleistung wird auf die Strähnen-Methodik zurückgegriffen. Dabei werden einzelne Haare in Gruppen gebündelt und als ein Mesh dargestellt. Bei genauer Betrachtung wird die Gruppe als Skeletal Mesh importiert und darauf wird dann die Kleidungsoption angewendet. Zur realistischen Simulation werden die Haare mit besonderen Eigenschaften versehen, die einem sehr leichten Stoff ähneln [42]. (Ergebnis: 7.4)

4.2.4 Erstellung eines vorgetäuschten Weichkörpers

Da die Unreal Engine keine Implementierung für eine vollständige Weichkörpersimulierung aufweist, kann diese, wie in Absatz 2.2 aufgezeigt, nur nachgeahmt oder mit Deformierung per Code umgesetzt werden. Um ein möglichst realistisches Fake Soft Body-Objekt zu erstellen, sollte ein rundliches Objekt verwendet werden. Dieses wird als Blueprint angelegt und mit einer vereinfachten Lerp-Funktion periodisch beim Aufprall mit dem Gegenspieler verformt. Als Hilfe wird eine Timer-Funktion herangezogen, welche zum Bewegen der Skalierung verwendet wird (siehe Abbildung 4.1). Hierbei wird aber die weitere Physik des Objektes nicht berücksichtigt. (Ergebnis: 7.6c)

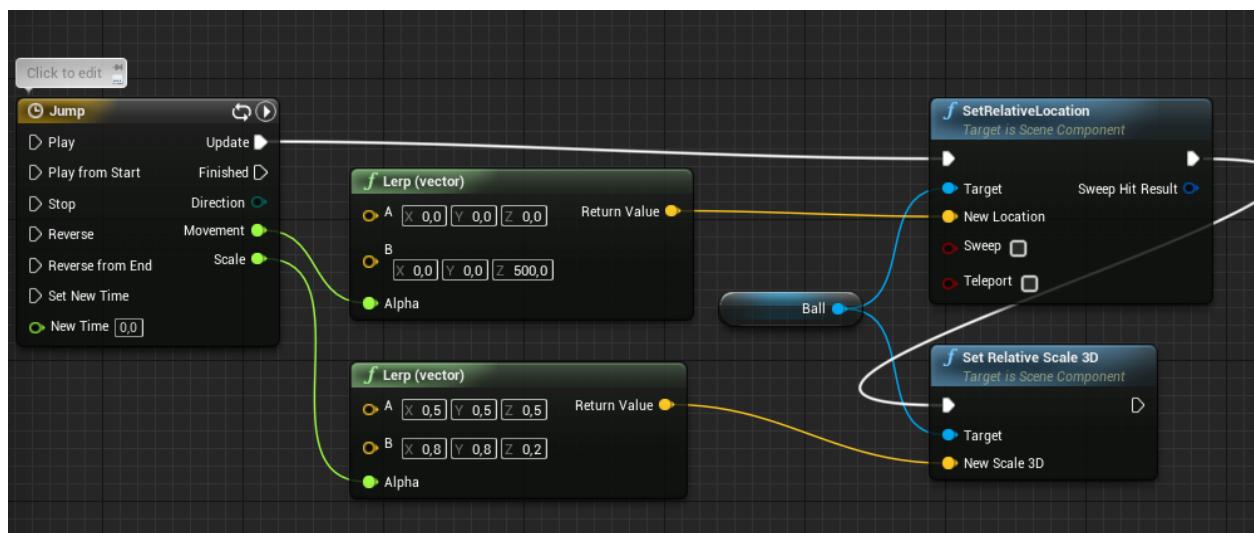


Abbildung 4.1: Visualcode zum Deformieren von Objekten

Diese Funktionalität kann durch einen größeren Kollider als das Objekt selbst erweitert werden. Dazu wird kurz vor dem Aufprall eine Funktion mittels Kolliderboxen mit einem Gegenspieler aufgerufen, anhand derer sich mit der entsprechenden Vektor-Transformation die Skalierung des Objektes verändert lässt.

Eine andere Methode ist das "Material Vertex Displacement". Dabei wird dem Objekt ein Material zugewiesen, welches die Material-Funktion "DistanceToNearestSurface" besitzt. Als Beispielobjekt wird ein Ball verwendet. Es werden hierbei weitere Oberflächen in der Nähe des Objektes gesucht und deren Richtung wird in der virtuellen Welt ebenfalls berücksichtigt. Das Ergebnis wird dann mit der "World Displacement"-Node auf das Objekt angewandt. Dabei werden nur die Vertices des Objektes beeinflusst, die Kollisionsbox des Objektes bleibt gleich (siehe 4.2). Ebenfalls wird ein Physik-Material zugewiesen, welches die Eigenschaften des Ball besitzt. Dieses sieht wie folgt aus: Resituation 0.8, Friction 0.2. Dadurch erhält das runde Objekt, folglich der Ball, seine Eigenschaft des Hüpfens. Die Voraussetzung damit diese Technik funktioniert ist die Option "Generate Mesh Distance Fields". Diese muss in den Projekteinstellungen aktiviert werden. Ein weiterer Schritt ist es bei dem Material die Option "Tessellation" zu setzen. (Ergebnis: 7.6b)

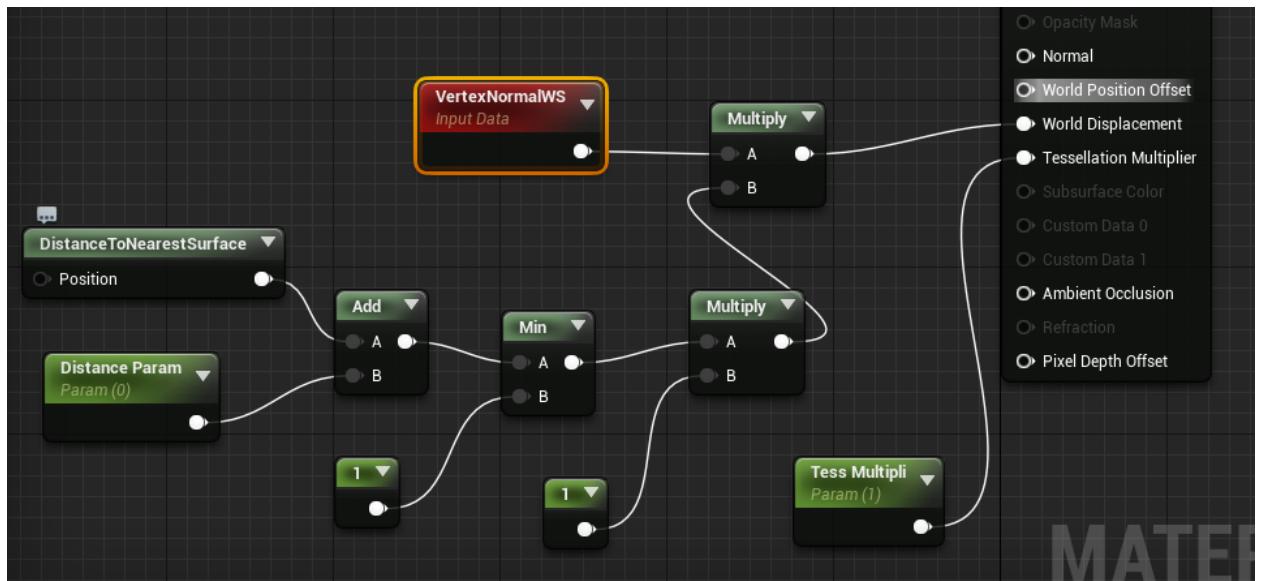


Abbildung 4.2: Material Visualcode zum erstellen des Tessellation-Effektes

4.3 Nvidia FleX Implementation

Im folgenden Abschnitt werden die genannten Spieleobjekte mit der Nvidia Flex umgesetzt.

4.3.1 Erstellung eines Seiles

Zum Erstellen eines Seiles stehen unter FleX zwei Möglichkeiten zur Verfügung: Zum einen die Kabel-Komponente, die bereits nativ in Unreal implementiert ist. Diese ist als externer Actor in die Umgebung zu importieren. Oder aber ein Mesh, welches ein Seil darstellt und mit der Soft Body-Komponente versehen wird.

Kabel Komponente

Hierbei wird die Kabel-Komponente, wie bei der nativen Vorgehensweise, eingefügt. Das Kabel, welches unter der Unreal Engine simuliert wurde, ist nun mit dem Nvidia FleX-Plugin ersetzt worden. Die Eigenschaften bleiben dieselben wie bei der nativen Implementation [43]. (Ergebnis: [7.5c](#))

FleX-Container

Es wird ein Static Mesh, welches einen Zylinder darstellt, in die Länge gezogen und dieses soll als Seil fungieren. Nach dem Konvertieren in ein FleX-Objekt wird dem Seil der Soft-Container zugewiesen. Als FleX-Eigenschaften werden die Werte "Cluster Spacing: 5, Cluster Raduis: 10 sowie Particel Spacing: 5" gesetzt.

4.3.2 Erstellung von Kleidung

Zur Darstellung von Kleidung wird nur ein Static Mesh benötigt. Dieses wird, wie jedes andere FleX-Objekt, dahin konvertiert. Dabei ist zu beachten, dass hierbei die Container-Option "Cloth" gesetzt wird. Wichtig ist ein Material hinzuzufügen, welches die Option von einer doppelseitigen Texturierung aufweist, da sonst nur eine Seite angezeigt wird [44]. (Ergebnis: [7.7b](#))

4.3.3 Erstellung von Haaren

Hierbei werden den Haaren die Containereigenschaft "Soft" hinzugefügt und weitere Einstellungen werden gesetzt. Ein wichtiger Wert hierbei ist die "Stiffness" der Haare. Dieser Wert beträgt bei diesem Beispiel 0.1, damit sie wie echte Haare locker fallen. Als Vorführobjekt wird ein Satz langer Haare verwendet (Ergebnis: [7.3b](#)).

4.3.4 Erstellung eines Weichkörpers

Wie bei der Kleidung wird für einen Weichkörper, der z. B. ein organisches Objekt simulieren soll, ein Static Mesh benötigt. Dies wird in ein FleX-Objekt umgewandelt und erhält die Container-Option "Soft". Nach dem Zuweisen eines FleX-Materials erhält man einen Weichkörper, der mit allen anderen Objekten in der Welt interagieren kann [44]. Als Beispielsobjekt wird hier eine Sphere verwendet. Das Material soll einen Fußball simulieren. Es werden hierfür keine weiteren Einstellungen verändert. (Ergebnis: [7.6a](#))



Abbildung 4.3: Charakter "Rampage" mit markierten Arealen

4.4 Anwendung an einem Charakter

Zum besseren Verständnis soll nachfolgend der Beispielcharakter "Rampage" [45] verwendet werden. Er ist aus dem Spiel Paragon. Dieses wurde von Epic Games entwickelt und wird nicht mehr aktualisiert. Charaktere und Objekte dieses Spiels gibt es für die experimentelle Verwendung im entsprechenden Marketplace. "Rampage" besitzt in seiner Standardausführung ausreichend Eigenschaften, die mit einer Soft Body-Mechanik ausgestattet werden können. Bei genauerer Betrachtung und in der nachfolgenden Ausarbeitung wird auf die muskulöse Struktur, seine Kleidung sowie seine kosmetischen Details eingegangen. Diese sind im Bild 4.3 gelb markiert.

4.4.1 Vorgehensweise bei einem Charakter

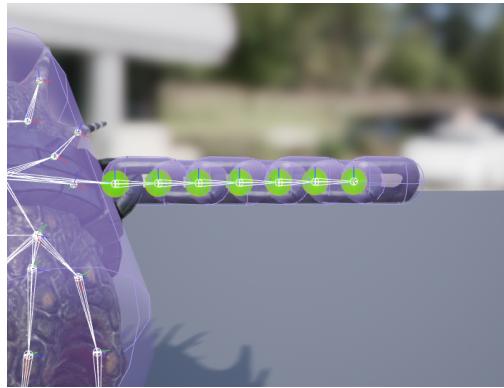
Folgende Vorgehensweisen sind geplant, um einen Charakter vollständig oder nur teilweise als Softbody zu simulieren: Zunächst sollen die bereits vorgestellten Varianten der Unreal Engine aufgezeigt werden. An dieser Stelle wird auf die typische Vorgehensweise in einer Game Engine zurückgegriffen. Dabei wird der gesamte Charakter als Skeletal Mesh gehandhabt und es werden alle Maßnahmen ausgeschöpft, um diesen so realistisch wie möglich zu gestalten.

Herangehensweise bezüglich verschiedene Areale

Wie das Bild 4.3 zeigt, sind verschiedene Areale am Charakter zu erkennen. Eine Herangehensweise ist, die einzelnen Teile, die eine andere Eigenschaft aufweisen, abzutrennen und als Kind-Objekt an den Hauptkörper anzubringen. Eine weitere Überlegung ist, nur vorher definierte Areale zu exportieren, diese vom Körper abzuschneiden und mit der FleX-Technologie auszustatten. Im Nachhinein sollen sie wieder an den Charakter angefügt werden. Dadurch wird der Mehraufwand für das Berechnen des kompletten Körpers minimiert. In diesem Fall erhält man einen festen Körper, der nicht, wie in der vorherigen Überlegung, mit einem Skelett oder ähnlichen Methoden gestützt werden muss.

4.4.2 Implementierung eines Charakters nur mit Techniken aus der Unreal Engine

Nachfolgend wird nun der Charakter, mit den verfügbaren Möglichkeiten der Unreal Engine erstellt und in die Bereiche von Muskeln, Kette und Kleidung unterteilt. Es wird dabei auf die gängigsten Methoden gesetzt. Demzufolge ist das Ziel ein möglichst effizientes Ergebnis, welches der typischen Vorgehensweise in der Unreal Engine entspricht, zu erhalten. Die Spielfigur "Rampage" besitzt, wie im Bild 4.3 zu erkennen, eine Kette. Sie kann am



(a) Endergebnis im -physik Editor der Kette.



(b) Endergebnis im Physik-Editor der Muskel.

Abbildung 4.4: Der Charakter im Physik-Editor

naheliegendsten mit der Kabel-Komponente umgesetzt werden, da die Kette genau in das Einsatzgebiet dieser Erweiterung fällt. Gleichzeitig kann sie auch durch ein Skeletal Mesh mit Physic Bodys simuliert werden, zumal am Skelett dafür Knochen gesetzt worden sind. Hintergrund hierbei ist, auf Außeneinwirkungen eingehen zu können. Technisch liegt nur die reine Animation mit Keyframe auch im Bereich des machbaren. Dennoch würde die Umgebung hier nicht berücksichtigt werden. Zur Veranschaulichung dient folgendes Beispiel: Es wird angenommen, der Charakter stützt seine Hände auf dem Boden ab. Simultan liegt die Kette in der Animation auf gleicher Höhe wie die Hand. Diese Position berücksichtigt dagegen nicht die Umgebung, in welcher die Kette ggf. mit einem anderen Objekt kollidiert oder in die Luft fliegt. Das Problem mit dem Schweben lässt sich durch inverse Kinematics bis zu einem gewissen Grad beheben. Die Kleidung kann entweder durch weitere Knochen im Skeletal Mesh simuliert werden oder durch Verwendung des Nvidia Cloth-Plugin bewerkstelligt werden. Für die Muskeln kann die Technik der Morph Targets in Betracht kommen. So können mehrere Zustände modelliert werden, die beim Abspielen der Animation die entsprechende Form ändern oder bei Kollision mit diesen die vordefinierte Position einnehmen. Dies kann mit Hilfe von Blend Spaces und dem Animator in der Unreal Engine für die gewissen Bereiche umgesetzt werden. Andernfalls bieten sich die bereits vorhandenen Knochen für die Erstellung der jeweiligen Muskelareale an.

Simulation der Kette

Durch Importieren des Charakters aus dem Marketplace besitzt dieser bereits vorgefertigte Assets. Dadurch ist auch eine Kollisionskomponente erstellt worden. In der Regel wird beim Importieren eines Objektes immer eine solche mit Knochen erstellt. Dabei erzeugt die Unreal Engine mit einem Algorithmus um das Mesh einen Kollisionskörper, welcher als physikalische Komponente mit der simulierten Umgebung interagiert. Da durch die Vorlage die Kette des Charakters bereits mit Knochen bestückt worden ist, kann diese infolgedessen bereits zur einer Simulation verwendet werden. Andernfalls müssten bei dieser Vorgehensweise durch ein separates 3D-Bearbeitungsprogramm weitere Knochen in den Charakter platziert werden. Um die Kette mit der Skeleton Mesh-Technik zu simulieren, müssen die noch benötigten Physical Bodies erzeugt werden. Diese lassen sich über die ausgewählten Knochen erzeugen. Danach werden die Bodies nachjustiert und somit ist dieser Schritt abgeschlossen (siehe 4.4a). Die Körper werden anschließend mit den Knochen verbunden und erhalten alle eine Verknüpfung, welche näher definiert werden kann. Jene lässt sich in den Knochen in

Abhängigkeit von einem Physical Constraint untergliedern, welche sich wiederum in Abhängigkeit zum Physical Body untergliedern lässt. Damit ein realistisches Verhalten der Kette simuliert werden kann, wird beim Physical Body jedes Kettengliedes die Simulationsart auf "Simulation" gestellt, da sie sich sonst nach den Vorgaben des Animationskontroller verhält. Weitere Werte wie "Linear Damping" sind auf den Wert 5 und "Angular Damping" auf den Wert 20 zu setzen. Diese Einstellungen verhindern das starke Beschleunigen der Kette beim Abspielen der Animationen. Die Knochen werden nämlich bei zu hohen Geschwindigkeiten voneinander getrennt und die Kette zerfällt für den Zeitraum der zu schnellen Bewegung in ihre Einzelteile. Abschließend muss, damit die Physik-Komponente im Spiel verwendet werden kann, im Charakter-Blueprint eine Funktion implementiert werden. Diese wurde aufgrund der Einfachheit als "Blueprint Visual Script" hinterlegt. Beim Spawning des Charakters soll in der Kette ab dem ersten Knochen die simulierte Physik aktiviert werden (siehe Abbildung 4.5).

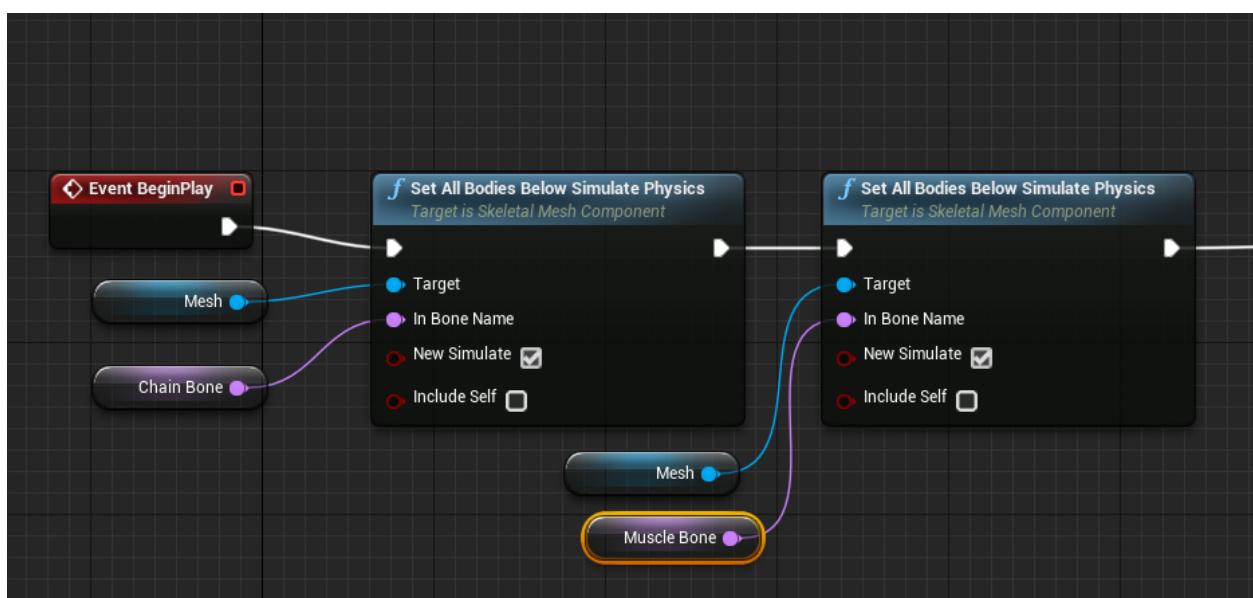


Abbildung 4.5: Aktivieren der simulierten Knochen im Code

Simulation der Muskeln

Eine ähnliche Vorgehensweise findet ebenso bei der Simulation der Muskeln oder größeren Gewebeteilen statt. Zugleich werden an den Knochen, wie in Abbildung 4.4b zu sehen ist, an der Brust des Charakters gleich große Physikkörper erstellt. Diese werden mit Einschränkungen versehen. Im Englischen werden sie "Contains" genannt. Sie lassen sich von der organischen Struktur eines Muskels ableiten. Sie lauten z. B. für den Bizeps "Motion: Locked, Swing 2 Motion Limit: 8 und Twist Limit: 15". Damit wird das Verhalten der Physik-Komponenten begrenzt und das dynamische Mesh des Charakters verhält sich nicht unnatürlich. Dieser Vorgang wird für alle weiteren Areale wiederholt mit angepassten Einstellungen. Dabei ist zu beachten, dass die Physik-Komponenten nicht größer als das eigentliche Mesh sein darf. Sonst würden die Physik-Kollider bei Bewegungen miteinander kollidieren und ein unerwünschtes Verhalten hervorrufen.

Simulation der Kleidung

Für die Simulation von Kleidung muss, wie bereits in Absatz 4.2.2 beschrieben, das Kleidungsstück selektiert und entsprechend umgewandelt werden. Dabei wird, wie in Abbildung

4.6 zu sehen ist, nicht das gesamte Kleidungsstück mit einer hohen Simulationsquote markiert, sondern die Ränder am Übergang der Rüstung bleiben statisch mit einem weichen Übergang zur Kleidungssimulation. Die Abstufung der Farbe pink zu weiß symbolisiert die Ausfallstärke. Die Stärke bei diesem Kleidungsstück ist hierbei mit 60 Prozent bemessen worden, da dieses Kleidungsstück nur einen kleinen Teil des Gesamtbildes des Charakters darstellt. Bei Verwendung höherer Werte würde der Stoff stärker von der Bewegung des Charakters und vom Wind beeinflusst werden und es würde so aussehen, als ob es ein leichtes Seidentuch wäre. Um das zu überprüfen, wurde im Physik-Editor Wind simuliert und das beschriebene Verhalten auch festgestellt. Infolgedessen kann Performance dadurch gespart werden, dass nicht jede Strukturfaser im Stoff berechnet werden muss.



Abbildung 4.6: Konfiguriertes Kleidungsstück des Charakters "Rampage"

4.4.3 Implementierung mit Hilfe des FleX Plugins

Das typische Vorgehen in der Spieleentwicklung sieht es vor den Charakter in ein Skeletal Mesh umzuwandeln. Dabei ist zu beachten, dass jener vorher in einer 3D-Grafik-Software mit einem Skelett (Rig) versehen werden sollte, um diesen auch animieren zu können. Vor teilshalber wurde dieser Schritt im verwendeten Template bereits erledigt als dieser aus dem Marketplace importiert worden ist. Damit der beabsichtigte Effekt eines Weichkörpers erzielt werden kann, muss nun der "gerigge" Charakter mit deinem Nvidia FleX-Mesh ersetzt werden. Dies ist nicht möglich, da in der Unreal Engine die Skeleton Meshes mit einem Dynamic Mesh dargestellt werden. Ursächlich hierfür ist das Skelett, welches an ein Static Mesh gekoppelt ist und dadurch dynamisch ist. Dies lässt sich nur durch Umprogrammierung der Unreal Engine umbauen. Deshalb bleibt für dieses Problem nur die Möglichkeit den Softbody an ein bereits bestehendes System zu koppeln und das bestehende System in der Spielewelt unsichtbar zu machen. Zuvor wurde bereits aufgezeigt, dass ein Static Mesh in ein FleX-Mesh verwandelt werden kann. Deshalb wird erstes nun angelegt. Der Charakter "Rampage" muss erst aus dem Template exportiert werden und in einer 3D-Grafik-Software, in diesem Fall Blender, bearbeitet werden. Das Zielformat wird von Unreal vorgegeben. Dies ist eine FBX-Datei. Darin sind zusätzliche Informationen, wie z. B. Knochen im Spielobjekt definiert sind, vorhanden, welche für die Umwandlung in ein Static Mesh nicht benötigt werden. Diese müssen manuell aus dem Charakter entfernt werden, um diesen dann erneut als FBX zu exportieren. Ansonsten wird beim erneuten Import das Objekt als Skeletal Mesh erkannt. Dabei ist zu beachten, dass die Skalierung des Exportes mit der Maßeinheit der Unreal Engine übereinstimmen muss. Nach dem erneuten Importieren kann der Charakter in



Abbildung 4.7: Charakter "Rampage" als FleX Static Mesh-Objekt

ein Flex-Objekt umgewandelt werden. Dazu wird ein Flex-Material erstellt und mit folgenden Materialeigenschaften versehen, um es an einen organischen Körper nachzuempfinden:

Einstellung	Wert
Particel Spacing	6
Surface Sampling	0
Cluster Sampling	10.0
Cluster Radius	19.0

Tabelle 4.1: FleX-Einstellungen des Charakters

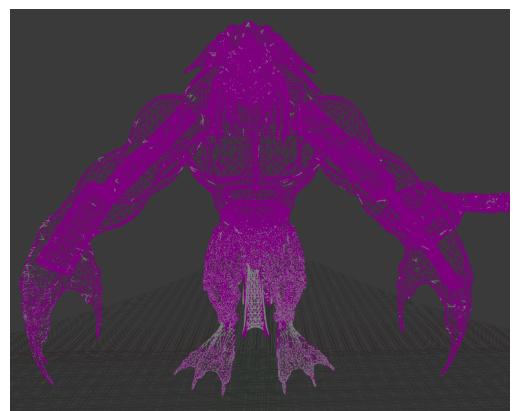
Eine weitere Eigenschaft, die essentiell markiert werden soll, ist "Attach to Ridgets". Dabei wird das Objekt an die umliegenden Objekte gekoppelt. Dies greift zugleich die zuvor angedachte Problemlösung auf. Der FleX-Körper sollte denselben Nullpunkt in der Welt aufweisen, wie das Skeleton Mesh, welches platziert wurde, ansonsten greifen die Physical Bodys nicht. Dadurch sollte der FleX-Körper von diesen mitgezogen werden.

Nachahmung der Physical Bodys

Da die Kollisionskörper nicht den gesamten Weichkörper beeinflussen, müssen an die entsprechenden Knochen des Skelettes weitere dieser Kollisionkörper angebracht werden. Bevor dieser Arbeitsschritt erledigt wird, sollte ein Testversuch durchgeführt werden, der hervorbringen sollte, dass die Physical Bodies eine Auswirkung auf das Static Mesh FleX-Objekt haben. Es zeigte sich, dass diese durch den Körper hindurch gleiten und ihn nur kurz beeinflussen. Die Physical Bodies sind entfernt worden, zumal diese ein unerwünschtes Verhalten des Charakters provozieren. Wie bereits erwähnt wurde, wurde bei dem FleX-Objekt die Option "Attach To Ridgets" ausgewählt. Daher liegt die Überlegung nahe Ridgetbodys als Skelett zu verwenden. So ist das weitere Vorgehen an den entsprechenden Knochen des Skelettes per Anhang weitere Objekte anzubringen. Diese müssen mit der Option "Hide on Play" versehen werden, da sie sonst während der Simulation den gesamten Charakter verdecken. Hierbei werden die Ridgetbodys als primitive Objekte in den FleX-Körper eingesetzt, sodass dieser sich daran befestigen kann. Die Funktion der Sockets der Skeletal Meshes wird dazu verwendet. Zeitgleich werden die primitiven Objekte an die Knochen-Ansätze gehaftet und durch die Animation bewegt. Es werden zur Erstellung zwei Varianten verwendet, welche nachfolgend im Punkt Bewertung näher erörtert werden: Einmal die Möglichkeit mit größeren primitiven Objekten (siehe Abbildung 4.8a). Dabei heften sich alle Partikel in dem



(a) Knochen um den simulierten Weichkörper



(b) Knochen im simulierten Weichkörper

Abbildung 4.8: Charakter mit Knochen im und um den simulierten Weichkörper

FleX-Mesh an dem Objekt fest und werden als gesamte Einheit bewegt. Als zweite Variante werden kleine primitive Objekte als knochenähnliche Struktur in den FleX-Körper eingebettet (siehe Abbildung 4.8b).

LOD Loading

Ein weiteres Problem, welches beim Umsetzen auftritt, ist, dass der Charakter beim Verlassen einer festen Distanz ausgeblendet wird. Dies röhrt aus der automatischen Ersetzung der LODS im Unreal-System. Beheben lässt sich dies durch den fixen Wert der LOD-Anzahl. Dieser beträgt Eins. Das Nvidia FleX-Plugin für Unreal zeigt hier seine Schwächen und lässt keine dynamische Ersetzbarkeit von Objekten zu.

FleX Material

Beim Simulieren des Charakters fällt auf, dass die Textur der LODs nicht geladen wird. Da der Charakter auf Basis des Standard Static Meshes erstellt worden ist, wurden die Texturen aus diesem übernommen. Deshalb müssen die Texturen mit den Parameter “Used with FleX Texturing” erneut angelegt werden. Andernfalls wird das standardisierte voreingestellte Material der Unreal Engine geladen.

Blueprint Code

Die Problematik des noch sichtbaren Meshes des Skelettes kann gelöst werden, indem die gesamte Struktur in ein Blueprint umgewandelt wird. Durch die Option “Hide on Play” wird das Skelett zwar ausgeblendet, aber es spielt nicht die Animation des Animationskontrollers ab. Dies lässt sich mit einem eigenen Script lösen. Außerdem lassen sich die Komponenten nicht per Code lokal ansteuern. So kann der Charakter z. B. nicht ohne großen Aufwand zu betreiben als NPC oder spielbaren Charakter verwendet werden. Im folgenden Bild 4.9 wird das genannte Problem per Visual Scripting gelöst. Dabei wird beim Spawning des Blueprints beim Charakter durch ein Event ein Signal ausgelöst, welches dieses startet. Es wird mit der Funktion “set Visibility” und dem Eingabeparameter des Skelet Meshes unsichtbar gemacht. Als nachfolgende Funktion wird der Animationskontroller, welcher dem Skeletal Mesh zugewiesen wurde, abgerufen und durch einen Cast spezifiziert. Damit wurde nun der Zugriff auf die Klassenvariablen geschaffen. In dieser ist nämlich eine Variable, die als Parameter für ein

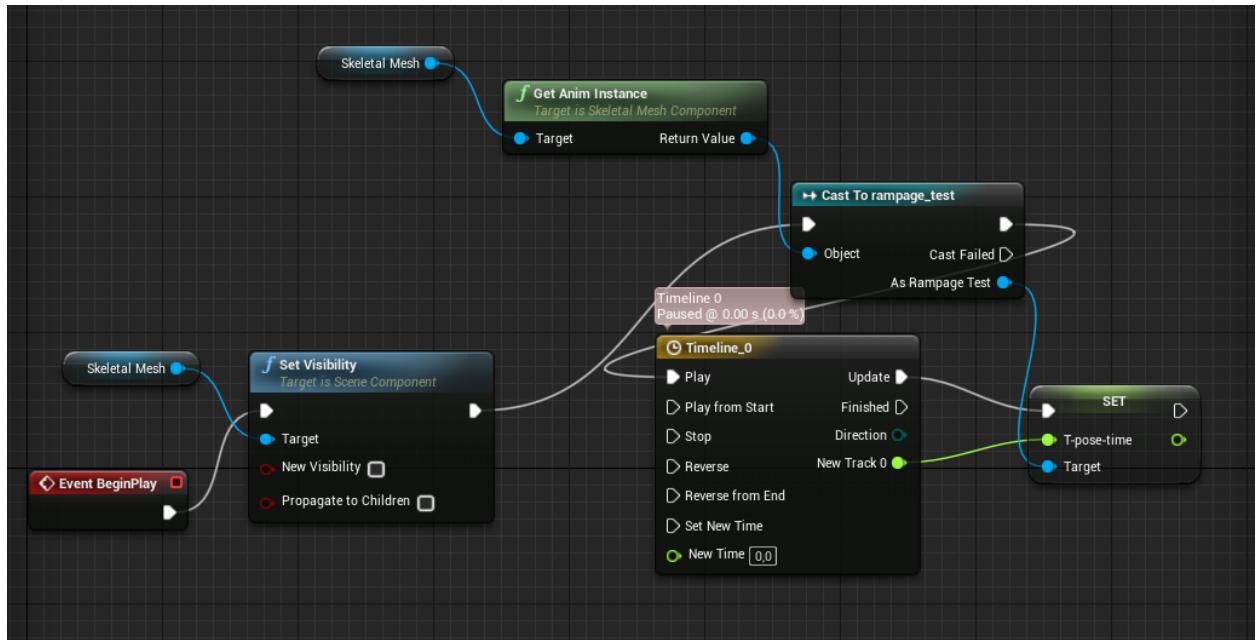


Abbildung 4.9: Code zum beheben auftretender Probleme beim FleX-Objekt

Blend Space der Animation dient, vorhanden. Das FleX-Objekt befindet sich in der Startposition namens T-Pose. Der bereits mitgelieferte Animationskontroller lädt beim Starten der Simulation die erste Pose, welche eine Idle-Animation ist. Die Animation besitzt allerdings für die Positionen der Knochen andere Keyframes als die Startpose des FleX-Meshes. Deswegen startet das Skelett an einer anderen Stelle und das FleX-Objekt hängt sich an die falsche Stelle. Somit wird ein Blendspace von der T-Pose zu der Ausgangsanimation angelegt, welcher über die bereits genannte Variable gesteuert wird.

4.4.4 Separierung der kosmetischen Objekte

Beim Separieren der kosmetischen Objekte wird wie folgt vorgegangen: Es werden die Meshes in einer 3D-Grafik-Software, die mit dem gewünschten Material versehen werden sollen, abgesondert. Dafür ist die Vorgehensweise wie im vorherigen Absatz erläutert. Die notwendigen Areale werden vom Charakter getrennt und als separate Objekte in Unreal importiert. Alle Objekte werden in FleX-Objekte umgewandelt und in die gleichen Stellen des Blueprints eingefügt.

Simulation der Kette

Es werden zwei Möglichkeiten in Betracht gezogen, um dies zu bewerkstelligen: Bei der ersten Variante wird die gesamte Handfessel abgetrennt, mit folgenden Werten "Cluster Stiffness 0.1" versehen und in den Blueprint an die exakte Position gesetzt, wo sie vorher war. Als zweite Option besteht die Möglichkeit, die Kette von der Handfessel abzutrennen und dabei aber in Kauf zu nehmen, dass sich die Handfessel nicht mehr bewegen lässt. Nach dem Platzieren des Blueprints fällt auf, dass nach Initialisierung der Kette in der Welt alle FleX-Objekte, die als Anhang an das Hauptobjekt angekoppelt wurden, doch nicht daran befestigt sind. Diese fallen ohne Kollision durch die Gravitation ab. Dieses Problem taucht nicht auf, wenn die Objekte separat in die Spielwelt platziert werden. Dafür muss im Blueprint an der entsprechenden Verbindungsstelle ein primitives Objekt an einen Sockel des Knochens platziert werden, damit dieses auch durch die Animation beeinflusst wird und

das kosmetische Objekt mitzieht.

Simulation der Kleidung

Es hat sich bereits gezeigt, dass das gleiche Verfahren auf die Kleidung angewendet werden kann. Folgende Einstellung wird verwendet, um den Stoff realitätsnah darzustellen: "Tearing Enabled". Wichtig ist beim Material, welches der Kleidung zugewiesen wird, dass es mit der Option "Double Sided" markiert wird, andernfalls ist eine Seite der Kleidung transparent.

4.4.5 Separierung der Muskeln

Es werden die markierten Muskeln, die in dem Schaubild 4.4a zu sehen sind, von dem Charakter getrennt. Danach werden diese separat als Static Mesh in das Projekt importiert und in ein FleX-Objekt umgewandelt. Die Einstellungen der FleX-Eigenschaften sind der Tabelle 4.1 zu entnehmen. Als nächstes müssen die FleX-Objekte an den Charakter geheftet werden, welche mit dem Skeletal Mesh in Verbindung stehen. Das wird wie im Absatz Nachahmung der Physical Bodys aufgeführt, bewerkstelligt. Jedoch ist beim Testen aufgefallen, dass die Muskelobjekte zu klein sind und sich durch die Hilfsobjekte in ungewollte Positionen verschieben. Demnach wird dieser Ansatz nicht weiterverfolgt. Die Fehlerquote ist dazu in der Konzeption bereits zu hoch.

4.5 Anwendung an Vegetationen

In der realen Welt gibt es unterschiedliche Pflanzen und so soll dies auch auf die fiktive Welt übertragen werden. Kategorisiert werden sie durch organische Weichkörper. Es soll ebenso zwischen Kräutern, Büschen, Sträuchern und Pflanzen mit einem Stamm unterscheiden werden. Kleine Pflanzen können als ein einheitliches Objekt betrachtet werden, wobei größere meist kein ähnliches Verhalten aufweisen und deshalb separat betrachtet werden sollen.

Unterscheidung anhand des Beispiels eines Baumes

Bäume werden in den meisten Spielen als Festkörper visualisiert und deren Blätter anhand von Texturen simuliert. Kleinere Bäume sollten aufgrund ihres kleineren Radiusse auch ein geringes Maß an Elastizität aufweisen, denn sie bewegen sich beispielsweise im Wind mit. Große Bäume hingegen sind so fest mit dem Boden verwurzelt, dass dies nicht der Fall ist. Dieses Konzept lässt sich wahlweise auch auf Sträucher übertragen.

Unterscheidung anhand des Beispiels der Kräuter, Büsche und restlichen Pflanzen

Büsche und Kräuter bzw. Gräser werden in den Spielen meist als Überlagerung verschiedener zweidimensionaler Sprites erzeugt. Dabei wird dieselbe Textur mit dem gleichen Nullpunkt mehrfach platziert. Eine andere Technik ist ein Static Mesh, welches die tatsächliche Form der Pflanze aufweist.

Ziel und Vorgehen

Sträucher sowie die kleinen Pflanzen sollen zur Evaluation mit einer Weichkörperdynamik versehen werden. Dafür werden freie Texturen und 3D-Modelle aus dem Unreal Marketplace heruntergeladen und zur Weiterverwendung vorbereitet. Wie zuvor wird auch hier versucht mit der Unreal Engine das gesetzte Ziel zu erreichen und danach mit der FleX-Erweiterung fortzufahren. Als Beispielobjekt soll nun ein Grasbüschel dienen. Dieses und die Textur dafür wurden aus dem Epic Marketplace heruntergeladen und die überflüssigen Komponenten entfernt [46].

4.5.1 Nativ Unreal-Implementation

Vegetation hat in der Unreal Engine eine eigene Sektion. Diese kann mit dem eingebauten Foliage-Tool platziert werden. Dabei werden die Pflanzen auf die Umgebung in der Spielewelt mit einem Pinsel-Werkzeug „aufgemalt“. Hierdurch wird eine Testfläche mit entsprechenden Objekten angelegt. Beide Gras-Objekte erhalten unterschiedliche Materialien und werden zur besseren Unterscheidung auch in verschiedene Areale gesetzt. Da es sich um ein kleines Objekt handelt, welches keinen großen Einfluss auf das Spielgeschehen hat, ist die Technik des „Material Vertex Displacements“ denkbar. Diese kann in zwei Ausführungen verwendet werden. Um einen relativ einfachen Shader zu programmieren, ist die Verwendung des Knotens „DistanceToNearestSurface“ am naheliegendsten. Da sich der Shader das Gras, egal an welchem Objekt es sich befindet, sich bewegen lässt. Dies wird im Code wie folgt umgesetzt: Zuerst wird ein Skalar mit dem Knoten „DistanceToNearestSurface“ erzeugt, welcher den Wert zur nächsten Oberfläche zurückgeben soll. Dieser wird mit einem Schwellwert verrechnet, welcher angibt, wann sich das Objekt verformen soll. Der Wert wird auf eine Normale zwischen 1 und 0 begrenzt. Nachfolgend wird dies mit dem Vektor, mit dem die Oberfläche registriert wurde, multipliziert und mit einem festgelegten Wert erneut multipliziert. Dies

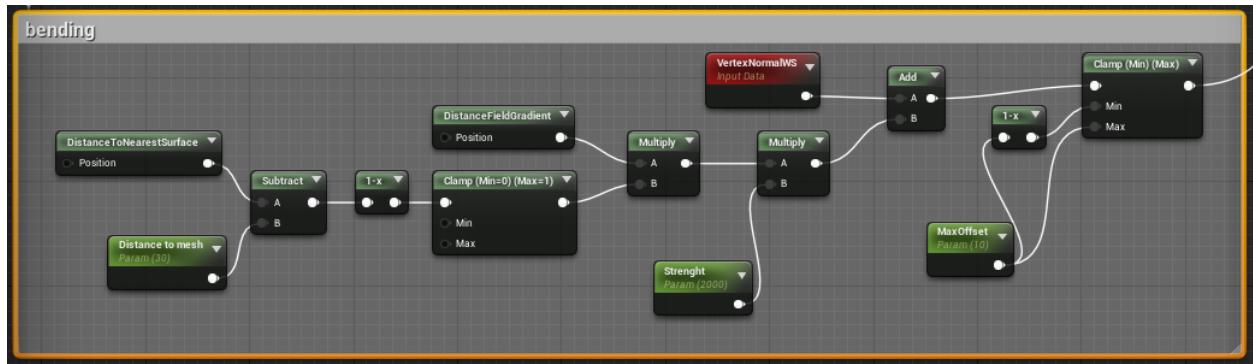


Abbildung 4.10: Code für den Material Shader

hat den Hintergrund, dass die Verbiegung des Materials unabhängig von dem eingehenden Vektor einstellbar sein soll. Der modifizierte Vektor wird nun mit der Vertex-Normalen in der Welt addiert und mit einem “Begrenzer Knoten” vor dem Überstrecken geschützt. Die Vertex-Normale ist ein Richtungsvektor, die dem Objekt die Richtung vorgibt, in welche es sich verbiegen soll. Zuallerletzt wird dies mit dem Input des Materials namens “World Position Offset” verbunden (siehe 4.10). Die gesetzten Materialparameter im Code können in der Materialvorschau nachträglich bearbeitet werden ohne, dass im Code Änderungen vorgenommen werden müssen. Als nächstes muss der Bug, der durch diese Vorgehensweise entsteht, behoben werden. Durch das Platzieren auf dem Grund wird das Objekt nämlich durch den Material-Shader in der Z-Achse in die positive Richtung verbogen. Dies kann durch das Setzen der Eigenschaft am Bodenobjekt “Affect Distance Field Lighting” korrigiert werden. (Ergebnis: 7.9a)

Eine zweite Möglichkeit einen Shader zu programmieren, um eine genauere Kollision mit dem Gras zu erzeugen, ist die Methode eines Material-Shaders basierend auf der Position in der Welt. Hierbei soll für den Material-Shader die absolute Weltposition eines jeden Akteurs mitgeteilt werden, um diese als Eingabeparameter mit einzubeziehen. Dadurch steigt die Komplexität der Technik, da für jedes Spielobjekt jeweils ein Parameter im Shader berücksichtigt werden muss.

Für die Umsetzung im Rahmen dieser Bachelorarbeit sind zwei Akteure herangezogen worden. Diese sind die beiden Füße eines Charakters. Zunächst wird eine Materialfunktion angelegt, welche die zwei Vektoren speichert. Damit die Vektoren die Position des jeweiligen Fußes des Charakters erhalten, muss das Charakter-Blueprint mit einem Event erweitert werden. Bei jedem Tick in der Engine werden die Positionen beider Füße an die Materialfunktion übergeben. Die Position der Füße wird durch zwei Kapsel-Kollisionsboxen bestimmt. Diese werden durch das Koppeln an die Knochen im Skeletal Mesh an die Füße “gehängt” und bewegen sich durch die gleiche Rotation und Position mit.

Der Material-Shader ist ähnlich zu dem "DistanceToNearestSurface" aufgebaut. Dabei wird die Position des Fußes mit der des Objektes in der Welt verrechnet und als normalisiertes Skalar weiter verarbeitet. Jedoch wird davor ein Schwellwert definiert, der besagt, wann das Gras mit dem Fuß interagieren soll. Der Wert des Skalars wird mit den Texturkoordinaten multipliziert. Dies verhindert das Verbiegen an falschen Stellen bzw. berichtet die Verschiebung der Textur von der Wurzel des Büschels bis zu den Stängeln. Das Produkt der Multiplikation wird mit dem Produkt des Biegefaktors und dem eingehenden Richtungsvektor berechnet. Der Richtungsvektor wird aus Weltposition minus Position des Fußes gebildet. Die Berechnung wird jeweils für das andere Bein wiederholt und beide Werte werden miteinander summiert. Die Summe kommt in den “World Position Offset”-Knoten (siehe 7.9a).

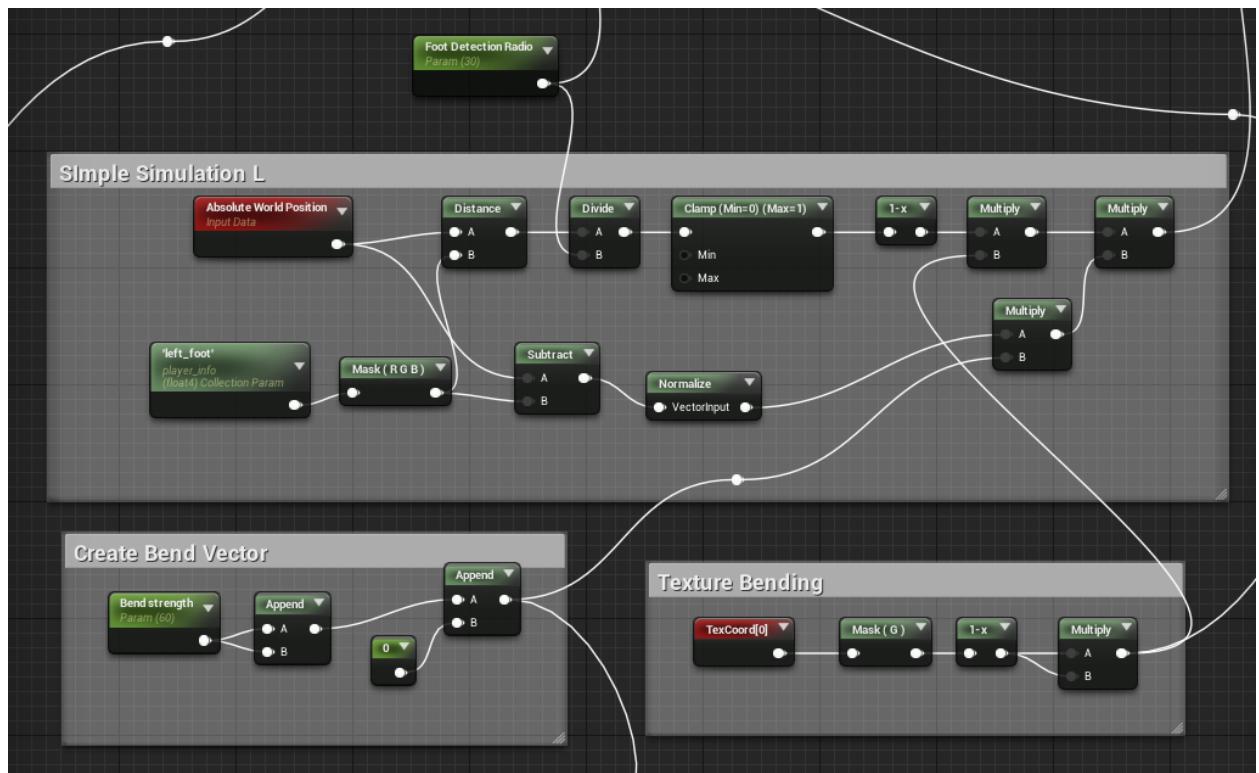


Abbildung 4.11: Code für den Material Shader mit Fuß Erkennung

4.5.2 Weiterführung der Implementierung

Das Verbiegen kann durch weitere Ursachen, wie Wind in der virtuellen Welt, hervorgerufen werden. Dafür gibt es bereits eine vorgefertigte Funktion in der Unreal Engine namens "SimpleGrassWind". Hierfür werden Materialparameter benötigt, die von einer Windquelle gespeist werden und der Funktion als Eingabewerte dienen. Jedoch sollte hierbei beachtet werden, dass ein "Overbending" durch Minimal- und Maximalwerte verhindert werden sollte.

4.5.3 Nvidia FleX-Implementation

Für den FleX-Effekt des Grasbüschels wird sein Static Mesh in ein FleX-Objekt umgewandelt. Nach anpassen des zugewiesenen Materials wird dieses mit dem Kontainertyp "Soft" kategorisiert. Weitere Einstellung, wie das Zuweisen eines eigenen Materials, werden vorgenommen, damit das Verbiegen des Grases nicht unnatürlich erscheint. Sogleich muss das Büschel immer an der gleichen Stelle verweilen, weshalb der Parameter "Attach to Ridget" aktiviert wird. Folglich wird die Pflanze dann in den Boden in der Z-Achse gesetzt. (Ergebnis: 7.9b)

5 Ergebnisse und Bewertung

Nachdem nun alle Programmierpunkte aufgezeigt wurden, soll nun alles zu einem Ergebnis zusammengefasst und bewertet werden. Dazu wird in die Problematik des Nvidia FleX-Plugins, die Umsetzung des Charakters sowie die Umsetzung kosmetischer Objekte gegliedert. Für eine nachvollziehbare Bewertung wird eine benutzerfreundliche Testumgebung angelegt. Dafür wird auf die Unreal Tutorial Maps zurückgegriffen. In diesen wird zu jeder Thematik eine entsprechende Umgebung mit Räumen geschaffen, die jeweils einige Vorführplattformen enthalten. Die Umgebungen sollen in folgende Räume unterteilt werden: Unreal native Implementation mit Seilen, Haaren, Kleidung und die Fake Softbodies. Im Vergleich dazu stehen die Nvidia FleX-Objekte. Als letzte Umgebung werden alle erarbeiteten Objekte verglichen und veranschaulicht.

5.0.1 Seil/Kette

Das Kabelsystem in Unreal ist, wie auch die Kabelkomponente, nicht für die aktive physikalische Verwendung gedacht. Dies ist dem Design der Implementierung geschuldet. Dadurch kommt diese in den Verruf nur als Dekorationsobjekt verwendet werden zu können. Jedoch bietet der "Physikal Constraint Actor" eine Erweiterung. Die Einstellbarkeit des "Physical Constraint Actors" beinhaltet jede Eigenschaft, die für die Simulation von Seilen in Betracht kommen kann. Zu bedenken gilt, dass das Überstrecken des Seiles nicht simuliert werden kann. Dadurch bleiben nur die zwei Zustände "komplett" oder "gerissen". Das Überstrecken kann man mit weiterem Code kaschieren und aufwendig nachimplementieren. Ein weiteres Problem ist es herauszufinden, wie auf das Seil eingehende Kräfte die Bewegung verändern können, wenn es nur als Dekorationsobjekt fungiert. Hierbei könnte sich jeder Entwickler mit der Technik der Skeletal Meshes helfen, indem der längliche Gegenstand mit Knochen versehen wird. Folglich resultiert aber daraus, dass es für eine grafische, akkurate Simulation eine Vielzahl an Knochen sein müssen. Sonst sieht die Verbiegung unnatürlich aus. Dieser Umstand ist auch bei der Kabelkomponente zu betrachten, lässt sich dort aber auch durch das Erhöhen der Bindeglieder lösen. Abhilfe schafft das FleX-Plugin in der Hinsicht, dass es die Seile akkurater simuliert. Außerdem hat man die Option, ein FleX-Objekt als Soft-Container Seil zu verwenden. So können durch weiteren Mehraufwand die Eigenschaften konfiguriert werden und mit einem "Joint" die Objekte verbunden werden. Weitere Programmertätigkeiten bleiben dabei aus. Wenn die Zertrennung des Seiles gewünscht ist, kann der gesetzte "Joint" entfernt werden und das Seil reißt an der Verbindungsstelle.

5.0.2 Kleidung

Die Nvidia Cloth-Implementation in der Unreal Engine hat für ihren begrenzten Bereich eine beträchtliche Menge an Einsatzmöglichkeiten. Die Begrenzung umfasst die Notwendigkeit das Objekt als Typ Skeletal Mesh anzulegen, die Nicht-Interaktion aller Spielobjekte sowie die fehlende Zerreißbarkeit der Kleidung. Jedoch lassen sich diese Schwächen mit einigen umständlichen Techniken ausgleichen. Die Kleidungssimulation des Nvidia FleX-Plugins kann hingegen mit dem Vorteil punkten, dass diese auf jedes Objekt angewendet werden kann. Dadurch entsteht ein schnellerer Workflow. Durch Simulation von Partikeln lässt sich die Kleidung ohne weiteren Programmieraufwand zerreißen. Zu bedenken ist, dass die Intensität der aufzuwendenden Kraft nicht konfigurierbar ist. Ein Negativpunkt ist, dass mit einem Hilfsscript der Wind in der globalen Welt auf den FleX-Container übertragen werden muss.

Besonders unvorteilhaft sind Kleidungsstücke, die nicht an einen Charakter angehängt werden können, ohne dafür weitere Hilfsobjekte zu verwenden. Schlussendlich ist die Verwendung der Kleidung mit dem FleX-Framework als dekorative Komponente eine Überlegung wert. Jedoch schließt sich damit die Verwendung auf Konsolen aus.

5.0.3 Haare

Mit dem Hinzukommen der neuen Erweiterung "Groom" und der Möglichkeit die Haare mit dem "Nvidia Cloth Solver" zu bearbeiten, ist ein ausreichendes Tool in der Unreal Engine vorhanden. Die FleX-Plugin Haare sehen durch den Softcontainer und den richtigen Einstellungen zufriedenstellend aus. Jedoch fallen sie durch die zusammenhängende Form des Beispielobjektes bei dichteren Stellen unrealistisch aus. Dadurch leitet sich die Verwendung nur für die Haarspitzen ab.

5.0.4 Weichkörper

Durch die vorhandenen Methoden in der Unreal Engine können in einem abgesteckten Rahmen Fake Softbodys erstellt werden. Dies setzt aber voraus, dass das Objekt ein Skeleton Mesh ist oder Techniken verwendet werden, die die Texturen beeinflussen. Die genannten Techniken gehen jedoch zu Lasten der Präzision der Simulation. Keine anspruchsvollen Gegenstände, wie Pflanzen, können mit der Methode "Vertex Displacement" verbogen werden. Wird das Objekt jedoch komplexer, muss auf weitere Techniken wie mit dem Skeleton Mesh und den simulierten Knochen zurückgegriffen werden. Es ist aber zu erwähnen, dass sich die Problematik damit nicht komplett löst. Probleme wie die Plasilität werden nicht berücksichtigt. Das FleX-Framework löst diese Probleme dadurch, dass es komplett in die Unreal Engine implementiert wird. Allerdings ist die Weiterverarbeitung des einzelnen FleX-Objektes sehr beschränkt, wodurch es schlussendlich nicht für den produktiven Einsatz zu verwenden ist.

5.1 Visueller Vergleich der umgesetzten Ergebnisse

Für die genauere Bewertung wurde ein Slow Motion-Modus implementiert, der per Tastendruck aktiviert werden kann.

5.1.1 Testen der Charaktere

Zum Testen wird eine Reihe von Animationen, die der Charakter bereits besitzt, verglichen. Diese sind in dem Template, welches vom Marketplace heruntergeladen wurde, vorhanden. Um diese anzuzeigen und mit der Spielsimulation zu verknüpfen, wird ein Animationskontroller verwendet. In der Gesamtbetrachtung sieht der Charakter mit den nativen Unreal-Techniken realistischer aus und interagiert auf eingehende Kräfte. Jedoch ist zu bemängeln, dass beim Laden anderer LODs die Kleidung in ihren Ausgangszustand zurückfällt und erneut simuliert werden muss. Charaktere, die auf der FleX-Technologie basieren, haben folgende Probleme: Durch die zu großen primitiven Objekte, die als Ersatzskelett dienen, wird die Kraft bei Kollisionen auf die Objekte verlagert statt auf den eigentlichen Körper des Charakters und der gewünschte Effekt bleibt aus. Die Version mit den Knochen im Körper macht einen realistischeren Eindruck. Dennoch trennen sich die Gliedmaße des Charakters bei zu vielen Interaktionen automatisch ab. Durch zu wenig Knochen im Körper werden

gewisse Animationen nicht ordnungsgemäß abgespielt und die Gliedmaße verformen sich unnatürlich. Als weiterer Kritikpunkt fällt auf, dass bei Spheren, die als Skelett verwendet werden, eine Art Zittern in diesem Bereich vorzufinden ist.

5.1.2 Testen der Vegetation

Wie erwartet, führt die zweite Methode mit den Kollisionsboxen am Charakter zu einem genaueren Ergebnis. Dennoch ist zu bedenken, dass diese Technik einen deutlichen Mehraufwand bei erhöhter Zahl von Objekten, die sich in der virtuellen Umgebung befinden, mit sich bringt. Das Ergebnis der FleX-Variante ist dennoch ausreichend. Obwohl die gesamte Struktur des Büschels ab und zu aus dem Boden gerissen wird und sich zurücksetzt.

5.2 Problematiken des Nvidia FleX-Frameworks

Im Zuge der Evaluation soll auf das Plugin eingegangen werden. Außerdem sollen dadurch entstehende Probleme, die die Umsetzung der Objekte behindern oder unverwendbar machen, näher beschrieben werden.

5.2.1 Umsetzung der Charaters

Bei der Umsetzung kommt es zu den folgenden Problemen: Der eigentliche Charakter ist als Doublette vorhanden und erzeugt einen Mehraufwand beim Berechnen der Objekte. Folglich muss eine tiefere Analyse der Implementierung gemacht werden. Die Unreal Engine implementiert ein Skeletal Mesh-Objekt der "USkeletalMeshComponent"-Klasse, welche mit der Renderpipeline in Verbindung steht. Deshalb muss die Klasse durch die FleX-Klassen erweitert werden und die Eigenschaften der FleX-Objekte im Bereich Positionierung müssen eingeschränkt werden. Letztere werden nämlich unabhängig vom Objekt-Nullpunkt berechnet. Gleichzeitig wird der Charakter als ganzes Objekt in ein FleX-Objekt umgewandelt. Dadurch erhalten alle Komponenten, die an ihn angehängt sind, dieselben Containereigenschaften. Dazu zählen die Kette und die Kleidung. Beide verhalten sich nicht wie gewünscht. Dieses Problem lässt sich mit einer Maskierung der einzelnen Areale umgehen. Ein Lösungsvorschlag hierfür ist die Erweiterung der FleX-Struktur im Bereich der Partikel, da diese mit einem zusätzlichen Faktor versehen werden können. Angewandt könnte dies z. B. auf ähnliche Implementationen, wie bei beim Kleidungstool, werden. Außerdem ist zu beachten, dass der Charakter von einem Konstrukt gelenkt wird, an das er befestigt ist. Es bleibt ein Restrisiko, welches den gesamten Charakter bei zu hoher Krafteinwirkung aus der Stützstruktur katapultieren könnte, vorhanden.

5.2.2 Umsetzung der kosmetischen Objekte

Hierbei wurden zwei Vorgehensweise verwendet: Bei der ersten wurde die Handfessel als gesamtes Objekt vom Charakter-Mesh abgetrennt und in ein FleX-Objekt konvertiert. Folglich erhält dieses die gleichen Containereigenschaften und die gleiche Partikelstruktur sowie die daraus resultierenden Probleme. Bei jeder Bewegung des Charakters wird ebenso die Handfessel mitbewegt, was auch das erwartete Verhalten ist. Allerdings ist beim genaueren Hinsehen eine dauerhafte Vibration des Objektes aufgefallen. Darüber hinaus fliegt die Handfessel bei schnellen Bewegungen vom Grundkörper ab. Daraus lässt sich schlussfolgern, dass diese Verbindung so nicht geeignet ist. Eine Überlegung dazu ist, einen "Joint" aus dem FleX-Framework zu verwenden. Diese sollen verschiedene FleX-Objekte verbinden und

dadurch überflüssiges aneinander koppeln durch ständige Kollision mit anderen Objekten verhindern. Im Nachhinein ist dazu jedoch nicht geeignet (siehe Absatz 5.2.3). Außerdem sollte bei diesem Lösungsansatz am Skeletal Mesh ein FleX-Körper hängen. Da diese sich von Grund auf nicht damit verbinden lassen, ist diese Lösung beim zu Grunde liegen von Software hinfällig. Die zweite Methode besteht darin, die Kette zu separieren und diese an den Charakter anzuhängen. Dieses Problem konnte nur durch externe primitive Objekte gelöst werden, da, wie festgestellt, die FleX-Technik bei Blueprints nicht greift. Zusammenfassend bringt die erste Variante wohl das realistischere Ergebnis, denn das Verrutschen an den Handgelenken würde simuliert werden. Jedoch ist diese Methode nur mit einer komplexeren Implementation lösbar. Das Umsetzen der Kleidung des Charakters hat sich durch die bereits aufgezeigten Problematiken ähnlich schwierig gestaltet. Das Platzieren in der globalen Spielwelt hat dies jedoch gelöst. Dennoch stellt es ein Problem dar, welches gelöst werden sollte. Schließlich sollten Konstrukte in Blueprints verlagert werden können, ohne dass diese gesondert gehandhabt werden müssen.

5.2.3 Sonstige Probleme

Eine weitere Schwierigkeit im FleX-Plugin ist die Verwendung der bereitgestellten "Joints". Allerdings sind sie in ihrer Funktion eingeschränkt. Es können nur FleX-Objekte desselben Typs verbunden werden. Dadurch lässt sich die Verbindung verschiedener FleX-Container nicht bewerkstelligen. Beim Testen des Cloth-Containers mit einem "Joint" ist die Engine abgestürzt. Als nächster Punkt ist die Unhandlichkeit von kleinen Objekten aufzuführen. Da die Partikel im FleX-Plugin eine Mindestgröße aufweisen, lassen sich kleine Objekte damit schlecht simulieren. Beim Vegetationsbeispiel sind die einzelnen Halme nicht berücksichtigt worden und somit ist die Anordnung der Partikel nicht wie gewünscht dargestellt gewesen. Als letztes ist das bereits erwähnte LOD-Problem zu erwähnen. Dies verhindert das Optimieren der Vertex-Anzahl in der Szene.

Zum Abschluss sollen nun die Erkenntnisse und Ergebnisse zusammengefasst werden und es soll ein Ausblick gegeben werden.

6 Fazit und Ausblick

6.1 Fazit

Nach Durchführung der Analyse und der Umsetzung der einzelnen Techniken lässt sich feststellen, dass eine große Schnittmenge der geforderten Mechaniken in der Unreal Engine vorhanden ist. Es fehlt in der Engine jedoch die tatsächliche Implementierung eines Weichkörpers. Welche Technologie dafür verwendet werden kann, liegt der Effizienz der Methode zugrunde und darin welche Hardware genutzt werden kann. Außerdem werden die Spieleentwickler vor die Herausforderung gestellt, die Methode zu finden, die für den Spieler realistisch genug ist, ohne zu Lasten der Leistung zu gehen. Ob dies nun durch eine wissenschaftliche Methodik geschieht oder nur ein verfälschtes Ergebnis ist, steht jedem Entwickler offen. Es sollte aber bedacht werden, dass die meisten Lösungsverfahren zu Lasten der CPU erfolgen. Ob es nun eine sinnvolle Entwicklung oder nur der Umstand ist, dass es auf jeder Plattform laufen soll, bleibt der Zukunft überlassen. Nvidia zeigt einen interessanten Ansatz darin, die Problematik auf die Grafikkarte auszulagern. Grafikkarten sind mittlerweile für ihre übermäßige Parallelisierung bekannt. Hierdurch können auch neue Forschungserkenntnisse genutzt werden. Mit dem Framework FleX ist eine gute Grundlage für die Weichkörpertechniken geboten. Das macht sich darin deutlich, dass einige Spiele mit diesen ausgestattet worden sind. Leider hat man sich auch an die Empfehlungen des Herstellers gehalten und nur visuelle Effekte verwendet, die bei einem Großteil der Effekte mit dem Spielgeschehen nicht in Verbindung stehen. Aber aufgrund dessen, dass nur einzelne Teile implementiert wurden und viele Stellen ihre Probleme zeigen, ist eine marginale Nutzung des Frameworks möglich. Abschließend ist zur Erweiterung zu sagen, dass diese auf den ersten Blick einen guten Eindruck macht und den Entwickler mit guten Features lockt, nach Validierung des Plugins hingegen schwach abschließt und keine gute Wahl ist. Zu bedenken sind hierbei auch die nicht vorhandene Plattformunabhängigkeit und die Fehler in der Implementierung. Wäre das Plugin für die Unreal Engine weiterentwickelt worden, wäre wohl nun angesichts der vergangenen Zeit bereits ein Toolset auf dem Markt, mit dem wahrscheinlich mehr Spiele ausgestattet worden wären. Resümierend ergibt diese Ausarbeitung eine Widerlegung der zugrundeliegenden These am Anfang der Ausarbeitung, dass es für die Spiele kaum Techniken gibt und auch bereits viele Probleme gelöst worden sind. Obwohl es nicht in einem Ausmaß geschehen ist, welches wünschenswert ist.

6.2 Ausblick

Wünschenswert wäre eine größere Auswahl an Frameworks, die sich der Weichkörpersimulation annehmen. Da aber zur aktuellen Zeit die mediale Aufmerksamkeit auf Themen wie Raytracing liegt, werden Entwickler sich selbst bei dem Themenkomplex behelfen müssen. Dahingehend wäre eine Evaluation des Plugins VICODynamics und dessen genaue Implementation interessant. Zum mindest wird die Problematik der Implementation eines Charakters beim Skeletal Mesh in der Weichkörperperform vom Entwickler erwähnt und es wird an einem System gearbeitet [47]. Demnach ist zu erwarten, dass in absehbarer Zeit eine tauglichere Variante des komplett in Echtzeit simulierten Weichkörpers als spielbarer Charakter zu sehen sein wird. Dadurch eröffnen sich weitere Möglichkeiten wie die Darstellung von realistischen Verletzungen, die mit entsprechender Entwicklungsarbeit das gesamte Spielgeschehen revolutionieren können. Möglich sind aber auch realistische Charaktere, die durch das gesamte

Erscheinungsbild dem Spiel eine weitere Tiefe geben können. Infolgedessen fallen dann die komplexen Konstruktionen der Charaktercontroller weg. Führt man konzeptionell Nvidia FleX etwas weiter, kommen folgende Schlussfolgerungen zustande: Das FleX-Plugin ist mit einer guten Voraussetzung gestartet, nämlich einer Technik, die für den Markt relativ neu gewesen ist und viele Menschen überzeugen konnte, sich mit dem Thema zu befassen. Leider ist dies zum Zeitpunkt der Veröffentlichung mit der Hardwareleistung nicht einhergegangen, was einer der Punkte gewesen sein kann, warum das Plugin nicht weiter aktualisiert wurde. Zweitens gibt es ein Feature, welches in der Vorschau zu sehen war, die "Plastic Deformation", aber nicht in das Plugin implementiert wurde. Deshalb wäre die Erstellung einer solchen Option und der damit verbundenen Anwendungen interessant. Es gibt bereits Spiele, die den Fokus auf diese Technik gelegt haben und für diese Nische recht bekannt geworden sind. Erweitern würde das Ganze die Möglichkeit, die Objekte zu zerschneiden, welche nicht explizit implementiert ist und deshalb eigenständig programmiert werden müsste. Der Ausgangspunkt dafür wird in Zukunft Nvidia mit PhysX 5.0 sein. Ob dadurch ein erneuter Konkurrenzkampf mit anderen Herstellern entfacht wird sowie ob eine grundlegende Erweiterung in den bekannteren Spiele Engines stattfindet, bleibt abzuwarten.

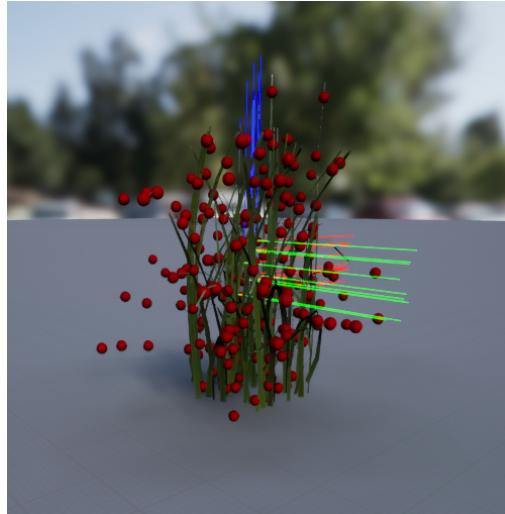
7 Weitere Abbildungen



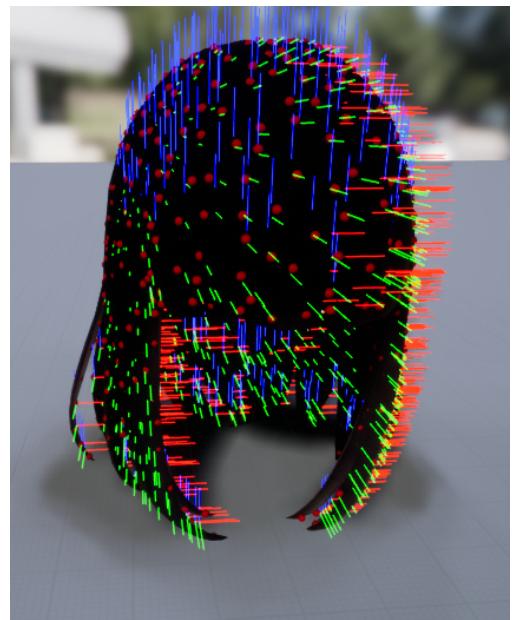
Abbildung 7.1: Testareal für die Basis-Komponenten



Abbildung 7.2: Testareal für die umgesetzten Beispiele



(a) FleX-Parikel im Gras-Objekt



(b) FleX-Parikel im Haar-Objekt

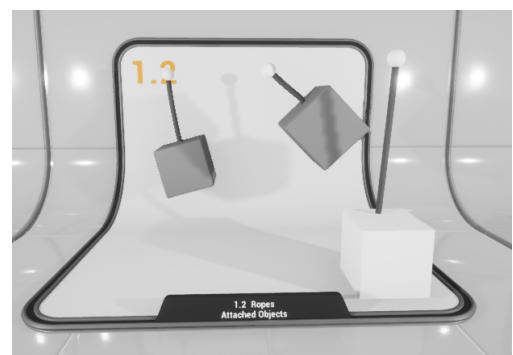
Abbildung 7.3: Übersicht der FleX-Objekte



Abbildung 7.4: Übersicht beider Haar-Umsetzungen



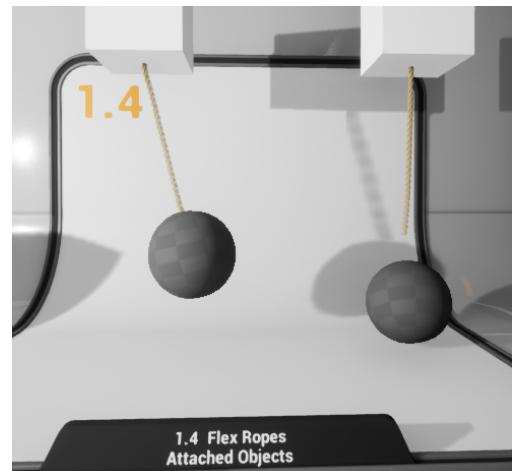
(a) Kabelkomponente



(b) Kabelkomponente mit Anhang

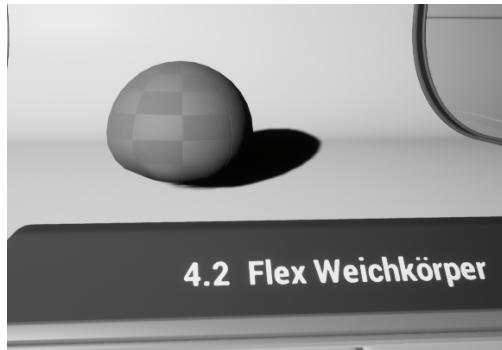


(c) FleX-Implementierung des Seils

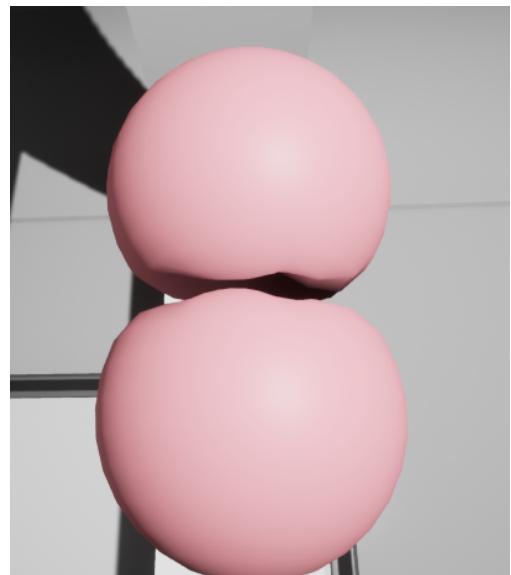


(d) FleX mit Anhang und Joints, Rechts ohne

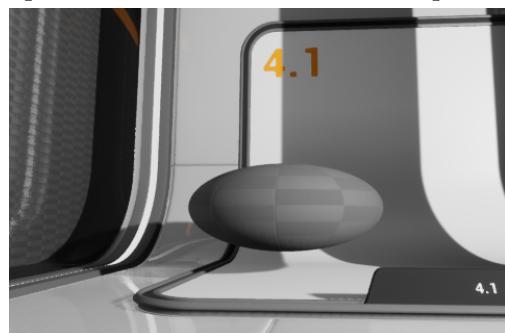
Abbildung 7.5: Übersicht aller Seil-Implementationen



(a) Weichkörper der FleX-Implementierung beim Aufprall

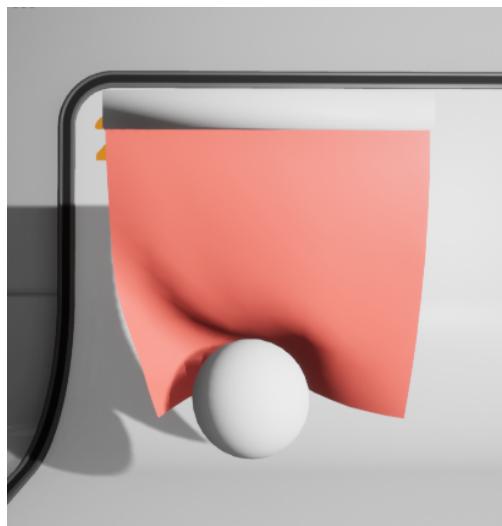


(b) Shader-Umsetzung der Weichkörper per



(c) Ergebnis der 3D-Transformation

Abbildung 7.6: Übersicht aller Weichkörper



(a) Native Kleidungssimulation



(b) FleX-Implementierung

Abbildung 7.7: Übersicht aller Kleidungs-Implementierungen

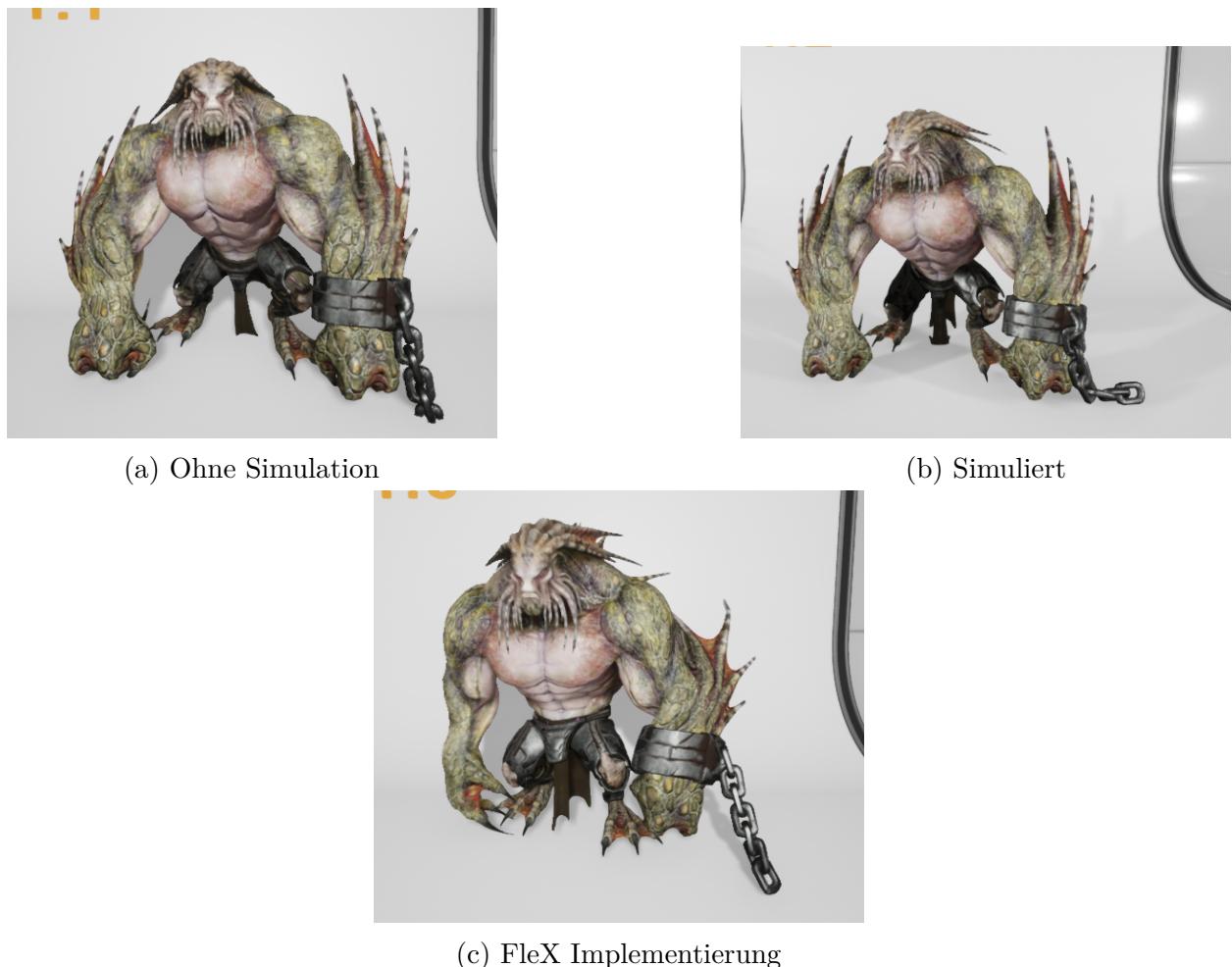


Abbildung 7.8: Übersicht aller Charakter-Implementierungen



(a) Distance-Shader mit einer Sphere als Testobjekt

(b) FleX-Implementierung mit einer Sphere als Testobjekt

Abbildung 7.9: Übersicht aller Gras-Implementierungen

Literaturverzeichnis

- [1] Jiayi Eris Zhang, Seungbae Bang, David I.W. Levin, and Alec Jacobson. Complementary dynamics. *ACM Transactions on Graphics*, 2020.
- [2] Fotorealismus in Spielen: Hellblade-Entwickler zeigt, wie beeindruckend Spiele der Zukunft aussehen könnten. <https://www.gamestar.de/artikel/hellblade-entwickler-project-dreadnought,3366085.html>. [Online; aufgerufen 19.07.2021].
- [3] Cyberpunk 2077 patch 1.10/1.11 - has the game improved on PS4 and Xbox One? <https://www.eurogamer.net/articles/digitalfoundry-2021-cyberpunk-2077-110-111-ps4-xbox-one-tests>. [Online; aufgerufen 19.07.2021].
- [4] BeamNG.drive. <https://www.beamng.com/game/>. [Online; aufgerufen 19.07.2021].
- [5] Wreckfest game. <https://order.wreckfestgame.com/>. [Online; aufgerufen 19.07.2021].
- [6] Epic Games Inc. www.epicgames.com, 1991. [Online; aufgerufen 14.07.2021].
- [7] What game engines do you currently use? <https://www.statista.com/statistics/321059/game-engines-used-by-video-game-developers-uk/>, 2019. [Online; aufgerufen 19.07.2021].
- [8] Unreal Engine 5: Early Access verfügbar. <https://www.pcgamehardware.de/Unreal-Engine-Software-239301/News/Unreal-Engine-5-Early-Access-Version-jetzt-als-Download-verfuegbar-1372773>. [Online; aufgerufen 19.07.2021].
- [9] Subsystem that calculates collision and simulate physical actors. <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/>. [Online; aufgerufen 19.07.2021].
- [10] Creation and Import of Destructibles using the NVIDIA APEX Toolset. <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/Apex/>. [Online; aufgerufen 19.07.2021].
- [11] Hans Lüth Harald Ibach. *Festkörperphysik*, pages 3–35. Springer-Verlag Berlin Heidelberg, 2009.
- [12] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics, 2005.
- [13] Mayer Humi. *Introduction to mathematical modeling*, pages 28–45. Boca Raton, FL: CRC Press, 2017.
- [14] Making Your Game Fully Interactive by NVIDIA FleX. <http://developer.download.nvidia.com/assets/frameworks/downloads/regular/events/cgdc15/Making%20Your%20Game%20Fully%20Interactive%20by%20NVIDIA%20FleX-ENG.pdf>. [Online; aufgerufen 19.07.2021].

- [15] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics*, 2014.
- [16] Fritz Klocke. *Finite Elemente Methode (FEM)*, pages 407–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.
- [17] Nvidia. <https://www.nvidia.com>. [Online; aufgerufen 19.07.2021].
- [18] GameWorks | Nvidia Developer. <https://developer.nvidia.com/gameworks>. [Online; aufgerufen 19.07.2021].
- [19] NVIDIAGameWorks/FleX . <https://github.com/NVIDIAGameWorks/FleX>. [Online; aufgerufen 19.07.2021].
- [20] NVIDIAGameWorks/FleX . <https://github.com/NvPhysX/UnrealEngine/tree/FleX-4.19.2>. [Online; aufgerufen 19.07.2021].
- [21] Olento/UnrealEngine Branch: 4.21-FleX . <https://github.com/Olento/UnrealEngine/tree/4.21-FleX>. [Online; aufgerufen 19.07.2021].
- [22] NVIDIA Flex documentation. <https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/flex/index.html>. [Online; aufgerufen 19.07.2021].
- [23] Flex Artist Tools 1.0 documentation. <https://docs.nvidia.com/gameworks/content/artisttools/Flex>. [Online; aufgerufen 19.07.2021].
- [24] Release Notes Flex Artist Tools 1.0 documentation. <https://docs.nvidia.com/gameworks/content/artisttools/Flex/releasenotes.html>. [Online; aufgerufen 19.07.2021].
- [25] Introduction Flex Artist Tools 1.0 documentation. https://docs.nvidia.com/gameworks/content/artisttools/Flex/FLEXUe4_Intro.html. [Online; aufgerufen 19.07.2021].
- [26] Killing Floor 2 mit Nvidia Flex - Grafikvergleich und Benchmarks. <https://www.gamestar.de/artikel/killing-floor-2-mit-nvidia-flex-grafikvergleich-und-benchmarks,3305619.html>. [Online; aufgerufen 19.07.2021].
- [27] Fallout 4 Patch 1.3: Benchmarks mit neuen Treibern und HBAO+. <https://www.computerbase.de/2016-02/fallout-4-patch-1.3-benchmarks-treiber-hbao-plus/2/>. [Online; aufgerufen 19.07.2021].
- [28] Advanced Micro Devices Inc. <https://www.amd.com>. [Online; aufgerufen 19.07.2021].
- [29] GPUOpen. <https://gpuopen.com/>. [Online; aufgerufen 19.07.2021].
- [30] FEMFX Multithreaded deformable physics . <https://gpuopen.com/femfx/>. [Online; aufgerufen 19.07.2021].
- [31] GPUOpen-Effects/FEMFX. <https://github.com/GPUOpen-Effects/FEMFX>. [Online; aufgerufen 19.07.2021].
- [32] ProjectBorealisTeam/UnrealEngine/tree/FEMFX-4.24 . <https://github.com/ProjectBorealisTeam/UnrealEngine/tree/FEMFX-4.24>. [Online; aufgerufen 19.07.2021].

- [33] Our Unreal Engine performance guide. <https://gpuopen.com/unreal-engine/>. [Online; aufgerufen 19.07.2021].
- [34] VICO Game Studio. <https://www.vicogamestudio.com>. [Online; aufgerufen 19.07.2021].
- [35] VICODynamics: Rope/Cloth/Soft-Body Simulation Plugin. <https://www.unrealengine.com/marketplace/en-US/product/vico-dynamics-plugin>. [Online; aufgerufen 19.07.2021].
- [36] Unreal Engine 4: Künstler zeigt beeindruckenden Sturm in Next-Gen-Optik. <https://www.pcgameshardware.de/Unreal-Engine-Software-239301/News/Kuenstler-zeigt-beeindruckenden-Sturm-in-Next-Gen-Optik-1335056/>. [Online; aufgerufen 19.07.2021].
- [37] Announcing NVIDIA PhysX SDK 5.0. <https://developer.nvidia.com/blog/announcing-nvidia-physics-sdk-5-0/>. [Online; aufgerufen 19.07.2021].
- [38] Unreal Engine 4 - Omniverse Connect Documentation. https://docs.omniverse.nvidia.com/con_connect/con_connect/ue4.html. [Online; aufgerufen 19.07.2021].
- [39] Cable Component. <https://docs.unrealengine.com/4.26/en-US/Basics/Components/CableComponent/>. [Online; aufgerufen 19.07.2021].
- [40] Physics Constraints. <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/Constraints/>. [Online; aufgerufen 19.07.2021].
- [41] An overview of Cloth creation using the in-Editor tools with Unreal Engine. <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/Cloth/Overview/>. [Online; aufgerufen 19.07.2021].
- [42] Hair Rendering. <https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Hair/Overview/>. [Online; aufgerufen 19.07.2021].
- [43] Nvidia Flex: Ropes. https://docs.nvidia.com/gameworks/content/artisttools/FLEXUe4_Ropes.html. [Online; aufgerufen 19.07.2021].
- [44] Nvidia Flex: Assets Overview. https://docs.nvidia.com/gameworks/content/artisttools/FLEXUe4_Assets.html. [Online; aufgerufen 19.07.2021].
- [45] Paragon: Rampage. <https://www.unrealengine.com/marketplace/en-US/product/paragon-rampage>. [Online; aufgerufen 19.07.2021].
- [46] temperate Vegetation: Foliage Collection. <https://www.unrealengine.com/marketplace/en-US/product/interactive-foliage-collection>. [Online; aufgerufen 19.07.2021].
- [47] Unreal Marketplace: Skeletal Mesh Per Poly Collision? <https://www.unrealengine.com/marketplace/en-US/product/vico-dynamics-plugin/questions>. [Online; aufgerufen 19.07.2021].

Eidesstattliche Erklärung

Hiermit erkläre ich eidesstattlich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt wurde, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen und unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Ort, Datum

Unterschrift