Entwicklung virtueller Kreaturen in 3D- und AR-Umgebungen

Christian Geiger[1], Tim Schmidt[2], Jörg Stöcklein[2]
cgeiger@hs-harz.de, tisch@upb.de, ozone@upb.de
[1]Hochschule Harz, Medieninformatik, www.medieninformatik.de
[2]Universität Paderborn, www.upb.de

Abstract: Wir betrachten die Entwicklung und Visualisierung virtueller Kreaturen nach Sims [Sims94] und stellen eine Softwarebibliothek vor, mit der die Evolution virtueller Kreaturen sowie deren Visualisierung in 3D-Welten und „Augmented-Reality"-Umgebungen in Echtzeit möglich ist. In diesem System werden verschiedene Software-Komponenten für Soft-Computing (GA, NN), physikalische Animation und „Augmented Reality" mit einer High-Level-3D-Bibliothek (Java3D) kombiniert. Um eine möglichst wirksame Darstellung der virtuellen Kreaturen zu ermöglichen, erweitern wir Java3D um einen eigenen Renderer, der u.a. Schatten und Verdeckung realer Objekte durch virtuelle ermöglicht.

Stichworte: Virtual Life, Augmented Reality, fortgeschrittene Rendertechniken

1 Einleitung

Diese Arbeit wurde durch Karl Sims SIGGRAPH Paper „Evolving Virtual Creatures" [Sims94] motiviert. Als Aufgabenstellung der Diplomarbeit eines der Autoren (Tim Schmidt) soll eine Software-Umgebung realisiert werden, in der die Entwicklung virtueller Kreaturen in Anlehnung an Sims modelliert wird. Ziel dabei ist neben einer möglichst einfachen Art der Beschreibung und einer effizienten Ausführung der Simulation auch eine ansprechende Visualisierung in Echtzeit. Weiterführendes Ziel ist es, über eine intuitive Art der Interaktion den Entwicklungsstand von Kreaturen testen zu können, die auf Umgebungseinflüsse sowohl virtueller Welten (z.B. durch andere Kreaturen) als auch realer Welten (z.B. durch den Benutzer oder Licht) reagieren. Das System soll am Ende einfach erweiterbar sein und unterschiedliche Experimente in diesem Kontext ermöglichen.

Als Software-Basis haben wir uns für Java und Java3D entschieden. Dies ermöglicht prinzipiell eine flexible und effiziente Simulation und bietet darüber hinaus eine angemessene, interaktive 3D-Visualisierung. Das geplante Vorhaben schließt jedoch eine Reihe zusätzlicher Module mit ein, z.B. für physikalische Simulation, „Augmented Reality" und die parallele Ausführung des evolutionären Algorithmus. Durch das Java Native Interface (JNI) kann elegant eine Kapselung nativer Bibliotheken erreicht werden. Somit erreichen wir z.B. im Falle der 3D-Visualisierung zwar den hohen Abstraktionsgrad einer Szenegraphbibliothek wie Java3D, können jedoch gleichzeitig auf die Funktionalität nativer AR-Bibliotheken (ARToolKit) zugreifen oder einen eigenen OpenGL-Renderer nutzen, der fortgeschrittene Darstellungstechniken wie Schatten und Verdeckungen zwischen virtuellen und realen Objekten ermöglicht.

Der Beitrag skizziert zunächst einige Grundbegriffe und referenziert kurz vergleichbare Arbeiten. Auf Basis der Anforderungen an diese Arbeit wird das Konzept vorgestellt, mit dem virtuelle Kreaturen für 3D und VR-Umgebungen evolutionär entwickelt werden können. Ausgewählte Implementierungsaspekte beleuchten die Integration der verschiedenen Bibliotheken und fokussieren auf den eingesetzten Renderer. Schließlich wird in einem kurzen Beispiel der aktuelle Stand der Arbeit illustriert und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Grundbegriffe

Starrkörpersimulation: In diesem Modell wird ein beliebiges 3-dimensionales Objekt auf seinen Schwerpunkt reduziert. Der Zustand eines solchen Objektes wird durch Position, Orientierung, Masse, Trägheitsmoment, Vorwärts- und Rotationsgeschwindigkeit beschrieben. Eine Zustandsänderung über die Zeit ergibt sich aus der Beeinflussung durch Kräfte und Drehmomente. Eine Kollision von Starrkörpern wird durch Generierung von entsprechenden Rückstoßkräften an den Überschneidungspunkten der zugehörigen Geometrien simuliert.

Virtuelle Kreaturen: Eine Kreatur besteht aus verschieden dimensionierten Quadern, welche über virtuelle Gelenke miteinander verbunden sind (Abbildung 1). Sensoren liefern Daten aus der simulierten physikalischen Umgebung, z.B. Kontaktinformationen, Beugungswinkel der Gelenke, visuelle Reize, etc. Ein neuronales Netz verarbeitet diese Daten und leitet sie weiter an Effektoren, die wiederum direkten Einfluss auf das Verhalten der Kreatur selbst haben, beispielsweise indem Beugungsmotoren in den Gelenken aktiviert werden. Eine Kreatur ist somit prinzipiell in der Lage, seine Umwelt wahrzunehmen und über autonomes, nicht zielgerichtetes Verhalten auf sie zu reagieren.

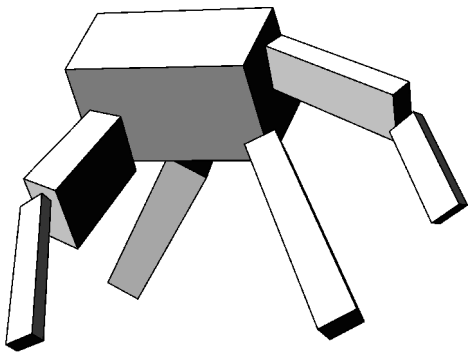


Abbildung 1: Beispiel einer virtuellen Kreatur

Entwicklungsmodell genetischer Algorithmus: Der genetische Algorithmus [Hol75] ist ein heuristisches Optimierungsverfahren, das die Prinzipien der Evolution anwendet, um Lösungen für sehr komplexe Probleme zu finden. Er behandelt in jedem Durchlauf (Generation) eine Anzahl von Lösungskandidaten (Population) und weist jedem einzelnen anhand einer vorgegebenen, so genannten Fitnessfunktion eine bestimmte Güte zu. Dieser Wert ist in der anschließenden Selektionsphase der bestimmende Faktor dafür, in welchem Umfang eine bestehende Lösung ihre Eigenschaften in neu zu generierende Lösungen einbringt. In diesem Prozess der Neugenerierung werden Operatoren benutzt, die man auch aus dem Bereich der Genetik kennt. Zum einen ist die Rekombination zu nennen, bei der Merkmale aus zwei Lösungen zu einer neuen Lösung vereinigt werden, zum anderen die Mutation, bei der einzelne Merkmale einer Lösung zufällig verändert werden. Aus der Tatsache, dass auf diese Weise vermehrt Eigenschaften weitervererbt werden, die im Sinne der Fitnessfunktion als positiv anzusehen sind, während negative eher aussterben, ergibt sich eine stetige Verbesserung der in jedem Durchlauf gefundenen besten Lösung.

3 Vergleichbare Arbeiten

Karl Sims: In [Sims94] wurde gezeigt, wie sich durch Simulation evolutionärer Prozesse 3D-Charaktere entwickeln können, deren Körperbau und Verhalten am Ende ein bestechendes Maß an Zweckdienlichkeit bzw. Zielgerichtetheit aufweist. Trotz einfacher Strukturen und ohne vom Menschen einprogrammierte Logik waren sie hervorragend an ihre jeweilige Umgebung angepasst. Die Berechnung erfolgte damals auf einem Parallelrechner (CM-5) und das Rendern als Film, der anschließend betrachtet werden konnte.

Framsticks: Framsticks [KoUl99] ist ein fortgeschrittenes Projekt zur Simulation 3-dimensionaler Lebensformen, das sowohl den mechanischen Aufbau als auch die Kontrollsysteme der Wesen in einen Evolutionsprozess einbezieht. Experimente können sehr variabel definiert werden. Die Anwendung verschiedener Arten der Genkodierung ist möglich.

Golem: Das Project Golem (Genetically Organized Lifelike Electro Mechanics) [LiPo00] zielt darauf ab, Möglichkeiten aufzuzeigen, wie Roboter durch einen automatischen Designprozess und anschließende, automatische Herstellung produziert werden können. Es wurden Experimente durchgeführt, in denen einfache elektro-mechanische Einheiten aus Plastik erzeugt wurden, die sich mithilfe von Elektromotoren und künstlichen Neuronen fortbewegen können. Die Entwicklung dieser Einheiten vollzog sich auf der Basis eines genetischen Algorithmus in Verbindung mit einer physikalischen Simulation. Die Baupläne der besten Kreaturen wurden am Ende einem 3D-Drucker übergeben, der die jeweiligen Bauteile ohne menschliches Dazutun Schicht für Schicht herstellte. Sensoren wurden nicht modelliert.

4 Anforderungen

Auf Grundlage der Ergebnisse der bisherigen Arbeiten auf diesem Gebiet stellten wir uns die Frage, wie man eine direktere, möglichst intuitive Art der Interaktion zwischen dem Menschen und den Kreaturen erreichen kann. Eine wichtige Voraussetzung dafür ist einerseits, dass die Kreaturen in Echtzeit auf ihre Umgebung eingehen können, und andererseits, dass der Mensch in der Lage ist, Änderungen in der von den Kreaturen wahrgenommenen Welt vorzunehmen. In Anlehnung an das Golem-Projekt könnte man die Kreaturen ebenfalls als Hardware bauen und sie damit aus der virtuellen in die reale Welt holen. Zusätzlich müsste man sie allerdings mit geeigneten Sensoren ausstatten. Der technische und zeitliche Aufwand dafür ist jedoch sehr hoch. Der andere Weg ist, die Kreaturen in ihrer virtuellen Welt zu belassen und stattdessen dem Benutzer einen intuitiven Zugang zu dieser zu verschaffen. Hier bietet es sich an, die virtuellen Kreaturen in eine markerbasierte AR-Umgebung einzubetten, um so die Benutzung von tangible Userinterfaces zu ermöglichen. Für eine bessere Tiefenwahrnehmung seitens des Anwenders ohne den Einsatz eines stereoskopischen Systems ist es dabei notwendig, verschiedene „Visual Cues“ wie die korrekte Verdeckung realer und virtueller Objekte und Schattenwurf bereitzustellen. Zusammengefasst ergeben sich also folgende Anforderungen:

- *Direkte Interaktion über tangible Userinterfaces*
    - Registrierung realer Objekte im virtuellen Raum

- *Realtimefähigkeit*
    - Echtzeit-3D-Grafik
    - performante Physik-Simulationsbibliothek
- *Verbesserte Tiefenwahrnehmung durch fortgeschrittene Rendertechniken*
    - Occlusion-Culling zwischen registrierten realen und virtuellen Objekten
    - Schattenwurf registrierter realer und virtueller Objekte

5 Konzepte

Im Folgenden soll zunächst ein Einblick in die Funktionsweise und das evolutionäre Entwicklungsmodell der virtuellen Kreaturen gegeben werden. Des Weiteren wird verdeutlicht, was eine VR-Visualisierung in diesem Themenkomplex leistet und welche Umstände letztendlich trotz der thematischen Entfernung zum Einsatz von AR im Bereich „Evolutionary Computing“ geführt haben.

5.1 Physikalische Simulation virtueller Kreaturen

Zum Erreichen natürlicher Bewegungsabläufe werden die Körper der virtuellen Kreaturen als Elemente einer Starrkörpersimulation („Rigid Body Dynamics“) modelliert. Ihr dynamisches Verhalten unterliegt somit künstlicher Schwerkraft und Reibung und den eigenen Massenträgheitsmomenten. Gelenke zwischen den Starrkörpern und ihre Beugung durch Anwendung entsprechender Kräfte sind die Grundlage für die eigenständige Bewegung der Kreaturen.

Die simulierte physikalische Umgebung und damit der „Lebensraum“ einer Kreatur besteht mindestens aus einer ebenen Bodenplatte, auf der eine Fortbewegung möglich ist, kann aber zusätzlich weitere Objekte wie z.B. Begrenzungen, einzelne Hindernisse oder auch weitere Kreaturen enthalten. Eine Interaktion der Kreaturen mit ihrer Umwelt wird über künstliche Sinnesorgane erreicht. Sowohl Umweltinformationen als auch Informationen über den eigenen Status werden hierzu parametrisiert und in das neuronale System einer Kreatur eingegeben.

Eine Interaktion zwischen Mensch und Kreatur vollzieht sich auf zweierlei Weise. Zum einen können Körperteile fixiert und bewegt werden, zum anderen kann die Umwelt, etwa durch Umpositionierung von Gegenständen, gezielt verändert werden, so dass die Interaktion indirekt stattfindet.

5.2 Evolutionärer Entwicklungsprozess

Aufgrund der Komplexität des Zusammenspiels der einzelnen Komponenten sind sowohl der Körperbau als auch die Topologie der neuronalen Verschaltung und damit das resultierende Verhalten der Charaktere nicht das Ergebnis eines manuellen Designvorgangs, sondern sie entwickeln sich im Verlaufe eines evolutionären Berechnungsprozesses, dem das Prinzip des genetischen Algorithmus zugrunde liegt.

Der Genotyp der virtuellen Kreaturen, d.h. die Codierung ihrer Eigenschaften, orientiert sich an dem beim genetischen Programmieren verwendeten Modell [Koza94]. Es werden demnach

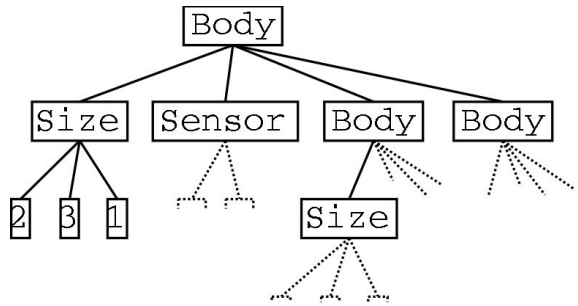


Abbildung 2: Der Genotyp als Baumstruktur

Baumstrukturen verwendet, deren Knoten und Blätter Funktionen und Konstanten repräsentieren. Die Kindelemente der einzelnen Knoten entsprechen den Parametern der jeweils zugehörigen Funktion (Abbildung 2). Geeignete genetische Operatoren sorgen für Mutation und Rekombination des so aufgebauten Gencodes.

In der Evaluationsphase des genetischen Algorithmus (Abbildung 3) wird zunächst der Genotyp in den Phänotyp, also das äußere Erscheinungsbild, übersetzt. Bei den virtuellen Kreaturen bedeutet dies, dass anhand der Gene geeignete Starrkörper mit den entsprechenden Gelenkverbindungen erzeugt werden. Hinzu kommt die Generierung des neuronalen Netzes für die Steuerung und dessen Ankopplung an die Sensorik und Aktorik einer Kreatur. In einem weiteren Schritt wird die Kreatur in eine problemspezifische Testumgebung gesetzt und die Simulation wird gestartet. Nach Ablauf einer vorher festgelegten Zeit wird schließlich dem zugehörigen Genom über eine ebenfalls problemspezifische Fitnessfunktion ein Gütewert zugewiesen.

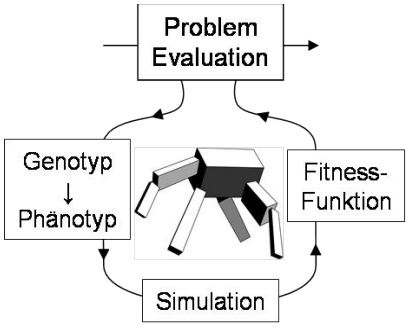


Abbildung 3: Stationen der Problem Evaluation

Dieser Fitnessfunktion kommt eine besondere Bedeutung zu, da durch sie definiert wird, in welche Richtung sich die Kreaturen entwickeln sollen. Eigenschaften und Verhaltensweisen, die über diese Funktion als gut bewertet werden, entwickeln sich über die Generationen hinweg immer weiter heraus.

5.3 Einbettung der virtuellen Kreaturen in eine VR-Umgebung

Der genetische Algorithmus ist ein Suchalgorithmus, der während seiner Laufzeit fortwährend neue Lösungen generiert und testet. Für die Betrachtung durch den Anwender sind jedoch nicht alle diese Lösungen interessant. Um hier also eine maximale Performance zu erreichen, wird während des normalen Berechnungsprozesses auf eine Visualisierung verzichtet. Für eine finale Präsentation der Ergebnisse eines Experiments, für zwischenzeitliche Beurteilungen des Entwicklungsstands der Kreaturen sowie für eine Kontrolle des korrekten Experimentverlaufs ist eine Visualisierung aber generell erforderlich. Sie erlaubt es dem Anwender, die Simulation an die Framerate gekoppelt, also in Realzeit, mitzuverfolgen. In einer Sekunde wirklicher Zeit vergeht also auch nur eine Sekunde simulierter Zeit.

Über einen Maus-Picking-Mechanismus ist der Anwender zusätzlich in der Lage, mit den virtuellen Kreaturen zu interagieren. Es können sowohl in der Szene vorkommende Gegenstände als auch die Kreaturen selbst mit der Maus anvisiert, hochgehoben, gedreht und positioniert werden. So ist es möglich, sich nicht nur einen äußerlichen Eindruck vom Aufbau der Kreaturen zu verschaffen, sondern durch die intuitive Art der Beschäftigung mit ihnen auch einen Einblick in ihr Verhalten zu erlangen. Ein Wesen beispielsweise, das während seiner phylogenetischen Entwicklung, also von Generation zu Generation, „gelernt“ hat, bestimmte

Gegenstände als Nahrung zu betrachten, kann daraufhin untersucht werden, wie gut es in der Lage ist, seine Nahrung zu finden, zu fixieren und zu verfolgen. Der Anwender kann in diesem Fall die Position eines solchen Nahrungsstückes gezielt und gegebenenfalls abrupt verändern und die Reaktion der Kreatur auf sein eigenes Handeln beobachten. Gegebenenfalls kann durch die so erworbenen Erkenntnisse eine vorteilhafte Anpassung des Experimentaufbaus gefunden und vorgenommen werden.

5.4 Einbettung der virtuellen Kreaturen in eine markerbasierte AR-Umgebung

Für erste, gelegentliche Versuche, in die virtuelle Welt der Kreaturen einzugreifen, und für eine Beurteilung des korrekten Verlaufs von Experimenten ist die Benutzung einer VR-Umgebung zwar durchaus zweckmäßig, es zeigen sich jedoch bei längerem Arbeiten und bei komplexeren virtuellen Versuchsaufbauten die begrenzten Fähigkeiten von 2D-Interfaces wie Maus und Monitor. Die Interaktion mit Kreaturen und die Navigation in der Szene sind gleichzeitig kaum möglich. Die Interaktion allein gestaltet sich besonders dann schwierig, wenn Tiefeninformationen richtig eingeschätzt werden müssen. So kommt es vor, dass Objekte beim Aufbau der virtuellen Elemente eines Versuchs nicht aufeinander, sondern hintereinander gestellt werden.

In einem letzten Schritt wurden daher die reale und die virtuelle Welt über den Einsatz von „Augmented Reality“ enger miteinander verknüpft. Eine mit einem Marker besetzte reale Szene wird dabei mit einer Kamera aufgenommen. Der Marker steht für den Mittelpunkt der simulierten physikalischen Welt. In die Szene werden ebenfalls mit Markern besetzte Gegenstände gestellt, die eine bezüglich der Ausmaße exakte Entsprechung in der Simulation haben. Verändert man die Position dieser Objekte, so verändert sich auch die Position der jeweils zugehörigen virtuellen Zwillinge relativ zum Weltmittelpunkt. Will man beispielsweise, wie oben bereits angesprochen, eine Kreatur mit virtuellem Futter „locken“, kann man den als Nahrung definierten Gegenstand direkt in die Hand nehmen und vor der Kreatur hin- und herbewegen.

Bei diesem Ansatz erfolgt die Darstellung und das Filmen der Szene idealerweise über ein Head-Mounted-Display (HMD) mit integrierter Kamera, ein solches stand uns jedoch nicht zur Verfügung. Wir gingen daher von einer Desktop-PC-Lösung mit einem Monitor und einer Kamera aus. Um hier bei der Betrachtung die Tiefenwahrnehmung zu verbessern, sind zusätzliche „Visual Cues“ von großem Nutzen.

5.5 Visual Cues

Wird ein nichtstereoskopisches System in AR-Anwendungen verwendet, hat der Benutzer bei der Interaktion mit rein virtuellen Objekten über ein tangible Userinterface oft das Problem mangelnder Tiefenwahrnehmung. Es ist ihm nicht möglich zu entscheiden, ob sich ein tangible Userinterface vor oder hinter einem virtuellen Objekt befindet. Auffällig wird dies unter anderem in Experimenten mit virtuellem Futter, in denen es schwierig zu erkennen ist, wie weit man das Lockmittel von einer Kreatur entfernt hält, bzw. ob es vor oder hinter der Kreatur positioniert ist. Die zwei für eine Entscheidungsfindung wichtigen visuellen Hinweise sind die korrekte Verdeckung realer und virtueller Objekte und deren Schattenwurf.

6 Realisierung

In diesem Kapitel soll der technische Aufbau des hier vorgestellten Systems näher beleuchtet werden. Es wird deutlich, dass der Szenegraph der 3D-Bibliothek Java3D das Element ist, das die Funktionalitäten der visuellen Komponenten unseres Systems bündelt und ein Zusammenspiel gewährleistet. Zunächst ist die Physik-Engine ODE (http://www.ode.org/) zu nennen, deren Java- und Java3D-Anbindung ODEforJava/3D (http://odeforjava.sourceforge.net/) für die Starrkörpersimulation benutzt wird. Die Marker-Registrierung ist die Aufgabe des in [GSS04] vorgestellten jARToolKits in der Version 2.0. Die für die „Visual Cues“ notwendigen fortgeschrittenen Rendertechniken werden über einen eigens entwickelten Renderer realisiert. Dieser setzt auf JOGL, einer Java-Anbindung an OpenGL (https://jogl.dev.java.net/), auf.

Die Durchführung des evolutionären Berechnungsprozesses übernimmt die reine Java-Bibliothek „Evolutionary Computation in Java“ (ECJ, http://cs.gmu.edu/~eclab/projects/ecj/).

6.1 Aufbau der virtuellen Kreaturen

Die Körper der virtuellen Kreaturen setzen sich aus Komponenten der Softwarebibliothek ODEforJava3D zusammen. Die Segmente bestehen dabei jeweils aus einer RigidBodyTransformGroup und einer CollidableSolidBox und stellen somit massetragende Würfel unterschiedlicher Größe dar. Über PoweredJointTransformGroups, also motorisierte Kugelgelenke, können Segmente miteinander verbunden werden (Abbildung 4). Diese Gelenke sind die Basis für die Aktorik einer Kreatur. Sie stellen ein Interface zur Verfügung, das Stimuli [-1,1] empfängt und in Beugungsbewegungen umsetzt. Zur Erzeugung von Stimuli können Sensoren definiert werden, die beliebige Größen aus der simulierten Welt auslesen und weiterleiten. Die Verarbeitung der gewonnenen Daten und die Übergabe an die Aktorik erfolgt mithilfe eines selbstentwickelten neuronalen Netzes. Dessen Neuronen entsprechen in ihrer Arbeitsweise dem im Framsticks Projekt [KoUl99] verwendeten Modell. Somit kann die interne Funktion jedes Neurons durch drei Parameter (force, inertia, sigmoid) manipuliert werden.

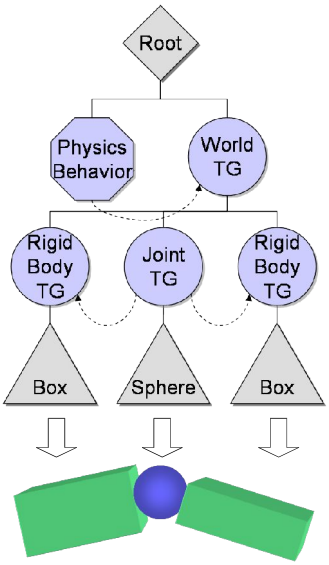


Abbildung 4: Physikalische Objekte im Szenegraphen und ihre Visualisierung

6.2 Der genetische Algorithmus

Für die Durchführung des evolutionären Prozesses wurde die freie Bibliothek ECJ verwendet. Diese bietet ein umfangreiches Framework austauschbarer Komponenten, die für den Ablauf des genetischen Algorithmus nötig sind. Insbesondere werden die wesentlichen Strukturen für das genetische Programmieren zur Verfügung gestellt.

Wie in Abbildung 5 zu sehen ist, übernimmt ECJ die Verwaltung der Erbanlagen einer Population von Kreaturen. Ein Evaluator sorgt dafür, dass jedes einzelne Genom einer Bewertungsinstanz zugeführt wird. ECJ sortiert anschließend die Genome nach ermittelter Güte und führt

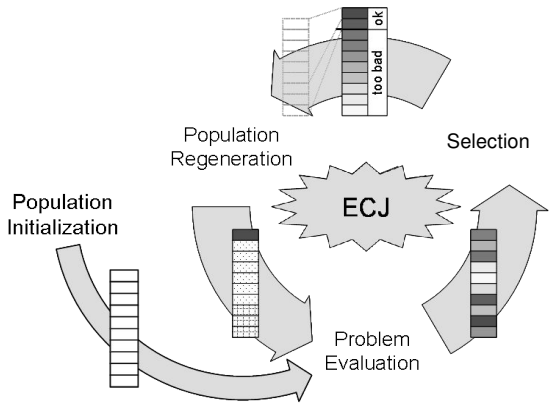


Abbildung 5: Genetischer Algorithmus mit ECJ

die Selektion durch. Die nun in ihrer Größe dezimierte Population wird in der Regenerationsphase wieder aufgefüllt. Wie das geschieht, wird über den Aufbau der so genannten Breedingpipeline spezifiziert. Hier kommen genetische Operatoren wie Rekombination und Mutation zum Einsatz. Während des Ablaufs eines Experiments überwacht ECJ sowohl die Anzahl der Generationen als auch die Güte des in jeder Generation ermittelten hochwertigsten Genoms. Erreicht einer dieser Parameter einen bestimmten Wert, wird das Experiment beendet.

6.3 Augmented Reality

In Abbildung 6 ist der Aufbau sowohl des jARToolKits als auch von ODEforJava dargestellt. Beide Module basieren auf einer C bzw. C++ Bibliothek, deren APIs mit Hilfe des JNI in Java zur Verfügung gestellt werden. Das jARToolKit zum Beispiel basiert auf dem sehr bekannten ARToolKit von Kato [Kato99a]. Darauf baut der Java3D-Abstraktionslayer auf, der die Funktionalität aus den Bibliotheken jeweils in passenden Szenegraphknoten kapselt.

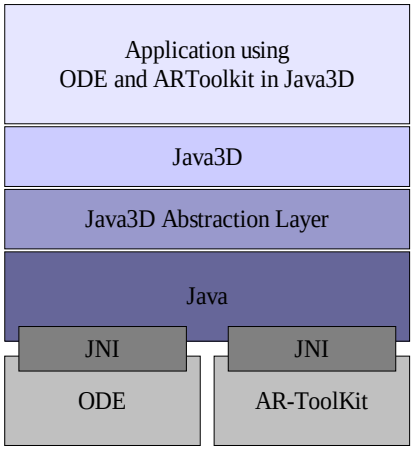


Abbildung 6: Abstraktionsebenen des jARToolKits und ODEforJava

Die Verbindung von „Augmented Reality“ und Physik wird, wie Abbildung 7 zeigt, an zwei Punkten hergestellt. Zum einen wird der Mittelpunkt der simulierten Welt an einen Marker geknüpft, indem eine WorldTransformGroup im Szenegraphen unter eine PatternTransformGroup gehängt wird. Wird ein Marker verdeckt oder nicht erkannt, wird die letzte bekannte Position beibehalten. Die Simulation läuft davon unabhängig weiter. Zur Vermeidung dieser Problematik ist auch die Verwendung von Multi-Markern denkbar.

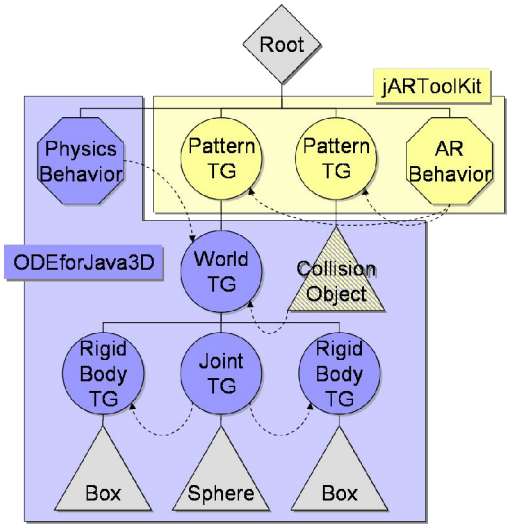


Abbildung 7: Zusammenspiel von jARToolKit und ODEforJava im Szenegraphen

Der zweite Verbindungspunkt ensteht durch den Einsatz externer Kollisionsobjekte. Die absolute Position ihrer visuellen Komponente ist an einen Marker gekoppelt, während die Position ihres physikalisch simulierten Repräsentanten relativ zur WorldTransformGroup berechnet wird. Eine Veränderung der Position des Kollisionsobjekts hat somit direkten Einfluss auf ein Objekt in der Simulation.

6.4 Integration fortgeschrittener Rendertechniken

Die Integration der beiden genannten „Visual Cues“ in die Visualisierung, also der Verdeckung von virtuellen und realen Objekten und der Schattenwurf, ist mit Java3D nicht möglich, da hier der Zugriff auf die Low-Level-Ebene der Grafikhardware nicht zur Verfügung steht, der aber für die Implementation notwendig ist. Um aber den Umgang mit den tangible Userinterfaces in der virtuellen 3D-Umgebung zu verbessern, mussten diese „Visual Cues“ bereitgestellt werden, die dem Benutzer helfen sollen, die Position im virtuellen Raum zu bestimmen. Aus diesem Grunde wurde ein zusätzlicher Renderer in OpenGL unter Benutzung von JOGL realisiert, der es ermöglicht, die Java3D Szene mit den entsprechenden „Visual Cues“ darzustellen. Die Benutzung dieses Renderers soll sich für die Applikation transparent gestalten, so dass möglich ist, sowohl den original Java3D-Renderer als auch den neuen OpenGL-Renderer zu benutzen, ohne dass an der Applikation selbst etwas geändert werden muss.

Damit die Benutzung dieses neuen Renderers für den Anwender und die Anwendung transparent bleibt, arbeitet er direkt auf dem von Java3D generierten Szenegraphen. Weiterhin benutzt er zum Event-Scheduling die Java3D-Infrastruktur, so dass kein eigener Scheduler implementiert werden musste.

Im Allgemeinen funktioniert der OpenGL-Renderer folgendermaßen:

1. Java3D wird aufgefordert alle Behaviors auszuführen.
2. Der OpenGL-Renderer traversiert den Szenegraphen und rendert alle ihm bekannten Szenegraphknoten.

Um einen Szenegraphknoten rendern zu können, muss dieser Knoten dem OpenGL-Renderer bekannt sein. Es wurden die wichtigsten Szenegraphelemente zur Verfügung gestellt. Da normalerweise viele dieser Objekte nicht gelesen werden müssen, sind die Default-Capabilities von Java3D nicht richtig gesetzt, um vom OpenGL-Renderer benutzt zu werden. Vor der Anbindung des Szenegraphen an das Universe muss eine spezielle Funktion aufgerufen werden, die die Capabilities des Szenegraphen für die Verwendung mit dem OpenGL-Renderer passend setzt. Damit ist es dann möglich, den kompletten Szenegraphen unter Verwendung von JOGL in OpenGL zu rendern. In Abbildung 8 ist die Integration schematisch dargestellt.

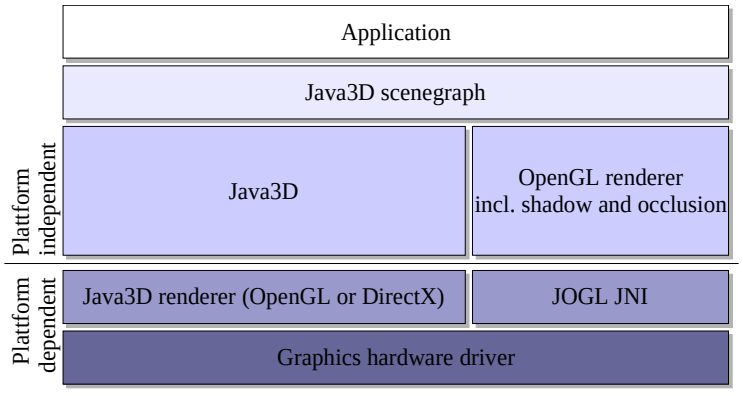


Abbildung 8: Aufbau der Renderpipeline

Zur Realisierung der „Visual Cues“ musste der Renderprozess modifiziert und für die Schattenberechnung in mehrere Durchgänge aufgeteilt werden. Um die Verdeckungsberechnung von realen und virtuellen Objekten zu erreichen, war eine Umstellung der Reihenfolge des Renderprozesses notwendig. Normalerweise wird in einer AR-Anwendung zuerst ein Videobild von

einer Kamera aufgenommen und als Hintergrund der Szene benutzt. Üblicherweise geschieht dies am Anfang eines Renderdurchlaufes, indem das Videobild komplett über den zuletzt berechneten Inhalt des Framebuffers gezeichnet wird. Nachdem das Videobild in den Framebuffer übertragen wurde, werden die virtuellen Objekte in selbigen Framebuffer gezeichnet. Bei dieser Methode ergibt sich damit folgendes Verdeckungsproblem: reale Objekte können niemals virtuelle Objekte überdecken, da

1. die realen Objekte nur als 2D-Videobild ohne Tiefeninformationen existieren, und
2. die virtuellen Objekte immer nach dem 2D-Videobild gezeichnet werden.

Um dieses Problem zu lösen ist es notwendig, sowohl die für die Verdeckung relevanten Objekte im 3D-Raum zu registrieren als auch diesen eine 3D-Repräsentation zu verleihen. Diese Repräsentation wird später nur zur Verdeckungsberechnung benötigt und ist nicht sichtbar. Weiterhin ist es notwendig, den Renderprozess so zu ändern, dass das Videobild erst am Ende eines Renderdurchlaufs in den Framebuffer kopiert wird. Zwar ist es auch möglich, durch eine Sortierung die Verdeckung zu berechnen und den Renderprozess wie gewöhnlich zu belassen, dieses würde aber eine weitere Traversierung des Szenegraphen mit sich führen, da die Reihenfolge der virtuellen und realen Objekte im Szenegraphen beliebig ist. Dies wäre unnötiger Aufwand, der Rechenzeit kosten würde. Unter Benutzung eines speziellen Buffers in OpenGL, des Stencilbuffers, und des dazugehörigen Stenciltests wurde folgendes Verfahren zur Berechnung von Verdeckung von realen und virtuellen Objekten implementiert (siehe Abbildung 9):

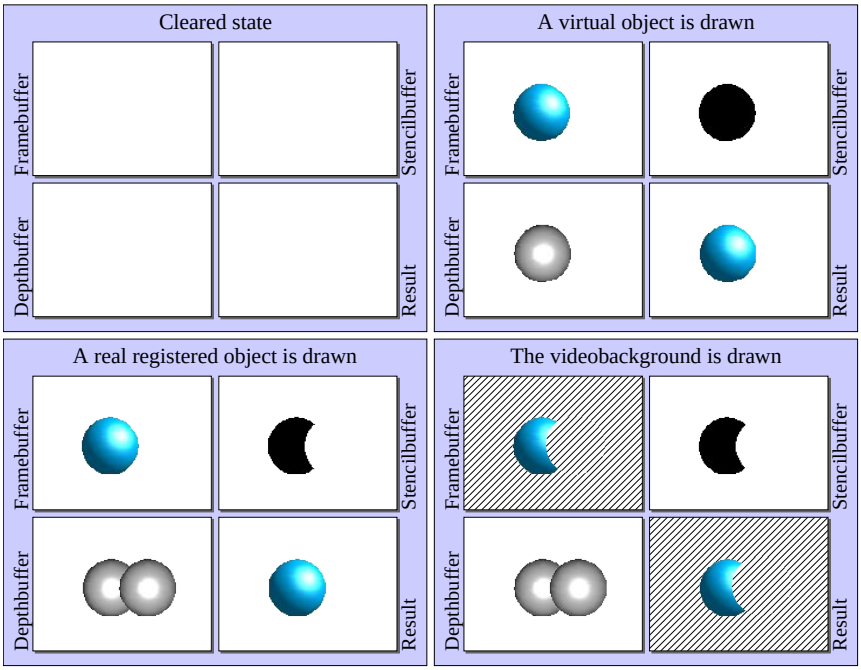


Abbildung 9: Schematische Arbeitsweise des Verdeckungsalgorithmus

1. Der Framebuffer und der Tiefenbuffer werden gelöscht. Der Stencilbuffer wird mit ‚0' gefüllt.
2. Virtuelle Objekte werden normal in den Framebuffer gezeichnet. In den Stencilbuffer wird an die entsprechende Stelle eine ‚1' geschrieben.
3. Reale Objekte, die im 3D-Raum registriert sind, werden *nur* in den Tiefenbuffer und nicht in den Framebuffer geschrieben. Somit sind die Tiefeninformationen des realen Objektes gespeichert, die Repräsentation dieses Objektes erfolgt später durch das Videobild. An die entsprechenden Stellen im Stencilbuffer wird eine ‚0' eingetragen.
4. Nachdem die komplette Szene gerendert wurde, wird nun das Kamerabild in den Framebuffer gezeichnet, aber nur an den Stellen, an denen im Stencilbuffer eine ‚0' steht.

Mit dieser Methode ist es möglich, beliebige virtuelle und reale registrierte Objekte in unsortierter Reihenfolge in einer Traversierung des Szenegraphen zu rendern und eine korrekte Verdeckung zu erhalten.

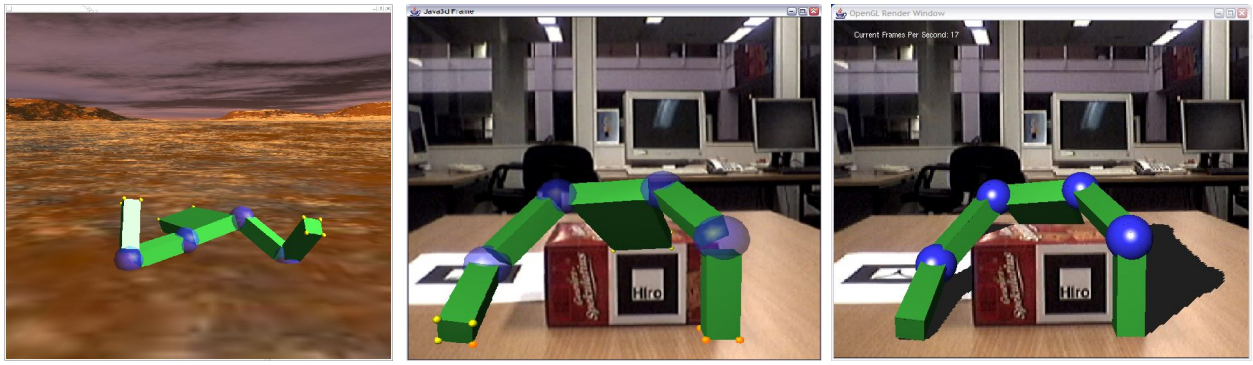
Die Erzeugung von Schatten auf sowohl virtuelle als auch reale registrierte Objekte erfordert die Erweiterung des Renderprozesses auf mehrere Durchläufe (Passes). Für die Schattenberechnung kam der „Depthmap Shadow"-Algorithmus zur Anwendung, der bei den meisten modernen Grafikkarten in Hardware unterstützt wird. Vorteil dieses Algorithmus ist, dass keine zusätzlichen Geometrieinformationen vorhanden sein müssen. Ein Nachteil ist die begrenzte Auflösung der Depthmaps, die bei bestimmten Lichteinfallswinkeln sehr grobe Schatten erzeugen. Die Implementation dieses Algorithmus wurde in 3 Phasen realisiert:

1. Rendern der Szene aus der Lichtposition und Speichern des Z-Buffers in eine Depthmap Textur.
2. Rendern der Szene nur mit diffuser abgedunkelter Beleuchtung, der Schattenfarbe.
3. Rendern der Szene mit der vollen Beleuchtung aber nur an den Stellen, an denen die Depthmap und der Z-Buffer übereinstimmen.

Eine genaue Beschreibung des Algorithmus kann z.B. in der Präsentation von Cass Everitt „Shadow Mapping" [Nv00] nachgelesen werden.

7 Beispielapplikation

In diesem Kapitel wird das oben erwähnte Szenario der virtuellen Kreaturen beispielhaft vorgestellt. Dabei sollen die unterschiedlichen Stufen hin zu besserer Interaktion mit den Charakteren deutlich werden.



a) VR mit Java3D b) AR mit Java3D c) AR mit JOGL & Visual Cues

Abbildung 10: Kreatur in VR und AR und zusätzlich mit „Visual Cues"

Wie in Abbildung 10 a) zu sehen, reicht die Darstellungsqualität von Java3D für eine reine VR-Sicht auf eine virtuelle Kreatur völlig aus. In b) wird klar, dass eine inkorrekte Darstellung der Verdeckung sehr verwirrend ist. Zudem wirkt das Bild sehr planar, die Tiefenwahrnehmung ist nicht ausreichend. Bild c) jedoch stellt die Szene korrekt dar. Durch Schatten und Verdeckung bekommt der Anwender einen sehr guten Eindruck sowohl von der Position der Kreatur in der Szene als auch von der Position des mit dem „Hiro"-Pattern markierten tangible Userinterfaces.

8 Zusammenfassung und Ausblick

Es wurde gezeigt, dass die evolutionäre Entwicklung von virtuellen Kreaturen unter Verwendung von ODEforJava, Java3D und ECJ möglich ist. Bei der Beurteilung des Entwicklungsstandes der virtuellen Kreaturen hat sich als Interaktionsmethode die Verwendung von „Augmented Reality“ als grundsätzlich vorteilhaft erwiesen. Jedoch hat erst der Einsatz fortgeschrittener Rendertechniken dazu geführt, dass der Anwender wirklich intuitiv mit den Kreaturen umgehen kann.

In Zukunft wird daran gearbeitet, das Modell der virtuellen Kreaturen dahingehend zu erweitern, dass diese auch miteinander kooperieren können. Die dadurch wachsende Komplexität der Experimente dürfte die Fähigkeit zur Interaktion über AR noch notwendiger machen.

9 Literatur

[GSS04] C. Geiger, Jörg Stöcklein, Tim Schmidt: “Entwicklung physikbasierter AR-Anwendungen mit Java”, Augmented & Virtual Reality in der Produktentstehung, HNI Verlag, Paderborn, 2004

[Hol75] John H. Holland: “Adaption in Natural and Artificial Systems”, University of Michigan Press, Ann Arbor, 1975.

[Kato99a] H. Kato, M. Billinghurst: “Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System”, In Proc. IWAR '99, 85-94ff, San Francisco, 1999

[KoUl99] M. Komosinski, Sz. Ulatowski: “Framsticks: towards a simulation of a nature-like world, creatures and evolution”, Proceedings of 5th European Conference on Artificial Life (ECAL99), Springer-Verlag (LNAI 1674), 261-265, Switzerland, 1999

[Koza94] Koza, John R.: “Genetic Programming II: Automatic Discovery of Reusable Programs”, MIT Press, Cambridge, Massachusetts, 1994

[LiPo00] H. Lipson and J. B. Pollack: “Automatic design and Manufacture of Robotic Lifeforms”, Nature 406, 974-978ff, 2000

[Nv01] Cass Everitt: “Shadow Mapping”, NVIDIA Corporation Webseite, 2001

[Sims94] K. Sims: “Evolving Virtual Creatures”, Siggraph '94 Proceedings, 15-22ff, 1994