HAW HAMBURG

Bachelorarbeit

Devran Akca

Evaluation von Skelett-Animationen für Anwendungen in Augmented Reality

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Devran Akca

Evaluation von Skelett-Animationen für Anwendungen in Augmented Reality

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Kai von Luck

Eingereicht am: 04. April 2019

Devran Akca

Thema der Arbeit

Evaluation von Skelett-Animationen für Anwendungen in Augmented Reality

Stichworte

Animation, Skelett-Animation, Skinning, Blending, Mesh-Deformation, Augmented Reality

Kurzzusammenfassung

Animationen sind ein fester Bestandteil unseres Alltags und man begegnet ihnen bei der Nutzung verschiedenster Medien.
Das Animieren von Objekten ist jedoch keine triviale Aufgabe, sodass verschiedene Herangehensweise entwickelt wurden, um diese Herausforderung elegant zu lösen.
Als Skelett-Animation wird eine dieser Herangehensweisen betitelt. Sie verbindet die Oberflächenrepräsentation eines Objektes mit einem Skelett, sodass die Manipulation einzelner Knochen auf die Oberfläche des Modells übertragen wird, um dieses zu deformieren und eine Animation zu erzeugen.
In dieser Arbeit werden zunächst verschiedene Methoden der Skelett-Animation erläutert. Zwei besonders relevante Methoden werden dann im Kontext einer Augmented Reality Applikation implementiert und abschließend evaluiert.

Devran Akca

Title of Thesis

Evaluation of Skeletal-Animations for Applications in Augmented Reality

Keywords

Animation, Skeletal-Animation, Skinning, Blending, Mesh-Deformation, Augmented Reality

Abstract

Animations are an integral part of our everyday life and you will encounter them in the use of various media.

However, animating objects is not a trivial task, so various approaches have been developed to elegantly solve this challenge.

One of these approaches is titled skeletal-animation. It combines the surface representation of an object with a skeleton, so that the manipulation of individual bones is transferred to the surface of the model in order to deform it and create an animation.

In this work first of all different methods of skeleton animation will be explained. Two particularly relevant methods are then implemented in the context of an augmented reality application and evaluated.

Inhaltsverzeichnis

1 Einleitung

Die Computeranimation ist bereits ein fester Bestandteil unseres digitalen Alltags und findet sich bereits in verschiedensten Bereichen wieder, von Computerspielen über Simulationen bis hin zur Unterstützung bei der Erklärung von Sachverhalten in der Lehre. Mit der immer populärer werdenden Augmented Reality, die sich vor allem auf mobilen Geräten durchsetzt, stellen sich der Animation neue qualitative wie auch quantitative Herausforderungen.
Diese Arbeit beschäftigt sich im Kern mit der Skelett-Animationstechnik und stellt diese in den Kontext der Augmented Reality. Durch diese Technik ist es möglich Objekte durch die Deformation ihrer Oberfläche zu animieren. Dafür wird für ein Objekt ein Skelett angelegt, dessen Deformation auf besagtes Objekt übertragen wird. Ob es sich bei dem Objekt um einen humanoiden Charakter, eine Pflanze oder eine Maschine handelt ist irrelevant.
Am Anfang der Arbeit werden die benötigten Grundlagen aus den Bereichen der Mathematik und der Computergrafik geschaffen. Darauf folgt eine Einführung in die Skelett-Animationstechnik und eine Untersuchung des Forschungsstandes. Anschließend wird ein Konzept formuliert, welches in seiner Umsetzung die Grundlage schafft verschiedene Skelett-Animationstechniken miteinander zu vergleichen und in den Rahmen der Augmented Reality zu stellen. Nach einer abschließenden Evaluation der Ergebnisse folgt dann der Ausblick für weitere Arbeiten.

2 Grundlagen

2.1 Euler-Winkel

Unter Euler-Winkeln versteht man einen Satz von drei Winkeln um drei Achsen, mit denen man die Orientierung eines Körpers im euklidischen Raum beschreiben kann. In der Regel werden die Winkel mit α, β, γ oder yaw, pitch und roll gekennzeichnet. Man unterscheidet bei den Drehungen zwischen intrinsischen und extrinsischen Drehungen. Als intrinsische Drehung bezeichnet man eine Drehung um eine zuvor gedrehte Drehachse, als extrinsische hingegen eine Drehung um die ursprüngliche Koordinatenachse. [17]

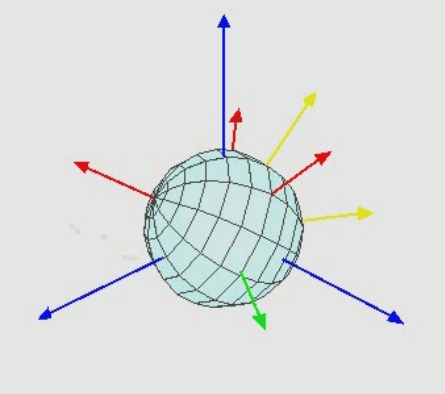

Abbildung 2.1: Gedrehter Körper im 3D-Raum. In blau ist das Referenz-Koordinatensystem dargestellt, in rot, gelb und grün die Rotationen. Quelle: https://de.wikipedia.org/wiki/Eulersche_Winkel (Letzter Zugriff 15.03.2019)

Ein großer Nachteil von Eulerwinkeln ist der Gimbal-Lock. Bei einer Rotation einer Achse um 90° rotiert diese unweigerlich auf eine andere Koordinatenachse. Diese kann dann nicht mehr weg rotiert werden, da zwei vorher verschiedene Rotationen dann identisch sind.

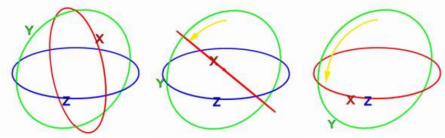

Abbildung 2.2: Gimbal-Lock durch Übereinanderlegen der Rotationsachsen. Die x-Achse wird durch Rotation auf die z-Achse gelegt, wodurch der Gimbal-Lock entsteht.
Quelle: Human Media Interaction - Luke Niesink

Hinzu kommt, dass Rotationen mit Eulerwinkeln mehrdeutig sind. Um eine eindeutige Endposition zu erhalten sind mehrere Euler Rotationen möglich. Die Orientierung einer Rotation der x-Achse um 180° ist identisch mit der Orientierung einer Rotation um die y-Achse um 180° und dann um die z-Achse um 180°. Um dieses Problem zu lösen können Konventionen getroffen werden, wie beispielsweise dass die erste und dritte Drehung immer um die gleiche Koordinatenachse stattfindet.

2.2 Quaternionen

Quaternionen sind ein vierdimensionaler Zahlenbereich, der die reellen Zahlen erweitert und eine reale und drei imaginäre Komponenten besitzt. Sie sind für die Computergrafik besonders interessant, weil sie Rotationen durch einfache Multiplikation elegant beschreiben. Im Kapitel Skinning werden die Vorteile von Quaternionen detailliert.

Quaternionen werden mit $\mathbb{R} = \{a + xi + yj + zk \mid a, x, y, k \in \mathbb{R}\}$, wobei x, j und z imaginär sind, definiert. Multiplikativ werden die neue Zahlen i, j und k gemäßg den Hamilton Regeln $i^2 = j^2 = k^2 = ijk = -1$ verknüpft.

2.2.1 Notation

Ist x eine Quaternion, so wird sie $x = x_0 + x_1 i + x_2 j + x_3 k$ notiert. Eine weitere Schreibweise formuliert den imaginären Teil als dreidimensionalen Vektor $(x_0, \vec{x}) = (x_0, x_1, x_2, x_3)$.

Die Rechenregeln zu Quaternionen kann man in [9] nachlesen.

2.2.2 Rotation mit Quaternionen

Das Euler Rotationstheorem besagt, dass jede Rotation oder Sequenz von Rotationen eines Koordinatensystems oder starren Körpers um einen fixen Punkt äquivalent zu einer einzelnen Rotation um einen Winkel alpha um eine fixe Achse ist, die durch den fixen Punkt läuft. Dabei wird meist der Einheitsvektor $\vec{u}$ als Achse genutzt. Den euklidischen Einheitsvektor $\vec{u} =< u_1, u_2, u_3 >$ kann man nun als den imaginären Teil einer Quaternion beschreiben, also $u_1 i + u_2 j + u_3 k$. Das heißt die Rotation um einen Winkel alpha kann beschrieben werden als $r = \cos(\frac{\alpha}{2} + (u_1 i + u_2 j + u_3 k)\sin(\frac{\alpha}{2})$. Um diese Rotation auf einen Vektor $\vec{t}$ mit (t_1, t_2, t_3) anzuwenden, bildet man zunächst eine Quaternion t_q mit einem Realteil von 0, also $t_q = 0 + t_1 i + t_2 j + t_3 k$ und multipliziert $t' = r * t_q * r^{-1}$.

2.3 Duale Quaternionen

Das Nutzen von dualen Quaternionen erweitert die Möglichkeiten statt Rotationen auch Translationen, die relevant in der Computergrafik sind, mit Quaternionen zu beschreiben. Wie man Matrizen in duale Quaternionen konvertiert (und wie man sie zurückkonvertiert) kann den Quellen J.B. Kuipers [19] und Kavan et al. [16] entnommen werden.

2.3.1 Duale Zahlen

Duale Zahlen erweitern ebenfalls die reellen Zahlen um eine zweite Komponente, indem der zweiten Zahl ein Element ϵ mit der Eigenschaft $\epsilon^2 = 0$ hinzugefügt wird. Sie sind notwendig, um in der Computergrafik mit dualen Quaternionen zu arbeiten.

2.3.2 Notation dualer Zahlen

Duale Zahlen notiert man $\hat{a} = a_0 + \epsilon * a_\epsilon$, bei der a_0 den reellen Anteil, ϵ die duale Einheit und a_ϵ den dualen Anteil der Zahl kennzeichnet.

2.3.3 Notation dualer Quaternionen

Duale Quaternionen notiert man $\hat{q} = q_0 + \epsilon * q_\epsilon$.

2.3.4 Rotation mit dualen Quaternionen

Eine dreidimensionale Rotation wird mit dualen Quaternionen so repräsentiert, dass der duale Part $q_\epsilon = 0$ ist. Als Konsequenz dessen wird die Quaternionen-Rotation genutzt, indem die Quaternion den reellen Teil der dualen Quaternion bildet. Für einen Vektor $v = (v_x, v_y, v_z)$ ist die korrespondierende duale Quaternion $\hat{v} = 1 + \epsilon(v_x i + v_y j + v_z k)$. Die Rotation eines Vektors um die duale Quaternion $\hat{q}$ wird berechnet durch $\hat{q} * \hat{v} * \hat{q}^*$. Bedenkt man, dass der duale Part der dualen Quaternion 0 ist, so kann man vereinfachen zu $\hat{q}\hat{v}\hat{q}^* = q_0(1 + \epsilon(v_x i + v_y j + v_z k))q_0^* = 1 + \epsilon q_0(v_x i + v_y j + v_z k)q_0^*$.

2.3.5 Translation mit dualen Quaternionen

Die 3D Translation um den Vektor $t = (t_x, t_y, t_z)$ wird mit der dualen Quaternion $\hat{t} = 1 + \frac{\epsilon}{2}(t_x i + t_y j + t_z k)$ gebildet. Es wird also mit der halben Translationsdistanz gerechnet, ähnlich dem halben Rotationswinkel der Rotation mit der Quaternion. Die Translation kann man ebenfalls der Einfachheit halber vereinfachen zu $\hat{t}\hat{v}\hat{t}^* = \hat{t}(1 + \epsilon(v_x i + v_y j + v_z k))\hat{t}^* = 1 + \epsilon((v_x + t_x)i + (v_y + t_y)j + (v_z + t_z)k)$.

2.4 3D-Modell

Die Oberfläche eines dreidimensionalen Objektes kann durch verschiedene geometrische Modelle beschrieben werden, unter anderem durch ein polygonales Netz (im folgenden auch Mesh). Die Grundlage für ein solches Mesh sind Polygone. Für eine Menge $\{V_0, ..., V_n\}$ von Punkten im zwei- oder dreidimensionalen Raum nennt man die Menge $Q = \{(V_0, V_1), ...(V_{n-1}, V_n)\}$ ein Polygon. Die Paare aus Punkten werden dabei als Kanten bezeichnet. Liegen alle Kanten eines Polygons in einer Ebene, so nennt man das Polygon planar.
Eine Menge von planaren, geschlossenen Polygonen, das heißt für jedes Polygon gilt $V_0 = V_n$, nennt man polygonales Netz, falls folgende Kriterien erfüllt sind:

1. Je zwei Polygonen haben entweder keinen Punkt oder eine Ecke oder eine ganze Kante gemeinsam.

2. Jede Kante eines Polygons gehört zu einem oder höchstens zwei Polygonen.

Außerdem umschließt jedes Polygon eine Fläche, die auch als Face oder Facette bezeichnet wird. Da der Oberfläche, also das Netz aus Polygonen, neben den Raumkoordinaten auch weitere Angaben wie zum Beispiel Informationen über Texturkoordinaten, Farbe oder Normalen (zur Beleuchtungsrechnung) zur Verfügung stehen sollen, werden solche Informationen als Vertex zusammengefasst.

Das Mesh besteht folgend aus einem Graphen, dessen Knoten Vertex genannt werden. Die durch Kanten verbundenen Vertices bilden ein Netz und repräsentieren somit die Oberfläche [6]. Eine weit verbreitete Form des Polygonnetzes ist das Dreiecksnetz, wel-

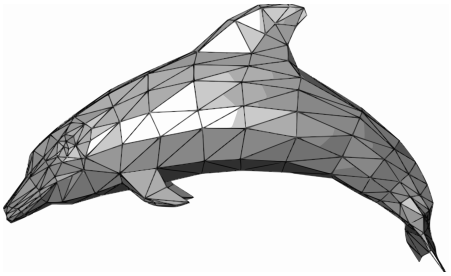

Abbildung 2.3: Beispiel eines Polygonalen Netzen. Der abgebildete Delphin besteht aus einer Vielzahl von Dreiecken.
Quelle: https://en.wikipedia.org/wiki/Polygon_mesh (Letzter Zugriff 15.03.2019)

ches die zu repräsentierende Oberfläche aus einer Vielzahl aneinander liegender Dreiecke beschreibt und verschiedene Vorteile mit sich bringt, zum Beispiel sind Dreiecke immer planar. Die Dreiecke können auf verschiedene Arten gespeichert werden. In dieser Ausarbeitung werden die Dreiecke in Listen verwaltet.

2.5 Augmented Reality

Augmented Reality (kurz AR) ist ein Teilbereich der Mixed Reality. Mit AR bezeichnet man verschiedene Displaytechnologien, die die Realitätswahrnehmung computergestützt erweitern. [10]

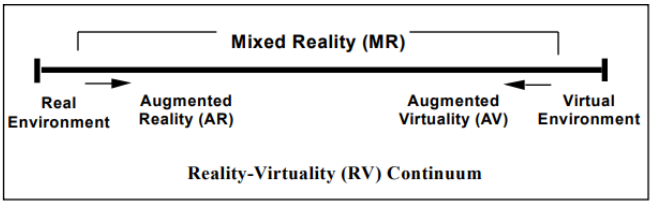

Abbildung 2.4: Das Reality-Virtuality Continuum
Quelle: https://medium.com/@marknb00/what-is-mixed-reality-60e5cc284330 (Letzter Zugriff: 12.03.2019)

Billinghurst et al. diskutieren Charakteristiken und Herausforderungen beim Planen von AR-Systemen.[5]

2.5.1 Immersion

In den meisten Fällen erwartet man, dass AR-Systeme die Realität erweitern indem sie dieser etwas hinzufügen, aber selten durch das Entfernen von realen Komponenten. Als Beispiel wird das digitale Zeichnen auf eine reale Wand genannt, bei der man ein Objekt, das vor der Wand steht, entfernen könnte. Außerdem zielen AR-Systeme oft die visuelle Wahrnehmung ab, sollten jedoch auch die Möglichkeit in Betracht ziehen andere Sinne einzubeziehen. Haptisches Feedback durch Handschuhe oder Rauschentfernen eingehender Geräusche durch Kopfhörer [22] sind beispielhafte Möglichkeiten.

2.5.2 Optisch vs. Video

Eine weitere Herausforderung ist die Art des Kombinierens reeller und virtueller Komponente. Dabei werden optical see-through und video see-through Konzepte gegenübergestellt. Bei ersterem handelt es sich um die Technologie mit Monitoren zu arbeiten, die die Möglichkeit besitzen projizierte Bilder zu reflektieren und gleichzeitig lichtdurchlässig zu sein. So sieht der Nutzer die reale Welt und gleichzeitig die virtuelle Komponente. Mit einem monochromen Monitor wäre dies ideal, aber in der Praxis reduzieren die meisten Monitore einen Teil des Lichts, das sie hindurchlassen. Bekannt dafür sind zum Beispiel Head-mounted Displays. Video see-through Systeme kombinieren stattdessen Monitore mit Kameras, indem sie eine Aufnahme der realen Welt aufzeichnen und diese zusätzlich

mit der virtuellen Komponente auf dem Monitor darstellen, wie es Smartphones und Tablets tun.

2.5.3 Fokus und Kontrast

Bei der Version der optical see-through Displays wird immer auf dieselbe Distanz zum Auge des Nutzers projiziert, während die reelle Welt verschiedene Distanzen aufweist, wodurch der Fokus des Nutzers auf einer der beiden Komponenten verloren gehen kann. Außerdem sollten virtuelle und reelle Szenen eine ähnliche Helligkeit aufweisen, da bei heller realer Umgebung und dunkler virtueller Komponente, die virtuelle Komponente ausgeblendet wird, oder im Fall dunkler realer Umgebung und heller virtueller Komponente, die virtuelle Komponente den realen Anteil überblendet. Video see-through Systeme haben dieses Problem typischerweise nicht.

3 Animation

Eine Animation im klassischen Sinne beschreibt die Technik, durch eine Abfolge von Einzelbildern, ein scheinbar bewegtes Bild zu schaffen. Einzelbilder stellen das elementare Medium dar und werden auch als Frame bezeichnet.
Überträgt man diese Idee auf Computeranimationen, werden durch verschiedene Techniken scheinbar bewegte Objekte oder bewegte Szenen erstellt. Zwei geläufige, im folgenden näher erläuterte Animationstechniken sind die Keyframe-Animation und die Skelett-Animation. Einen breiten Überblick bezüglich Animationstechniken der Computergrafik bietet die Quelle [24].

3.1 Keyframe-Animation

Bei der klassischen Keyframe-Animation werden von einem Hauptzeichner besonders wichtige Bilder, sogenannte Keyframes, erstellt. Ein Zwischenphasenzeichner erstellt im nächsten Schritt die Zwischenbilder, auch Interframes oder Inbetweens genannt, die zwischen einem Keyframe und seinem folgenden Keyframe liegen. Mithilfe dieser Technik wurden unter anderem Zeichentrickfilme erstellt.

In der Computergraphik werden analog zur Zeichnung Keyframes für das zu animierende Objekte erstellt, indem Parameter, zum Beispiel Raumkoordinaten, verändert werden. Die Parameter der Inbetweens werden dann interpoliert. Für die Interpolation gibt es verschiedene Verfahren, zum Beispiel lineare oder sphärische Interpolationen oder durch Interpolationen durch Splines.

Zwei wesentliche Nachteile von Keyframe-Animationen sind zum einen die aufwändige Erstellung der einzelnen Objekt-Keyframes, da jeder Vertex eines Polygonnetzes neu parametrisiert werden muss und zum anderen ein hoher Speicherbedarf.

3.2 Skelett-Animation

Bei der Skelett-Animation besteht das zu animierende Objekt aus zwei Komponenten. Zum einen aus dem Polygonnetz zur Oberflächenrepräsentation, und zum anderen aus einem Skelett, welches zur Animation des Objektes dient.

Das Skelett selbst besteht aus einem hierarchischen Baum, dessen Knoten man als Knochen (des Skeletts) bezeichnet. Die Wurzel des Baumes ist ein Anker, an dem die Unterbäume hängen und der sich nicht ändert. Jeder Knochen wird dabei aus einem Startpunkt, Endpunkt und einem eigenen Koordinatensystem beschrieben.
Das Koordinatensystem setzt sich aus einer Rotation und einer Translation zusammen, der Startpunkt des Knochens ist abhängig von der Transformation des Koordinatensystems des Elternknochens und der Endpunkt ist abhängig vom Startpunkt und einer Verschiebung vom Startpunkt [20]. Das Skelett wird im Rigging-Prozess erstellt und im Skinning-Prozess mit der Oberfläche verbunden, sodass das Deformieren des Skeletts auch auf die Oberfläche übertragen wird (dieser Vorgang wird als Blending bezeichnet). Die Knochen dienen dabei als Animation-Controller. In der Praxis werden die Animations-Controller des Skeletts auch als Gelenke bezeichnet, dabei ändert sich lediglich die Bezeichnung.

Abbildung 3.1: 3D-Modell mit einem Skelett (in blaue)
Quelle: https://de.wikipedia.org/wiki/Rigging_(Animation) (Letzter Zugriff: 01.02.2019)

Vorteile der Skelett-Animation sind, dass Animationen über die Bewegung des Skeletts definiert werden und nicht das gesamte Polygonnetz parametrisiert werden muss, wie bei den Keyframe-Animationen. Dies vereinfacht die Arbeit von Animationskünstlerin und bietet Raum für komplexere Animationen. Zudem kann die Skelett-Animation mit Motion-Capture Verfahren kombiniert werden, um Animationen zu erstellen.

3.2.1 Skinning

Als Skinning werden die zwei Schritte bezeichnet, in denen das Polygonnetz mit dem Skelett verbunden wird und die Bewegungen des Skeletts dann auf die Oberflächendarstellung übertragen wird. Dazu bringt man das Objekt in eine Default-Position, die sogenannte Rest-Pose oder Bind-Pose. In dieser werden Vertices und Knochen miteinander verbunden, indem eine Vertex-Knochen-Gewichtung definiert wird. Die Gewichtung legt den Einfluss eines Knochens auf einen Vertex fest. Diese Vertex-Knochen-Gewichte werden auch als Blending-Weights bezeichnet.

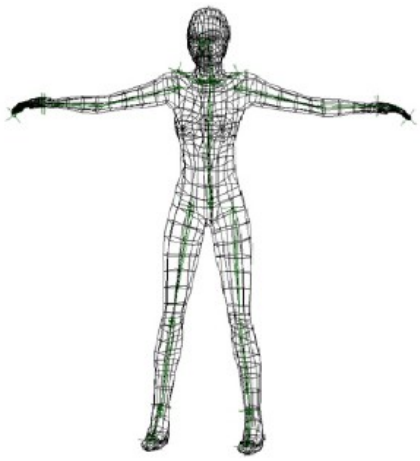

Abbildung 3.2: Rest-Pose eines humanoiden Charakters mit (in grün) Skelett
Quelle: http://www.cveld.net/modules/xpwiki/589.html (Letzter Zugriff 04.02.2019)

3.2.2 Nächster Knochen

Ein Ansatz ist es jedem Vertex genau einen, und zwar den nahsten, Knochen zuzuweisen. Dazu wird der kleinste Abstand von Vertex zu Knochen berechnet. Wichtig dafür ist,

dass eine sinnvolle Rest-Pose erstellt wird. Die Arme eines humanoiden Charakters sind beispielsweise in der Rest-poste ausgestreckt. Hingen die Arme am Körper herunter, könnten Torso-Vertices an die Armknochen gebunden werden.

Für eine bessere Animationsqualität bietet es sich jedoch an, die Vertices an mehrere Knochen zuzuweisen.

3.2.3 Wärmegleichgewicht (Pinocchio)

Bei dieser Methode betrachtet man das 3D-Objekt als isoliertes, wärmeleitendes Volumen. Man iteriert über jeden Knochen und setzt die Temperatur des aktuellen Knochens auf 1, die aller anderen Knochen auf 0. Der aktuelle Knochen strahlt die Wärme auf die Oberfläche des 3D-Objektes aus und es wird die Gleichgewichtstemperatur an jedem Vertex des Polygonnetzes berechnet.

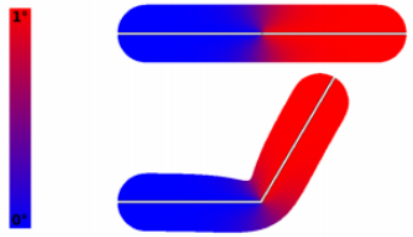

Abbildung 3.3: Gleichgewichtstemperatur des rechten Knochens auf ein Arm ähnliches 2D-Modell [3]

Die genaue Berechnung der Gleichgewichtstemperatur kann in [3] nachgeschlagen werden.

3.2.4 Bounded Biharmonic Weights

Jacobson et al. definieren die Blending-Weights indem sie mit ihrer Methode die die laplacesche Energie, mit Ober- und Unterbegrenzung, minimieren und die Berechnung der Gewichte automatisieren. Diese Methode kann man für verschiedene Blending-Techniken verwenden, im folgenden werden fünf von Jacobson et al. formulierte Kriterien im Kontext der Skelett-Animation erläutert.

Keine Negativen Gewichte: Frameworks und Algorithmen zur automatisierten Berechnung der Gewichte, die ebenfalls biharmonische Funktionen nutzen (siehe [**?**])),

führten teilweise zu negativen Gewichten. Negative Gewichte müssen manuell korrigiert werden, da sich die zu animierende Form andernfalls entgegengesetzt zur gewünschten Richtung bewegt.
Form-bewusst: Die Deformation durch das Skelett soll besondere Eigenschaften der Form beibehalten.
Einheitlichkeit: Wird dieselbe Transformation T durch alle Knochen angewendet, so wird die gesamte Oberfläche mit T transformiert.
Lokalität: Jeder Knochen beeinflusst nur lokale Eigenschaften. Liegen zwischen einem Vertex v und einem Knochen K_j andere Knochen die näher am Vertex v sind, wird der Knochen K_j ausgeschlossen, das heißt das Gewicht w_j für den Vertex v ist 0.
Keine lokalen Maxima: Jedes Gewicht w_j soll sein globales Maximum (wenn gewichtet den Wert 1) am Knochen K_j und keine anderen lokalen Maxima besitzen.

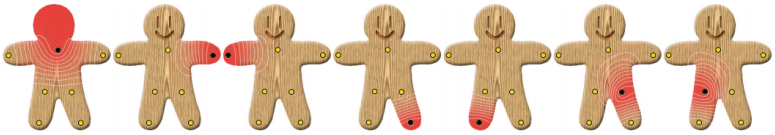

Abbildung 3.4: Mit biharmonischen gewichtetes 2D Objekt. Die Punkte sind äquivalent zu Knochen eines Skeletts. Jeder Punkt hat positive Gewichte, ist lokal und hat nur ein einziges Maxima, sodass bei der Deformation keine Extremen auftreten [13]

Bounded Biharmonic Weights erfüllen diese Kriterien.[13]

3.2.5 Manuelles Skinning

Um eine hohe Animationsqualität zu erzielen, werden die Blending-Weights häufig von Hand eingestellt. Software zum Modellieren von 3D-Objekten und Animationen bietet Künstlern und Entwicklern in den meisten Fällen die Möglichkeit einzelne Knochen des Skeletts auszuwählen und die Gewichte auf die Oberflächenrepräsentation zu malen.

3.3 Blending

Als Blending bezeichnet man das Deformieren der Oberflächenrepräsentation durch Bewegen des Skeletts, somit ist das Blending ein Unterschritt des Skinning. Dazu gibt es

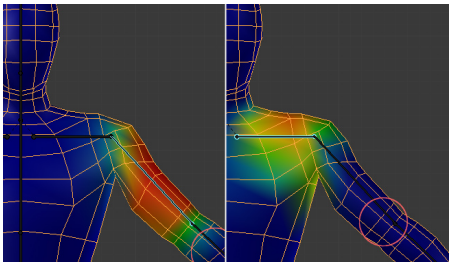

Abbildung 3.5: Humanoider Charakter, der ausgewählte Knochen ist in türkis markiert. Eine rote Färbung der Oberfläche repräsentiert hohe Gewichte für die Vertices zum markierten Knochen.
Quelle: https://cellcode.us/quotes/low-tutorial-ninja-poly-blender.html (Letzter Zugriff: 02.03.2019)

einige Methoden, die sich als Standard durchgesetzt haben und Vor- und Nachteile bieten.

3.3.1 Linear Blend Skinning (LBS)

Das Linear Blend Skinning ist in der Fachliteratur auch unter anderen Namen bekannt, zum Beispiel Matrix Palette Skinning oder Vertex Skinning, und ist die Standard-Blending-Methode bei Skelett-Animationen.

Vorausgesetzt die Summe aller Blending-Gewichte w sei normiert, das heißt es gilt $\sum_{i=0}^{n} w_i = 1$ und $w_i \geq 0$, also dass die Summe aller Blending-Gewichte eines Vertex eins ergeben und größer null ist.

Die Idee ist es die Raumkoordinate eines jeden Vertices mit Hilfe der Transformationen der ihn beeinflussenden Knochen zu bestimmen, indem man über diese Transformationen linear interpoliert. Das Mesh ist dazu in der Bind-Pose.

Jeder Vertex v wird im ersten Schritt mit der inversen Bind-Pose-Matrix eines Knochen B in das Weltkoordinatensystem des Wurzelknochens verschoben. Die Bind-Pose-Matrix ist das Resultat des Rigging-Prozesses und bereits bekannt. Im nächsten Schritt multipliziert man den Vertex mit der mit w_B gewichteten Transformationsmatrix T_B des Knochens B in Animations-Pose in das Koordinatensystem des Knochens zurück. Die Linearkombination für alle Knochen von $i = 0$ bis $i = N$ ergibt dann die neue Vertexposition: $v' = \sum_{i=0}^{N} w_i T_B(i) B(i)^{-1} v$.

Ein Vorteil dieser Methode ist die Einfachheit, da der Algorithmus leicht zu implementieren und nachzuvollziehen ist. Der große Nachteil ist jedoch das Auftreten von unerwünschten Artefakten in der Rotation.

Candy-Wrapper-Effekt

Die resultierende Matrix die den Vertex v transformiert, ist nicht mehr starr (euklidisch, rigid) sondern eine affine Transformation, kann also möglicherweise Skalierung und Scherung beinhalten. Dadurch kann bei der Animation ein Volumenverlust an den Gelenken des Skeletts entstehen, der auch als Candy-Wrapper Effekt bekannt ist. Durch Umschreiben der Skinning-Gleichung, indem man erst die Linearkombination bildet und sie dann auf den Vertex anwendet, erkennt man das eigentliche Problem: $v' = (\sum_{i=0}^{N} w_i T_B(i) B(i)^{-1}) v$. Durch komponentenweises Interpolieren wird die Orthogonalität der Rotation nicht beibehalten, manchmal selbst nicht der Rang. Genau das passiert auch beim Candy-Wrapper-Effekt.

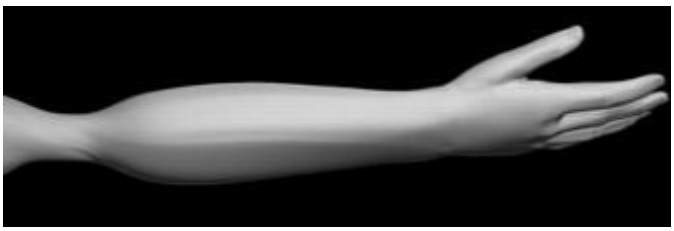

Abbildung 3.6: Candy-Wrapper Effekt: Volumenverlust an einem Gelenk
Quelle: http://imagine.inrialpes.fr/people/Francois Faure/htmlCourses/PoseSpaceDeformation.html (Letzter Zugriff: 02.02.2019)

3.3.2 Euler Angles Skinning (EAS)

Eulerwinkel eignen sich um Rotation zu repräsentieren und werden ebenfalls zum Blending genutzt. Diese Methode bietet den Vorteil den Candy-Wrapper-Effekt zu umgehen, da keine Scherungen oder Skalierungen auftreten, außerdem ist eine Rotationsdarstellung als Rotation eines Winkels um eine Achse leicht verständlich. In der Praxis rechnet man die Eulerwinkel in Rotationsmatrizen um und konkateniert diese, um eine komplexe Rotation zu erstellen. Für jeden Vertex kann man nun über die Rotationen der ihn

beeinflussenden Knochen interpolieren.

Neben dem Gimbal-Lock und der nicht-identischen Drehungsreihenfolge ist ein weiterer Nachteil von Eulerwinkeln, dass sie nur Rotationen darstellen, sodass für die Transformationsmatrix der Knochen die Translation zusätzlich erstellt und verwaltet werden muss.

3.3.3 Spherical Blend Skinning (SBS)

Eine Blending-Methode die Effekte wie den Candy-Wrapper-Effekt behebt ist das Spherical Blend Skinning. Der Rotationsteil der Transformationsmatrix im LBS sorgt für den Volumenverlust und genau den adressiert das SBS. Hierbei wird der Rotationsanteil der Transformationsmatrix nicht linear interpoliert, sondern nach Shoemake sphärisch mit Einheitsquaternionen (QLERP). [25]

Die erste Herausforderung liegt darin, das richtige Rotationszentrum zu finden, da sich andernfalls ein Drift in das Ergebnis einschleichen kann. Das Rotationszentrum ist abhängig von der Transformation der entsprechenden Knochen und somit kann das gleiche Rotationszentrum für alle Verticis genutzt werden, die von den gleichen Knochen deformiert werden.

Zwischen dem sphärischen linearen Interpolieren und QLERP besteht weiterhin ein Unterschied. Während SLERP auf einem Bogen interpoliert, interpoliert QLERP auf dem kürzesten Segment zwischen zwei Quaternionen und projiziert das Ergebnis dann auf den Bogen. Hierdurch kommt es zu einem kleinen Unterschied in beiden Interpolationsmethoden, der aber vernachlässigt werden kann. Das nutzen von QLERP statt SLERP ist mit einer besseren Performance begründet [15].

Der Blending-Algorithmus funktioniert so, dass im ersten Schritt der Rotationsanteil aller Knochen-Transformationsmatrizen, die den Vertex deformieren, in Quaternionen umgeformt werden. Folgend wird das Rotationszentrum berechnet und die Quaternionen werden linear interpoliert, das Ergebnis wird dann in eine Matrix umgewandelt und mit dem Vertex verrechnet.

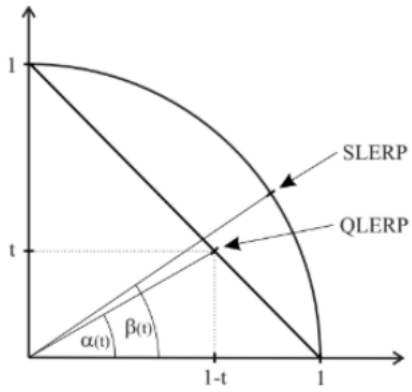

Abbildung 3.7: Beispielhafte Demonstration der Abweichung der Winkel zwischen den Interpolationsmethoden SLERP und QLERP

3.3.4 Dual-Quaternion Blend Skinning (DQS)

Das Dual-Quaternion Skinning, auch Dual-Quaternion Blending, ist dem Linear Blend Skinning in seinem Algorithmus ähnlich, lediglich die Berechnung auf dem Vertex v ändert sich und erfolgt mit Einheits-dualen Quaternionen d_q, die sich aus dem einem Translations- und Rotationsquaternion zusammensetzen, die im Kapitel Grundlagen näher beschrieben wurden.

Mit normierten Gewichten w_i ergibt sich folgende Skinning-Formel:

$$v' = v * \frac{\sum_{i=0}^{N} w_i \hat{q}}{||\sum_{i=0}^{N} w_i \hat{q}||}$$

Abbildung 3.8: Links eine Deformation mit LBS: Die Drehung an der Schulter führt zu einem Volumenverlust. Bei der rechten Deformation mit DQS tritt dieser Effekt nicht auf. Quelle: [16]

Direkte Vorteile gegenüber den bisher erläuterten Skinning-Methoden sind, dass ein Gimbal-Lock mit dualen Quaternionen nicht möglich ist und deutlich weniger Artefak-

te, wie der Candy-Wrapper Effekt, während der Animation auftreten, da Quaternionen keine Scherungen oder Skalierungen ermöglichen.[16]
Außerdem kann diese Technik mit dem selben Datensatz des Linear Blend Skinnings verwendet werden.

DQS - Wölbungen

Ein Nachteil des Blendings mit Dualen Quaternionen, ist dass manchmal Wölbungen an den Gelenken als Artefakte auftreten können. Kim und Hun formulieren deshalb eine Methode, um diese Wölbungen zu entfernen.
Im ersten Schritt berechnet man die Entfernung eines jeden Vertices zum nächsten Knochensegment, während das Mesh in der Rest-Pose ist, und speichert diese Entfernung. Diese Entfernung wird als dv_{rest} bezeichnet. In der Regel ist dies auch der Knochen mit dem höchsten Vertex-beeinflussenden Gewicht. Während der Animation führt man diese Kalkulation weiterhin durch, im weiteren $dv_{current}$ und vergleicht das Ergebnis mit der initialen Berechnung. Sollte $dv_{current} > dv_{rest}$ gelten, so wird der Vertex in die Richtung des nächsten Knochensegments mit einem Faktor von $\frac{dv_{rest}}{dv_{current}}$ projiziert.[18]

3.3.5 Stretchable and Twistable Bones for Skeletal Shape Deformation (STBS)

Jacobson et al. [14] formulieren das Problem, dass die bisher vorgestellten Skinning-Techniken die qualitativen Anforderungen von Streckungen in einer Animation nicht erfüllen. Streckt man einen Knochen wirkt sich die Streckung auf die alle betroffenen Vertices des Polygonnetzes aus und die Streckung resultiert in einem unnatürlichen Ausbruch der Oberfläche. Außerdem löst das DQS den Candy-Wrapper-Effekt, aber die Rotation konzentriert sich noch immer an den Gelenken. Das Volumen fällt zwar nicht gegen einen Punkt zusammen, kann aber unnatürlich wirken.
Deshalb schlagen Jacobson et al. vor zusätzlich zu den Vertex-Knochen-Gewichten für jeden Knochen Endpunkt-Gewichte einzuführen. Das Besondere an dieser Methode ist, dass sie Techniken wie LBS oder DQS ergänzt. Während die Vertex-Knochen-Gewichte normiert sind, dürfen die Endpunkt-Gewichte eines Knochens auch nicht normierte Werte < 1 annehmen.
Setzt man die Endpunkt-Gewichte $e(v)$ in die aus dem LBS bekannte Skinning-Gleichung ein ergibt sich eine erweiterte Skinning-Gleichung $v' = \sum_{i=0}^{N} w_i(T_B(e_i(v)) + R_B(e_i(v)))B(i)^{-1}v$,

mit der Transformationsmatrix des Knochens aufgeteilt in den Translationspart T_B und den Rotationspart R_B. Die Vertices v werden vor der Multiplikation mit den Matrizen mit den Endpunkt-Gewichten e multipliziert. Erweitert man die DQS-Gleichung ergibt sich äquivalent zur erweiterten LBS-Gleichung $v' = \sum_{i=0}^{N} w_i * (\hat{d}_q * ((e) * v) * \hat{d}_q^*)$.

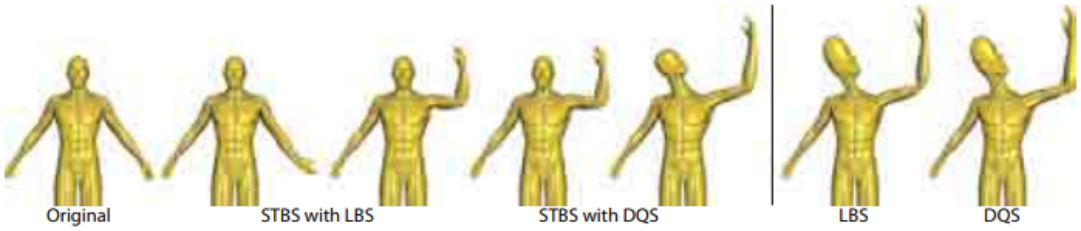

Abbildung 3.9: Ein humanoides Model ist mit Biharmonic Weights und Endpoint Weights an ein Skelett geskinnt. DQS sorgt dafür, dass keine Volumenverluste stattfinden. STBS verhindert außerdem den Ausbruch an Hand und Kopf des Models. Quelle: [14]

3.3.6 Spline-based Skinning

Ein weiterer Skinning-Ansatz geht einen ganzen Schritt zurück und überarbeitet das Datenmodell des Skeletts. Die Knochen eines Skeletts werden beim Spline-based Skinning nicht mehr durch eine Transformation oder End und Startpunkt, sondern durch ein Spline repräsentiert.

Um also Spline-basiert zu deformieren, muss eine passende Spline gewählt werden.Forstmann et al.schlagen aus GPU-Performancegründen vor Bezierkurven mit drei Kontrollpunkten $p1$, $p2$ und $p3$ zu benutzen und modifizieren die konventionelle Bezierkurve, um die Steifheit der Kurve zu beeinflussen, ohne einen weiteren Kontrollpunkt hinzufügen zu müssen.[12]

Konventionelle Bezierkurve:
$f_b(x) = p_1(1-x)^2 + p_2 * (2x(1-x)) + p_3x^2$
$f_b'(x) = p_1 2(x-1) + p_2(2-4x) + p_3 2x$

Modifizierte Bezierkurve:
$f_m(x) = p_1 + \Delta_{12}(1-(1-x)^a) + \Delta_{23}x^a$
$f_m'(x) = \Delta_{12}a(1-x)^{a-1} + \Delta_{23}ax^{a-1}$

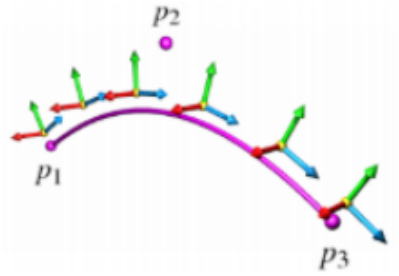

Abbildung 3.10: Koordinatensystem entlang einer Spline. Quelle: [12]

Außerdem muss ein vollständiges Koordinatensystem für die Spline erstellt werden. Die Berechnung der orthonormalen Basis für jeden Punkt auf der Spline kann in [12] nachgelesen werden.

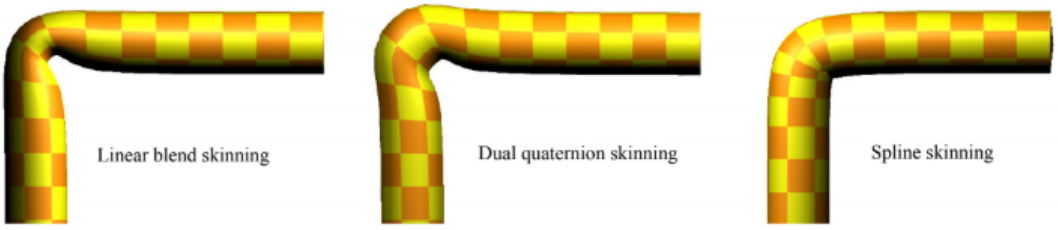

Abbildung 3.11: Muskelkontraktion durch Spline-basierte Deformation

Beim Rigging gibt es in diesem Verfahren einen Unterschied. Vertices werden nicht mehr an die Knochen, in diesem Fall Splines, gebunden, sondern an genaue Punkte auf den Knochenelementen. Dazu wird für jeden Vertex eine Binärsuche in der Mitte einer Spline gestartet, die dann endet, wenn der Vertex auf der Ebene liegt die vom Koordinatenursprung der Spline und der Tangente am entsprechenden Punkt aufgespannt wird. Das Blending selbst wird durch lineares Interpolieren über alle beeinflussenden Splines eines Vertex durchgeführt. Vorteile dieser Methode sind, dass die Ergebnisse einer Deformierung unabhängig von der Oberflächenrepräsentation sind und dass Rundungen und Biegungen dargestellt werden können, wie zum Beispiel bei der Muskelkontraktion eines humanoiden Charakters. Besonders gut sieht man die natürlichen Rundungen im Vergleich vom Spline-based Skinning zu LBS oder DQS.

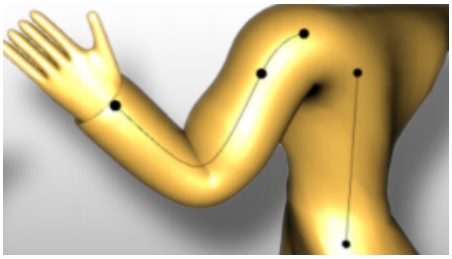

Abbildung 3.12: Muskelkontraktion durch Spline-basierte Deformation. Quelle: [12]

3.3.7 Example-based Skinning

Um Artefakte wie den Candy-Wrapper-Effekt zu vermeiden und Deformationen wie eine Muskelkontraktion zu berechnen, stellen Mohr und Gleicher ein weiteres Skinning-Model vor. Hierbei wird dem Skelett an den Stellen weitere Knochen hinzugefügt, an denen beim LBS starke Artefakte auftreten.
Um die neuen Knochen zu berechnen wird zwischen der Rotation eines Knochens in animierter Pose und der Rotation eines Knochens in Bind-Pose sphärisch linear interpoliert [25]. Die Position eines Knochens liegt dann bei der Hälfte dieses Ergebnisses. Mit diesem neuen Satz Knochen wird ein weiteres Skelett erstellt, ein sogenanntes Example. Auch das Skelett in Bind-Pose und das Skelett in Animations-Pose gelten als Example.
In einem weiteren Schritt werden Skalierungs-Knochen mitsamt Parametern hinzugefügt, um Effekte wie Muskelkontraktionen darzustellen. Zuerst wird ein Knochen ausgesucht, der als Driver bezeichnet wird.
Von diesem Knochen hängt es ab, wie sich die Skalierungsparameter ändern. Im nächsten Schritt bekommt der Treiber zwei Knochen-Sätze, von denen einer als Upstream-Knochen und der andere als Downstream-Knochen bezeichnet wird. Der Upstream-Satz liegt in der Mitte zwischen Treiber und seinem Elternknochen und der Downstream-Satz in der Mitte zwischen Treiber und seinem Kindknochen.

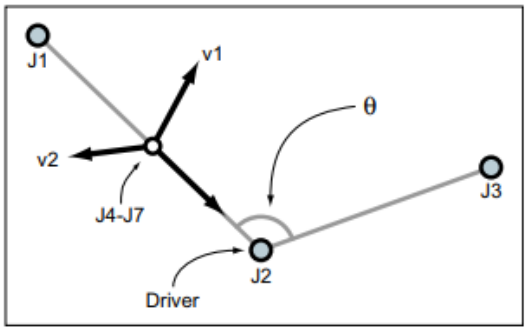

Abbildung 3.13: Ein Beispiel für den Upstream-Satz zwischen Driver J2 und seinem Elternknochen J1. Am markierten Punkt liegen die Knochen J4-J7. Der Vektor v1 liegt orthogonal zu den Knochen J1 und J2, der Vektor v2 liegt orthogonal zum Knochen J2 und zum Vektor v1. J4 skaliert in Richtung v1, wenn der Winkel θ kleiner wird und J5 skaliert in Richtung v2, wenn der Winkel θ kleiner wird. Die Knochen J6 und J7 verhalten sich ähnlich, skalieren aber in entgegengesetzte Richtung. Downstream-Knochen liegen zwischen J2 und J3.
Quelle: [21]

Im nächsten Schritt muss ein neuer Satz beeinflussender Knochen für jeden Vertex und ein entsprechendes Gewicht berechnet werden. Dazu wird die für jeden Vertex die Position an allen Knochen der Examples Berechnet und die Positionen gleicher Knochen werden zu einer Punktwolke zusammengefasst. Je kleiner die Punktwolke eines Vertex-Knochen-Paares ist, desto einflussreicher ist dieser Knochen auf den Vertex.

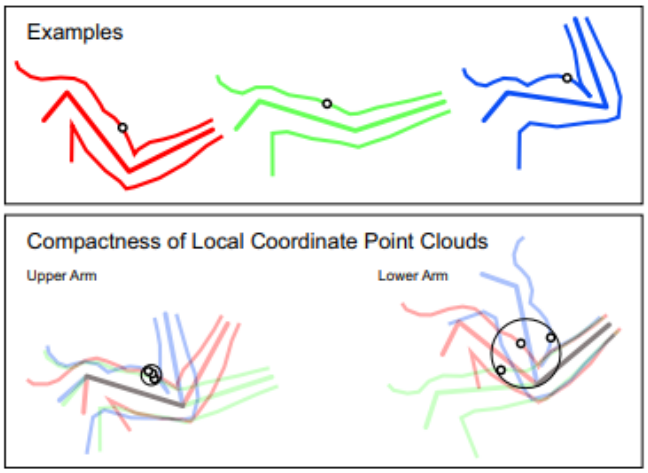

Abbildung 3.14: Im oberen Bild ist derselbe Vertex an drei Examples abgebildet. Im unteren Bild links ist die Punktwolke dieses Vertices in Relation zum Oberarmknochen und rechts die Punktwolke dieses Vertices in Relation zum Unterarmknochen abgebildet. Aus dem Vergleich der Punktwolken-Durchmesser folgt, dass der Oberarmknochen einen größeren Einfluss auf den Vertex hat, als der Unterarmknochen.
Quelle: [21]

An der Abbildung 3.15 erkennt man den Unterschied dieser Skinning-Technik zum Linear Blend Skinning. Es können gezielt natürliche Deformationen erzeugt werden, der Overhead liegt jedoch in einer neuen Vorberechnung des Meshes und des Skeletts.

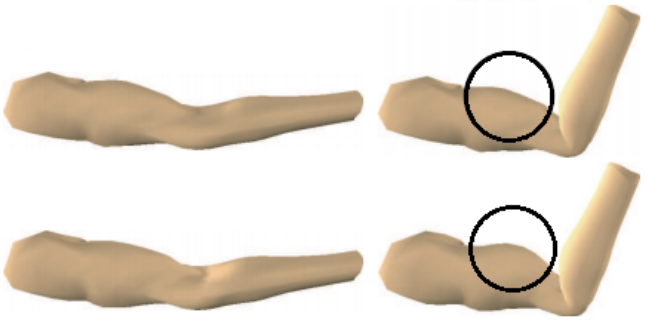

Abbildung 3.15: Der obere Arm wurde mit LBS deformiert und produziert keine Muskelkontraktion, der untere hingegen wurde mit Example-based Skinning deformiert und erzeugt eine Muskelkontraktion.
Quelle: [21]

3.3.8 Implicit Skinning

Vaillant et al. stellen eine Methode vor die auf Feldfunktionen und Impliziten Flächen basieren. Als Grundlage dient ein Mesh mit einem Skelett und Knochen-Gewichten. Das Mesh muss außerdem bereits in einzelne Mesh-Segmente unterteilt sein, die sich auf die Skelettstruktur beziehen.

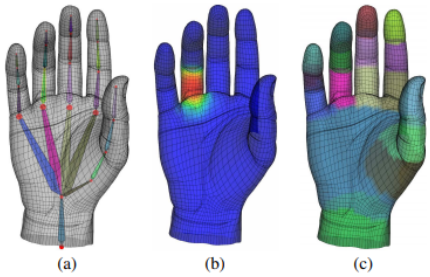

Abbildung 3.16: a) Ein Input-Mesh mit Skelett
b) Die Blending-Weights an einem Knochen
c) Die Mesh-Segmentation
Quelle: [26]

Aus diesem Input ausgehend wird zunächst für jedes Mesh-Segment eine Implizite Fläche berechnet, sodass so das Gesamt-Mesh approximiert wird. Mithilfe einer Vereingungsfunktion werden die Impliziten Flächen dann zusammengeführt.

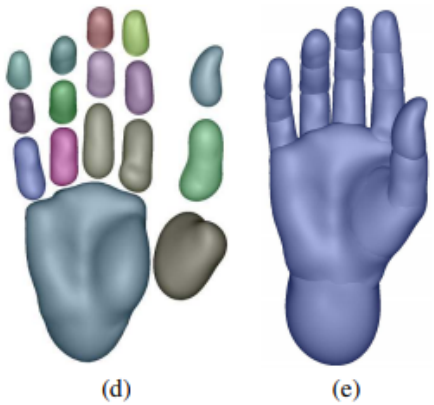

Abbildung 3.17: d) Die Impliziten Flächen der einzelnen Mesh-Segmente
e) Die Komposition der Impliziten Flächen durch die Vereinigungsfunktion

Quelle: [26]

Beim Deformieren der Vertices werden auch die Feldfunktionen mit transformiert, sodass ein Vertex solange entlang des Feldgradienten wandert, bis er seinen individuellen Isowert erreicht oder er auf den Feldgradienten eines anderen Vertices stößt. [26]

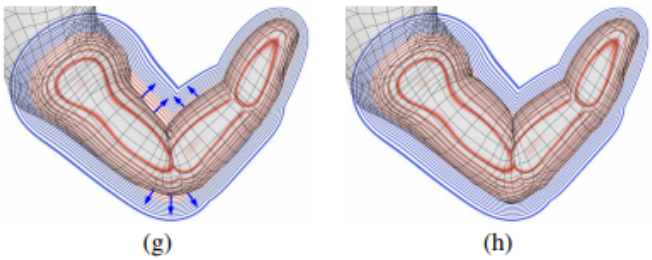

Abbildung 3.18: g) Die Vertices wandern (kleine blaue Pfeile) entlang der Feld-Gradienten
h) Das finale deformierte Mesh

Quelle: [26]

3.3.9 Weitere Eigenschaft

Obwohl das Skinning nicht nur auf organische Objekte abzielt, wird die Darstellbarkeit organischer Deformationen, zum Beispiel die Muskelkontraktion (siehe Spline-based Skinning), als ein wichtiges Merkmal hervorgehoben. Zum Beispiel entwickelten Nedel und Thalmann zu diesem Zweck einen Anatomie-basierten Ansatz, um humanoide Körper natürlich zu deformieren. Dabei erweitern sie ein Skelett um Muskeln, die aus Partikelsystemen bestehen. Die Interaktionen zwischen den einzelnen Partikeln basiert dabei auf einem Feder-System, sodass natürliche Deformationen von Muskeln möglich sind. [23]
Auch Wilhelms et al. nutzen ein mehrschichtiges Modell, welches sich für das Modellieren jeglicher Wirbeltiere eignet und die Datenstruktur aus Mesh, Skelett und Muskeln um Gewebe erweitert. Hier werden die Muskeln jedoch als deformierte Zylinder repräsentiert, deren Achse eine Kurve ist und den Ursprung in der Mitte des Zylinders hat. Der Zylinder spannt einen Bereich von zwei Einschubspunkten bis zwei Endpunkten auf. [27]

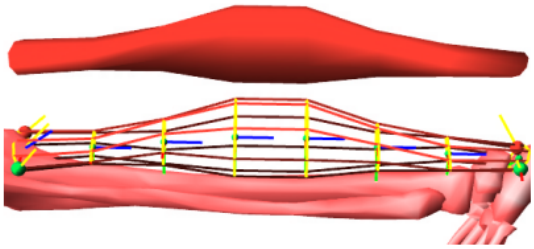

Abbildung 3.19: Die grüne und rote Kugel auf der linken Seite des unteren Bildes sind Einschubspunkte, rechts hingegen Endpunkte. Das obere Bild zeigt das daraus resultierende Muskel-Mesh.
Quelle: [27]

3.3.10 Performanz

Ein weiterer Wichtiger Aspekt ist die Performanz von Skinning-Techniken. Kavan et al. vergleich die Ergebnisse des Dual Quaternion Blend Skinnings unter anderem mit dem Log Skinning und dem Linear Blend Skinning [8] und stellen fest, dass das Linear Blend Skinning schneller berechnet wird. Dies wird damit begründet, dass das Linear Blend Skinning keine weiteren Kosten verursacht. Die Anzahl der Instruktionen steigt auf das fast doppelte an, wenn die Antipodalität der Quaternionen (dazu mehr in Kapitel 5) im Vertex Shader, eine programmierbare GPU-Komponente, die Operationen auf den

Vertices anwendet, gelöst wird. Forstmann et al. formulieren das Kriterium Vertices pro Sekunde und messen daran die Performanz verschiedener Deformations-Stile durch Splines. [12]

Ein weiteres relevantes Kriterium ist das Maß für die Bildfrequenz (Framerate), Bilder pro Sekunde (im Folgenden auch Frames per Second oder fps). Apteker et al. klassifizieren Videos über drei Merkmale in acht Klassen und lassen Probanden diese in 30, 15, 10 und 5fps ansehen.

	T_{lo}	T_{hi}
$A_{lo}V_{lo}$	Logo/test pattern	Station break
$A_{lo}V_{hi}$	Snooker	Sporting highlights
$A_{hi}V_{lo}$	Talk show	Advertisements
$A_{hi}V_{hi}$	Stand-up comedy	Music video

Abbildung 3.20: Die Video-Klassen nach Apteker et al.
Quelle: [1]

Dann untersuchen sie die Bereitschaft der Probanden die Videos anzusehen und stellen fest, dass die Reduzierung der Frames per Second auch die Bereitschaft des Zusehens senkt. Bei den Klassen mit dem Merkmal T_{lo} (siehe Abbildung 3.20) sinkt die Bereitschaft um Teilweise 50%, bleibt aber bei allen Klassen mit 30fps bei 100%.

	115 Kbps (5 fps)	200 Kbps (10 fps)	260 Kbps (15 fps)	480 Kbps (30 fps)
$T_{lo}A_{lo}V_{lo}$	90	89	91	100
$T_{lo}A_{lo}V_{hi}$	47	69	73	100
$T_{lo}A_{hi}V_{lo}$	50	74	84	100
$T_{lo}A_{hi}V_{hi}$	43	73	81	100

	165 Kbps (5 fps)	300 Kbps (10 fps)	350 Kbps (15 fps)	670 Kbps (30 fps)
$T_{hi}A_{lo}V_{lo}$	84	90	93	100
$T_{hi}A_{lo}V_{hi}$	60	80	86	100
$T_{hi}A_{hi}V_{lo}$	70	84	91	100
$T_{hi}A_{hi}V_{hi}$	64	81	86	100

Abbildung 3.21: Auffällig ist die teilweise niedrige Bereitschaft bei 5fps und die 100%ige Bereitschaft bei 30fps.
Quelle: [1]

Claypool et al. stellen die Wichtigkeit der Bildfrequenz in den Kontext von First Person Shooter Spielen. Probanden spielten bei den Untersuchungen verschiedene Quake 3 (First Person Shooter Videospiel) Konfigurationen für 30 Sekunden und bewerteten diese dann bezüglich der Spielbarkeit, Bildqualität und Anstrengung zu spielen. Für die Konfigurationen wurden fünf verschiedene Framerates von 3, 7, 15, 30 und 60fps und verschiedene Bildauflösungen von 640 x 840, 512x 384 und 320 x 240 gewählt. Zusätzlich gibt es Spiele-Karten in denen die Probanden die Figur bewegen müssen oder gegen einen virtuellen Gegner (Bot) spielen müssen.

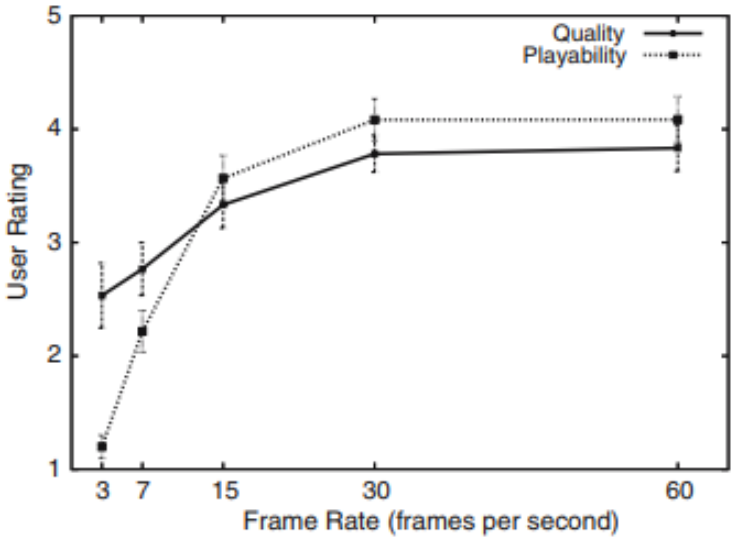

Abbildung 3.22: Auf der y-Achse: Die Nutzer-Bewertung und auf der x-Achse: Bildfrequenz.
Quelle: [7]

Die Ergebnisse zeigen, dass die Probanden die Spielfigur schlechter steuern, je niedriger die Framerate ist, da die Probanden mit sinkender Framerate eine längere Zeit brauchten, um eine Lauf-Karte abzuschließen. Bei den Karten mit dem virtuellen Gegner zeigen sich ähnliche Resultate.
Die Nutzer-Bewertung weist außerdem auf, dass sie abnehmend steigt, je höher die Framerate wird.[7]

4 Konzept

Nachdem nun die Grundlagen für verschiedene Skinning-Techniken geschaffen wurden, wird im folgenden Kapitel das Konzept für eine Anwendung vorgestellt, die eine Grundlage für die Evaluation verschiedener Skinning-Techniken mit einem AR-Kontext bietet.

4.1 Use-Case

Der Anwendungsfall ist die Animation in einer Augmented Reality Applikation mit einer User-Interaction, bestehend aus einer darstellenden und einer simulierenden Komponente. Um die Evaluation zu unterstützen wird ebenfalls eine Desktop-Applikation entwickelt, die dieselbe simulierende Komponente nutzt. Spezifische Herausforderungen die sich an Skelett-Animationen im AR-Kontext stellen, werden im Rahmen der AR-App betrachtet.

4.2 Anforderungen

Als nächstes werden die Anforderungen an die Software formuliert.

4.2.1 Darstellung in AR

Die Animation soll in Augmented Reality dargestellt werden. Hierfür ist ein AR-Framework notwendig.

4.2.2 Laden von Animationdateien

Da die Skinning-Techniken selbst implementiert werden, müssen die Rohdaten eines 3D-Objektes geladen werden. Hierzu werden, falls vorhanden, Bibliotheken genutzt und sonst wird ein eigenes Import-Programm geschrieben. Das Laden der Animationsdateien soll maximal 10 Sekunden dauern.

4.2.3 Framerate

Um eine flüssige Animation abzuspielen, soll eine Framerate von 30 Frames pro Sekunde erreicht werden (siehe Kapitel 4).

4.2.4 Animationsdateien

Es wird ein 3D-Objekt mit Animation beschafft, welches in der AR-App dargestellt wird. Außerdem werden zwei Zylinder mit jeweils zwei Animationen in einer Grafiksuite selbst erstellt. Die Zylinder unterscheiden sich in der Menge Vertices an den Gelenken. Eine Animation wird das Abknicken des Zylinders an der halben Höhe und die andere Animation eine Drehung um den Zylinder selbst sein.
Die Animationsdateien werden einen einheitlichen Datentyp besitzen.

4.2.5 Interaktion mit der Animation bzw. dem Objekt

Es soll die User-Interaktion ‘Folgen der Kamera’ mit dem animierten Objekt implementiert werden. Weiterhin sollen dem Nutzer Funktionen zur Verfügung stehen, um die Animation zu stoppen, zu starten, zu verlangsamen und zu verschnellern.
Das Programm soll zwei Skinning-Techniken implementieren und dem User die Möglichkeit geben zwischen diesen zu schalten. Die zwei Skinning-Techniken sollen sein:

1. Linear Blend Skinning: das LBS ist der Skinning-Standard und bildet die Grundlage in diesem Bereich. Diese Methode ist leicht verständlich und reicht für viele Anwendungen aus.

2. Dual Quaternion Blend Skinning: das DQS baut direkt auf dem LBS auf und kann auf denselben Datensatz angewendet werden und verspricht laut der Literatur eine wesentlich bessere Animationsqualität.

4.3 Projectpipeline

Die Schritte vom Spezifizieren bis hin zur Evaluation in den nachfolgenden Kapiteln sieht folgendermaßen aus:

1. Festlegen von Frameworks und Dateitypen
2. Beschaffung und Erstellen von Animationsdateien
3. Beschaffen oder Entwickeln eines Programms zum Importieren der Animationen
4. Entwickeln der konzipierten Anwendung
5. Evaluieren der Animationen
6. Diskussion der Evaluation

5 Umsetzung

Im folgenden Kapitel werden die einzelnen Schritte der Umsetzung des Konzepts näher und leicht reproduzierbar erläutert.

5.1 Verwendete Hardware

Samsung Galaxy S7 Edge: Beim Samsung Galaxy S7 Edge handelt es sich um ein im April 2015 veröffentlichtes Smartphone von Samsung, mit dem aktuellen Betriebssystem Android 8.0 Nougat. Das Device eignet sich bezüglich der Technik für AR-Anwendungen und wird von der HAW Hamburg gestellt. Da es sich um ein Android-Betriebssystem handelt, liegt es nahe in der Programmiersprache Java zu entwickeln. Das Smartphone wird benötigt, um die AR-Applikation zu nutzen.

5.2 Verwendete Software

Android Studio: Das Android Studio Framework aus der Vorlesung WP-CG von Herrn Prof. Dr. Jenke wird als Basis für die AR-Applikation genutzt und bildet die darstellende Komponente, da es bereits die Grafikbibliothek OpenGL implementiert und Entwicklern die Möglichkeit gibt, auf einer Abstraktionsebene über der Grafikbibliothek zu entwickeln. Android Studio basiert auf der IntelliJ IDEA Community Edition und eignet sich, um Android Applikationen zu schreiben. Im folgenden wird das Framework als WPCG-Framework bezeichnet.
Vuforia SDK: Das Vuforia SDK ermöglicht das Erstellen von markerbasierten und markerlosen AR Anwendungen für Mobile Devices durch Tracking-Verfahren. Es wird im Kontext dieser Implementierung genutzt, um die Animationen auf einem Marker darzustellen.
Blender: Blender ist eine freie Grafiksuite und ermöglicht das Erstellen von 3D-Objekten

und Animationen. Blender bietet auch die Möglichkeit erstellte Modelle in den gängigen Dateiformaten zu exportieren.

5.3 Gesamtarchitektur

Das Programm besteht aus drei Komponenten, die voneinander abgekoppelt sind und sich in das Framework einordnen. Eine Komponente ist für I/O-Funktionen zuständig und liest alle relevanten Animations-Informationen aus COLLADA-Dateien. Eine weitere Komponente führt nebenläufig die Simulation der Animation aus und eine dritte Komponente dient der Darstellung der Animation.

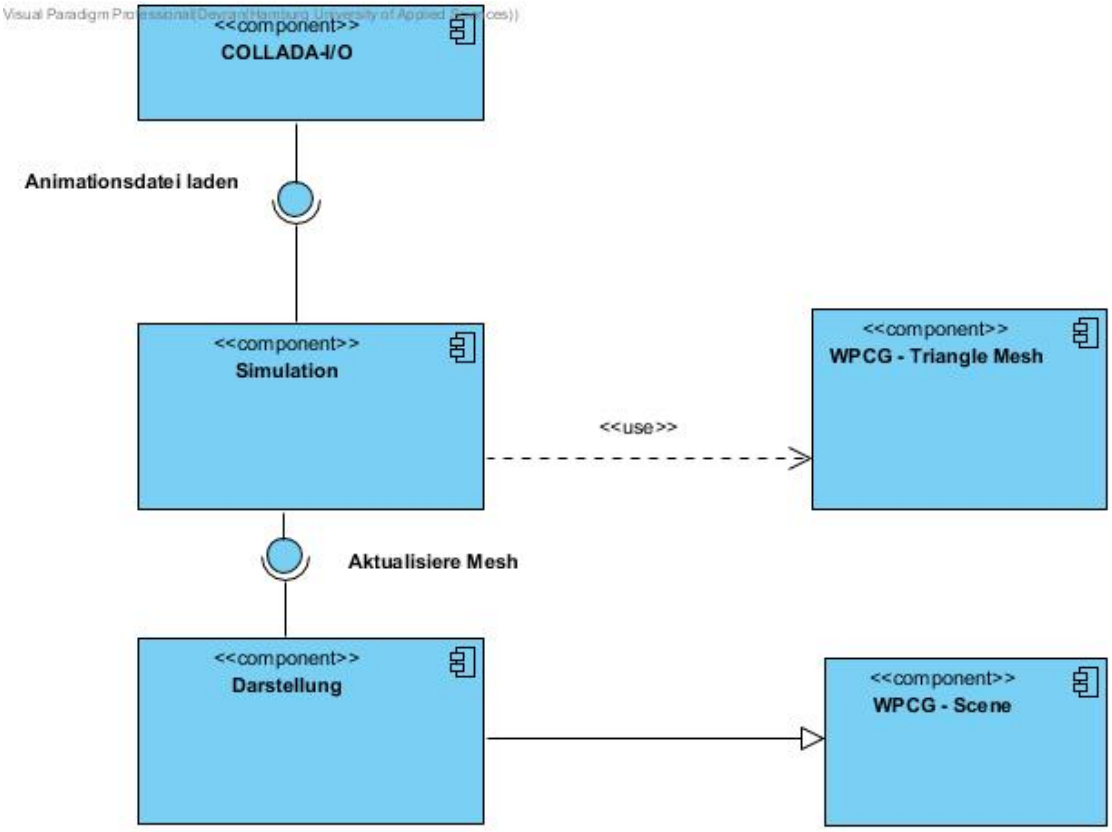

Abbildung 5.1: Die Software besteht aus drei Komponenten, von denen zwei Teile des WPCG-Frameworks nutzen.

5.4 COLLADA-I/O

Im folgenden Abschnitt wird erläutert wie man die Animations-Daten aus COLLADA-Dateien liest. Dem Import-Programm wurden außerdem eine Vielzahl an Funktionen hinzugefügt, um durch XML-Formate zu navigieren, da fertige Bibliotheksfunktionen teilweise nicht ausreichend waren.

5.4.1 COLLADA-Speicherformat

Das COLLADA (COLLAborative Design Activity) ist ein offenes Austauschformat für Daten verschiedener 3D-Programme und basiert auf XML. Die XML-Basis war der ausschlaggebende Grund den .dae (COLLADA) Datentyp zu wählen. [4] Zu COLLADA gibt es eine Spezifikation die zuletzt im April 2008 aktualisiert wurde. Die bekannten Grafiksuites, wie zum Beispiel Blender oder Maya, wie auch Entwicklungsumgebung aus der Spieleentwicklung, zum Beispiel Unity oder die Cryengine, implementieren das Dateiformat und ermöglichen es Grafikassets und Animationen zu importieren und zu exportieren. Außerdem gibt es mit ASSIMP eine gängige portable Open-Source C++ Programmbibliothek, welche COLLADA implementiert. Für Java fehlt ein robustes Pendant, welches die Möglichkeit bietet die rohen Daten abzufragen. Aus diesem Grund wurde ein eigenes I/O-Programm geschrieben.

Um Animationsinformationen aus einer COLLADA-Datei zu parsen, muss zunächst der Aufbau einer COLLADA-Datei verstanden werden. Der COLLADA-Knoten, welcher Informationen zur Version enthält, hat eine Reihe von Kindknoten. Relevant für reine Animationen eines Meshes, ohne Texturen, Materialien oder Effekte, sind dabei vier dieser Kinderknoten. Als Vorinformation für die einzelnen Unterpunkte sei gesagt, dass COLLADA Matrizen reihendominant speichert, also in folgendem Format:

$$\begin{pmatrix} x_1 & x_2 & x_3 & t_1 \\ y_1 & y_2 & y_3 & t_2 \\ z_1 & z_2 & z_3 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Durch das WPCG-Framework können die Matrizen so genutzt werden. Nutzt man ein anderes Framework kann es notwendig sein die Matrizen umzustrukturieren, falls das Framework eine spaltendominante Formatierung aufweist.

Abbildung 5.2: Das Klassendiagramm der Collada-Import-Klasse.

5.4.2 Mesh

Zunächst ist der Knoten mit dem Namen **library_geometries** wichtig, da dieser die Informationen zum Mesh enthält. Neben dem Aufbau des Meshes durch Vertices und Normalen, sind hier auch Farbinformationen und Texturkoordinaten einzelner Vertices enthalten. Im Kindknoten **polylist** (in manchen Exportern auch **triangles**) sind zwei relevante Informationen angegeben.

```
<library_geometries>
  <geometry id="Cube-mesh" name="Cube">
    <mesh>
      <source id="Cube-mesh-positions">
      <source id="Cube-mesh-normals">
      <source id="Cube-mesh-map-0">
      <source id="Cube-mesh-colors-Col" name="Col">
      <vertices id="Cube-mesh-vertices">
      <polylist material="Material-material" count="1420">
        <input semantic="VERTEX" source="#Cube-mesh-vertices" offset="0"/>
        <input semantic="NORMAL" source="#Cube-mesh-normals" offset="1"/>
        <input semantic="TEXCOORD" source="#Cube-mesh-map-0" offset="2" set="0"/>
        <input semantic="COLOR" source="#Cube-mesh-colors-Col" offset="3" set="0"/>
        <vcount>3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
        <p>97 0 0 0 36 0 1 1 93 0 2 2 93 1 3 3 56 1 4 4 100 1 5 5 104 2 6 6 23 2 7 7 13 2
      </polylist>
    </mesh>
  </geometry>
</library_geometries>
```

Abbildung 5.3: Polylist ist der Knoten mit den Informationen zur Zusammensetzung der Polygone.
Die schwarz unterstrichenen Knoten sind in dieser Sicht eingeklappt und enthalten andere Informationen.

1. Die Angabe über den Aufbau der Vertices und Polygon
2. Die Quellenangabe, wo Positions- und Normalenkoordinaten enthalten sind

Die **vcount** und **p** Knoten geben an wie ein Polygon zusammengebaut wird. In **vcount** ist angegeben, aus wie vielen Vertices ein Polygon besteht. In den Beispielen dieser Arbeit werden dort immer Dreien stehen, da mit Dreiecken gearbeitet wird. In **p** stehen nun die Indizes, um ein Polygon (in diesem Fall Dreieck) zu konstruieren. Da es vier **input**-Knoten gibt, in der Reihenfolge (an dem Attribut **semantic** geordnet) **VERTEX** (hier reine Positionskoordinaten gemeint), **NORMAL**, **TEXCOORD**, **COLOR**, bildet sich ein Offset von vier. Das heißt nach jedem vierten Eintrag im **p**-Knoten stehen die Informationen zu einem neuen Vertex, und nach drei in **p** gebildeten Vertices ein neues Dreieck.

Die Input-Knoten verweisen dabei auf den Quellknoten, in dem die entsprechenden Informationen stehen. Für den ersten Vertex aus diesem Beispiel muss also die **97.** Positionskoordinate und **0.** Normalenkoordinate angegeben werden. Diese Koordinaten stehen im Knoten mit der **id Cube-mesh-vertices** und **Cube-mesh-normals**, wobei **Cube-mesh-vertices** weiter auf **Cube-mesh-positions** verweisen.
Die Knoten **Cube-mesh-positions** und **Cube-mesh-normals** speichern eine lange Liste aus Fließkommazahlen. Alle drei Einträge bilden jeweils eine Koordinateninformation (siehe Unterknoten accessor mit dem Attribut **stride 3**). Zum Parsen der Meshinformationen legt man also eine Liste mit allen Positions- und Normalenkoordinaten an diesen zwei Knoten an und konstruiert dann Vertices und Dreiecke.

```
<library_geometries>
  <geometry id="Cube-mesh" name="Cube">
    <mesh>
      <source id="Cube-mesh-positions">
        <float_array id="Cube-mesh-positions-array" count="2220">0.3735479 0.8294312 7.561044 0.8573105 0.3869484
        <technique_common>
          <accessor source="#Cube-mesh-positions-array" count="740" stride="3">
            <param name="X" type="float"/>
            <param name="Y" type="float"/>
            <param name="Z" type="float"/>
          </accessor>
        </technique_common>
      </source>
      <source id="Cube-mesh-normals">
      <source id="Cube-mesh-map-0">
      <source id="Cube-mesh-colors-Col" name="Col">
      <vertices id="Cube-mesh-vertices">
      <polylist material="Material-material" count="1420">
    </mesh>
  </geometry>
</library_geometries>
```

Abbildung 5.4: Der Positions-Knoten liefert Informationen zu den Vertex-Koordinaten. Die schwarz unterstrichenen Knoten sind in dieser Sicht eingeklappt und enthalten andere Informationen.

Auffällig wird dabei, dass es mehr Normalen als Vertices gibt. Das hängt damit zusammen, dass Dreiecke, die sich einen Vertex teilen, verschiedene Normalen für diese Vertexposition besitzen können. Dies wird für das Erstellen von Smoothing-Gruppen genutzt. Außerdem sollten für das folgende Lesen der Skinning-Informationen bereits nun sogenannte Vertex-Positionsgruppen erstellt werden. Das heißt, alle Vertices werden entsprechend ihrer Positionsinformation gruppiert, da sich Vertex-Knochen-Gewichte immer auf die Position der Vertices beziehen. Dies wird im folgenden näher erläutert.

5.4.3 Skelett

Angaben zum Skelett stehen im Knoten **library_visual_scenes**. Die Kindknoten haben immer das Attribut **NODE**, wobei der Wurzelknochen mit den Skelettinformationen der Knochen ist, welcher mindestens einen Kindknoten mit dem Attribut **NODE** besitzt. Zum Parsen muss nun ausgehend vom Wurzelknochen, welcher Informationen zur Transformierung des gesamten Skeletts enthält, also die Bind-Shape-Matrix, eine Tiefensuche gestartet werden, und in dieser Reihenfolge werden die Knochen indiziert (relevant für das spätere Skinning). Die Knoten mit den Informationen zu den Knochen enthalten außerdem die Bind-Pose Matrix des Knochens, immer im Bezug zum Elternknochen, und den Namen eines einzelnen Knochens.

```
  <library_visual_scenes>
    <visual_scene id="Scene" name="Scene">
      <node id="Camera" name="Camera" type="NODE">
      <node id="Armature" name="Armature" type="NODE">
        <translate sid="location">0 0 0</translate>
        <rotate sid="rotationZ">0 0 1 0</rotate>
        <rotate sid="rotationY">0 1 0 0</rotate>
        <rotate sid="rotationX">1 0 0 0</rotate>
        <scale sid="scale">1 1 1</scale>
        <node id="Torso" name="Torso" sid="Torso" type="JOINT">
          <matrix sid="transform">1 0 0 0 0 -0.06466547 -0.997907 0 0 0.997907 -0.0646655
          <node id="Chest" name="Chest" sid="Chest" type="JOINT">
          <node id="Upper Leg L" name="Upper Leg.L" sid="Upper Leg L" type="JOINT">
          <node id="Upper Leg R" name="Upper Leg.R" sid="Upper Leg R" type="JOINT">
          <extra>
        </node>
      </node>
      <node id="Cube" name="Cube" type="NODE">
    </visual_scene>
  </library_visual_scenes>
  <scene>
</COLLADA>
```

Abbildung 5.5: Im Visual_Scene-Knoten ist die Hierarchie des Skeletts aufgeführt. Die schwarz unterstrichenen Knoten sind in dieser Sicht eingeklappt und enthalten Informationen zu den einzelnen Knochen.

5.4.4 Vertex-Knochen-Gewichte

Um das Mesh mit dem Skelett zu verknüpfen, müssen die Vertex-Gewichte gelesen werden, diese stehen im Knoten **library_controllers**. Im Kindknoten **vertex_weights** stehen, ähnlich der Dreiecksinformationen, wie die Skelett-Gewichte zu einem Vertex

konstruiert werden. Wichtig ist dabei zu beachten, dass die Gewichte sich auf die Positionen und nicht auf die tatsächlichen Vertices beziehen. Das heißt, dass das Gewicht für die Position p für alle Vertices gilt, welche als Positionskoordinate p besitzen. Aus diesem Grund wurden vorherigen Arbeitsschritt Vertex-Positionsgruppen erstellt, damit die Zuordnung zu den Gewichten vereinfacht wird. Die Bind-Shape-Matrix steht ebenfalls im Kindknoten **bind_shape_matrix**.

Im Kindknoten **vcount** steht die Anzahl Knochen die einen Vertex beeinflussen, in **v** die konkrete Zusammensetzung aus Knochen und Gewicht zu einem Vertex. Die Reihenfolge und Offsets werden wieder durch die **input**-Knoten bestimmt. Es gibt zwei **input**-Knoten mit den Attributen **JOINT** und **WEIGHT**, in denen die **Quell-IDs**-Knoten angegeben sind, um die Knochen und die Gewichte zu finden. Die Knochen sind in der selben Reihenfolge indiziert wie bereits beim Parsen des Skeletts indiziert wurde.

```
<library_controllers>
  <controller id="Armature_Cube-skin" name="Armature">
    <skin source="#Cube-mesh">
      <bind_shape_matrix>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</bind_shape_matrix>
      <source id="Armature_Cube-skin-joints">
      <source id="Armature_Cube-skin-bind_poses">
      <source id="Armature_Cube-skin-weights">
      <vertex_weights count="740">
        <input semantic="JOINT" source="#Armature_Cube-skin-joints" offset="0"/>
        <input semantic="WEIGHT" source="#Armature_Cube-skin-weights" offset="1"/>
        <vcount>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        <v>3 0 3 1 3 2 3 3 3 4 3 5 3 6 3 7 3 8 3 9 3 10 3 11 3 12 3 13 3 14 3 15 3
      </vertex_weights>
    </skin>
  </controller>
</library_controllers>
```

Abbildung 5.6: Im Library_Controllers-Knoten findet man die Blending-Informationen und die Bind-Shape-Matrix.
Die schwarz unterstrichenen Knoten sind in dieser Sicht eingeklappt und enthalten Informationen zu den einzelnen Knochen.

Eine weitere wichtige Information, die Inverse Bind Matrix eines Knochens, steht ebenfalls an dieser Stelle und muss nicht eigens berechnet werden, im **joints**-Knoten wird auf den Quellknoten verwiesen. Dort stehen $16 * n$ (n=Anzahl der Knochen) Einträge als Zeichenkette, wovon alle 16 Einträge eine Inverse Bind Matrix bilden. Die Indizierung der Matrizen ist wieder äquivalent der Indizierung der Knochen.

5.4.5 Keyframes

In **library_animations** stehen Informationen zu den einzelnen Keyframes einer Animation und zu der Art wie interpoliert wird. Die Knoten zu den Deformationsinformationen eines Knochen können über die **id Prefix + Name des Knochens + Suffix** gefunden werden. Der Prefix ist dabei der Name des Wurzelknochens und der Suffix _pose_matrix. In den Knoten mit den Animationsinformationen findet man den Kindknoten **sampler**, in dem drei Einträge mit Verweisen auf andere Knoten zu finden sind. Die Kindknoten des **sampler**-Knotens findet man mit dem Attribut **semantic** **INPUT**, **OUTPUT** und **INTERPOLATION**. Die Quellknoten sind Kindknoten des Knotens mit den Animationsinformationen.
Ein Keyframe besteht immer aus einem Zeitpunkt und einer Transformationsmatrix, der Pose-Matrix zum entsprechenden Zeitpunkt.
INPUT verweist auf den Knoten mit den einzelnen Zeitstempeln, **OUTPUT** auf die Pose-To-Parent-Matrix, und **INTERPOLATION** auf die Art der Interpolation wird (für Skelettanimationen **LINEAR**, weitere Arten sind in der COLLADA-Spezifikation aufgeführt). Die Output- Transformationen sind immer im Bezug zum Elternknochen eines Knochens. Ein Keyframe wird also definiert durch einen Zeitstempel und eine Transformation.
Da die Zeitstempel keine Grenzbedingung haben, werden sie beim Laden der Animationsdatei auf 0 bis 1 umgerechnet.

```
<library_animations>
  <animation id="Armature_Torso_pose_matrix">
    <source id="Armature_Torso_pose_matrix-input">
      <float_array id="Armature_Torso_pose_matrix-input-array" count="5">0 0.2083333 0.4166666 0.62
      <technique common>
    </source>
    <source id="Armature_Torso_pose_matrix-output">
    <source id="Armature_Torso_pose_matrix-interpolation">
    <sampler id="Armature_Torso_pose_matrix-sampler">
      <input semantic="INPUT" source="#Armature_Torso_pose_matrix-input"/>
      <input semantic="OUTPUT" source="#Armature_Torso_pose_matrix-output"/>
      <input semantic="INTERPOLATION" source="#Armature_Torso_pose_matrix-interpolation"/>
    </sampler>
    <channel source="#Armature_Torso_pose_matrix-sampler" target="Torso/transform"/>
  </animation>
```

Abbildung 5.7: Im Library_Animations-Knoten findet man die Keyframes und Animationsmatrizen der einzelnen Knochen.
Die schwarz unterstrichenen Knoten sind in dieser Sicht eingeklappt und liefern die einzelnen Animationsinformationen.

5.4.6 Konstruieren des animierbaren Meshes

Hat man die vier Knoten richtig geparst, erhält man ein Mesh und ein Skelett. Skelett und Mesh sind durch die Vertex-Knochen-Gewichte miteinander verbunden und das Mesh kann durch Deformieren des Skeletts anhand der Keyframes animiert werden.

5.5 Darstellende Komponente

Die Darstellung unterscheidet sich bei der Desktop und bei der Mobile-Variante der Applikation, da die Mobile-Variante im Augmented Reality Kontext steht und unter anderem die Nutzer-Interaktion Folgen der Kamera implementiert. In diesem Abschnitt wird die darstellende Komponente der Mobile-Variante näher erläutert, da der Unterschied zur Desktop-Variante dann leicht zu erklären ist.

5.5.1 AR-Tracking

Zum Tracken wird das markerbasierte Tracking aus dem Vuforia-Framework genutzt. Hier wird auf dem Vuforia Developer-Portal ein Image Target erzeugt, also ein Bild mit genügend Featurepunkten das vom Framework zum Tracking genutzt werden kann. Im Development-Portal gibt es zum Image Target die Möglichkeit die Featurepunkte anzeigen zu lassen und anhand einer Farbcodierung Feedback dazu, ob das genutzte Bild geeignet ist. Im WPCG-Framework muss dann der Datensatz mit den Image Targets, eine .xml und eine .dat-Datei, dann dem aktuellen Datensatz hinzugefügt werden.

5.5.2 Aufbau und Rendering der Szene

Um die Daten in der Darstellung zu verwalten wird ein sogenannter Szenengraph erstellt. Ein Szenengraph ist ein gerichteter Graph, dessen Wurzelknoten die gesamte Szene enthält. Dem Wurzelknoten sind dann weitere Kindknoten untergeordent, die Objekte einer Szene oder Eigenschaften, wie zum Beispiel eine Skalierung oder eine Transformation, enthalten. Auch die Image-Targets können als Knoten hinzugefügt und als Marker genutzt werden. In der AR-Applikation wird dazu einen Szenengraph erstellt, an dessen Wurzelknoten der Marker-Knoten hängt. An diesem hängt ein Knoten zur Skalierung und an diesem ein Knoten mit einer Transformation, dessen Kindknoten das Mesh ist.

Das WPCG-Framework bietet alle nötigen Knoten-Arten an, sodass kein zusätzlich implementiert werden muss.
Das Weltkoordinatensystem mit dem Vuforia-Framework ist so aufgebaut, dass die Kamera immer an der Position $(0, 0, 0)$, also dem Ursprung, liegt. Von der Kamera ausgehend zeigt die z-Achse immer in die Richtung in die auch die Kamera sieht, die x-Achse bildet die Längsachse und die y-Achse bildet die Höhenachse. In der Klasse **SkeletalAnimatedScene** ist die Szene aufgebaut. Immer wenn die Methode **onSceneRedraw()**, geerbt aus der Klasse **Scene** des Frameworks, aufgerufen wird, wird das Mesh aktualisiert. Da die Animation des Meshes in einem anderen Thread berechnet wird, wird zur Synchronisation ein Mutex eingesetzt, der immer dann frei ist, wenn ein voller Animationsschritt abgeschlossen ist. Ohne die Synchronisation entstehen Artefakte in der Animation des Meshes, da alte und aktuelle Animationsschritte miteinander vermischt werden können. Nach dem aktualisieren des Meshes wird dieses noch in die Richtung der Kamera gedreht.

5.5.3 Folgen der Kamera

Um die Transformation zu berechnen, mit dem das Mesh in Richtung der Kamera schaut, muss zunächst die Sichtlinie von Mesh zur Kamera berechnet werden. Dazu wird zunächst die Inverse Transformation des Markers gebildet, da das Mesh zur Berechnung der Sichtlinie aus dem Marker-Koordinatensystem in das Weltkoordinatensystem transformiert werden muss. Im folgenden Schritt wird diese Transformationsmatrix mit dem Positionsvektor der Kamera multipliziert. Von dem resultierenden Vektor wird dann noch der Translationsvektor des Meshes auf dem Marker abgezogen und das Ergebnis wird normiert. Somit ist die Sichtlinie berechnet.
Um aus der Sichtlinie eine Transformationsmatrix zu bilden, müssen zwei weitere Vektoren berechnet werden. Der Vektor der Sichtlinie wird nun als x bezeichnet. Aus x und einem Vektor $u_y = (0, 1, 0)$, also dem Einheitsvektor in y-Richtung, wird das Kreuzprodukt gebildet und als Vektor z bezeichnet. Aus z und x wird ebenfalls das Kreuzprodukt gebildet und das Resultat wird als Vektor y bezeichnet. Die drei Vektoren bilden ein Rechtssystem und beschreiben die Transformation $M = (x, y, z)$, um das Mesh in Richtung der Sichtlinie zu drehen.
Das Mesh wurde in Blender in einem Koordinatensystem erstellt, welches sich von dem Weltkoordinatensystem des Vuforia-Frameworks unterscheidet. Deswegen wird das Mesh noch entsprechend gedreht.

5.5.4 Funktionalitäten Mobile

Dem Nutzer stehen eine Reihe von Funktionen auf dem Display zur Verfügung. Sie sind durch Setzen von Zustandsvariablen in der Klasse **AnimationConfig** realisiert.

1. Starten oder Stoppen des Aktualisieren des Meshes (dies ist nicht gleichzusetzen mit dem Starten oder Stoppen der Animation)
2. Starten oder Stoppen der Animation
3. Starten oder Stoppen der Nutzer-Interaktion Folgen der Kamera
4. Wechseln zwischen den Blending-Techniken LBS und DQS
5. Wechseln zwischen verschiedenen Interpolationsmethoden zum Interpolieren der Rotation zwischen Keyframes
6. Verschnellern oder Verlangsamen der Animationsgeschwindigkeit

5.5.5 Desktop-Variante

Die Darstellung ist in der Klassen **SkeletalAnimatedScene** implementiert, welche in der **VuforiaActivity** (Mobile-Variante) oder **MeshApplication** (Desktop-Variante) gestartet wird. Im Unterschied zur Mobile-Variante wird in der Desktop-Variante ohne Vuforia gearbeitet, sodass die Marker-Knoten im Szenengraph wegfallen und auch die Nutzer-Interaktion fällt in der Desktop-Variante weg. Die Funktionen sind im Gegensatz zur Mobile-Variante auf eine Toolbar neben der Szene platziert.
Es kommen vier weitere Meshes hinzu, die über Pfeile auf der Toolbar durchgewechselt werden können.

5.6 Simulierende Komponente

Die Simulation läuft in einem Thread parallel zur rendernden Szene in der Klasse **SkeletalAnimatedMesh**. Diese Klasse instanziiert das Mesh, das Skelett und den Skelett-Controller. Nach Start des Threads aktualisiert dieses die Deformierung des Skeletts periodisch und über eine der beiden Skinning-Varianten wird die Deformation des Skeletts dann auf das Mesh übertragen.

Da das **SkeletalAnimatedMesh** die Klasse **TriangleMesh** aus dem WPCG-Framework nutzt, um das Mesh zu erstellen, gliedert sich die Umsetzung nahtlos in das Framework ein. Ein Klassendiagramm ist im Anhang A zu finden.

Den Ablauf kann man folgendermaßen darstellen:

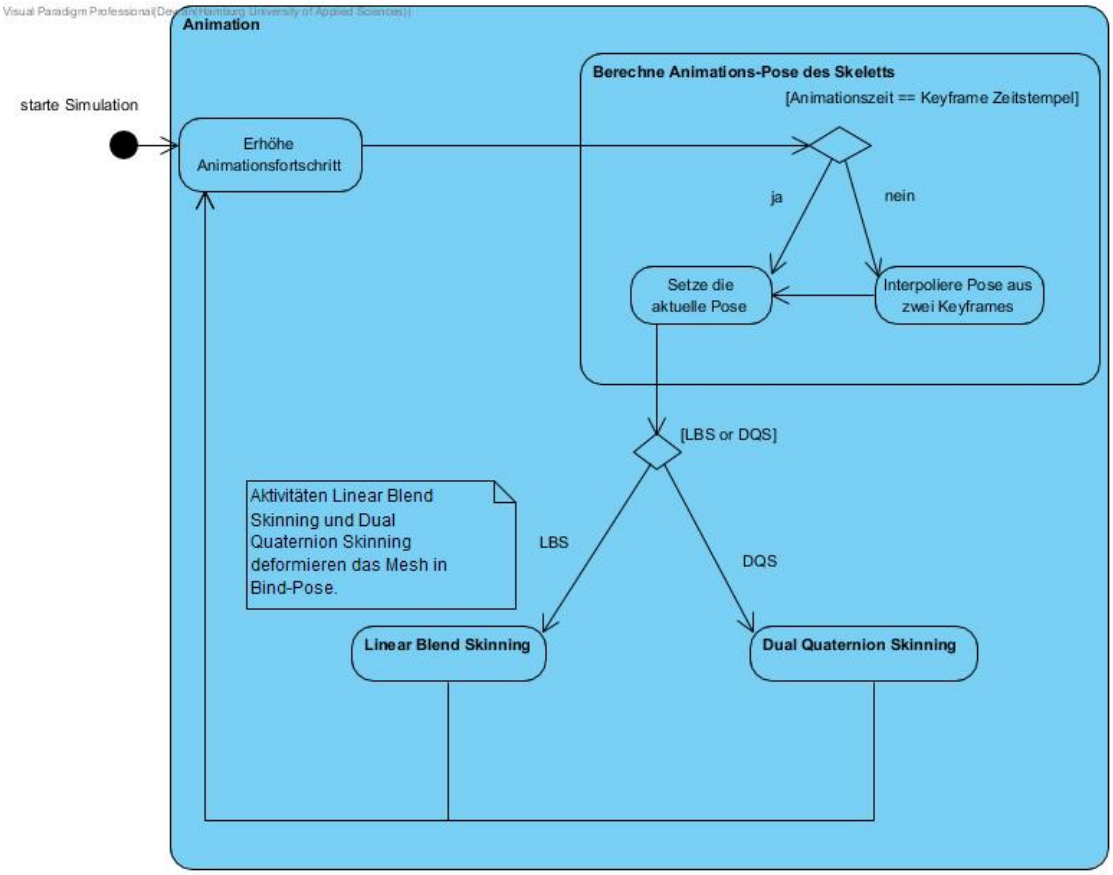

Abbildung 5.8: Das Ablaufdiagramm der Mesh-Deformation

5.7 Berechnen des deformierten Meshes

Mit jedem Aufruf der **update()**-Methode wird das deformierte Mesh berechnet. Hier muss zunächst das Skelett deformiert werden, wofür die Inverse-Bind-Matrix, die Bind-Shape-Matrix und die Animations-Matrix im Wurzelkoordinatensystem notwendig sind. Im nächsten Schritt wird die Deformierung des Skeletts durch Linear Blend Skinning oder Dual Quaternion Blend Skinning auf das Mesh in der Bind-Pose übertragen.

Der Lesbarkeit halber werden an dieser Stelle einige Matrizen erläutert:

1. Bind-Shape-Matrix: Die Transformation die für das gesamte Skelett gilt. Sie wird aus der COLLADA-Datei gelesen.

2. Pose-To-Parent-Matrix: Die Transformation der Deformierung eines Knochens B zum Animationszeitpunkt, in Relation zum Koordinatensystem des Elternknochen B_{parent}.Sie wird aus der COLLADA-Datei für jeden Keyframe gelesen.

3. Pose-To-World-Matrix: Die Transformation der Deformierung eines Knochens B zum Animationszeitpunkt, in Relation zum Koordinatensystem des Wurzelknochens.

4. Animations-Matrix: Die Transformation, die einen Vertex in Bind-Pose zum Animationszeitpunkt deformiert.

5. Inverse-Bind-Matrix: Die Inverse der Bind-Matrix, also die Transformation, um aus dem Koordinatensystem eines Knochens in das Koordinatensystem des Wurzelknochens zu transformieren. Sie wird aus der COLLADA-Datei gelesen.

5.7.1 Berechnung der deformierten Skeletts

Um das deformierte Skelett zu berechnen, wird entsprechend der Hierarchie über alle Knochen iteriert und zunächst die Animation-Matrix eines jedes Knochens berechnet. Die Animationsmatrix eines Knochens wird berechnet durch $Animations - Matrix = Bind - Shape - Matrix * Pose - To - World - Matrix * Inverse - Bind - Matrix$. Die dabei entstehende Transformationsmatrix beschreibt eine Transformation die einen Vertex in Bind-Pose durch die Inverse Bind Matrix aus dem Knochenkoordinatensystem in das Koordinatensystem des Wurzelknochens transformiert, und dann den Vertex im selben Koordinatensystem an die animierte Position im Koordinatensystem des Knochen transformiert. Die in dieser Kalkulation berechnete Pose-To-World-Matrix liegt bereits im Koordinatensystem des Wurzelknochens und wird berechnet, indem die Pose-To-Parent-Matrix eines Knochens B mit der Pose-To-World-Matrix des Elternknochens von B, also B_{parent}, multipliziert wird. Dafür muss die Pose-To-World-Matrix des Elternknochens B_{parent} bereits berechnet sein.
Die richtige Indizierung der Knochen-Hierarchie aus COLLADA-Datei vereinfacht diesen Schritt.
Liegt die Pose-To-Parent-Matrix nicht vor, weil der Animationszeitpunkt zwischen zwei

Keyframes liegt, muss diese erst berechnet werden, indem zwischen zwei Keyframes interpoliert wird.

5.7.2 Keyframe-Interpolation

Um eine flüssige Animation abzuspielen, muss zu einem Zeitpunkt t_i, der zwischen den Keyframes k_1 und k_2 mit den Zeitstempeln t_{k1} und t_{k2}, wobei $t_{k1} < t_i < t_{k2}$ gilt, interpoliert werden. Da die Pose-Matrizen Rotationsmatrizen enthalten und sich diese nur schwer linear interpolieren lassen, bietet es sich an die Pose-Matrix in ein Paar aus Quaternion (zur Beschreibung der Rotation) und Vektor (zur Beschreibung der Translation) umzurechnen. Die Translationsvektoren sind leicht linear interpolierbar mit $v_i = v_1 * (t - 1) + v_2 * t$.
Zur Interpolation der Quaternion sind folgende drei Methoden implementiert:

1. Lineare Interpolation (lerp)
2. Sphärische Lineare Interpolation (slerp)
3. Normalisierte Lineare Interpolation (nlerp)

5.8 Blending

Da das Skelett nun deformiert ist, wird die Deformation auf das Mesh in Bind-Pose übertragen. Dazu sind zwei Methoden implementiert **linearBlendSkinning()** und **dualQuaternionSkinning()**.
Die grundlegende Idee beider Methoden wurde im Kapitel Related Work erläutert.

5.8.1 Linear Blend Skinning

Es wird über alle Vertices v des Meshes iteriert. Für jeden Vertex wird eine Blend-Matrix (die Deformationsmatrix) berechnet, indem die Animations-Matrizen der v beeinflussen-

den Knochen $\{B_1, ..., B_n\}$ mit dem Gewicht $\{w_1, ..., w_n\}$ gewichtet linear interpoliert wird. Auch die Normalen der Vertices werden so berechnet berechnet.

```
foreach Vertex v do
    BlendMatrix = 0-Matrix;
    foreach Bone do
        BlendMatrix += Bone-Animation-Matrix * Weight;
    end
end
Vertex newPosition = BlendMatrix * v;
```

Algorithm 1: Linear-Blend-Algorithmus

5.8.2 Dual Quaternion Skinning

Das Dual Quaternion Skinning, oder Dual Quaternion Blending, baut direkt auf dem Linear Blend Skinning auf. Es gibt grundsätzlich zwei Herangehensweisen in der Deformation der Mesh-Vertices, die sich nach den genutzten Frameworks richten, entweder die Deformation durch das Blend Dual Quaternion (also das interpolierte duale Quaternion aller Vertex beeinflussender Knochen) oder eine Umrechnung des Blend Dual Quaternions in eine Blend Matrix, dann die Deformierung auf dem Vertex. Das Ergebnis ist dabei das selbe, weil in letzterem Fall die Matrix aus einem dualen Quaternion gebildet wird, welches keine Skalierungen oder Scherungen enthält.
In dieser Implementation werden die Animations-Transformationen der Knochen, die jeweils als Matrizen gespeichert sind, erst zur Animationszeit in duale Quaternionen umgewandelt (siehe Grundlagen, Abschnitt 2.3.6). Im darauffolgenden Schritt werden die dualen Quaternionen mit den entsprechenden Blending-Weights skaliert und addiert. Das resultierende duale Quaternion zum Deformieren eines Vertices deformiert dieses dann direkt, ohne in eine Matrix zurückgewandelt zu werden. Der Algorithmus, um einen Vertex

mit dualen Quaternionen zu deformieren, sieht dabei folgendermaßen aus:

```
foreach Vertex v do
    BlendDQ = 0-DQ;
    pivot = first convertMatrixToDQ( first Bone-Animation-Matrix associated with
     Vertex);
    foreach Bone do
        AnimationDQ = convertMatrixToDQ(Bone-Animaton-Matrix);
        if cross of pivot and AnimationDQ < 0 then
            weight = weight * -1;
        end
        AnimationDQ * weight;
        BlendDQ += AnimationDQ;
    end
end
Vertex newPosition = BlendDQ * v;
```

Algorithm 2: Dual-Quaternion-Algorithmus

Die if-Bedingung bezüglich des Kreuzproduktes aus pivot des Vertex und Animations-Matrix wird gesetzt, um Artefakte zu vermeiden. Das pivot bildet der Vertex beeinflussende Knochen eines Vertices, der in der Hierarchie der Knochen dem Wurzelknochen am nahesten ist. Quaternionen können eine Rotation über zwei Wege beschreiben. Durch das Bilden des Kreuzproduktes wird sichergestellt, dass der kurze Weg gewählt wird und keine Artefakte entstehen. Kavan et al. erläutern im Abschnitt 4.1 des Papers zum Dual-Quaternion Blend Skinning diese Problematik genauer (Antipodality).[16]

5.8.3 Mathematik Klassen

Die Klassen **Quaternion** und **DualQuaternion** implementieren die für die Applikation notwendige Mathematik. Die Konversion zwischen Matrix, Quaternion, duale Quaternion und Vektor, sowieso jede Interpolation, befindet sich hingegen in der Klasse **Animation-Helper**.

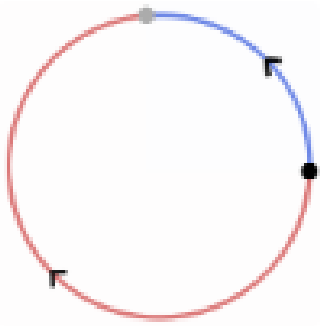

Abbildung 5.9: Beide Pfeile zeigen vom schwarzen Punkt ausgehend Rotationsrichtungen einer Rotation zum grauen Punkte an. Wählt man nicht die kürzeste Rotation, können Skinning-Artefakte entstehen.
Quelle: http://rodolphe-vaillant.fr/?e=29 (Letzter Zugriff: 24.03.2019)

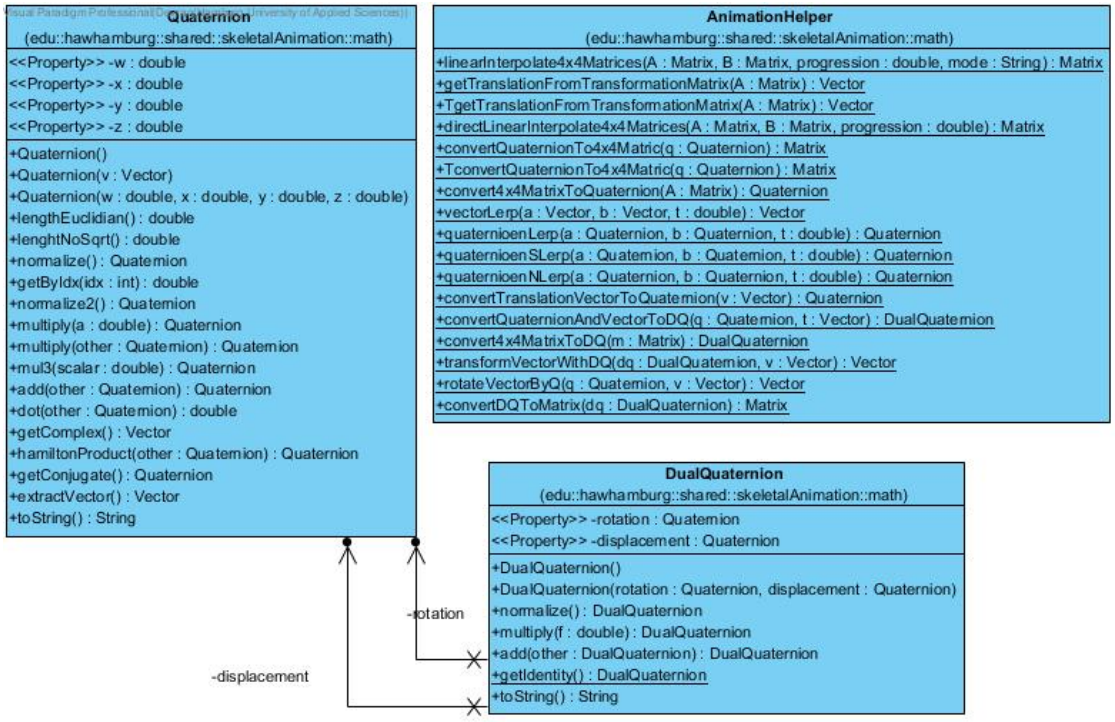

Abbildung 5.10: Das Klassendiagramm zu den Mathematik-Klassen.

Dateiname	Anzahl Polygone	Animation	Keyframes
cowboy.dae	1420	Laufanimation	5
cylinderXHigh.dae	2044	Abknicken an der Zylidermitte	2
cylinderYHigh.dae	2044	Drehung um die Höhenachse	2
cylinderYLow.dae	1148	Drehung um die Höhenachse	2

5.8.4 Wartbarkeit des Codes

Damit der Code gut gewartet werden kann, wurden notwendige Kommentare zur Erklärung der einzelnen Animationsschritte eingeführt. Außerdem wurde modular gearbeitet, sodass man die verschiedenen Komponenten des Systems austauschen kann. Dadurch wird dieselbe I/O-Komponente und dieselbe simulierende Komponente bei den Desktop und Mobile Applikationen genutzt.

5.8.5 Herausforderungen bei der Implementierung und Hinweise

Eine besonderes Herausforderung bei der Implementierung des COLLADA-Parsers war die COLLADA-Spezifikation selbst. Sie erklärt zwar den Aufbau einer .dae-Datei genau, aber erklärt nicht wichtige Zusammenhänge wie zum Beispiel die Assoziation der Vertex-Gewichte mit den Vertex-Positionen oder ob die Keyframe-Transformationen noch in das Koordinatensystem des Wurzelknochens gerechnet werden müssen oder bereits verrechnet sind.

5.9 Animations-Dateien

Es stehen insgesamt vier Animationsdateien zur Verfügung, von denen drei selbst in Blender erstellt wurden. Einem kurzen Überblick folgt eine genauere Beschreibung.

5.9.1 Selbst erstellte Animations-Dateien

In Blender wurden drei Animationsdateien mit je zwei Knochen erstellt. Die Animationsdatei cylinderXHigh.dae besitzt eine Animation mit dem Abknicken an der Zylidermitte, die Animationsdateien cylinderYLow.dae und cylinderYHigh.dae eine Animation mit der Drehung um die Höhenachse. Die Dateien cylinderXHigh.dae und cylinderYHigh.dae bestehen aus 2044 Polygonen, cylinderYLow.dae besteht aus 1148 Polygonen.

Die höhere Anzahl an Polygonen befindet sich am Gelenk, um mögliche Auswirkungen höherer Polygonanzahl im Blending festzustellen. Die Animationen bestehen aus zwei Keyframes zwischen denen Zwischenschritte interpoliert werden. Die Animationen wurden automatisch geskinnt, Blender benutzt dabei standardmäßig das Skinning nach der Wärmegleichgewichts-Methode. Die Zylinder-Animationen sind nicht in der Mobile-Applikation enthalten und dienen lediglich der Evaluation.

5.9.2 Beschaffte Animations-Dateien

Die beschaffte Datei, welche als Beispiel im Abschnitt COLLADA-I/O genutzt wurde, wird in der Mobile-Applikation genutzt. Das Cowboy-Mesh besteht aus 1420 Polygonen und 16 Knochen. Die Lauf-Animation besteht aus fünf Keyframes.

5.9.3 Unterschied Desktop und Mobile

Die Mobile-Variante nutzt aus Grunden ungelöster Probleme eine ältere Version des WPCG-Frameworks. Dabei ändert sich seitens der Implementierung oder der Nutzung der Software jedoch nichts.

6 Evaluation

Im folgenden Kapitel wird die Umsetzung der Desktop- und Mobile-Version evaluiert. Die Evaluation der Skinning-Technik wird auf Basis der Desktop-Variante durchgeführt

6.1 Implementierung Mobile

Die Implementierung der Mobile-Variante ist mit vollem Funktionsumfang gelungen. Das Mesh, im Fall der Mobile-Variante, läuft immer auf die Kamera zu. Der Nutzer hat die Möglichkeit über verschiedene Buttons mit dem Charakter zu interagieren.

Abbildung 6.1: Implementierung der Mobile-Variante

Er kann über die beiden Pfeile mit grünem Hintergrund die Animation beschleunigen bzw. verlangsamen und die Interaktion starten bzw. stoppen, indem er auf den Button

mit dem Auge drückt. Wird die Interaktion gestoppt, bleibt der Charakter fest in seiner Position und dreht sich nicht mehr der Kamera zu. Außerdem kann mit dem blauen Rechteck-Button das Aktualisieren des Meshes (also des Vertex Buffer Objects) gestartet bzw. gestoppt werden und mit dem blauen Dreieck-Button kann die Animation gestoppt werden. Der grüne Button mit den zwei Pfeilen kann genutzt werden, um zwischen den zwei Skinning-Modi Linear Blend Skinning und Dual Quaternion Blend Skinning zu wechseln. Der Unterschied zwischen dem rechteckigen Button und dem dreieckigen Button liegt darin, dass der Dreiecks-Button zwar die Animation anhält, aber das Mesh weiterhin aktualisiert wird, sodass ein Umschalten des Skinning-Modis sichtbar bleibt.

6.2 Implementierung Desktop

In der Desktop-Variante stehen dem Nutzer ähnliche Funktionen zur Verfügung. Es fällt die Interaktion mit der Rotation zur Kamera weg, zum Evaluieren kommen jedoch Funktionen hinzu, um den Rotationsanteil der Keyframes anders zu interpolieren. Ansonsten kann der Nutzer auch hier die Animation starten bzw. stoppen, den Skinning-Modus ändern und, weil drei Zylinder-Meshes hinzukommen, zwischen den vorhandenen Meshes schalten. Der Button, um die Kamera auf das Mesh zu zentrieren ist bereits im Framework enthalten.

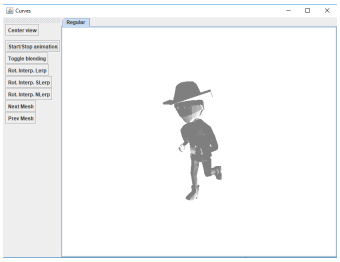

Abbildung 6.2: Implementierung der Desktop-Variante

6.3 Ladezeiten

Die Ladezeit von Meshes liegt bei beiden Versionen unter dem zuvor gesetzten Ziel von 10 Sekunden. Die maximale Anzahl Mesh-Polygone liegt bei 2044 Polygone, konstruiert aus 1024 Vertices (cylinderX.dae und cylinderYHigh.dae).

Anzahl Meshes	FPS DQS	FPS LBS
1	60	60
4	60	48
8	18	11
12	7	5

6.4 Framerate

Beim Ermitteln der Framerate wird das Cowboy-Mesh genutzt. Die erzielte Framerate in der Desktop-Version liegt für alle Meshes im Mittel bei 60 Frames pro Sekunde für ein Mesh und erfüllt somit das Ziel. Dies gilt für beide Skinning-Varianten.
Erhöht man die Anzahl der Meshes auf 4 Meshes, so erziel das Dual Quaternion Blend Skinning weiterhin eine Framerate von 60 Frames pro Sekunde, das Linear Blend Skinning fällt jedoch bereits auf 47 Frame pro Sekunde ab. Verdoppelt man die Anzahl auf 8

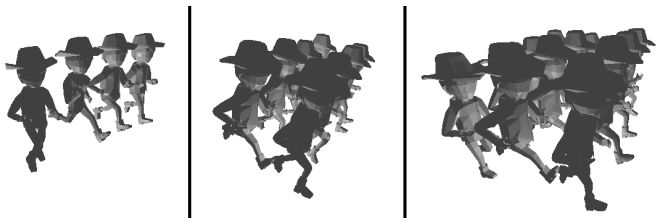

Abbildung 6.3: Je höher die Anzahl der Meshes, desto niedriger wird die FPS. Das DQS erzielt in jeder Variante eine höhere Framerate als das LBS

Meshes erkennt man bei beiden Skinning-Varianten einen deutlichen Abfall der Framerate. Das DQS liegt dabei bei 18 Frames pro Sekunde und das LBS erzielt 11 Frames pro Sekunde. Fügt man weitere vier Meshes hinzu, fällt das DQS auf 7 Frames pro Sekunde und das LBS auf 5 Frames pro Sekunde ab.
Anhand dieser Auswertung kann man ableiten, dass das Dual Quaternion Blend Skinning eine bessere Performance erzielt, als das Linear Blend Skinning, obwohl bei der DQS-Methode jede Knochen-Matrize zuvor in ein duales Quaternion umgewandelt werden muss, da die Datensätze als Matrizen gespeichert werden.

Die erzielte Framerate für ein Mesh in der Mobile-Version liegt bei 4 Frames pro Sekunde beim LBS und 3 Frames pro Sekunde beim DQS. Die Framerate steigt auf 30 Frames pro Sekunde, wenn das Mesh nicht weiter deformiert wird. Die niedrige Framerate scheint

daher an der Berechnung der Deformation zu liegen. Die Auslastung des Smartphones beträgt dabei bis zu 40% bei einem Speicherverbrauch zwischen 100-200MB.

6.5 Laufzeitmessungen

Folgende Laufzeiten für die Berechnung des Deformierten Meshes wurden gemessen.

Datei	DQS (in ms)	LBS (in ms)
cowboy.dae	4.35	4.55
cylinderYLow.dae	2.67	3.00
cylinderXHigh.dae	5.43	6.5
cylinderYHigh.dae	4.86	6.37

Man erkennt, dass das deformierte Mesh mit dem Dual Quaternion Blend Skinning schneller berechnet wird.

Optimierung Skinning

Das Dual Quaternion Blend Skinning erzielt bei der Desktop-Variante eine wesentliche bessere Performanz und steht dem Linear Blend Skinning in der Mobile-Variante nur um 1 Frame pro Sekunde nach.
Zum aktuellen Stand der Implementation wird jede Knochen-Animations-Transformation für jeden Vertex in ein duales Quaternion, zur Animationszeit, konvertiert. Eine Optimierungsmöglichkeit ist, alle Animations-Matrizen vor dem Berechnen der Mesh-Deformation in duale Quaternionen umzuwandeln, sodass $n_{Konvertierung} = n_{Knochen}$ anstelle von $n_{Konvertierung} = n_{Vertices} * n_{Vertex-Knochen}$ beträgt.

Optimierung durch Hardware-Skinning

Zum aktuellen Zeitpunkt wird das Skinning auf der CPU durchgeführt, diesen Prozess nennt man auch Software-Skinning. Eine Alternative dazu ist es, die Deformation des Skelettes weiterhin auf der CPU zu berechnen, aber die Deformation des Meshes, also aller Vertices, auf der GPU durchführen zu lassen. Dazu reicht man zusätzlich zu den Vertices

auch die Blending-Weights, Vertex-Knochen-Indices und Knochen-Transformationen an den Vertex-Shader der GPU weiter und lässt die Vertex-Deformation dort berechnen [11]. Vorteile, und Popularität, des Software-Skinnings liegen darin, dass Post-Skinning Berechnen, wie zum Beispiel die Kollisionsberechnung, einfach zu realisieren sind [2].

6.6 Volumenverlust

Aufgrund besserer Erkennbarkeit (flüssigere Animationen, größere Bilder) werden die Ergebnisse beider Skinning-Techniken hauptsächlich an der Desktop-Version verglichen, da die deformierten Meshes in Desktop- und Mobile-Applikation identisch sind.

6.6.1 Cowboy

Die Gegenüberstellung des Dual Quaternion Blend Skinning und Linear Blend Skinning zeigt bereits am Cowboy-Mesh die volumenerhaltende Eigenschaft des Dual Quaternion Skinnings. Auf der Abbildung 6.3 sieht man an den rot eingekreisten Bereichen, dass das Mesh nach Deformierung des DQS voluminöser wirkt. Jedoch fällt auch auf, dass der Candy-Wrapper-Effekt des Linear Blend Skinnings oder andere Artefakte nicht auftreten, da die Laufanimation des Cowboy-Meshes keine extremen Rotationen enthalten. Einen großen Vorteil bietet das DQS in diesem Fall nicht.

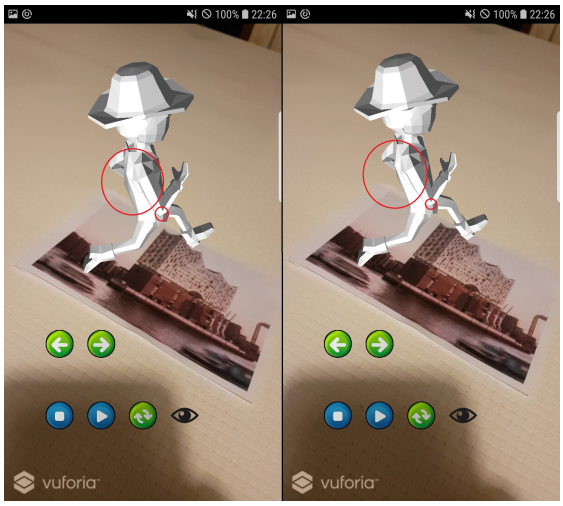

Abbildung 6.4: Auf der linken Bildhälfte das Dual Quaternion Blend Skinning und rechts das Linear Blend Skinning.

Auch auf der Abbildung 6.4 der Desktop-Version sieht man ein der Mobile-Version ähnliches Ergebnis. Das Cowboy-Mesh bekommt eine leichte Wölbung an der oberen Rückenpartie, welche rot eingekreist ist.

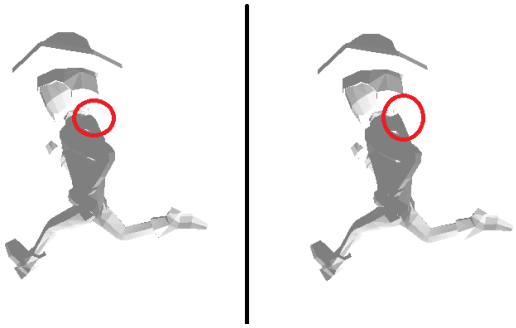

Abbildung 6.5: Auf der linken Bildhälfte das Linear Blend Skinning und auf der rechten Bildhälfte das Dual Quaternion Blend Skinning

6.6.2 Rotation des Zylinders um die Höhenachse

Am ersten Zylinderobjekt wird eine Rotation um 360 Grad um die Höhenachse animiert. Es handelt sich hierbei um einen Zylinder der aus 1148 Polygonen konstruiert ist, die gleichmäßig über das Mesh verteilt sind. Bei der extremen Rotation auf Abbildung 6.5 sieht man beim Linear Blend Skinning ganz klar den Volumenverlust durch den Candy-Wrapper-Effekt, da das Volumen am Gelenk auf einen Punkt zusammenfällt. Durch das Dual Quaternion Blend Skinning werden hier bereits bessere Ergebnisse erzielt, denn der Volumenverlust ist deutlich geringer.

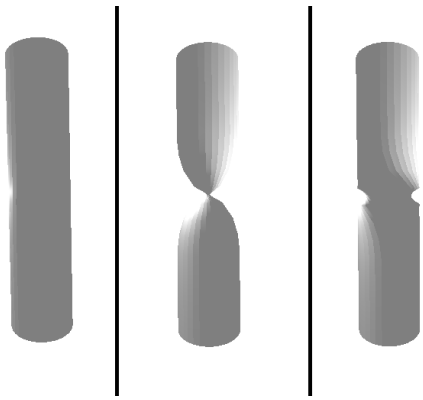

Abbildung 6.6: Links die Ursprungsposition, in der Mitte die Deformation nach LBS und rechts die Deformation nach DQS.

Vergleicht man die bisherigen Ergebnisse mit einem zweiten Zylinder, bei dem dieselbe Rotation animiert wird, der aber deutlich mehr (2044) Polygone am Gelenk hat, fällt sofort auf, dass der Volumenverlust durch das Linear Blend Skinning gleich bleibt. An Abbildung 6.6 sieht man, dass durch das Dual Quaternion Blend Skinning keinerlei Volumenverlust auftreten und sich die volumenerhaltende Eigenschaft der Quaternionen zeigt. Hierbei sei angemerkt, dass die Dreiecks-Normalen nicht neu berechnet werden, damit die Rotation auch ohne Volumenverlust erkennbar ist.

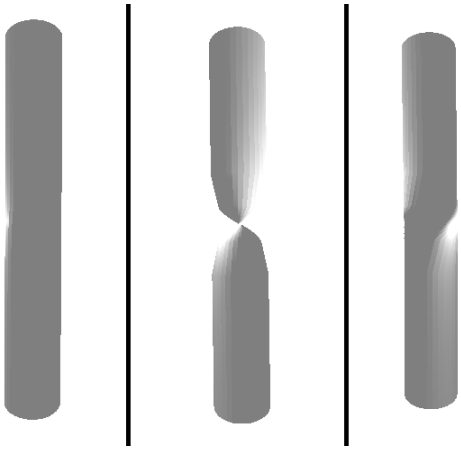

Abbildung 6.7: Links die Ursprungsposition, in der Mitte die Deformation nach LBS und rechts die Deformation nach DQS.

6.6.3 Abknicken an der halben Zylinderhöhe

Der selbe Zylinder der auf Abbildung 6.6 genutzt wurde, wird mit einer anderen Animation deformiert. Dabei wird der Zylinder an der halben Zylinderhöhe erst um etwa 90 Grad und dann um fast 180 Grad abgeknickt. Man erkennt auf Abbildung 6.7, dass die Deformation durch LBS zu einem deutlichen Volumenverlust führt.

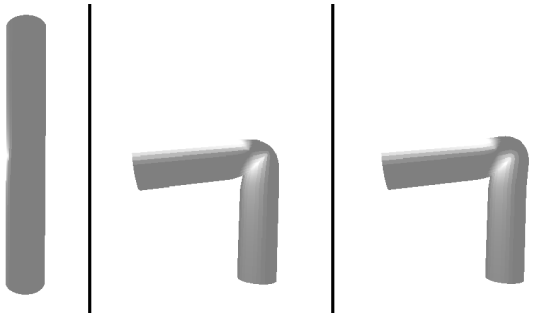

Abbildung 6.8: Links die Ursprungsposition, in der Mitte die Deformation nach LBS und rechts die Deformation nach DQS. Die Rotation beträgt 90 Grad.

Auf Abbildung 6.8, also wenn die obere Zylinderhälfte um fast 180 Grad abgeknickt wurde, ist am Mesh ein spitz zulaufender Bereich festzustellen. Durch DQS wird das Volumen wieder erhalten, aber gleichzeitig ist eine Wölbung am Gelenkbereich zu erkennen.

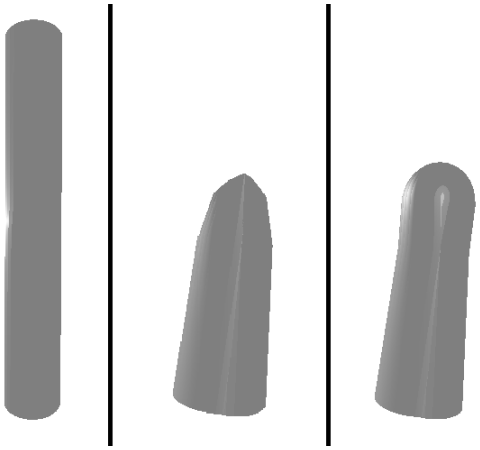

Abbildung 6.9: Links die Ursprungsposition, in der Mitte die Deformation nach LBS und rechts die Deformation nach DQS. Die Rotation beträgt annähernd 180 Grad.

7 Schluss

7.1 Zusammenfassung der Ergebnisse

Der Unterschied zwischen den beiden Skinning-Techniken zeigt sich vor allem dann, wenn extreme Rotationen berechnet werden. Dies hat man bei den Animationen mit den Zylindern besonders gut gesehen, da die Linear Blend Skinning Technik immer einen Volumenverlust am Mesh verursacht hat. Das volumenerhaltende Dual Quaternion Blend Skinning erhält das Volumen, sofern genug Vertices an den Gelenken enthalten sind, bringt aber in einer nicht optimierten Version Wölbungen mit sich, wenn die Rotationen extrem werden.
Auch am Cowboy-Mesh sind die Unterschiede beider Techniken zu erkennen, jedoch sind die Unterschiede deutlich geringer. Der Grund dafür ist, dass die Animation keine extremen Rotationen enthält und es durch die Linear Blend Skinning Technik zu keinem starken Volumenverlust kommt. Der entstehende Unterschied ist auf das Dual Quaternion Blend Skinning zurückzuführen, welches mit dem Wölben des Charakters ein eigenes Artefakt mitbringt.
Die Ursache für die niedrige Performanz (3-4 Frames per Second) der Smartphone-Applikation kann zwei Gründen unterliegen. Zum einen kann es sein, dass die Implementation nicht performant genug ist, die Deformation des Skeletts und die Übertragung auf das Mesh schnell genug zu berechnen. Weiterhin kann der hauptsächliche Grund an der Rechenleistung des Gerätes liegen.
Zusammenfassend kann man sagen, dass vor allem die optische Qualität des Dual Quaternion Blend Skinnings überzeugt.

7.2 Ausblick

Den nächsten Schritt kann man in verschiedene Richtungen setzen.
Eine Richtung beschäftigt sich mit der Frage der Effizienz der Implementation und ob eine

auf Smartphones performante Optimierung möglich ist. Dazu müsste man zunächst die aktuelle Implementation auf modernen Geräten testen und außerdem die Implementation untersuchen. Es bietet sich auch an die Evaluation auf weitere Meshes und Animationen zu erweitern, sodass Aussagen über Meshes mit noch höherer Polygonanzahl getroffen werden können.
Eine andere Richtung wäre die Frage nach den Grenzen von Skelett-Animationen. Hierfür könnte man das Dual Quaternion Blend Skinning mit der von Kim und Han entwickelten Lösung zum Beheben der Wölbungs-Artefakte erweitern und sich die Frage stellen. Im weiteren Verlauf bietet es sich an die Möglichkeit für Skalierungen zu schaffen die Quaternionen nicht mitbringen, indem man zusätzlich das Stretchable and Twistable Bones System und andere Vorteil bringende Systeme, wie das Spline-based Skinning, implementiert.
Die dritte Richtung entfernt sich von den Grundlagen und stellt die Frage, wie sich das DQS in andere Frameworks integrieren lässt. Zum aktuellen Zeitpunkt unterstützen viele Engines wie Unity oder die Unreal Engine 4 diese Skinning-Technik von Haus aus nicht, sodass es denkbar ist Erweiterungen zu entwickeln.

Literaturverzeichnis

[1] Apteker, R.T. ; Fisher, J.A. ; Kisimov, V.S. ; Neishlos, H.: Video acceptability and frame rate. In: *IEEE Multimedia* 2 (1995), Nr. 3, S. 32–40

[2] Ashraf, Golam ; Zhou, Junyu: Hardware accelerated skin deformation for animated crowds. In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, S. 226–237

[3] Baran, Ilya ; Popović, Jovan: Automatic rigging and animation of 3d characters. In: *ACM Transactions on Graphics* 26 (2007), jul, Nr. 3, S. 72

[4] Barnes, Mark: COLLADA. In: *ACM SIGGRAPH 2006 Courses on - SIGGRAPH 06*, ACM Press, 2006

[5] Billinghurst, Mark ; Clark, Adrian ; Lee, Gun: a survey of augmented reality. In: *Foundations and Trends® in Human–Computer Interaction* 8 (2015), Nr. 2-3, S. 73–272

[6] Brill, Manfred: *Computergrafik*. Hanser Fachbuchverlag, 2003. – ISBN 9783446221505

[7] Claypool, Kajal T. ; Claypool, Mark: On frame rate and player performance in first person shooter games. In: *Multimedia Systems* 13 (2007), apr, Nr. 1, S. 3–17

[8] Cordier, F. ; Magnenat-Thalmann, N.: A data-driven approach for real-time clothes simulation. In: *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, IEEE, 2004

[9] Dam, Erik B. ; Koch, Martin ; Lillholm, Martin: Quaternions, interpolation and animation / Department of Computer Science University of Copenhagen. 1998. – Forschungsbericht

[10] Dörner, Ralf ; Broll, Wolfgang ; Grimm, Paul ; Jung, Bernhard: *Virtual und augmented reality (vr / ar)*. Springer Berlin Heidelberg, 2014. – ISBN 3642289029

[11] Dudash, Bryan: Skinned instancing / NVIDIA Corporation. 2007. – Forschungsbericht

[12] Forstmann, Sven ; Ohya, Jun ; Krohn-Grimberghe, Artus ; McDougall, Ryan: Deformation Styles for Spline-based Skeletal Animation. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2007 (SCA '07), S. 141–150. – URL http://dl.acm.org/citation.cfm?id=1272690.1272710. – ISBN 978-1-59593-624-0

[13] Jacobson, Alec ; Baran, Ilya ; Popović, Jovan ; Sorkine, Olga: Bounded biharmonic weights for real-time deformation. In: *ACM Transactions on Graphics* 30 (2011), jul, Nr. 4, S. 1

[14] Jacobson, Alec ; Sorkine, Olga: Stretchable and twistable bones for skeletal shape deformation. In: *Proceedings of the 2011 SIGGRAPH Asia Conference*, ACM Press, 2011

[15] Kavan, Ladislav ; Žára, Jiří: Spherical Blend Skinning: A Real-time Deformation of Articulated Models. In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. New York, NY, USA : ACM, 2005 (I3D '05), S. 9–16. – URL http://doi.acm.org/10.1145/1053427.1053429. – ISBN 1-59593-013-2

[16] Kavan, Ladislav ; Zara, Jiri ; Collins, Steven ; O'Sullivan, Carol: Skinning with dual quaternions. In: *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2008)

[17] Kim, Phil: *Rigid Body Dynamics For Beginners: Euler angles & Quaternions*. CreateSpace Independent Publishing Platform, 2013. – ISBN 9781493598205

[18] Kim, YoungBeom ; Han, JungHyun: Bulging-free dual quaternion skinning. In: *Computer Animation and Virtual Worlds* 25 (2014), may, Nr. 3-4, S. 321–329

[19] Kuipers, J. B.: *Quaternions and Rotation Sequences*. Princeton University Press, 2002. – ISBN 0691102988

[20] Magnenat-thalmann, Nadia ; Laperrire, Richard ; Thalmann, Daniel ; Montréal, Université D.: Joint-Dependent Local Deformations for Hand Animation and Object Grasping. In: *In Proceedings on Graphics interface '88*, 1988, S. 26–33

[21] Mohr, Alex ; Gleicher, Michael: Building efficient, accurate character skins from examples. In: *ACM SIGGRAPH 2003 Papers on - SIGGRAPH 03*, ACM Press, 2003

[22] National Research Council, Computer S. ; Telecommunications, Committee on Virtual Reality Research a.: *virtual reality:: scientific and technological challenges*. NATL ACADEMY PR, 1995. – ISBN 0309051355

[23] Nedel, L. P. ; Thalmann, D.: Modeling and deformation of the human body using an anatomically-based approach. In: *Proceedings Computer Animation 98 (Cat. No.98EX169)*, IEEE Comput. Soc, 1998

[24] Parent, Rick: *Computer animation: algorithms and techniques*. Morgan Kaufmann, 2012

[25] Shoemake, Ken: Animating rotation with quaternion curves. In: *Proceedings of the 12th annual conference on Computer graphics and interactive techniques - SIGGRAPH 85*, ACM Press, 1985

[26] Vaillant, Rodolphe ; Barthe, Loïc ; Guennebaud, Gaël ; Cani, Marie-Paule ; Rohmer, Damien ; Wyvill, Brian ; Gourmel, Olivier ; Paulin, Mathias: Implicit skinning. In: *ACM Transactions on Graphics* 32 (2013), jul, Nr. 4, S. 1

[27] Wilhelms, Jane ; Gelder, Allen V.: Anatomically based modeling. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH 97*, ACM Press, 1997

A Anhang

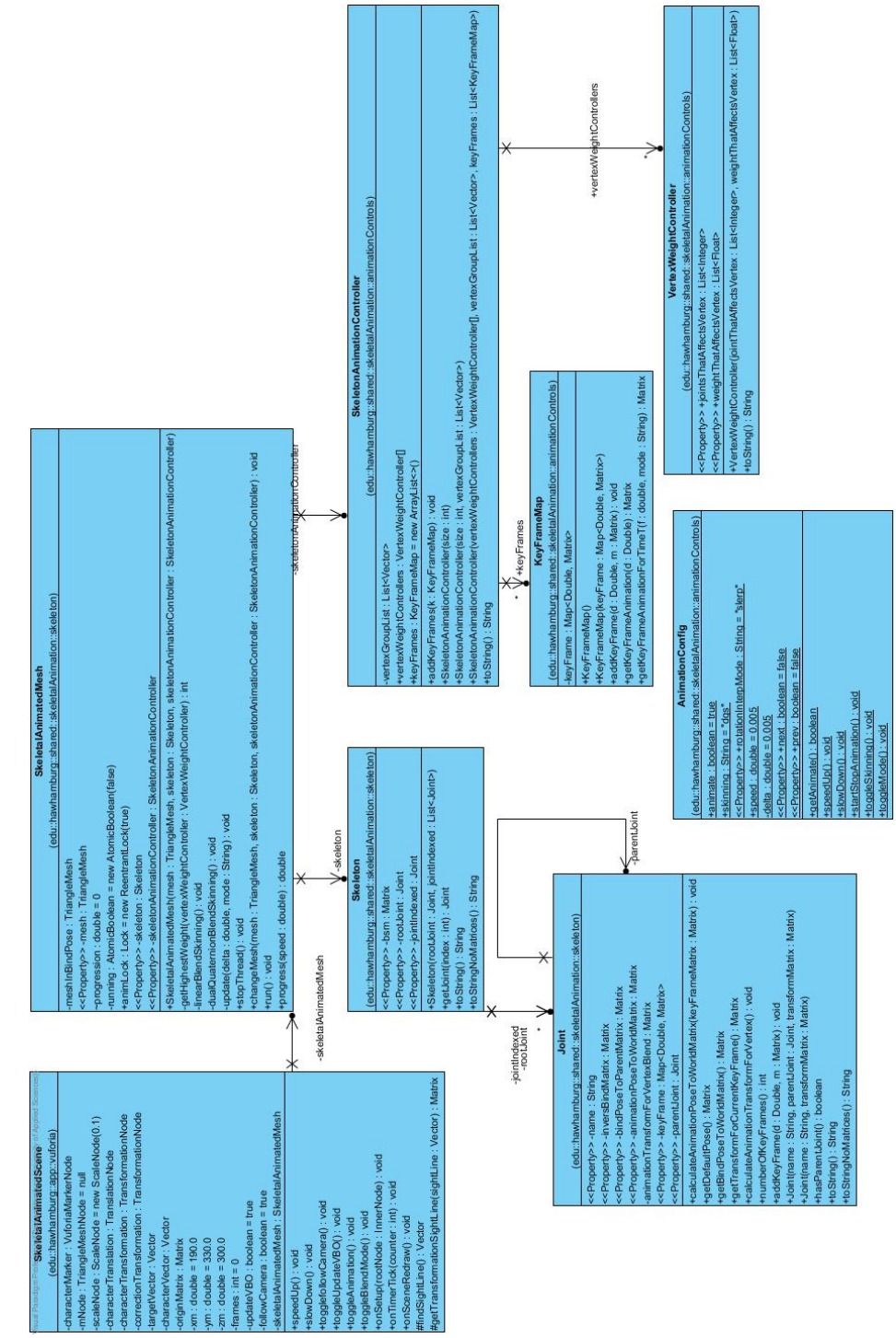

Abbildung A.1: Klassendiagramm der darstellenden und simulierenden Komponente

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original