Interaktives Motion-Capturing zur Echtzeitanimation virtueller Agenten

Bernhard-Andreas Brüning[1]
Marc Erich Latoschik[2]
Ipke Wachsmuth[3]

AG Wissensbasierte Systeme[1,3]
Technische Fakultät
Universität Bielefeld
P.O. 10 01 31
D-33615 Bielefeld

Tel.: +49 521 106-2917
Fax.: +49 521 106-2962
b.a.b@web.de[1]
ipke@techfak.uni-bielefeld.de[3]

International Media Informatics[2]
Department 4
FHTW Berlin
Treskowallee 8
D- 10318 Berlin

Tel.: +49 (0) 171 786 3265
marc.latoschik@fhtw-berlin.de

Abstract: Motion Capturing (MoCap) ermöglicht realistische Bewegungen virtueller Agenten. Hier wird ein System für die optische Bewegungserfassung und GPU-beschleunigte Bewegungssynthese vorgestellt. Das Verfahren berücksichtigt Clusterumgebungen heutiger VR Systeme. Mittels eines optischen Trackingsystems werden die Bewegungen eines Benutzers in einer immersiven virtuellen Umgebung auf Basis der Positions- und Ausrichtungsdaten einiger Marker an signifikanten Körperpunkten erfasst. Die Übertragung auf einen virtuellen Agenten erfolgt interaktiv und unterstützt die Anpassung unterschiedlicher Skelettproportionen. Signifikante Teile der benötigten kinematischen Berechnungen auf Basis einer Denavit-Hartenberg-Repräsentation sowie die für die Animation des Agenten notwendigen Meshdeformationen werden dabei in Echtzeit über eine GPU-basierte (GPU-Graphical Processing Unit) Implementierung der Algorithmen realisiert.

Stichworte: Motion Capturing, Charakteranimation, GPU-Kinematik

1. Einleitung

Motion Capturing (MoCap) ist eine Grundlage der Gestaltung interaktiver Mensch-Maschine-Schnittstellen. Das Ziel ist die möglichst genaue und vollständige Erfassung der Bewegungen eines oder mehrerer Benutzer. Die über MoCap gewonnenen Bewegungsinformationen können daraufhin vielfältig eingesetzt werden. Zwei wichtige Einsatzmöglichkeiten sind die Analyse der Bewegungen zur Interaktion mit virtuellen Umgebungen sowie die Synthese der Bewegungen zur Animation virtueller Agenten. Eine Übertragung der Bewegungsdaten eines menschlichen Akteurs auf einen virtuellen Agenten generiert hier sehr realistische Bewegungsprofile. Bei der

Bewegungsübertragung müssen mehrere Probleme gelöst werden. Zuallererst müssen die Bewegungsdaten erfasst werden. Je nach Sensoren können dabei direkt Gelenkstellungsdaten oder auch Positions- und/oder Ausrichtungsdaten (6DOF - Degree of Freedom) von Markern an signifikanten Körperpunkten eines Akteurs, in der Regel repräsentiert durch ein Skelettmodell, gemessen werden. Die Aufnahme der Bewegungen erfolgt in der folgenden Studie über ein optisches Trackingsystem in Verbindung mit einer CAVE™ - Cave Automatic Virtual Environment [CruzSan92] als immersive Projektionseinrichtung, welche auch in der späteren Agenteninteraktion eingesetzt wird. Der gerichtete Zusammenhang zwischen Gelenkwinkeln und 6DOF-Informationen wird durch die Vorwärtskinematik, die umgekehrte Richtung durch die inverse Kinematik bestimmt. Dazu müssen die Skelettproportionen des Benutzers auf die interne mathematische Repräsentation des zugrunde liegenden Agentenskeletts übertragen werden. Ausgehend von einer Oberflächenbeschreibung des Körpers über ein so genanntes Polygonmesh muss das Mesh gemäß des Agentenskeletts dergestalt verformt werden, dass es in glaubwürdig synthetisierten Agentenbewegungen resultiert. Beide Aufgaben, Kinematikberechung sowie Meshdeformation, sind rechenintensive Prozesse. Aktuelle Fortschritte in der Architektur computergraphischer Systeme, speziell der so genannten GPUs (Graphical Processing Units), erlauben die Nutzung der speziellen SIMD (Single Instruction Multiple Data) Einheiten auch für die Berechnung nicht-graphischer Prozesse. Der hier vorgestellte Ansatz lagert Teile der Kinematikberechnung sowie die Meshdeformation auf die GPU aus. Dieses ermöglicht eine interaktive Echtzeitanimation virtueller Agenten auch in Cluster-basierten VR-Systemen, da das Verfahren die Netzwerklast zwischen den Clusterknoten deutlich reduziert.

2. Grundlagen

Grundlage heutiger Echtzeit-Rastercomputergrafik ist die Benutzung dedizierter Prozessoren, der GPUs, für die Berechnung der Grafikalgorithmen innerhalb der Grafikpipeline. Die vier Hauptprozesse innerhalb der Pipeline sind „Vertex Transformationen“, „Primitive Assembly and Rasterization“, „Fragment Texturing and Coloring“ und „Raster Operations“. Heutige GPUs basieren dabei nicht mehr auf einer festen Funktionsverdrahtung, der sogenannten „Fixed Function Pipeline–FFP“, sondern ermöglichen weitgehende programmgesteuerte Eingriffe in den verschiedenen Stufen der grafischen Verarbeitung durch Vertex- und Pixel-Shader-Programme. Der Aufbau der Grafikpipeline ist in Abbildung 1 dargestellt.

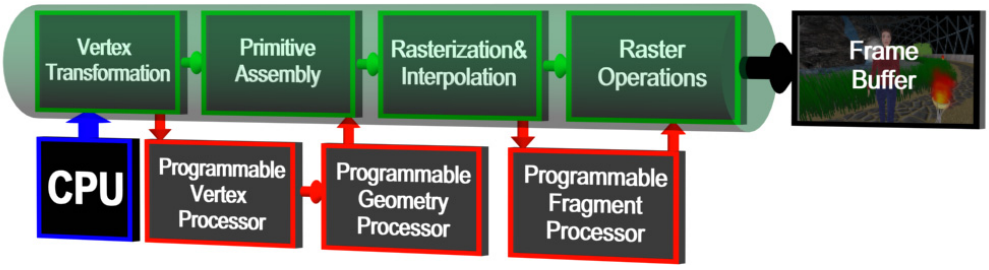

Abbildung 1: Die programmierbare Grafikpipeline (nach [RaKi03, Seite 17])

Hauptaufgabe eines Vertex-Processors ist die Manipulation von Vertexposition, Vertexfarbe

sowie Vertexnormale. Hauptaufgabe eines Pixel-Processors ist die Berechnung der Fragmente, der gerasterten Entitäten vor der eigentlichen Übertragung in den Framebuffer. Hier können die benötigten Fragmentparameter wie Farbe oder Textur berechnet und/oder manipuliert werden, etwa durch die Anwendung der Beleuchtungsmodelle pro Fragment anstatt pro Vertex wie in der FFP. Die Evolution von FFP zu frei programmierbaren Shadern hat in den letzten Jahren einen erheblichen Fortschritt gemacht. GPUs (nach dem zum Zeitpunkt der Veröffentlichung aktuellen) Shader Model 4 (SM4) stellen darüber hinaus auch geometrische Operationen zum Erstellen und Entfernen einzelner Vertices in einem Geometry-Shader zur Verfügung (ausgeführt auf dem „Programmable Geometry Processor“, s. Abbildung 1). Dieser ist zwischen den Vertex- und Pixel-Shaderstufen angeordnet. Shader-Programme werden heutzutage in Hochsprachen wie Nvidias CG - C for Graphics, GLSL - OpenGL Shading Language, HLSL - High Level Shading Language oder RenderMan beschrieben.

3. Stand der Forschung

Der vorgestellte Ansatz zur Bewegungserfassung orientiert sich an [Ma06]. Der Einsatz einer immersiven Projektionseinrichtung mit fest installiertem Trackingsystem resultiert in einem eingeschränkten Bewegungsraum für den Benutzer. Dies bedingt die teilweise Zerlegung der Bewegungserfassung bei Bewegungen mit relativ hohen Lokationsänderungen in Teilbewegungen. Der hier vorgestellte Ansatz betrachtet dagegen den gesamten Körper.
Wu beschreibt die Berechnung der benötigten Vorwärtskinematik auf der GPU innerhalb der Pixel-Shaderstufe [Wu06]. Erscheint die Verwendung eines Pixel-Shaders anstatt eines Vertex-Shaders für Kinematikberechnungen unter Berücksichtigung der jeweils zentralen Berechnungs-primitiven unangemessen, wird dies durch zwei Faktoren erklärt: Erstens war die Anzahl der Pixel-Shader auf der GPU in der eingesetzten Shaderarchitektur höher als die der Vertex-Shader-Einheiten (bei GPUs bis zum SM3). Zweitens unterstützen Pixel-Shader im SM3 komplexere Operationen. So gibt es dort etwa kein Limit für Texturinstruktionen [Re04], wodurch beliebig viele Matrizen gespeichert oder bearbeitet werden können. Da GPUs nach SM4 keine expliziten Vertex- oder Pixel-Shader-Einheiten besitzen und nur noch mit Unified-Shader-Einheiten (Streamprozessoren) bestückt sind, ist diese Verlagerung überflüssig. Die Shadereinheiten können im SM4 je nach Bedarf ohne signifikanten operativen Unterschied für die entsprechenden Aufgaben eingesetzt werden.

4. Setup der Cave-Tracking-System Kombination

Basis der Bewegungserfassung und der späteren Agentensimulation ist eine dreiseitige CAVE™ mit der Grundfläche von 2.65x2.65m und einer Höhe von 2.12m. Die stereoskopische Projektion wird mittels Zirkularpolarisationsfiltern realisiert. Das verwendete optische Trackingsystem der Firma AR-Tracking verwendet 9 Kameras, um die Positionen einzelner Marker oder die Positionen und Ausrichtungen eindeutig identifizierbarer so genannter Rigid-Body Marker (RBM) im Raum zu verfolgen. Die Positionierung der Kameras ist auf Grund des CAVE™-

Layouts eingeschränkt. Abbildung 2 zeigt das verwendete CAVE™-Tracking-Setup (links) im Vergleich zu einer idealisierten Positionierung (rechts) mit möglichst maximalem optischem Abdeckungsbereich. Das Rendering wird in der aktuellen Konfiguration auf zwei verschiedene Arten durchgeführt. Neben einer PC-basierten Einzelserverlösung mit 3 Grafikkarten und 6 Ausgängen [RaFröLa07] wird die Darstellung alternativ durch einen Verbund von 6 Renderknoten und einem Zentralserver realisiert. Durch die verteilte Darstellung des CAVE™-Setups auf mehrere Rechner muss eine Visualisierung verwendet werden, welche die Netzwerklast zwischen den einzelnen Rechnern möglichst gering hält.

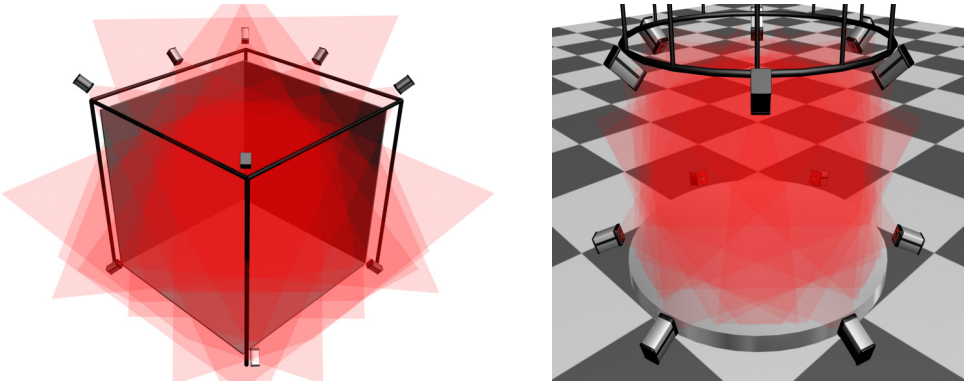

Abbildung 2: CAVE™-Tracking-System-Setup und ideale Tracking-Arena.

Die Animation innerhalb eines Vertex-Shader-Programms auf den GPUs der einzelnen Knoten unterstützt sowohl die Einzelserver-, als auch Clustervariante. Auf diese Weise wird vermieden, die veränderte Meshgeometrie zu übertragen. Übertragen werden wenige Parameter, z.B. von Winkelangaben.

5. Berechnung der Gelenkstellungen

5.1 Positionen der Marker

Um die aktuellen Gelenkstellungen des Akteurs zu ermitteln, sind die Positionen der RBM am Körper maßgeblich. Sie werden für das Tracking idealerweise so an den Körperteilen platziert, dass aus deren Positionen und Ausrichtungen relativ zu den Gelenken (z.B. an der Außenseite in der Mitte zwischen den Knochen des Ober- und Unterarms) diese Gelenkpositionen aus den aktuellen 6DOF-Daten der RBM ermittelt werden können. Um diese Berechnungen möglichst genau durchzuführen zu können werden die Positionen der Gelenke und nicht die der RBM verwendet. Insgesamt werden zehn RBM verteilt auf die Hände, Ellenbogen, Knie, die Füße, den Kopf und den Rücken verwendet. Die Positionen der RBM am Körper im Bezug zu den Gelenken sind in Abbildung 3 von der Hinter- und der Vorderansicht abgebildet. Für das Akteur-Skelettmodel werden die Distanzen zwischen Mittelpunkt der Schulterblätter (gegeben durch einen RBM) und den Schultern sowie der Hüfte gemessen. Ein entsprechender Offset wird zwischen RBM-Positionen und den Hand-, Ellenbogen-, Fuß- und Kniegelenken manuell ermittelt. Die RBM für die Handgelenke werden an den Handrücken befestigt, so dass die

Ausrichtung der Handgelenke mit aufgezeichnet werden können, und durch die manuellen Abmessungen die Position der Gelenkes im Skelett bekannt sind. Ein Ziel war es unter anderem, mit einer minimalen Anzahl von Markern auszukommen. Zum einen, da eine Erhöhung der Markeranzahl die Samplingrate des Trackingsystems belastet, zum anderen, da das Anbringen und Ausrichten der Marker ein aufwändiger und langwieriger Prozess ist.

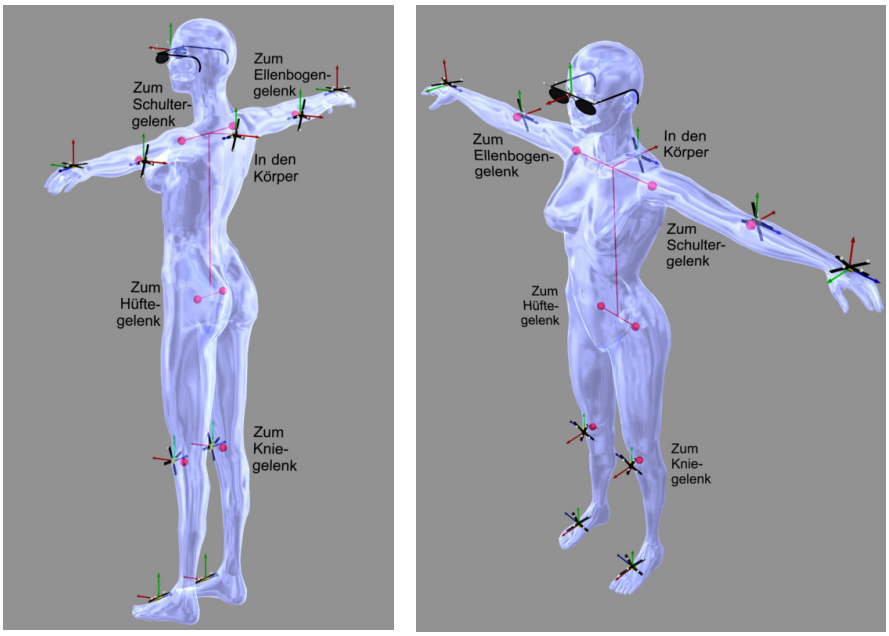

Abbildung 3: Positionen der RBM am Körper im Bezug zu den Gelenken.

5.2 Skelettrepräsentation

Um die benötigten Winkel zu berechnen und anzuwenden, wird zuerst ein Skelett in der *Denavit-Hartenberg*-Konvention – DH-Konvention aufgebaut. Die Gelenke des DH-Skeletts werden iterativ angepasst. Dieses startet bei dem Wurzelgelenk (erstes Gelenk, das nicht von anderen Gelenken beeinflusst wird) und geht nach und nach immer weiter bis zu den Endeffektoren (den Händen, Füßen und dem Kopf). In dieser Konvention wird ein Joint (Verbindung) zwischen zwei Gelenken durch vier einzelne homogene Transformationen (Rotations- und Translationsmatrizen) beschrieben, die miteinander multipliziert die Transformation A_i des Gelenkes ergeben:

$$A_i = R_{z_{i-1},\theta} * T_{z_{i-1},\mathrm{di}} * T_{x_i,\mathrm{ai}} * R_{x_i,\alpha}$$

In Abbildung 4 links wird ein Gelenk als Zylinder mit einer Rotationsachse parallel zur Höhe dargestellt, ein Gelenk mit mehreren DOFs wird aus einzelnen Gelenken zusammengebaut. Daher wird z.B. ein Schultergelenk mit 3 DOF mittels dreier einzelner Gelenke, wie in Abbildung 4 als Teil eines gesamten Arms gezeigt, repräsentiert. Auf der rechten Seite der Abbildung 4 ist das gesamte Skelett in der DH-Konvention dargestellt.

5.3 Gelenkwinkelberechnung

Die Winkelberechnung erfolgt auf Basis der RBM-Positionen und Ausrichtungsangaben in

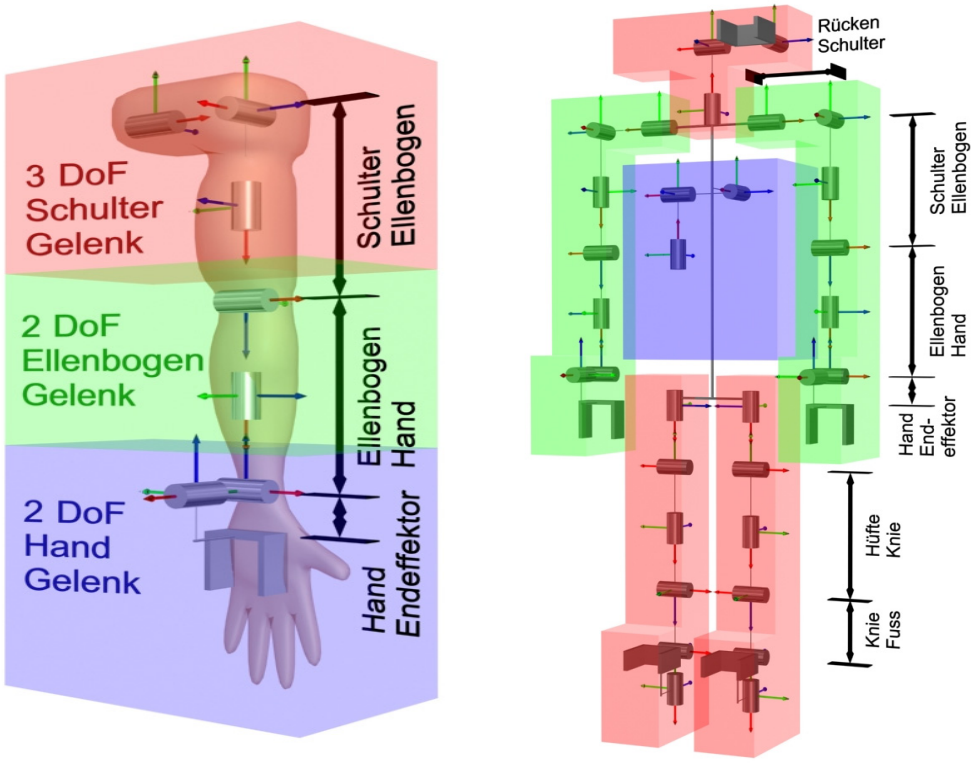

Abbildung 4: Arm und Skelett in der DH-Konvention.

Weltkoordinaten. Zur Berechnung des ersten Winkels (Schultergelenk) sind in der Abbildung 5 die Messdaten, die die Gelenke darstellen, durch rote Kugeln visualisiert. Für die Berechnung des ersten Winkels ist die Position des Schultergelenks und die Position des Ellenbogens wichtig. Als erstes muss die Position des Ellenbogens in Schultergelenkkoordinaten des DH-Skeletts transformiert werden. Gemäß der vereinfachten Darstellung in Abbildung 5 folgt:

$$\begin{pmatrix} 0 & 0 & 1 & x_0 \\ -1 & 0 & 0 & y_0 \\ 0 & -1 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} * \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{L0} \\ y_{L0} \\ z_{L0} \\ 1 \end{pmatrix}$$

Die erste Matrix beschreibt den Zusammenhang zwischen Weltkoordinaten und Gelenkkoordinaten. Das Gelenkkoordinatensystem ist mit dem Weltkoordinatensystem in einem rechtwinkligen Zusammenhang, so dass die rot gefärbte x-Achse des Gelenks in Weltkoordinaten der negativen y-Achse entspricht, die y-Achse des Gelenks in Weltkoordinaten der negativen z-Achse entspricht, und die z-Achse in Weltkoordinaten der x-Achse entspricht. Dieses trifft nur im vereinfachten speziellen Fall zu, bei dem die Koordinatensysteme wie in Abbildung 5 gezeigt ausgerichtet sind. Der Verschiebungsanteil der Matrix entspricht der Position des Schultergelenks. Um die Messdaten des Ellenbogengelenks in die lokalen Koordinaten des Gelenks zu transformieren, muss diese Matrix invertiert werden und mit den Koordinaten des Ellenbogengelenks multipliziert werden. Aus diesen lokalen Koordinaten kann durch die Verwendung der atan2-Funktion der Gelenkwinkel ermittelt werde.

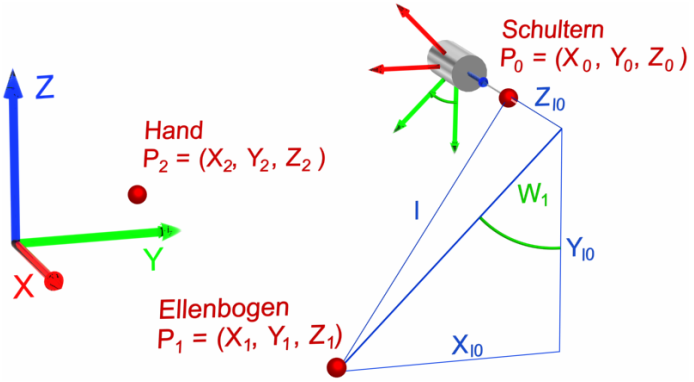

Abbildung 5: Berechnung des ersten Winkels des Armes entsprechend der Messdaten.

Daraus folgt der gesuchte Winkel als:

$$W_1 = \text{atan2}(y_{L0},\ x_{L0})$$

Die Länge l kann errechnet werden durch

$$l = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

sie besteht aus dem Abstand zwischen dem Schultergelenk und dem Ellenbogengelenk. Dieser Abstand wird zu Beginn der Aufnahme, wenn alle benötigten Gelenkpositionen durch die RBM-Daten verfügbar sind, einmalig auf das Skelett übertragen. Diese Anpassung des virtuellen DH-Skeletts an das reale ist notwendig, damit die Gelenkpositionen der beiden Skelette übereinstimmen. Die Berechnung des zweiten Winkels des Schultergelenks und die restlichen (Winkel des Armes) erfolgen nach dem folgenden Prinzip:

```
Function: berechneWinkelDerNachfolgendenGelenke (DH-Skelett.vomTyp(Gelenk))
    foreach (kinderGelenk of  DH-Skelett) do
        if  (kinderGelenk=Null) do break;
        kinderGelenk.berechneTransformationZurWelt();
        kinderGelenk.transformiereSensorDatenInGelenkKoordinaten();
        float winkel = kinderGelenk.atan2(Ankathete, Gegenkathete);
        kinderGelenk.setWinkel(winkel);
        berechneWinkelDerNachfolgendenGelenke (kinderGelenk);
    end
end
```

In Abbildung 6 wird dieser Zusammenhang verdeutlicht. Da das erste Schultergelenk das zweite beeinflusst, ist jetzt das Koordinatensystem nicht mehr in einer rechtwinkligen Beziehung zum Weltkoordinatensystem. Allerdings besteht ein rechtwinkliger Zusammenhang zwischen dem eingestellten Gelenk 1 und dem Gelenk 2 in Ausgangsstellung. Die Messdaten, die verwendet werden, sind in diesem Fall die gleichen wie im Fall des ersten Winkels, lediglich die Koordinatensysteme sind andere. Da das Skelett eine beliebige Ausrichtung im 3D-Raum einnehmen kann, kommen vor der Transformation des Armes noch weitere Transformationen. Die ersten drei Gelenke, die das gesamte nachfolgende Skelett beeinflussen, entsprechen den

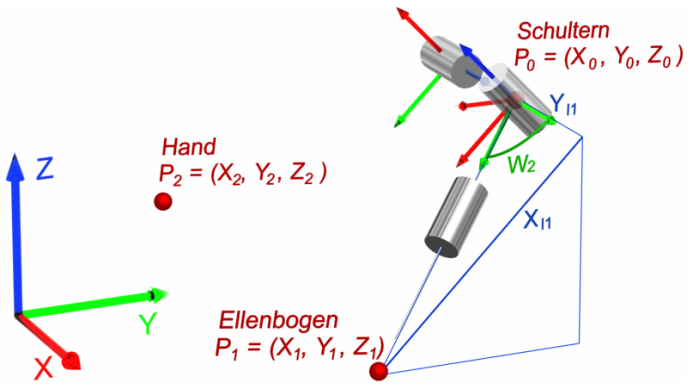

Abbildung 6: Berechnung des zweiten Winkels des Armes entsprechend der Messdaten.

Euler-Winkeln, die aus einer Matrix des RBM, welcher am Rücken befestigt wird, extrahiert werden können. Als nächstes kommen die Translationen in die Schulter- bzw. Hüftgelenke. Nun kann die Bestimmung der Winkel durchgeführt werden, da die nachfolgenden Gelenke entsprechend des menschlichen Skeletts liegen. Zum Schluss der jeweiligen kinematischen Kette (Folge von sich beeinflussenden Gelenken) muss die jeweilige Ausrichtung der Hand-RBM (bzw. auch die der Füße) auf das Skelett übertragen werden. Dazu müssen die Roll-Pitch-Yaw-Winkel aus den Weltkoordinaten in Euler-Winkel bezüglich der aktuellen Gelenkstellungen des Skelettes übertragen werden.

$$ErgebnisMatrix = (SkelettHandGelenkMatrix)^{(-1)} * HandRigid\ BodyMatrix$$

In der *ErgebnisMatrix* sind die einzustellenden Euler-Winkel enthalten, die zuvor erst aus der Matrix extrahiert werden. Bei den Füßen ist dieses im Grunde das gleiche Vorgehen wie bei der Hand, aber auch wie bei der Berechnung der Orientierung des Kopfes bezüglich der aktuellen Lage des Skelettes. Diese Berechnungen werden momentan noch auf der CPU durchgeführt.

6. Charakteranimation durch Vorwärtskinematik auf der GPU

6.1 GPU-basierte Segmentierung der Körperregionen

Die Zuordnung der einzelnen Meshpunkte zu den durch Gelenke definierten Körperregionen für Schultern, Ellenbogen, Hände, Hüften, Knie und Füße wird initial manuell einmal vorbestimmt. Dieses erfolgt entweder mittels eines speziell hierfür entwickelten Werkzeugs oder alternativ in Modellierungssystemen (Maya, 3DStudioMax etc.), welche Exporter für einer der spezielle Dateiformate bereitstellen (wrl, 3ds, obj, dxf oder ply). In diesen werden die Gelenkpunkte durch entsprechend bezeichnete Objekte repräsentiert, welche daraufhin in Beziehung zu bestimmten Meshregionen gesetzt werden. Durch diese Punkte ist eine Einteilung (Segmentierung) des Körpers in entsprechend zu animierende Körperteile möglich. Dazu wird die Lage der einzelnen Vertices in Bezug zu den Gelenken verwendet. Das Prinzip der daraufhin erfolgenden dynamischen Segmentierung und Zuordnung im Vertex-Shader ist in Abbildung 7 dargestellt. Der Vergleich einer einzelnen X- und/oder Y- Koordinate im Bezug zu einem Gelenk ist

Abbildung 7: Segmentiertes Mesh im Wireframe-Shading-Modus mit Gelenken und das resultierende animierte Mesh [Reo05].

schneller als der entsprechende Vergleich von IDs (einmalig auf entsprechende Weise von der CPU ermittelt). Bei Ermittelung der Zugehörigkeit eines Vertex zu einem Segment über IDs müssen alle Segmente durchgegangen werden. Dagegen steht die Ermittelung über den Bezug zum Gelenk, bei dem z.B. aus der Position des Vertex auf der x-Achse auf die linke oder rechte Seite geschlossen werden kann.

```
1    if ((Vertex.Position[x]>0) do
2        if (Vertex.Position[x]>rGelenk.Position[x]) do
3            Vertex = GelenkRotation(VertexPosition, rGelenk.Position, GelenkType, Winkel)
4            Normale = GelenkRotation(Normale, GelenkType, Winkel)
5        end
6    else if (Vertex.Position[x]<0) do
7        if (Vertex.Position[x]<lGelenk.Position[x]) do
8            Vertex = GelenkRotation(VertexPosition, lGelenk.Position, GelenkType, Winkel)
9            Normale = GelenkRotation(Normale, GelenkType, Winkel)
10       end
11   end
```

Der eingesetzte Vertex-Shader zur Segmentierung ist weitgehend unabhängig vom zugrundeliegenden Mesh, unter Berücksichtigung der initialen Körperregionzuordnung über Gelenkpositionen. Die eigentliche Deformation des Meshes in die entsprechende Pose wird durch einzelne Rotationen von Vertices um die entsprechenden Gelenke durchgeführt.

6.2 Durchführungsreihenfolge der Transformationen

Erst durch die Unterstützung des SM4 sind GPUs durch die Anzahl der Instruktionen und der verfügbaren Register in der Lage, komplexe Berechnungen wie die der Vorwärtskinematik bis in die Fingerspitzen durchzuführen (siehe Tabelle 1), so dass die Vorwärtskinematik in einem Vertex-Shader-Programm für die GPU geschrieben werden kann.

Shader Model	Erscheinungsjahr	GPU Serien Name	Instruktions Limit	Temp. Register Vertex-Shader
1	10/2002	Geforce 4	128	Nicht bekannt
2(a)	2003	Geforce 5	256	12(13)
3	2004	Geforce 6/ Radon 9600	512	32
4	2007	Geforce 8/ Radon HD 3000	4096	4096 (32*)

Tabelle 1: Limits der Shader Modelle (*in Verbindung mit CG 2 vom Januar 2008).

Um mit der verfügbaren Anzahl an temporären Registern die kinematischen Berechnungen bis zu den Fingerspitzen ausrechnen zu können, werden nicht die einzelnen Matrizen einzelner Gelenke bis zur Spitze multipliziert, sondern es wird zuerst die Transformation von der Spitze an mehrfach auf die Vertex-Position angewendet. Dieses spart eine explizite Speicherung der einzelnen Matrizen der kinematischen Kette für die Zwischengelenke. Auf diese Weise kommt der Shader mit den verfügbaren 32 Registern des SM4 aus, wenn die Rotationen der Gelenke die Oberflächennormalen mit beeinflussen. Eigentlich besitzt eine GPU, welche das SM4 unterstützt, 4096 Register, mit der verwendeten Shader-Sprache CG2 vom Januar 2008 von Nvidia sind nur 32 beim Kompilieren der Shader verfügbar. Abbildung 8 illustriert, wie die Position eines Vertex mehrfach iterativ, angefangen bei der Spitze, verändert wird. Ein analoges Vorgehen zur Hand- und Fingeranimation wird auch für die Arm- und Beinanimation verwendet.

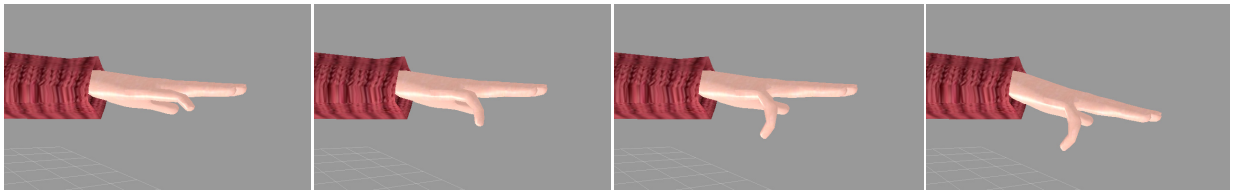

Abbildung 8: Reihenfolge der Transformation bei der Einstellung einer Fingerstellung.

Eine weitere Maßnahme, um mit den verfügbaren Registern auszukommen, ist die separate Speicherung der einzelnen Finger aus der Handgeometrie und das Zuweisen eines Shaders. Neben der Registerersparnis gestalten sich die einzelnen Shader-Programme dadurch auch weniger komplex.

6.3 Abrundung der Segmentübergänge

Durch die Segmentierung und anschließende Angabe der Gelenkwinkel für das Mesh können an den Stellen zwischen zwei verschieden zu animierenden Körperteilen harte Kanten oder auch Artefakte entstehen. Dieses wird in Abbildung 9 auf der linken Seite dargestellt. Um keine harten Ecken bei den Transformationen der Übergange der verschiedenen segmentierten Bereiche des Körpermeshes zu erhalten, wird über die Entfernung der Vertices zum Gelenk zwischen der untransformierten Position und der transformierten Position linear interpoliert.

$$Position = interpolation(OrginalPosition, TransformiertePosition, Entfernung)$$

Je weiter ein Vertex vom Gelenk entfernt ist, desto stärker wirkt sich die Rotationstransformation aus, dargestellt auf der rechten Seite in Abbildung 9. Diese Entfernung wird nur über die Lage

Abbildung 9: Harter und weicher Segmentübergang [Reo05].

des Vertex und des Gelenks entlang einer Achse ermittelt. Beim Kopf und bei den Füßen ist es die y-Achse (vertikal), im Gegensatz zu den Armen, bei denen die x-Achse verwendet wird (horizontal). Dabei stellt die im ersten Schritt gewählte lineare Interpolation einen guten Kompromiss zwischen Aufwand und Resultat in Bezug auf die visuelle Güte dar.

6.3 Instruktion- und Registeranzahl der einzelnen Shader

Das Modell der zu animierenden Figur ist aus mehreren geometrischen Unterteilen aufgebaut, die jeweils verschiedene Shader zur Darstellung der Gelenkwinkel oder Verschiebungswerte erhalten. Diese Werte werden dem Vertex-Shader als uniforme Parameter übergeben. In der Tabelle 2 sind verschiedenen Shader-Programme mit ihren Eigenschaften aufgeführt.

Name	Instruktionen	Register	Uniforme Parameter
Gesicht	**556**	14	81
Pullover	250	13	14
Finger	473	16	17

Tabelle 2: Eigenschaften der verschiedenen Shader.

Die in dieser Tabelle aufgeführten Instruktionen und Register repräsentieren die Anforderungen, die die kompilierten Programme an die GPU setzen. Die Limits, die zum Kompilieren der einzelnen Vertex-Shader-Programme benötig werden, liegen tatsächlich viel höher. Da der Vertex-Shader auf der GPU kompiliert wird, benötigen die Shader für die *Hand*, den *Finger*, das *Gesicht* und die *Hose* eine GPU mit mindestens Shader Model 4, um dargestellt zu werden.

7. Fazit

Das vorgestellte System erlaubt interaktives Echtzeit MoCap für die Animation virtueller

Agenten. Der Ansatz der Bewegungszerlegung und anschließenden kombinatorischen Synthese erlaubt die Verwendung eines optischen Trackingsystems innerhalb VR-typischer räumlich begrenzten Tracking- und Displayumgebungen.
Um die Bewegungen des Akteurs auch in Echtzeit zu visualisieren, wurden mehrere GPU-basierte Kinematik- und Meshdeformationsverfahren auf Basis der aktuellen GPU-Generation mit SM4 umgesetzt. Damit ist eine detaillierte, realistische und interaktive Bewegungssysnthese möglich. Der Ansatz der Übertragung von Teilen der benötigten Kinematik- und Animationsfunktionalität auf Shaderprogramme mit einem Satz weniger veränderlicher Parameter berücksichtigt darüber hinaus die Visualisierung ohne Performanzeinbrüche sowohl auf Einzelserverlösungen als auch auf Renderclustern.

8. Literatur

[CruzSan92] Cruz-Neira, C; Sandin, D.J; DeFaniti, T.A.; Kenyon, R.V.; Hart, J. C.: *The cave: audio visual experience automatic virtual environment*, Commun. ACM, 1992

[Ma06] Masaki, Oshita: *Motion-Capture-Based Avatar Control Person View Virtual Environments*. ACM Paper, Kyushu Technology 680-4 Kawazu, lizuka, Fukuoka Japan, 2006, June.

[Möl05] Möller, Ralf: *Vorlesung Robotik 1: Serielle Manipulatoren*. Skript, Universität Bielefeld Technische Fakultät AG Technische Informatik, 2005, Oktober.

[SponSeth06] Spong, M. W.; Hutchinson, S., Vidyasagar, M.: *Robotik Modeling And Control*. Wiley, 2006.

[RaFröLa07] Rabe, F.; Fröhlich, C.; Latoschik, M. E.: *Low-cost image generation for Immersive Multi-Screen Environmensts*, In Virtuelle und Erweiterte Realität, 4. Workshop of the GI special interest group VR/AR, pages 65-76

[RaKi03] Randima, F.; Kilgard, M. J.: *The CG Tutorial*. Addison Wesley, Amsterdam

[Re04] Rege, Ashu: *Shader Model 3.0*, NVIDIA Developer Technology Group, 2004

[Wu06] Wu, Owen: *Animation with R2VB*, ATI Technologies, Inc., 2006

[Reo05] Reorom (Member bei TurboSquid): *lady with gun*, Mesh, TurboSquid Inc, New Orleans, LA 70130, 2005

9. Vermerke

CAVE™ ist ein eingetragenes Markenzeichen von der *University of Illinois*