UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Skinning und dessen Optimierungsverfahren

Bachelorarbeit

zur Erlangung des Grades einer Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Judith Bauerdiek

Erstgutachter: Prof. Dr. Stefan Müller
Institut für Computervisualistik, Arbeitsgruppe Computergrafik

Zweitgutachter: Gerrit Lochmann, M.Sc.
Institut für Computervisualistik, Arbeitsgruppe Computergrafik

Koblenz, im September 2014

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Bachelorarbeit
Judith Bauerdiek
(Mat. Nr. 209 210 083)

Thema: Skinning und dessen Optimierungsverfahren

Die Animation von Charakteren wird in vielen Bereichen der Computergrafik benötigt. Skinning verwendet ein Skelett um die daran assoziierten und gewichteten Vertices eines Meshs zu deformieren. Dadurch lassen sich sich fließende Bewegungen zwischen Keyframes berechnen.

Ziel der Arbeit ist die Implementierung von Software- und GPU-beschleunigtem Hardware-Skinning und dieses durch beispielsweise Hardware-Instancing zu optimieren. In einer Evaluationsreihe soll die Software bezüglich des optischen Eindrucks und der Performanz der Algorithmen bewertet werden.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche über Skinning und dessen Optimierungsmöglichkeiten
2. Implementierung von Hardware- und Software-Skinning
3. Optimierung der Verfahren
4. Evaluation hinsichtlich der Framerate und des optischen Eindrucks
5. Dokumentation der Ergebnisse

Koblenz, 06.02.2014

J. Bauerdiek

– Judith Bauerdiek –

S. Müller

– Prof. Dr. Stefan Müller –

Abstract

The animation of models has become an important part in different areas of everyday life. It is a demanding task for computer graphics to generate a natural deformation of organic models. Skinning is a common method to animate models without animating each vertex individually. The skin of the model deforms automatically by manipulating individual bones of a skeleton.
This bachelor thesis deals with the most common algorithm, linear blend skinning and aims to find some optimizations regarding the visual effect and performance. Additionally it presents certain instancing methods which are combined with the skinning methods in the application to show the advantages and disadvantages of the latter.

Zusammenfassung

Die Animation von Modellen ist zu einem wichtigen Teil in den unterschiedlichsten und alltäglichen Bereichen unseres Lebens geworden. Es ist eine anspruchsvolle Aufgabe der Computergrafik eine natürliche Deformation von organischen Modellen zu generieren. Skinning ist eine übliche Methode um Modelle zu animieren ohne die Animation jedes Vertices. Die Oberfläche des Modells wird automatisch durch die Manipulation einzelner Knochen eines Skeletts deformiert.
Diese Bachelorarbeit befasst sich mit dem meist genutzen Algorithmus, dem Linear Blend Skinning Algorithmus, der bezüglich des visuellen Eindrucks und der Performanz optimiert werden soll. Zusätzlich werden Instancing Methoden vorgestellt und in der Anwendung mit Skinning Methoden kombiniert um deren Vor- und Nachteile aufzuzeigen.

Inhaltsverzeichnis

1 Einleitung

Animationen sind heutzutage in vielen Bereichen zu finden. Unter anderem in der Film- und Spielindustrie oder auch in der Lehre für ein besseres Verständnis. Wenn dabei einzelne Objekte animiert werden sollen, ist es von Vorteil nicht jeden einzelnen Vertex zu animieren, sondern eine darunter liegende Struktur zu manipulieren, wodurch die Oberfläche automatisch deformiert wird.
Die Animation von einem Objekt durch die Deformation der Oberfläche wird Skinning genannt. Dabei gibt es viele verschiedene Ansätze vom geometrischen bis physikalischen Skinning. Die Art des Objekts spielt keine Rolle, es kann sich beispielsweise um einen Charakter, eine Cartoon-Figur, eine Pflanze, eine Maschine oder auch um ein Kleidungsstück handeln. Für das Objekt muss nur ein Skelett angelegt worden sein, damit diesem Transformationen zugewiesen werden können.
Für die Wahl der richtigen Animationsmethode muss meist ein Kompromiss aus optischem Anspruch und Rechenaufwand geschlossen werden. So ist es zum Beispiel von Vorteil, Kleidung nicht durch physikalisches Skinning zu animieren, wenn wenig Rechenleistung zur Verfügung steht, sondern durch geometrisches Skinning. Im Vergleich zum physikalischen Skinning ist das geometische Skinning realitätsfern.

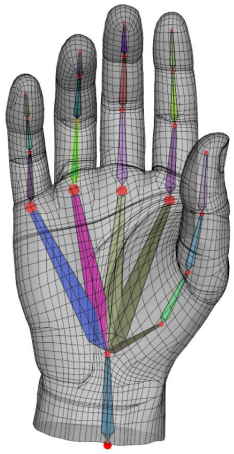


Abbildung 1: Beispiel für ein Handmodell mit Skelett [Vaillant]

In dieser Arbeit soll der Schwerpunkt auf das geometrische Skinning gelegt werden. Dabei werden sowohl Möglichkeiten für die Optimierung der Performanz als auch für den optischen Eindruck erarbeitet.
Die Grundlage bildet das Lineare Blend Skinning. Bei diesem wird die Skinning Transformation durch Matrizen repräsentiert, welche linear in-

terpoliert werden. Es ist das einfachste und effizienteste Verfahren, weswegen es weit verbreitet ist. Jedoch gibt es einige visuelle Nachteile, die hier durch die Verwendung von dualen Quaternionen, anstelle von Matrizen, vermindert werden sollen. Dabei sollen keine Veränderungen am Modell selbst vorgenommen werden. Für die Verbesserung der Performanz wird das Software Skinning auch als Hardware Skinning implementiert.
In vielen animierten Szenen werden gleiche oder ähnliche Instanzen in großer Anzahl benötigt, dies ist vor allem bei einfachen Strukturen, wie Grashalme und Blätter, kann aber auch für große Objekte, wie Charaktere für eine Menschenmasse, verwendet werden. Anstatt jedes Objekt einzeln anzulegen, werden lediglich mehrere Instanzen eines Objekts erzeugt.
Das sogenannte Instancing wurde als Software Instancing und mit zwei Varianten als Hardware Instancing implementiert. Ausgewertet wurde es hinsichtlich der Performanz. In Kombination mit den Skinning Verfahren sollen weitere Vor- und Nachteile dieser herausgestellt werden.

Abbildung 2: Beispiel für eine Szene, in der ein Instancing Verfahren angewendet worden ist. [Nguyen]

Am Anfang dieser Arbeit steht eine theoretische Einführung in das Skinning und dessen Optimierungen. Darauf folgt in Kapitel 3 die Erläuterung der Implementierung. Eine Auswertung hinsichtlich des optischen Eindrucks und der Performanz folgt im Kapitel 4. Die Arbeit wird mit einem Fazit und einem Ausblick abgeschlossen.

2 Skinning in der Theorie

Durch Skinning soll die Deformation eines Objekts ermöglicht werden. Ein Objekt besteht aus einzelnen Vertices, die als Primitive miteinander verbunden sind. Diese bilden wiederum ein Mesh. Für die Deformation wird zum Mesh ein Skelett erstellt, welches aus verschiedenen Bones besteht. Am Ursprung jedes Bone liegt dessen Rotationszentrum.
Ein Vertex wird von einem oder mehreren Bones beeinflusst. Dafür werden die Vertices mit den Bones assoziiert und jede Assoziation wird gewichtet. Durch die Gewichtung nehmen die Bones unterschiedlich viel Einfluss auf die Transformation eines Vertex, sodass sich eine weiche Deformation ergibt. Die Pose, in der sich das Mesh und das Skelett befinden, wenn die Vertices mit den Bones assoziiert werden, wird Bindpose genannt.

2.1 Linear Blend Skinning

Linear Blend Skinning (LBS) ist in der Literatur unter verschiedenen Namen bekannt, unter anderem als „Vertex Skinning"[Fernando].
Wie der Name schon sagt, wird bei LBS zwischen den Transformationen der einzelnen Bones, die den Vertex beeinflussen linear interpoliert.

$$v' = \sum_{n=0}^{N} w_n T(n) B(n)^{-1} v \tag{1}$$

Dabei ist N die Anzahl der Bones die den Vertex v beeinflussen. Um wie viel ein Bone n den Vertex v beeinflusst, wird durch die Gewichtung w_n festgelegt. Für diese gilt $\sum_{n=0}^{N} w_n = 1$. Die Transformation für Bone n ist $T(n)$. Der vollständig transformierte Vertex ist v'. Die inverse Bindpose-Matrix $B(n)^{-1}$ transformiert den Vertex in das lokale Koordinatensystem des Bones.
Die Vorteile von LBS liegen in der Einfachheit des Algorithmus. Sowohl die Implementierung als auch die spätere Anwendung sind leicht nachzuvollziehen, da die Eulersche Winkel nur drei Parameter benötigen und diese geometrisch nachvollziehbar sind.
Ein Nachteil ist die Nichtkommutativität von Matrizen, sodass zwölf unterschiedliche Rotationsreihenfolgen um die drei Achsen möglich sind. [**?**]
Ein weiterer Nachteil ist der Volumenverlust durch die lineare Interpolation, welcher in Abbildung 7(a) gezeigt wird. Hier beeinflussen zwei Bones den Vertex v. Wird v nur durch einen dieser Bones beeinflusst, so liegt er entweder bei v_1 oder v_2. Werden diese Werte mit einer Gewichtung von 0,5 interpoliert, so erhält man v'_{LBS}. Bei einer anderen Gewichtung liegt der transformierte Punkt auf der gepunkteten Linie. Für ein Skinning ohne Volumenverluste müssen nicht zwei transformierte Vertices linear interpoliert werden, sondern die Winkel. Man spricht von einer sphärischen Interpolation bzw. Spherical Blend Skinning (SBS). Dabei liegt der transformierte

Vertex auf der Kreislinie, wie in diesem Beispiel V'_{SBS}. [KavanSBS] Durch diese sphärische Interpolation können Volumenverluste vermieden werden. Je stumpfer der Winkel zwischen zwei Bones, desto größer wird auch der Abstand und damit der Fehler zwischen v'_{LBS} und v'_{SBS}, der schließlich zum „candy-wrapper"-Effekt führt[KavanDQ07]. Dabei wird, wie in Abbildung 3 zu sehen ist, durch eine große Rotation das Mesh am Bone-Ursprung auf einen einzigen Punkt deformiert.

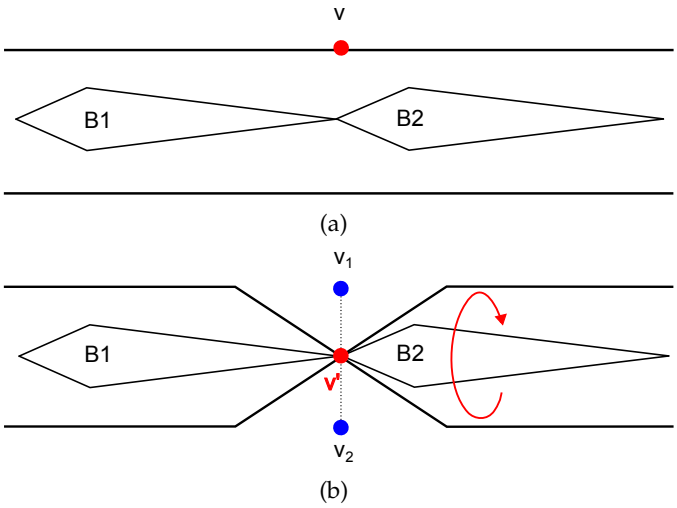


Abbildung 3: Darstellung des „Candy-Wrapper "-Effekts. 3(a) zeigt den zu transformierenden Vertex v und die Bones $B1$ und $B2$ in Bindpose. In 3(b) wird Bone $B2$ um $180°$ rotiert, dabei zeigen $v1$ und $v2$ die ungewichteten Transformationen durch den jeweiligen Bone $B1$ und $B2$, v' zeigt die interpolierte Transformation

Der erste Bone soll unverändert bleiben und der zweite um $180°$ um die Längsachse rotiert werden. Vertex v wird durch beide Knochen gleich beeinflusst $w_1 = w_2 = 0.5$, sodass v'_{LBS} schließlich auf dem Gelenk liegt, wie die folgende Gleichung zeigt:

$$\begin{aligned} v' &= \sum_{n=0}^{N} w_n T(n) B(n)^{-1} v \\ &= w_1(T_1 * B_1 * v) + w_2(T_2 * B_2 * v) \\ &= 0.5v_1 + 0.5v_2 \end{aligned} \tag{2}$$

Aber auch bei kleineren Rotationen gibt es Volumenverluste, wie man in Abbildung 4 erkennen kann. Dies liegt an der Interpolation. Dabei werden nicht einzelne Positionen, sondern Matrizen interpoliert, wie in (3) durch

die Umstellung von (1) gezeigt wird.

$$v' = (\sum_{n=0}^{N} w_n T_n(v) B(n)^{-1}) v \tag{3}$$

Durch die Interpolation der Matrizen bleibt keine starre Transformation [1] erhalten, sondern es entsteht eine affine Transformation, bei der die Skalierung mit eingeschlossen ist. Dies ist mit der nicht Abgeschlossenheit unter Addition von orthogonalen Matrizen, so wie es bei Rotationsmatrizen der Fall ist, zu begründen. [KavanDQ07]

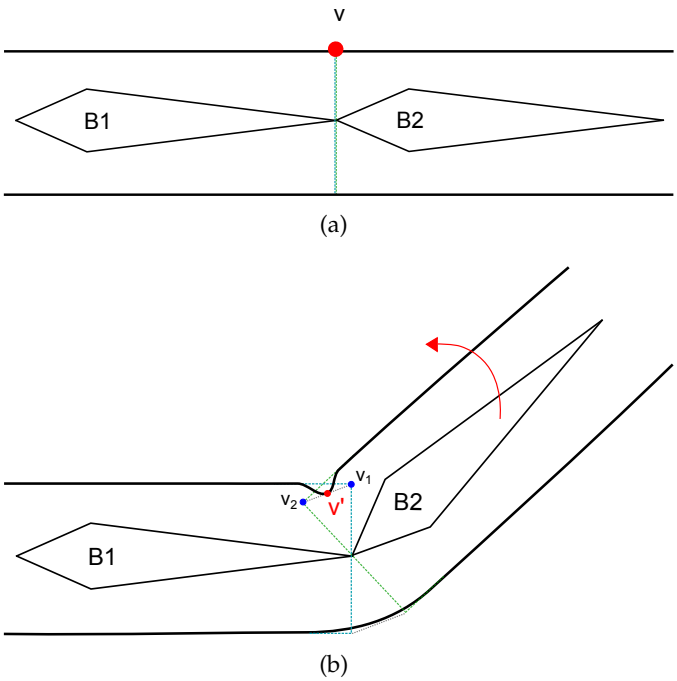


Abbildung 4: Volumenverlust durch Interpolation von Matrizen. 4(a) zeigt die Ausgangsposition mit dem zu transformierenden Vertex v. 4(b) zeigt die Rotation von Bone B_2, wodurch der Vertex v an die Position von Vertex v' transformiert wird. Die Vertices v_1 und v_2 verdeutlichen die ungewichtete Transformation des Vertex v nur durch den Bone $B1$ beziehungsweise $B2$.

Möchte man einen Volumenverlust vermeiden, so muss dafür gesorgt werden, dass die Transformationsmatrix zu keiner affinen Transformation wird.

[1] mit starrer Transformation ist die aus der Literatur bekannte rigid Transformation genannt, die lediglich Translation und Rotation mit einbezieht

2.2 Optimierung durch Duale Quaternionen

Bei einer Optimierung von LBS sollten Volumenverluste vermieden werden. Eine Option dafür sind duale Quaternionen. Im Folgenden werden lediglich die für das Skinning interessanten Operationen mit dualen Quaternionen eingeführt. Für eine detailliertere Einführung wird „A Beginners Guide to Dual-Quaternions“[Kenwright] empfohlen.
Eine Quaternion ist definiert als

$$q = w + (xi + yi + zk) \tag{4}$$

Dabei sind w, x, y und z die numerischen Werte und i, j, k die imaginären Komponenten der Quaternion. Meist werden sie jedoch nur als Vektorteil $v_q = (x, y, z)$ und Skalarteil $s_q = w$ notiert

$$q = (v_q, s_q) \tag{5}$$

Mit Hilfe einer Einheitsquaternion $\parallel q \parallel = 1$ lässt sich eine Rotation beschreiben. Die Rotation um einen Winkel θ und einer Einheitsachse n wird durch

$$q = (cos(\frac{\theta}{2}), nsin(\frac{\theta}{2})) \tag{6}$$

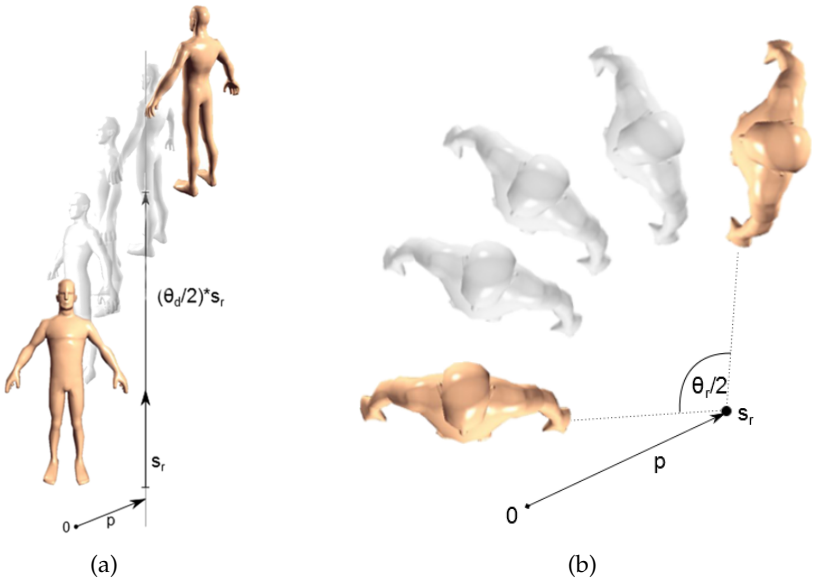
repräsentiert. Da beim Skinning nicht nur die Rotation von Interesse ist, sondern auch die Translation, reichen klassische Quaternionen nicht aus.
Allerdings kann aus einer Quaternion und einem Vektor, welche Rotation und Translation beschreiben, die gewünschte Transformation gebildet werden. Diese Transformation hat jedoch zum Nachteil, dass die Vertices um den Urpsrung des Meshs rotieren würden und nicht um den gewünschten Bone–Ursprung. [KavanSBS]
Um dies zu vermeiden, können durch die Algebra der dualen Zahlen duale Quaternionen gebildet werden. Sie bestehen aus zwei klassischen Quaternionen, dem realen Teil q_r, der die Rotation beschreibt und dem dualen Teil $q_d\varepsilon$, der die Translation beschreibt. [Kenwright]

$$\hat{q} = q_r + q_d\varepsilon \tag{7}$$

Dabei besteht der reale Teil aus einer Einheitsquaternion $q_r = r$. Der duale Teil setzt sich aus $q_d = \frac{1}{2}rt$ zusammen. Die Einheitsquaternion für die Translation besteht demnach aus der halben Rotation und der Quaternion für die Translation $\hat{t} = (0, t_x, t_y, t_z)$. Eine Alternative zu dieser Notation ist ein 8-Tupel aus reelen Zahlen.
Die Berechnung der dualen Einheitsquaternion erfolgt ähnlich wie bei einer Einheitsquaternion:

$$\hat{q} = (cos(\frac{\hat{\theta}}{2}), \hat{s}sin(\frac{\hat{\theta}}{2})) \tag{8}$$

Mit dem dualen Winkel $\hat{\theta} = \theta_r + \theta_d\varepsilon$ und dem dualen Vektor $\hat{s} = s_r + s_d\varepsilon$. Dabei beschreibt $\theta_r/2$ den Winkel der Rotation und $\theta_d/2$ die Größe der Translation. s_r gibt die Richtung der Achse an und s_d ist das Moment der Achse, welches sich aus $s_d = p \times s_r$ ergibt. Hierbei ist p ein Vektor, der vom Ursprung auf einen beliebigen Punkt auf die Transformationsachse s_0 zeigt. Dies wure in Abbildung 5 visualisiert.
Duale Quaternionen können eine Achse festlegen, die anders als bei klassischen Quaternionen, nicht durch den Ursprung verlaufen muss und um die rotiert und verschoben werden kann. Diese Transformation wird *Screw Motion* genannt, da sich der Vertex wie bei einer Schraube um die Achse dreht und gleichzeitig verschoben wird.[KavanDQ07]



(a) (b)

Abbildung 5: Visualisierung einer Transformation durch ein duales Quaternion. In der linken Abbildung 5(a) ist die Transformation von der Seite abgebildet. Dabei ist p der Vektor, der vom Ursprung auf die Transformationsachse s_r zeigt. Die Länge der Translation wird durch $\theta_d/2 * s_r$ beschrieben. Das Objekt wird schraubenartig durch das duale Quaternion entlang der Achse verschoben und um sie rotiert. Das gleiche Vorgehen, nur von oben betrachtet, ist in der rechten Abbildung 5(b) zu sehen.

Bei der linearen Interpolation von dualen Quaternionen (DQLB) werden die gewichteten Transformationen addiert und anschließend normiert:

$$DQLB(w;\hat{q}) = \frac{\sum_{n=0}^{N} w_n \hat{q}_n}{\| \sum_{n=0}^{N} w_n \hat{q}_n \|} \tag{9}$$

Dabei entspricht w_n einer Gewichtung und $\hat{q_n}$ einer dualen Quaternion. Die Normierung ist erforderlich, da für die Transformation eine duale Einheitsquaternion benötigt wird. Die Einheitsquaternion wird nun auf den Vertex angewendet. Bei der linearen Interpolation von Quaternionen bleibt eine starre Transformation erhalten, wodurch es nur zu minimalen Volumenverlusten kommt.
Duale Quaternionen sind wie klassische Quaternionen antipodal. Dies bedeutet, dass $\hat{q}$ und $\hat{-q}$ die gleiche Transformation repräsentieren.
Bei der Interpolation von zwei Quaternionen kann durch das Vorzeichen bestimmt werden, ob über den kürzesten oder längsten Pfad transformiert werden soll. Die Richtung der Interpolation kann durch den Winkel zwischen den beiden Quaternionen bestimmt werden:

$$cos\phi = q_0 * q_1, mit 0 \leq \phi < \pi \tag{10}$$

Bei dualen Quaternionen werden in der Gleichung (10) jeweils die Quaternionen für die Rotation ($\hat{q_0}$) multipliziert. Ist der Winkel $\phi > \frac{\pi}{2}$, so wird der längere Pfad für die Rotation gewählt. Durch die Negation einer Quaternion wird entlang des kürzeren Pfades rotiert. [Hanson]

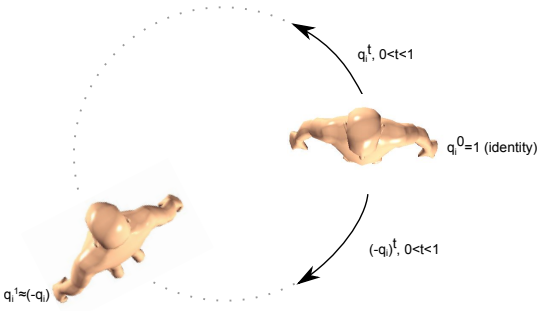


Abbildung 6: Visualisierung der Antipodalität von Quaternionen

Ein großer Vorteil von dualen Quaternionen ist, dass bei der linearen Interpolation das Ergebnis auf der Kreislinie zwischen den Quaternionen liegt. Abbildung 7 vergleicht die lineare Interpolation von LBS und die mit dualen Quaternionen jeweils mit der sphärischen Interpolation. Dabei ist zu erkennen, dass sich durch duale Quaternionen die sphärische Interpolation annähern lässt (Abbildung 7(b)). Der interpolierte Vertex v'_{DQLB} liegt immer auf der Kreislinie, weicht jedoch von dem sphärisch interpolierten Vertex v'_{SBS} leicht ab. Die Obergrenze dieser Abweichung liegt in der Theorie bei 8,15°. In der Praxis ist die Abweichung meist geringer. [KavanSBS]

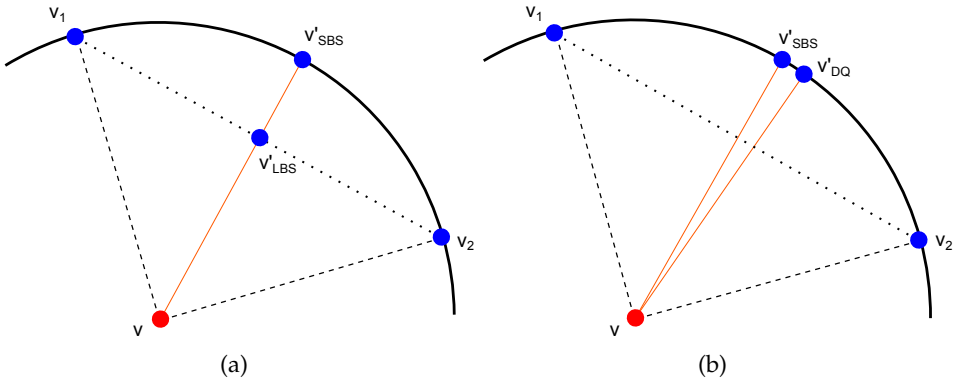


Abbildung 7: Zwei Bones beeinflussen v mit einer Gewichtung von 0,5. Wenn v nur vom ersten Bone beeinflusst wird, ist die neue Position v_1. Beeinflusst nur der zweite Bone v, so liegt das Ergebnis bei v_2. v' ist das berechnete Ergebnis. v'_{SBS} das gewünschte Ergebnis.7(a) zeigt den linear interpolierten Punkt (v'_{LBS}) 7(b) zeigt das Ergebnis linearer Interpolation von dualen Quaternionen (v'_{DQ})

2.3 Instanced Rendering

Mit Hilfe der Instacing-Technik soll, wie der Name schon sagt, eine große Anzahl von Instanzen eines Objektes in einer Szene gerendert werden. Oft wird Instanced Rendering bei der Darstellung von Blättern, Gras oder anderen kleinen Geometrien, die in großer Zahl in einer Szene vorkommen, verwendet. Aber auch für das Rendern größerer Geometrien kann es genutzt werden, wie zum Beispiel für die Darstellung einer Menschenmenge. Ziel ist es, das Instancing mit möglichst wenig Renderingaufrufen zu erreichen und die Anzahl der Daten gering zu halten.
Alle Instanzen haben die gleiche Anzahl an Primitiven und bestehen aus denselben Primitiv-Typen, wie das ursprüngliche Objekt. Den einzelnen Instanzen können unterschiedliche Parameter zugeordnet werden, wie zum Beispiel eine Farbe, sodass sie sich von anderen Instanzen unterscheiden. Zu den Parametern zählt auch eine eigene Model-Matrix, die für jede Instanz generiert wird. Dadurch erhält jedes Objekt seinen eigenen Platz in der Szene.
Für das Instancing sollen im Folgenden drei Methoden vorgestellt und anschließend verglichen werden:

2.3.1 Software Instancing

Bei dieser Methode werden durch erneute Renderingaufrufe mehrere Instanzen erzeugt. Da die Instanzen auf der CPU generiert werden, spricht man vom Software Skinning.

2.3.2 Instancing über eine OpenGL Funktion

Die OpenGL Funktion (ab Version 3.1) $glDrawElementsInstanced(...)$ erzeugt intern ebenfalls mehrere Renderingaufrufe, wie Algorithm 1 zeigt.

```
1  glDrwElementsIanstanced(mode, count, type, indices, primcount) {
2  if mode, count, or type is invalid then
3  |   generate appropriate error
4  else
5  |   for i ← 0 to primcount do
6  |   |   instanceID = i;
7  |   |   glDrawElements(mode, count, type, indices);
8  |   end
9  |   instanceID = 0;
10 end
11 }
```

Algorithm 1: Pseudocode der OpenGL Funktion $glDrawElementsInstanced(...)$ [Shreiner]

Durch eine Abfrage der Built-In Variable $instancedID$ kann im Vertex Shader auf die einzelne Instanz Einfluss genommen werden. Wie auch schon beim Software Instancing wird bei dieser Methode der Vertex Shader so oft aufgerufen, wie es Instanzen gibt. [Shreiner]

2.3.3 Hardware Instancing mit einem Geometry Shader

Bei der Hardware Instancing Methode mit einem Geometry Shader erfolgt lediglich nur ein Renderingaufruf. Der Vertex Shader wird somit nur einmal durchlaufen, sodass die Berechnung der Transformationen ebenfalls nur einmal statt findet. Die transformierten Vertices werden an den Geometry Shader übergeben, welcher die Vertices vervielfältigt und damit die Instanzen erzeugt. Schließlich wird der Fragment Shader aufgerufen. [Wright]

Geometry Shader Dieser optionale Shader wird zwischen Vertex und Fragment Shader aufgerufen. Eine der Hauptfunktion, das Vervielfältigen von Primitiven, kann für das Instancing verwendet werden. Als Input erhält er einzelne Primitive, deren Typ festgelegt sein muss. Der Zugriff auf alle Vertices des Primitivs ist innerhalb des Geometry Shaders möglich. Der Output besteht aus keinem oder mehreren Primitiven. Auch hier muss vorher der Primtivtyp und die maximale Anzahl der Vertices festgelegt werden. [Wright]

3 Umsetzung

Es wurde eine Anwendung implementiert, in der die in Kapitel 2 vorgestellten Methoden umgesetzt wurden.
Das dafür benötigte Modell wird mit einem eigenen Loader (dem FBXLoader, siehe dafür Kapitel 3.5) aus einer .fbx-Datei geladen, die zuvor mit Blender angefertigt worden ist. Die relevanten Daten werden in Vektoren gespeichert.
Zu jedem Vertex gehört eine Indexliste für die Bones, die den Vertex beeinflussen und eine gleich lange Liste mit den dazugehörigen Gewichtungen. Dabei wird die Anzahl der beeinflussenden Bones auf vier begrenzt und die Gewichtungen anschließend normalisiert. Die Bone-Hierarchie wird durch eine gerichtete „in-tree"-Struktur festgehalten, das heißt, dass jeder Bone seinen Parentbone referenziert. Pro Bone-Knoten wird der Name und der Index des Bones gespeichert. Eine Bindpose-Matrix beschreibt die gloable Transformation und somit die Position eines Bones zu dem Zeitpunkt an dem die Vertices mit den Bones assoziert wurden. Für das Skinning wird nur die inverse Bindpose-Matrix benötigt und eine relative Bindpose-Matrix berechnet. Die relative Bindpose-Matrix beschreibt die Position des Bones relativ zu seinem Parentbone.

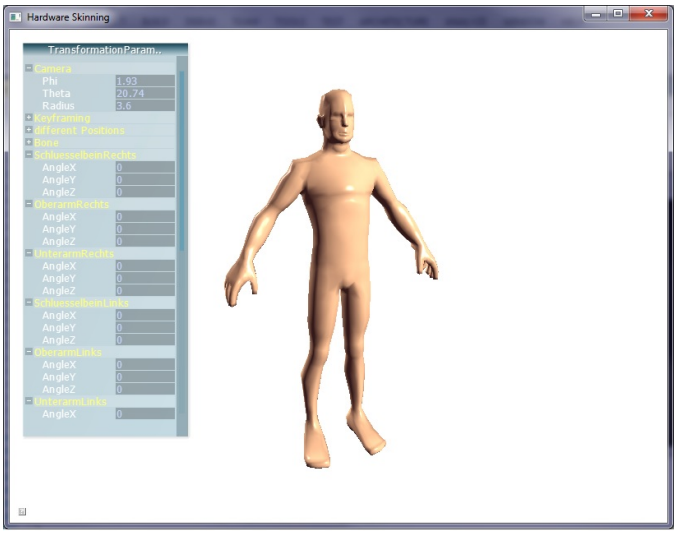


Abbildung 8: Beispiel eines Fensters der implementierten Anwendung. Neben dem Modell ist ein Widget eingefügt, über das man die Kameraposition auslesen und ändern kann. Die Rotationswinkel sind pro Bone einstellbar. Außerdem lässt sich eine Keyframing-Animation starten und die einzelnen Instanzen können beim Instancing in verschiedenen Positionen dargestellt werden.

Inwieweit die einzelnen Bones rotiert werden sollen, wird vom Benutzer

über ein Widget eingestellt. Dabei handelt es sich um die Winkel bezüglich der lokalen Achsen der Bones, die aufmultipliziert im *BoneTransformationSet* gespeichert werden. Da die Bones in einer Hierarchie angeordnet sind, werden sie nicht nur von ihrer eigenen Rotation beeinflusst, sondern auch durch die in der Hierarchie höher liegenden. Die Gesamttransformation, unter Berücksichtigung der Bone-Hierarchie, wird im *TransformationSet* gespeichert und kann so für die Skinning-Berechnung einfach ausgelesen werden.
Beim Skinning ist die Darstellung der Bones nicht vorgesehen. Sie sind nur als Hilfsmittel für die Deformation anzusehen. Deswegen wurde bei dieser Anwendung ebenfalls auf eine Darstellung verzichtet.

3.1 Linear Blend Skinning

Linear Blend Skinning wurde als Software Skinning und als Hardware Skinning implementiert.

3.1.1 Software Skinning

Die Berechnungen für Software Skinning werden vollständig auf der CPU ausgeführt.
Damit später die einzelnen Transformationen für einen Vertex einfach interpoliert werden können, wird für jeden Bone ein Eintrag im TransformationSet angelegt. Das TransformationSet ist in der Formel (1) als $T(n)$ notiert und setzt sich wie folgt zusammen:

$$T(n) = T(parent_n) * B_r(n) * t_n \tag{11}$$

Für den Bone n besteht das *TransformationSet* aus seiner vom Benutzer eingestellten Transformation t_n, welche in seinem lokalen Koordinatensystem festgelegt ist. Darauf wird die relative Bindpose-Matrix $B_r(n)$ und das *TransformationSet* seines Parents $T(parent)$ multipliziert. So wird die Transformation der Bones repräsentieren, die in der Hierarchie über dem Bone liegen.
Ist das *TransformationSet* aktualisiert, können die Vertices transformiert werden. Dafür wird zwischen den Einträgen im *TransformationSet* interpoliert, die Einfluss auf den Vertex haben. [Lewis]

Input : Vertex, BoneIndices, BoneWeights, BindPoseInverse, TransformationSet
Output: transformierter Vertex

```
1  for b ← 0 to 4 do
2      if boneIndices(b) ≠ −1 then
3          index ← boneIndices(b)
           boneTransformation ← transformationSet(index) ∗
           boneWeight(b) ∗ bindPoseInverse(index)
4          if b==0 then
5              transformation = boneTransfromation
6          else
7              transformation+ = boneTransformation
8          end
9      end
10 end
11 vertex = transformation ∗ vertex
```

Algorithm 2: Linear Blend Skinning Implementation

Algorithm 2 zeigt die Implementation der linearen Interpolation aus der Formel (1). Dazu werden die Koordinaten des zu transformierenden Vertices mit der Indexliste für die Bones, die Einfluss auf diesen haben, sowie ihre Gewichtungen benötigt. Mit Hilfe der Boneindices können die entsprechenden Transformationen aus dem *TransformationSet* und die dazugehörige invere Bindpose-Matrix ausgelesen werden. Letztere wird benötigt, damit der Vertex im lokalen Koordinatensystem des Bones liegt. Die Transformationen der einzelnen Bones wird addiert und die entstehende Matrix auf den zu transformierenden Vertex angewendet.
Beim Software Skinning werden diese Berechnungen in der Renderloop aufgerufen, sodass die fertig transformierten Vertices an die Shader weiter gegeben werden können. Hier ist die Deformation des Meshs bezüglich der Bone-Transformation abgeschlossen.

3.1.2 Hardware Skinning

Die Transformationsberechnungen erfolgen beim Hardware Skinning auf die gleiche Weise wie beim Software Skinning. Die beiden Verfahren unterscheiden sich lediglich dadurch, wo diese Berechnungen ausgeführt werden.
Das *TransformationSet* wird wieder auf der CPU berechnet und an den Vertex Shader übergeben. Hier werden auf der GPU die Berechnungen für LBS ausgeführt.

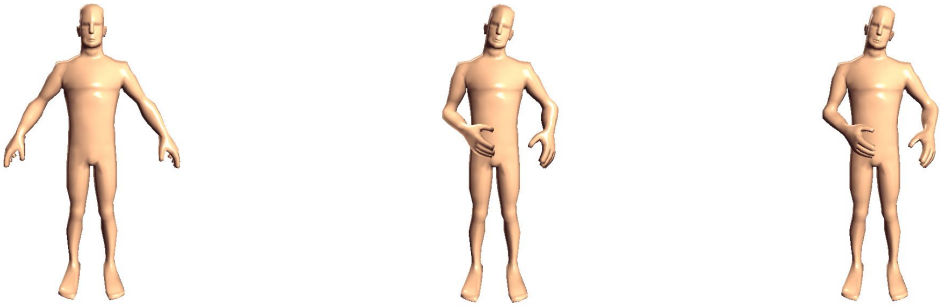
```
1  layout(location=0)in vec4 position;
2  layout(location=1)in vec4 normal;
3  layout(location=2)in vec4 boneIndices;
4  layout(location=3)in vec4 boneWeights;
5  uniform mat4 transformationSet[20];
6  uniform mat4 bindPoseInverse[20];
7  uniform mat4 modelViewProjection;
8  out vec3 passPosition;
9  for b ← 0 to 4 do
10     if boneIndices[b] ≠ −1 then
11         index ← boneIndices[b]
           boneTransformation ← transformationSet[index] ∗
           boneWeight[b] ∗ bindPoseInverse[index]
12         if b==0 then
13             transformation = boneTransfromation
14         else
15             transformation+ = boneTransformation
16         end
17     end
18 end
19 vertex = transformation ∗ vertex
   passPosition = modelViewProjection ∗ vertex;
```

Algorithm 3: Vertex Shader für Hardware Skinning

Der Vertex Shader erhält für jeden Aufruf und somit für jeden Vertex die entsprechenden Indices und Gewichtungen der Bones. Als uniform Variable wird das gesamte *TransformationSet* und die inversen Bindposematrizen übergeben. Die Berechnung der neuen Position des Vertex erfolgt wie in Algorithm 2. Der transformierte Vertex wird an den Fragment Shader übergeben.[Fernando]
In Abbildung 9(a) wird das Modell in Bindpose dem deformierten Modell durch Hardware Skinning 9(b) gegenüber gestellt.

3.2 Skinning mit dualen Quaternionen

Für das Skinning mit dualen Quaternionen wurde die Mathebibliothek *glm* verwendet. Die dualen Quaternionen werden hier als zwei Quaternionen, real und dual, abgespeichert, die jeweils durch ein 4-Tupel repräsentiert werden.
Die Bindpose-Matrizen und das *TransformationSet* werden wie gewohnt als 4x4 Matrix angelegt. Dies ist nötig, da bei Sonderfällen kein einheitliches Rotationszentrum berechnet werden kann und es dadurch zu fehlerhaften Darstellungen kommt. [KavanSBS]



(a) (b) (c)

Abbildung 9: 9(a) zeigt das Modell „Mensch“in BindPose. Das Ergebnis von einer Transformation der beiden Unterarme, des rechten Oberarms und des Kopfes mit LBS durch Hardware Skinning ist in 9(b) und durch DQLB in 9(c) zu sehen.

Für die Interpolation wird die Matrix, die sich aus der inversen Bindpose-Matrix und dem TransformationSet zusammensetzt, in eine duale Quaternion (siehe $transformationAtBone$ in im Algorithm 4) umgerechnet. Mit Hilfe einer Schleife werden die Transformationen interpoliert. Damit der kürzeste Pfad bei der Rotation garantiert ist, wird die Richtung der Quaternionenrotation bei jedem Schleifendurchlauf wie folgt abgefragt:

Input : $b \rightarrow$ Laufvariable der Schleife über beeinflussende Bones
Output: Transformationsquaternion entlang des kürzesten Pfades

```
1  if b == 0 then
2  |   q0 = transformationAtBone(b).real;
3  |   transformation = weight * transformationAtBone;
4  else
5  |   if dot(q0, transformationAtBone(b).real) < 0 then
6  |   |   weight* = −1;
7  |   end
8  |   transformation =
   |   weight * transformationAtBone + transformation;
9  end
10 normalize(transformation);
```

Algorithm 4: Überprüfung der Rotationsrichtung bei dualen Quaternionen

Die im Kaptiel über die Theorie des Skinnings mit dualen Quaternionen vorgestellte Gleichung (10) zeigt, dass sich aus dem Skalarprodukt zweier Quaternionen der Winkel zwischen diesen ergibt. Aus dieser Gleichung kann geschlussfolgert werden, dass wenn der Winkel zwischen der ersten Rotation (siehe $q0$ in Algorithm 4) und den darauf folgenden Rotationen

größer Null ist, der kürzeste Pfad gewählt wird. Andernfalls muss die Quaternion negiert werden. Dies ist durch die Negation des Gewichts umgesetzt, welches anschließend auf die Rotation multipliziert wird. [Mukundan] Danach wird normalisiert und die duale Quaternion kann auf den Vertex angewendet werden. Dies kann auf zwei verschiedene Weisen erfolgen:

$$\hat{v}' = \hat{q} * \hat{v} * \hat{q}^* \tag{12}$$

Dabei ist $\hat{q}^*$ die konjugierte duale Quaternion und $\hat{v}$ der zu einer dualen Einheitsquaternion umgerechnete Vertex. Alternativ kann die duale Quaternion auch in eine Transformationsmatrix umgewandelt werden und analog zu LBS auf den Vertex multipliziert werden: $v' = T * v$ Zwar verbraucht diese Methode mehr Speicherplatz, spart jedoch laut Eberly Performanz. [Eberly]

3.3 Instancing

Die Model-Matrix, die die Instanz an ihre individuelle Position transformiert, wird bei allen drei Instancing-Verfahren gleich berechnet. Durch die gewünschte Anzahl der Instanzen und der maximalen Anzahl pro Reihe, werden die Objekte in einem Rechteck angeordnet (Abbildung 10).

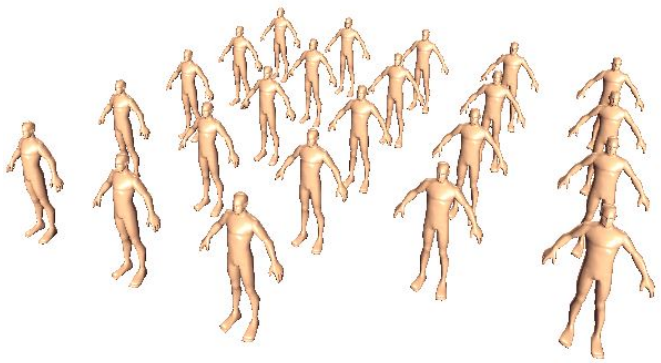


Abbildung 10: In diesem Beispiel wurde eine Model-Matrix generiert, die maximal 5 Objekte pro Reihe zulässt.

3.3.1 Software Instancing

Beim Software Instancing befindet sich der Renderaufruf in einer Schleife, die so oft durchlaufen wird, wie Instanzen erstellt werden sollen. Im Schleifenrumpf wird die Model-Matrix auf das Modell angewendet. Danach wird der Buffer mit den Vertices überschrieben. Durch das Aktualisieren des Buffers wird vermieden, dass es für jede Instanz einen eigenen

Buffer gibt, wodurch der Speicherverbrauch möglichst klein bleibt. Die nun folgenden Shader, der jeweiligen Skinning Methode, bleiben unverändert.

3.3.2 Hardware Instancing über eine OpenGL Funktion

OpenGL stellt ab Version 3.1 die Funktion *glDrawElementInstanced* zur Verfügung. Diese wurde bereits in Kapitel 2.3.2 eingeführt.
Mit dieser Funktion ist der Zugriff im Vertex Shader auf die Built-In Variable $instanceID$ möglich. Diese gibt die aktuelle Instanz an, sodass die entsprechende Model-Matrix generiert und schließlich auf den Vertex angewendet werden kann. Dieses Hardware Instancing hat gegenüber der Software Instancing Methode aus 3.3.1 den Vorteil, dass die Model-Matrix erst im Vertex Shader auf die Vertices angewendet wird. Dadurch lässt sich eine Performanz Verbesserung vermuten, da die Berechnungen für die Instanzen auf der Grafikkarte ausgeführt werden. [Shreiner]

3.3.3 Hardware Instancing mit einem Geometry Shader

Für Hardware Instancing mit einem Geometry Shader wird nur ein Renderaufruf benötigt. Dabei werden die Instanzen im Geometry Shader erzeugt:

```
1  layout (triangles) in;
2  layout(triangleStrip, maxVertices = 100) out;
3  in mat4 viewProjectionMatrix[];
4  uniform int instanceNumber;
5  void main(){
6  for i ← 0 to instanceNumber do
7  |   generate Modelmatrix model
8  |   for v ← 0 to vertices do
9  |   |   position = viewProjectionMatrix[n] ∗ model ∗ inPosition[n];
10 |   |   EmitVertex();
11 |   end
12 |   EndPrimitive();
13 end
14 }
```

Algorithm 5: Pseudocode des Geometry Shaders für Hardware Skinning

Algorithm 5 ist eine Vereinfachung des Geometry Shaders. In den ersten beiden Zeilen wird der Paramtertyp für die eingehenden und ausgehenden Primitive festgelegt. Mit $maxVertices$ wird angegeben, wie viele Vertices pro Shader-Aufruf weiter gegeben werden dürfen. Alle in-Variablen, wie die $viewProjection$-Matrix werden als Array angelegt, sodass es für jeden

Vertex des Primitivs eine entsprechende Variable gibt. Die uniform Variable $instanceNumber$ gibt die Gesamtanzahl der gewünschten Instanzen an.
Für jede Instanz wird eine Model-Matrix generiert und diese zusammen mit der $viewProjection$-Matrix auf alle Vertices des Primitivs angewendet (vgl. Zeile 9). Mit dem Geometry Shader spezifischen Befehl $EmitVertex()$ wird ein neuer Vertex im Primitiv erzeugt. Die Berechnung des aktuellen Primitivs wird durch $EndPrimitive()$ beendet, sodass der nächste *EmitVertex()*-Aufruf einen Vertex für die neue Instanz erzeugt. Da hier *EndPrimitive()* immer aufgerufen wird, sobald ein Triangle generiert wurde, erhält man in der Ausgabe Triangles und keinen TriangleStrip. Nach dem Geometry Shader wird wie gewohnt der Fragment Shader ausgeführt.
Beim Hardware Instancing mit einem Geometry Shader liegt der Vorteil gegenüber den anderen beiden Methoden darin, dass es nur einen Renderingaufruf gibt und der Vertex Shader nur für eine Instanz ausgeführt wird. [Wright]

3.4 Zusätzliche Features

Es wurden zwei zusätzlichen Features entwickelt:
Zum Einen eine Keyframing-Funktion. Die Animation startet dabei in der Bindpose und endet in der eingestellten Transformation. Dafür wurde bei LBS zwischen den Positionen und bei den Quaternionen zwischen den Winkeln interpoliert.
Zum Anderen ist es mögliche alle Instanzen in unterschiedlichen Posen darzustellen. Die individuelle Pose einer Instanz ist eine Interpolation zwischen der Bindpose und der eingestellten Pose. Die Implementation ist bei allen Kombinationen möglich. Beim Hardware Skinning mit Software Instancing ist eine Implementation jedoch ineffektiv. Denn damit der Vertex im lokalen Koordinatensystems des Bones liegt, müsste die inverse Model-Matrix angewendet werden um nach der Transformation erneut die Model-Matrix anzuwenden. Das Instancing würde also zweimal erfolgen.

3.5 FBXLoader

Fbx ist ein Dateiformat, welches die Elemente einer 3D Szene, wie Kamera, Licht, Mesh und NURBS speichert.
Für den FBXLoader wird gefordert, dass die 3D Szene mit Blender angelegt und als .fbx exportiert worden ist, da davon der Aufbau der .fbx-Datei abhängig ist. Beim Anlegen müssen einige Kriterien beachtet werden:

- Das Skelett darf nur einen root Bone haben.
- Der Mesh Ursprung muss im Weltkoordinatenursprung liegen.
- Für eine erleichterte Bedienung sollten die einzelnen Bones angemessen benannt werden.

- Es darf eine Boneanzahl von 25 nicht überschritten werden. Diese Grenze wurde für das Hardware Skinning festgelegt.

Ist die Datei richtig angelegt, können alle relevanten Daten ausgelesen werden. Für die spätere Darstellung werden die Vertices und Normalen benötigt. Die *Vertices* werden ausgelesen und in einem Vektor gespeichert, sodass die Stelle im Vektor gleichzeitig die ID ist.
Das globale Koordinatensystem in Blender ist ein Rechtssystem und wie folgt aufgebaut: Blickt man von vorne auf den Ursprung, zeigt die x-Achse nach rechts, die y-Achse nach hinten und die z-Achse nach oben. Dies stimmt nicht mit dem OpenGL Rechtssystem überein. Wenn man von vorne auf den Nullpunkt blickt, zeigt die X-Achse nach rechts, die y-Achse nach oben und die z-Achse nach vorne, wenn man von vorne auf den Nullpunkt blickt. Dies führt zu dem Problem, dass die Bindposematrizen der Bones nicht mehr mit den Koordinaten der Vertices übereinstimmen. Deswegen müssen alle Vertices vor dem Abspeichern noch um 90°um die x-Achse rotiert werden.
Für die Vertices wird eine indizierte Liste angelegt, die für das Rendering verwendet wird.
Da in der .fbx-Datei nur die Flächennormalen abgespeichert werden, für das in der Anwendung verwendete Phong Shading aber Vertexnormalen nötig sind, werden diese aus den Flächennormalen berechnet.
Für das Skelett werden die Bones in einer gerichtete „in-tree"-Struktur gespeichert. Dabei hat jeder Bone–Knoten eine BoneID, einen Namen und die Referenz zu seinem Parentbone. Die einzlenen Bones lassen sich durch ein Widget in der Anwendung manipulieren, wofür der Bone–Name verwendet wird. Die Referenz zum Parentbonte wird für die vollständige Berechnung der Transformation benötigt.
Die Bones werden im Ursprung erzeugt und mit Hilfe der Bindpose-Matrix an die richtige Position verschoben. Dies macht das Abspeichern der Koordinaten für Bones überflüssig. Von der Bindpose-Matrix wird jeweils nur die Inverse gespeichert und es wird die relative Bindpose-Matrix zum Parentbone berechnet.

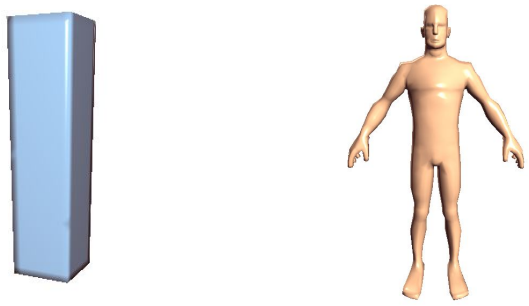
$$bindPoseRelative_n = bindPoseRelative^{-1}_{parent} * bindPose_n \qquad (13)$$

Beide Matrizen werden in Vektoren gespeichert, sodass diese mit der BoneID ausgelesen werden können. Die Bindpose-Matrizen wurden nicht im Bone-Knoten gespeichert, da so die spätere Übergabe an einen Shader oder an eine Funktion erleichtert wird.
Blender speichert die Länge eines Bones nicht. Deshalb müsste diese für eine Visualisierung erst nocht mittels der relativen Bindpose-Matrix berechnet werden. Da eine Darstellung der Bones jedoch beim Skinning nicht von Interesse ist wurde darauf verzichtet.

Pro Bone wird auch die Indexlist für Vertices mit den jeweiligen Gewichtungen ausgelesen, die durch diesen Bone beeinflusst werden. Da beim Skinning vom Vertex auf die Bones geschlossen werden soll, wird für jeden Vertex eine Indexliste für Bones angelegt und passend dazu die Gewichtungen gespeichert. Dabei wird die Bone-Anzahl auf maximal vier begrenzt und anschließend die Gewichtung normalisiert. [KavanDQ07]

4 Ergebnisse

In diesem Kaptiel sollen die einzelenen Skinning Methoden verglichen werden. Dafür liegen zwei Testmodelle vor: Ein einfacher Quader 11(a) und ein Mensch 11(b) mit mehreren Modellversionen, die sich in ihrer Anzahl an Vertices unterscheiden.



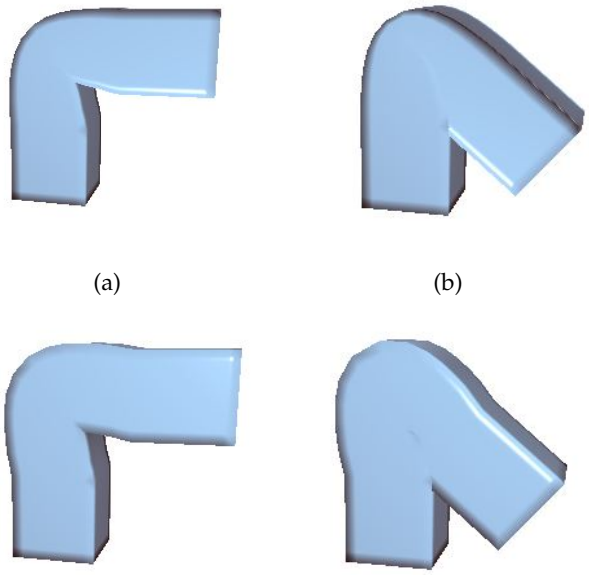
(a) (b)

Abbildung 11: Modelle in der Bindpose, die für die Auswertung verwendet worden sind: 11(a) „Quader“und 11(b) „Mensch“mit 3938 Vertices

Im Unterkapitel 4.1 „Visuelle Auswertung “werden die Ergebnisse der Skinning Verfahren LBS und DQLB bezüglich der Optik verglichen. Im zweiten Unterkaptiel werden die Performanz Unterschiede von Soft- und Hardware Skinning herausgestellt und die Instanzing Verfahren verglichen.

4.1 Visuelle Auswertung

Es wurden zwei unterschiedliche Skinning Berechnungen vorgestellt: Lineares Blend Skinning und Skinning mit dualen Quaternionen. In der Theorie wurde gezeigt, dass sich die Ergebnisse der beiden Methoden unterscheiden. Durch das angenäherte Sphärische Blend Skinning mittels dualen Quaternionen soll ein Volumenverlust, wie er bei LBS entsteht, vermieden werden.
In Abbildung 12 werden die Ergebnisse von LBS und DQLB mit einem ein-

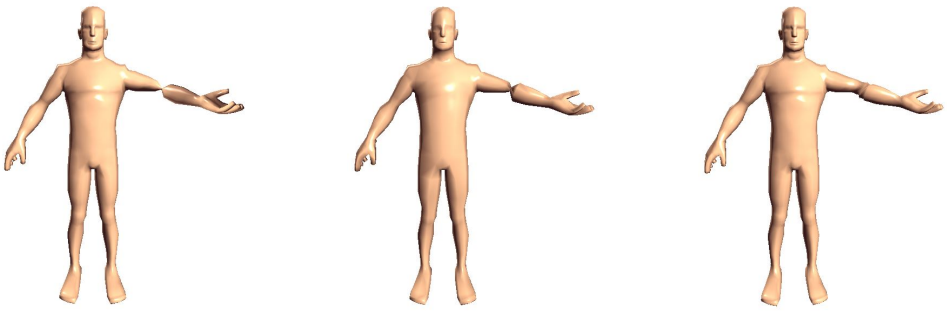


(a) (b)

(c) (d)

Abbildung 12: Das Modell „Quader “wurde am zweiten Bone um -90°und um -130°entlang der z-Achse rotiert. Die oberen Abbildungen 12(a) und 12(d) zeigen das Ergebnis mit LBS. Die unteren Abbildungen 12(c) und 12(d) zeigen das Ergebnis mit DQLB.

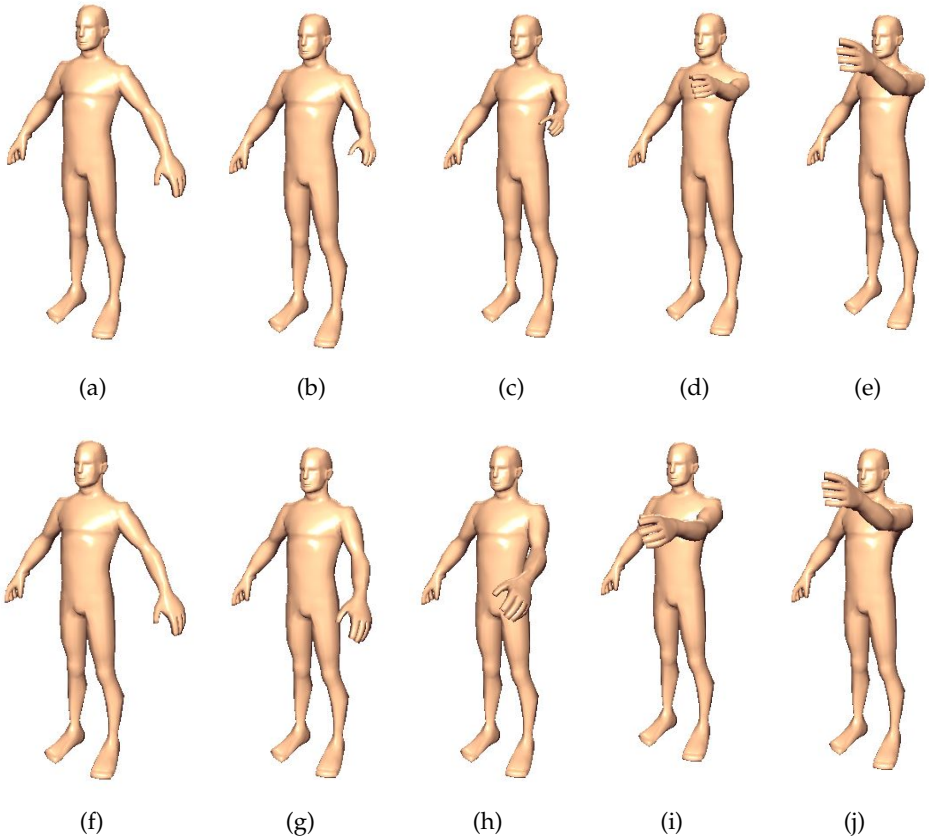
fachen Modell gegenüber gestellt. Dabei fällt positiv auf, dass der Quader, der mit dualen Quaternionen transformiert worden ist, keinen Volumenverlust aufweist. Die äußere Seite der Verformung ist weiter ausgeführt als die bei LBS.
Ein Nachteil ist bei beiden Methoden, dass bei einer zu großen Rotation die eine Oberfläche in den Körper eindringt. Dies ist deutlich in den Abbildungen 12(b) und 12(d) zu erkennen. Dieses Defizit lässt sich durch eine im Ausblick im Kapitel 5 vorgestellte Methode beheben.
In Kaptiel 2.1 wurde bereits der „candy-wrapper"-Effekt vorgestellt, bei dem das Mesh durch eine Rotation um 180°entlang der Längsachse auf einen Punkt zusammen geschnürt wird. Mit der Anwendung von dualen Quaternionen konnte dieser erfolgreich verhindert werden, wie Abbildung 13(c) zeigt. Besteht das Modell aus wenigen Primitiven, wie in Abbildung 13(b), so ist ein abgeschwächter „candy-wrapper"-Effekt auch bei dualen Quaternionen zu erkennen.



(a) (b) (c)

Abbildung 13: In 13(a) wurde der „candy-wrapper"-Effekt durch eine Rotation um 180° um die Längsachse des Unterarms mit LBS erzeugt (Modell „Mensch "mit 3938 Vertices). Die gleiche Rotation wurde mit dualen Quaternionen ausgeführt. Das Modell „Mensch "mit 986 Vertices 13(c) zeigt eine schwächere Form des „candy-wrapper"-Effekts. Bei Erhöhung auf 3938 Vertices kann der „candy-wrapper"-Effekt vermieden werden.

Auch beim Keyframing lassen sich unterschiedliche optische Ergebnisse feststellen. Das Keyframing mittels LBS wurde dem durch duale Quaternionen in Abbildung 14 gegenüber gestellt. Die Transformation durch LBS wirkt unnatürlich. Beim Keyframing wird sichtbar, dass um jede einzelne Achse (X,Y,Z) rotiert wird. In Abbildung 14(c) wird zusätzlich der unerwünschte Skalierungseffekt sichtbar. Die Transformation durch Quaternionen erfolgt aufgrund der „screw motion"-Achse auf direktem Wege.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Abbildung 14: Keyframing mit der linearen Interpolation von Positionen 14(a) –14(d) im Vergleich zum Quaternionen Keyframing 14(e) –14(i) bei dem die Winkel interpoliert werden. Dabei zeigt 14(c) deutlich den unerwünschten Skalierungseffekt. Beim Keyframing durch Quaternionen ist der direkte Weg zu erkennen.

4.2 Performanz

Neben den Ansprüchen an die visuelle Qualität ist die Performanz ein Hauptbewertungspunkt des Skinnings. Sie wurde auf einem System mit einem Intel Core i3-4130 Prozessor gemessen. Die Ergebnisse werden anhand von Graphen dargelegt, bei denen auf der Y-Achse jeweils die Millisekunden pro Frame aufgetragen sind.
Zunächst soll der Unterschied zwischen Soft- und Hardware Skinning herausgestellt werden. Dafür wurde in den einzelnen Testläufen die Vertexanzahl von 986 auf 3938 bis 15746 des Modells „Mensch “erhöht. Bei einer weiteren Unterteilung der Primitive, sodass 62978 Vertices entstehen, war die Anwendung nicht mehr ausführbar. Die Anzahl der Vertices wurden im Diagramm in Abbildung 15 auf der X-Achse aufgetragen.

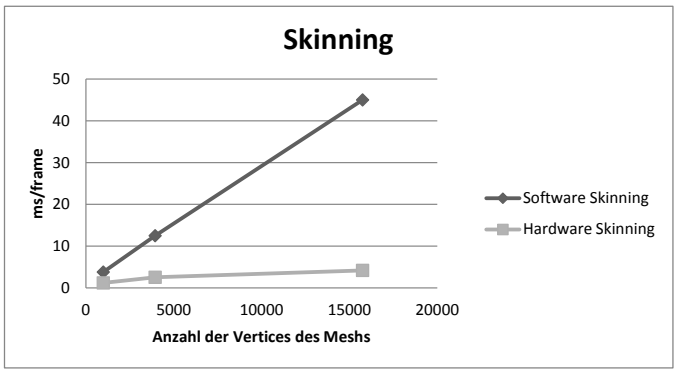
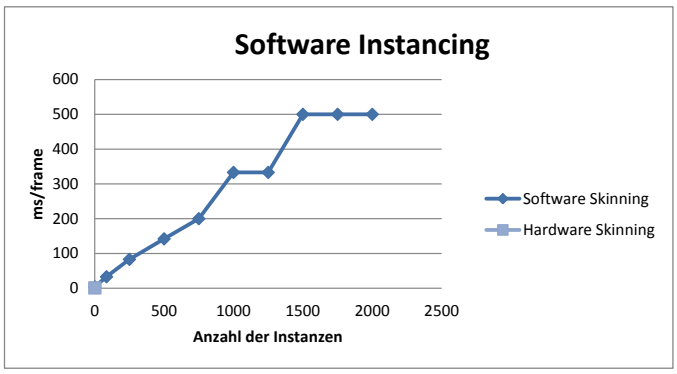
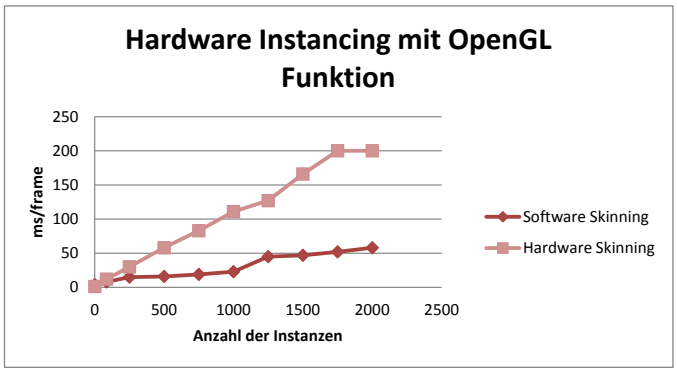


Abbildung 15: Performanz von Soft- und Hardware Skinning beim Erhöhen der Vertices des Meshs

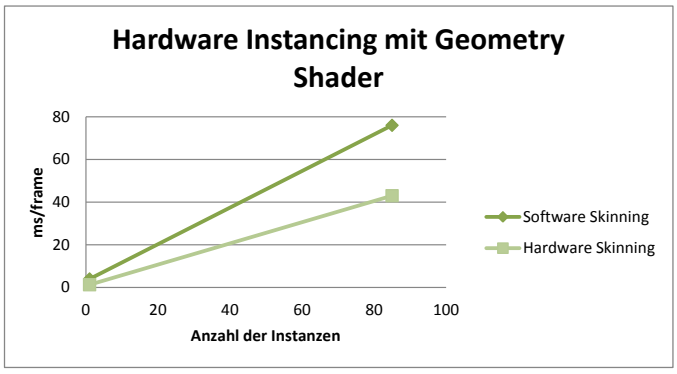
Je höher die Anzahl der Vertices ist, desto langsamer wird die Anwendung. Obwohl es sich um den gleichen Algorithmus handelt, steigt die Bildfrequenzrate beim Hardware Skinning wesentlich langsamer an als beim Software Skinning.
Im Folgenden werden die einzelnen Instancing Verfahren in Kombination mit Soft- und Hardware Skinning verglichen. Dabei wurde das Testmodell „Mensch “mit 986 Vertices verwendet. Die Unterschiede gehen aus den einzelnen Diagrammen in Abbildung 16 hervor, bei denen auf der X-Achse die jeweilige Instanzen-Anzahl notiert wurde.
Das Software Instancing funktioniert nur in Kombination mit Software Skinning. Denn werden zuerst mehrere Instanzen durch Software Instancing erzeugt und anschließend erst durch Hardware Skinning deformiert, so haben die erzeugten Instanzen beim Transformieren das falsche Rotationszentrum und es entstehen unerwünschte Deformationen.
Hardware Instancing über die OpenGL Funktion eignet sich besonders in Kombination mit Software Skinning. Der Vorteil liegt darin, dass die Ver-



(a)



(b)



(c)

Abbildung 16: Performanz der unterschiedlichen Instancing Methoden, jeweils mit Soft- und Hardware Skinning gemessen

tices nur ein einziges Mal transformiert werden, dann an den Vertex Shader weiter gegeben werden und dort vervielfältigt werden. In Kombination mit Hardware Skinning werden die nicht transformierten Vertices an den Vertex Shader weitergegeben. Da für jede Instanz der Vertex Shader aufgerufen wird, werden auch für jede Instanz die Skinning Transformationen berechnet. Dadurch entsteht ein größerer Rechenaufwand, womit ein Performanzverlust einhergeht.
Das Hardware Instancing mittels Geometry Shader zeigt einen wesentlich kleineren Messbereich, da die hier verwendete Hardware für mehr Instanzen nicht geeignet ist. Die maximale Anzahl an output-Vertices des Geometry Shaders liegt bei 255, mit denen man 85 Instanzen erzeugen kann. Überschreitet man diese Grenze, kann es zu fehlerhaften Darstellungen wie in Abbildung 17 kommen.

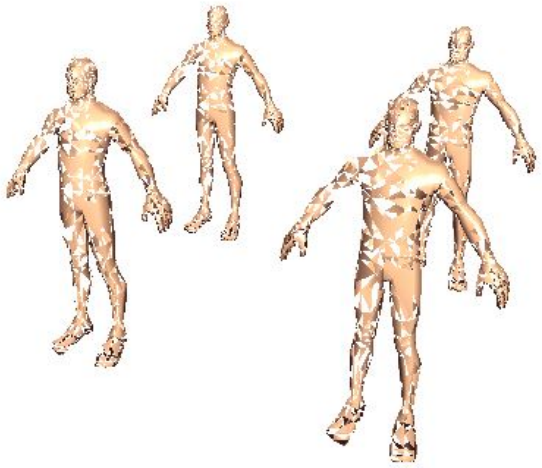


Abbildung 17: Ausschnitt aus einer Instancing Szene, bei der ein fehlerhaftes Rendering aufgetretetn ist, da die maximalen output-Vertices vom Geometry Shader überschritten wurden.

Solange die Hardware für die Anzahl der Instanzen ausgelegt ist, ist die Kombination mit Hardware Skinning schneller. Da der Instancing Schritt noch weiter nach hinten in der Rendering Pipeline gewandert ist, wird nun auch beim Hardware Skinning nur einmal die Skinning Transformation durchgeführt und das transformierte Mesh im Geometry Shader vervielfältigt.
Um alle drei Instancing Methoden miteinander vergleichen zu können wurde eine weitere Messung mit Software Skinning durchgeführt und die Instanzschritte verkleinert. Die Ergebnisse sind in Abbildung 18 zu sehen.

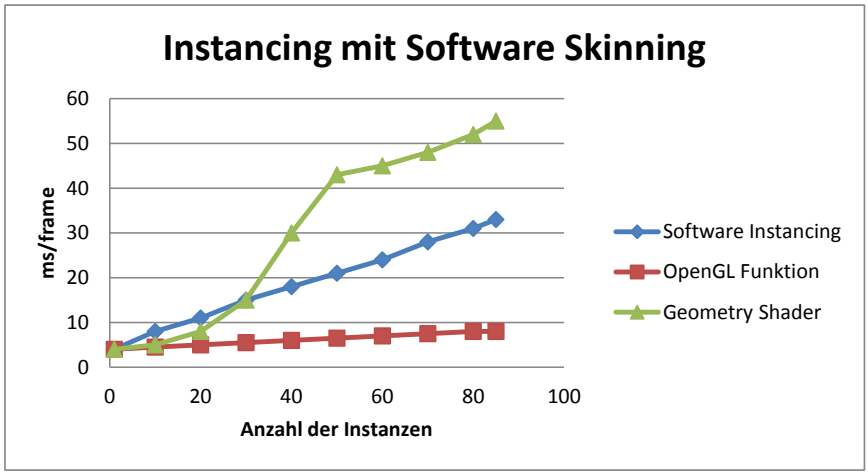


Abbildung 18: Instancing Methoden im direkten Vergleich in Kombination mit Software Skinning.

Die Geschwindigkeit von Software Instancing nimmt pro Instanz durchschnittlich um 0,4 ms/frame ab. Dieser gleichmäßige Abfall ist beim Hardware Instancing durch den Geometry Shader nicht gegeben. Bei 30 Instanzen fällt die Geschwindigkeit sogar noch unter die Geschwindigkeit des Software Skinnings. Scheinbar ist hier eine Grenze an Instanzen erreicht worden, bei der die on-board Grafikkarte des Prozessors ausgelastet ist. Man kann vermuten, dass Ergebnisse auf dem Prozessorspeicher zwischengespeichert werden. Die Hardware Instancing Methode mit der OpenGL Funktion erzeugt am schnellsten alle Instanzen und hat im Vergleich zum Software Skinning nur einen schwachen Abfall der Performanz.

5 Fazit und Ausblick

5.1 Fazit

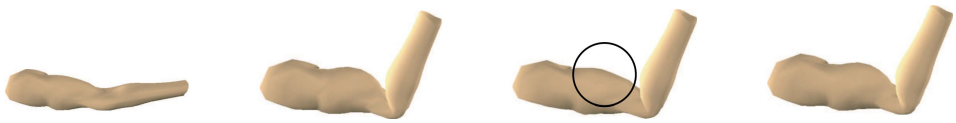
Diese Arbeit hat sich mit der Optimierung von Skinning beschäftigt. Dabei wurde der LBS Algorithmus als Software Skinning implementiert und als Grundgerüst verwendet.
Durch duale Quaternionen konnte eine natürlichere Deformation des Mesh erzielt werden. Die gewünschten Verbesserungen von LBS, Volumenverlust und den „candy-wrapper"-Effekt zu vermeiden, wurden durch die angenäherte sphärische Interpolation erreicht. Die entstandenen Wölbungen entsprechen jedoch nicht den Wölbungen von Muskeln, wodurch keine naturalistische Darstellung erreicht werden konnte. Beim Keyframing stellte sich die „screw motion"-Achse als Vorteil heraus, da die Animation so auf direktem Weg erfolgt.
Zusätzlich wurden drei verschiedene Instancing Verfahren implementiert und mit Soft- und Hardware Skinning kombiniert. Aus deren Ergebnissen folgendes festzuhalten ist:

- Die Verwendung von Software Instancing bietet sich nur an, wenn in den Shadern die zu rendernden Vertices nicht mehr verändert werden (vgl. Hardware Skinning kombiniert mit Software Instancing).
- Die beste Peformanz bildet die Kombination aus Software Skinning und Hardware Instancing mittels OpenGL Funktion.
- Der Geometry Shader eignet sich kaum für die Vervielfältigung eines Objekts. Zum Einen, weil die möglichen Instanzen stark begrenzt sind und zum Anderen kann es bei schlechter Grafikhardware zu einer schlechteren Performanz als beim Software Instancing kommen.
- Software Skinning hat gegenüber dem Hardware Skinning den Vorteil, dass die Vertices schon auf der CPU fertig transformiert sind. Dies ist zum Beispiel bei der Verwendung einer Physik Engine, die unter Umständen die Kollision zweier Objekte berechnen soll, notwendig, da deren Berechnungen ebenfalls auf der CPU laufen und dafür das Objekt fertig transformiert sein muss.

5.2 Ausblick

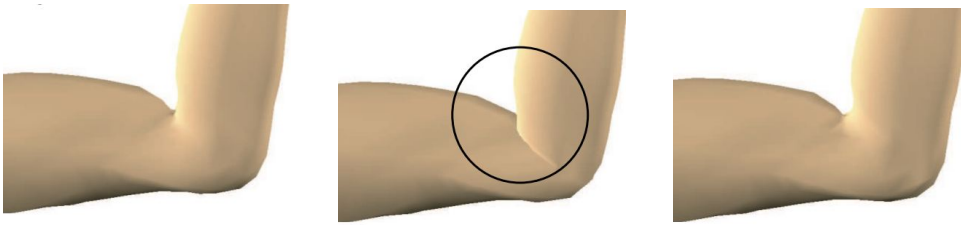
Neben dem hier vorgestellten geometrischen Skinning durch LBS oder DQLB gibt es noch weitere Möglichkeiten plausible Skin-Deformationen zu berechnen. Die zwei folgenden Methoden verbessern den visuellen Effekt durch zusätzliche Muskelwölbungen und das Verhindern von Mesh Überschneidungen.

Die Methode von Mohr und Gleicher generiert zusätzliche Bones. Für diese wird zuvor das skelettierte Mesh in einem Modellierungsprogramm in verschiedenen Posen abgespeichert. Mit Hilfe dieser *examples* werden die Gewichtungen für die zusätzlichen Bones ermittelt, welche bei der Transformation die Vertices auf die gleiche Weise beeinflussen. Durch die generierten Bones werden Muskelwölbungen möglich (Abbildung 19) und die Überschneidung der Oberflächen wird verhindert (Abbildung 20). [Mohr]



(a) (b) (c) (d)

Abbildung 19: 19(a) Arm in Bindpose, 19(b) *example* des Originals, 19(c) Deformation mittels LBS, 19(d) Deformation mit generierten Bones [Mohr]



(a) (b) (c)

Abbildung 20: 20(a) ein *example* des Originals, 20(b) Deformation mittels LBS, 20(c) Deformation mit generierten Bones [Mohr]

Die zweite Methode ist von Vaillant et al.. Bei dieser Methode wird nicht nur verhindert, dass sich die Oberflächen nicht überschneiden, sondern es wird zusätzlich ein natürlicher Kontakt der Oberfläche berechnet. Dabei wird das gewichtete und skelettierte Mesh in Segmente eingeteilt, zu denen mit Hilfe von Feldfunktionen, Isoflächen gebildet werden. Bei der Transformation werden die Vertices des Meshs transformiert, wie auch die Feldfunktion. Dabei wandert der Vertex solange entlang des Feldgradienten bis sein individueller Isowert erreicht ist oder er auf einen Feldgradienten eines anderen Vertex stößt, sodass eine Kontaktregion entsteht. Das Resultat dieses Algorithmus wird in Abbildung 21 dem LBS gegenüber gestellt. [Vaillant]

Um eine noch bessere Performanz beim Instancing im Zusammenhang mit Skinning zu erreichen, veröffentlichten Ashraf und Junyu ihre Arbeit [Ashraf].

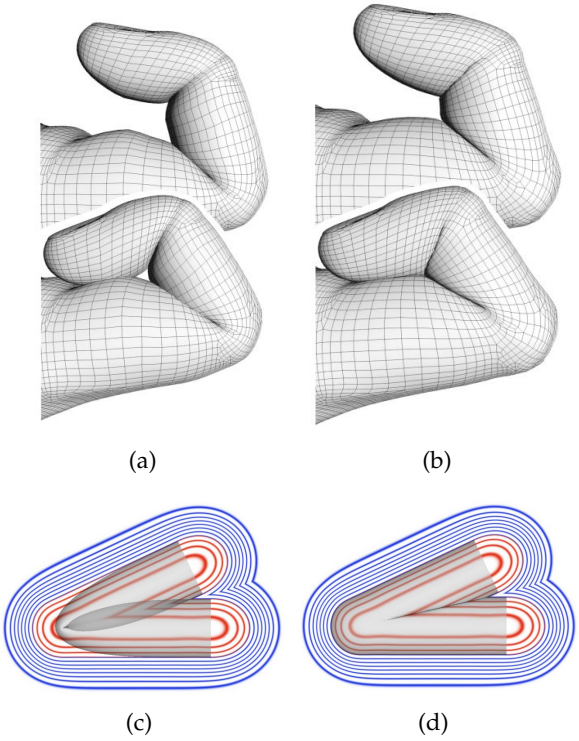


Abbildung 21: Die erste Spalte stellt die Transformation mit LBS dar, die zweite zeigt das implizierte Skinning. Die obere Zeile zeigt die praktische Anwendung der Verfahren. Die untere Zeile zeigt die Isoflächen. [Vaillant]

Dazu werden zwei Renderdurchläufe benutzt. Die zum Skinning benötigten Informationen werden in 2D-Texturen gespeichert, sodass die Transformationen im Fragment Shader berechnet werden können. Die Ergebnisse werden in einer output-Textur gespeichert und entweder an die CPU zurück gegeben (*Read Back*-Variante) oder direkt an den Vertex Shader des zweiten Renderdurchlaufs weiter geleitet (*Vertex Texture Fetch*-Variante). Im zweiten Renderdurchlauf erfolgt das Instanzing durch eine globale Transformation pro Instanz. Damit die Duplikate nicht sofort erkannt werden, werden den Instanzen zusätzliche Parameter mitgegeben. Anschließend wird wie gewohnt beleuchtet. Die Variante, in der die transformierten Werte zurück an die CPU gegeben werden, stellt sich als die mit der besseren Performanz heraus (Abbildung 23). [Ashraf]

Abbildung 22: Eine Beispielszene, die mit der Methode von Ashraf und Junyu erzeugt wurde.[Ashraf]

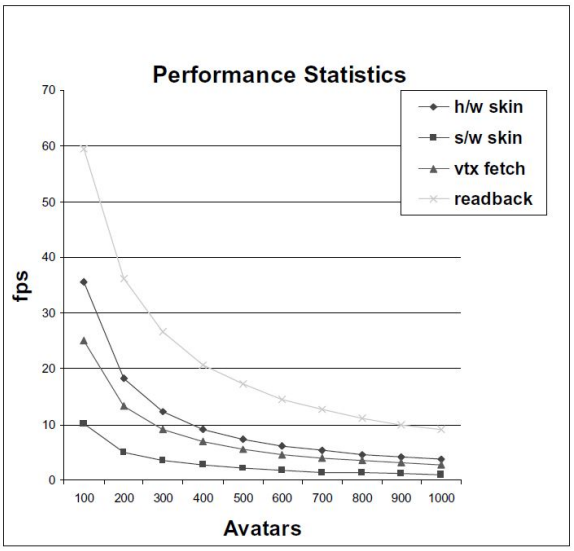


Abbildung 23: Die Performanzergebnisse für Instancing mit einer GeForce6600: Hardware Skinning (h/w), Software Skinning (s/w), Vertex Texture Fetch (vtx fetch), Read Back (readback) [Ashraf]

Dieser Ausblick zeigt, dass mit geometrischem Skinning noch wesentlich

natürlichere Deformationen berechnet werden können als in dieser Arbeit vorgestellte Methoden. Dabei muss nur ein Kompromiss mit zusätzlichem Aufwand geschlossen werden.

A Danksagung

Nach 32 Seiten Ausarbeitung und 2563 Zeilen Code in 209 Tagen möchte ich allen Danken, die meine Launen ertragen haben.

Ganz besonderer Dank geht dabei an ...

Norman, für Kaffeekränzchen, motivierende Worte, Abrufbereitschaft, Denkfehler finden und die Drecksarbeit

Magdalena, fürs immer da sein und für ihre Grammatikkünste

Daniela, fürs tägliche Zuhören bei völliger Ahnungslosigkeit

Jonas, fürs Jonas sein

Sebastian, für 6 verschwendete Stunden und fürs Tür hinter mir zu machen

Jule, der zweiten Grammatikfee

Lubosz, der sich tapfer durch meinen chaotischen Code gewühlt hat

Lisa, für morgendliche Motivationsnachrichten

Elena, für den alten Deal, der scheinbar immer noch steht

Arne, der mir eine vernünftige Arbeitsumgebung geschaffen hat

Chrstioph, fürs 1... 2... 3...

meine Eltern, die sich nie beschweren, dass ich so selten zu Hause bin

funny frisch und **Ambros Senseo**, für die tägliche Dosis

Literatur

[Ashraf] Golam Ashraf and Junyu Zhou. Hardware accelerated skin deformation for animated crowds. In *Proceedings of the 13th International Conference on Multimedia Modeling - Volume Part II*, MMM'07, pages 226–237, Berlin, Heidelberg, 2006. Springer-Verlag.

[Eberly] David H. Eberly. *3D game engine design - a practical approach to real-time computer graphics.* Morgan Kaufmann, 2001.

[Fernando] Randima Fernando and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[Hanson] A.J. Hanson. *Visualizing Quaternions*. Morgan Kaufmann series in interactive 3D technology. Morgan Kaufmann, 2006.

[KavanDQ06] Ladislav Kavan, Steven Collins, and Jiri Zara. Dual quaternions for rigid transformation blending. Technical report, 2006.

[KavanDQ07] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 39–46, New York, NY, USA, 2007. ACM.

[KavanSBS] Ladislav Kavan and Jiří Žára. Spherical blend skinning: A real-time deformation of articulated models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 9–16, New York, NY, USA, 2005. ACM.

[Kenwright] Ben Kenwright. A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3d character hierarchies. pages 1–13, 2012. WSCG 2012 Communication Proceedings, Conference June. 2012.

[Lewis] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Mohr] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. *ACM Trans. Graph.*, 22(3):562–568, July 2003.

[Mukundan] R. Mukundan. Quaternions: From classical mechanics to computer graphics, and beyond. In *Proceedings of the 7 th Asian Technology Conference in Mathematics, 2002*, 2002.

[Nguyen] Hubert Nguyen. *Gpu Gems 3*. Addison-Wesley Professional, first edition, 2007.

[Shreiner] D. Shreiner, G. Sellers, J.M. Kessenich, and B.M. Licea-Kane. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*. OpenGL. Pearson Education, 2013.

[Vaillant] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. Implicit skinning: Real-time skin deformation with contact modeling. *ACM Trans. Graph.*, 32(4):125:1–125:12, July 2013.

[Wright] Richard S. Wright, Nicholas Haemel, Graham Sellers, and Benjamin Lipchak. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional, 5th edition, 2010.