TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

Diplomarbeit

Der Türkische Schachspieler
Prototyp einer Augmented Reality Applikation

ausgeführt am

Institut für Computergraphik und Algorithmen

der Technischen Universität Wien

unter Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Michael Gervautz

durch

Alexander Schälss

Matrikelnummer 8425511

Bernardgasse 17/4, 1070 Wien

Wien, 24. Jänner 2004

Kurzfassung

Im Jahr 1769 präsentierte der kaiserlich-königliche Hofkammerrat Wolfgang von Kempelen seinen mechanischen Schachspieler am Wiener Hof Maria Theresias. Der Schachautomat war ein eleganter Holzkasten, hinter dem eine lebensgroße Puppe in türkischer Kleidung saß. Während der Vorführung konnte ein Herausforderer gegen die mechanische Figur eine Partie Schach spielen. Nach mehreren Tourneen durch Europa und die USA wurde der „Automat“ im Jahr 1854 bei einem Brand im Peale‘s Museum in Philadelphia zerstört.

Im Rahmen des „Virtual Showcase“-Projektes, welches ein Teil des IST-Programms (Information – Society – Technologies) der Europäischen Union ist, soll eine zeitgemäße Reproduktion des historischen Automaten für das Technische Museum Wien entwickelt werden. Der Autor beschreibt in dieser Diplomarbeit die Hardware- und Softwarekomponenten, die für den Prototypen des „Türkischen Schachspielers“ eingesetzt und entwickelt wurden.

Die Grundlage für den Prototyp ist das Augmented Reality-System STUDIERSTUBE. Das 3D-Modell des „Türkischen Schachspielers“ wird mittels passiver Stereo-Projektion dem Besucher präsentiert. Das Bewegen der virtuellen Schachfiguren erfolgt direkt durch die Hand des Benutzers unter Verwendung eines Bildverarbeitungssystems.

Abstract

Hofkammerrat Wolfgang von Kempelen presented his mechanical chess player to the Viennese court of empress Maria Theresia in the year 1769. The chess automaton was an elegant wooden showcase. A life-sized figure in Turkish clothes sat behind a table. During a demonstration a human challenger could play chess against the mechanical figure. After several tours in Europe and the U.S.A. the “automaton” was destroyed by fire in the Peale`s Museum in Philadelphia in the year 1854.

In the context of the “Virtual Showcase”-project, which is part of the IST initiative (Information – Society – Technologies) of the European Union, a modern reproduction of the historical automaton for the Technisches Museum Wien was to be developed. In his master thesis the author describes the hardware and software components which were used and developed for the prototype of the “Turkish Chess Player”.

The prototype is based on the Augmented Reality framework STUDIERSTUBE. The 3D-Model of the “Turkish Chess Player” is presented to the user by means of a passive stereo projection. The virtual chessmen are moved directly by the hand of the user via an image processing system.

Inhaltsverzeichnis

Kapitel 1

Einführung

> *„Die Erscheinung einer mechanischen Figur, die mit einem denkenden beseelten Wesen das schwerste aller Spiele spielt, sich seinem belebten Gegner gleich bewegt, von dessen Willen und Spiel abhängt, gleich ihm oft gewinn, oft verliehrt, kurz der kühnste Gedanke eines Mechanikers, das Meisterstück der Schöpfung in einem beweglichen Bilde nachzuahmen, war zu auffallend, um nicht die größte Aufmerksamkeit zu erregen.“*
>
> – Chrétien de Mechel, aus dem Vorwort zu Karl Gottlieb von Windisch, *Briefe über den Schachspieler des Herrn von Kempelen*, 1783

1.1 Motivation

Der intuitive Zugang zu Informationen in unserem täglichen Umfeld wird immer wichtiger. Eine Möglichkeit, dies zu erreichen, ist der Einsatz von neuen Informationstechnologien in Verbindung mit Gegenständen aus dem alltäglichen Gebrauch.

Im Rahmen des Projekts „Virtual Showcase“ werden neue Arten von stereoskopischen Anzeigesystemen entwickelt und deren Einsatz erforscht. Das Virtual Showcase ist ähnlich einer herkömmlichen Vitrine, wie sie in Museen eingesetzt wird, aufgebaut. Dadurch gelingt es, die gewünschte Information dem Betrachter möglichst einfach und intuitiv näher zu bringen. Neben wissenschaftlichen Ausstellungsstücken können auch kulturelle Artefakte im Virtual Showcase ausgestellt werden. Im Virtual Showcase wird das reale Exponat mit computergenerierter dreidimensionaler Stereographik kombiniert. Das reale Schaustück und die virtuelle Repräsentation teilen sich den gemeinsamen Raum innerhalb der Vitrine.

Die Verbindung von virtuellem und realem Inhalt erlaubt neue Wege in der Vermittlung von Information. Der virtuelle Teil des Ausstellungsstücks kann auf verschiedene Art und Weise mit dem Betrachter interagieren und erlaubt so neue Sichtweisen auf das ausgestellte Material. Bei diesen intelligenten Schaukästen steht die Bedienung des Computers im Hintergrund und der Besucher kann seine Aufmerksamkeit auf die Erforschung des Inhalts richten. Dem Besucher soll auch die Angst im Umgang mit neuen Technologien genommen werden. Die Interaktion mit dem Virtual Showcase soll dem natürlichen menschlichen Verhalten zum Erwerb von Wissen entsprechen. Im Weiteren soll diese Art der Vermittlung von Inhalten beim Betrachter das Interesse nach zusätzlichem Wissen wecken.

Die Museumsinstallation

Im Jahr 2004 jährt sich der Todestag von Wolfgang von Kempelen zum 200sten Mal. Im gleichen Jahr plant das Technische Museum Wien dem Publikum ein neues Ausstellungsstück zu präsentieren – den wiedererweckten „Türkischen Schachspieler".

Im Rahmen des „Virtual Showcase"-Projektes entwickeln mehrere Projektpartner im europäischen Raum gemeinsam die technischen Grundlagen, um dies zu ermöglichen [BFS+01a] [BFS+01b]. Diese Arbeit soll den Hardware- und Software-Prototypen für die Installation liefern, aus dem die eigentliche Museumsinstallation hervorgeht.

1.2 Der Türkische Schachspieler

1.2.1 Geschichte

Im Jahr 1769 präsentierte der kaiserlich-königliche Hofkammerrat Wolfgang von Kempelen (1734 - 1804) seinen „Schachautomaten" erstmals der Weltöffentlichkeit am Wiener Hofe von Maria Theresia. Die Herrscherin fand zu dieser Zeit Gefallen an der Vorführung von Kunststücken, die auf physikalischen und chemischen Gesetzmäßigkeiten beruhten. Nach einer dieser Vorführungen, welche einer heutigen Zaubervorführung nicht unähnlich gewesen sein dürfte, fragte sie den anwesenden Wolfgang von Kempelen, was er davon halte. Er erklärte zum Erstaunen der Anwesenden, daß er eine Maschine bauen könne, welche sehr viel spektakulärer als das eben Gezeigte sei.

Sechs Monate später – während dieses Zeitraums war Kempelen von seinen Pflichten am Hofe von Maria Theresia entbunden – stellte er den fertigen Schachautomaten am Wiener Hofe vor. Die Kritiker waren sich nach der ersten Vorstellung einig, man hatte eine technische Sensation gesehen. Nach der Premiere gab Kempelen noch weitere gut besuchte Vorstellungen für die Wiener Gesellschaft. Unter dem Vorwand, die Maschine sei irreparabel beschädigt, bekam der Schachautomat eine Pause von zehn Jahren.

Erst im Jahre 1781, als der Sohn Katharinas II, Großfürst Paul in Wien zu Besuch war, erinnerte man sich am Wiener Hof wieder des Automaten. Zur Belustigung der Gäste wurde der türkische Schachspieler wieder reaktiviert. Auf Anregung des Großfürsten Paul machte Kempelen im Folgenden eine mehrjährige Tournee durch die Städte Europas. Die Reise führte ihn über Frankfurt, Paris, Amsterdam auch nach London. 1785 kehrt Kempelen wieder nach Wien zurück. Der Apparat blieb danach bis zum Tod von Kempelen im März 1804 in den Lagerräumen des Schlosses Schönbrunn.

1806 verkaufte Carl von Kempelen, Wolfgang von Kempelens Sohn, den Türken an Johann Nepomuk Maelzel. Der Schachautomat befand sich bei Mechanikus Maelzel in der Gesellschaft anderer Automaten, mit denen ihr Besitzer Vorstellungen in ganz Europa gab. Ein weiterer Höhepunkt in der Geschichte des türkischen Schachspielers ist der 9. Oktober 1809. An diesem Tag spielte Napoleon gegen den Automaten und verlor. Maelzel verkaufte den Schachautomaten an Eugène de Beauharnais und der Apparat bezog in Italien Quartier. 1817 kaufte Maelzel den Türken wieder zurück und präsentierte ihn wieder in Paris.

Ende 1825 beginnt ein weiterer Abschnitt in der Geschichte des Türken. Maelzel überquert mit ihm den Atlantik auf einem Postboot in Richtung Neue Welt. Dort wird er in meist ausverkauften Vorstellungen u.a. in New York, Boston, Philadelphia und Baltimore gezeigt. Nachdem die Popularität des Schachspielers sich 1837 in Amerika dem Ende zuneigt, bricht Maelzel nach Havanna auf. Nach einigen gut besuchten Vorstellungen in Havanna muß Maelzel nach Philadelphia zurückkehren und stirbt während der Rückreise im Jahr 1838.

Der Türke gelangt über einen Zwischenhändler in den Besitz des schachbegeisterten Physikers John K. Mitchell. Dieser schenkt den Automaten, nach einigen privaten Vorführungen, im Jahr 1840 dem Peale's Museum in Philadelphia. Nach vierzehn Jahren als Ausstellungsstück verbrennt der türkische Schachspieler am 5. Juli 1854 bei einem Feuer im Museum.

1.2.2 Vorführung

Es gibt einige Aufzeichnungen und Bilder über den Automaten, die während seines Daseins entstanden sind. Karl Gottlieb von Windisch, ein Freund von Kempelen, gibt in [Win83] einen Überblick über das Aussehen des Automaten und wie er von Kempelen bei seinen Auftritten vorgeführt wurde. Joseph Friedrich Freyherr zu Racknitz gibt in [Rac89] nicht nur eine detaillierte Beschreibung, ähnlich wie Windisch, sondern versucht auch die Funktionsweise des Automaten zu erklären.

Der türkische Schachspieler bestand aus einem eleganten Holzkasten in der Form eines Tisches, hinter dem eine lebensgroße Puppe in türkischer Kleidung saß. An der Vorderseite wies der Kasten drei Türen und darunter eine Schublade auf. Auf dem Tisch war das Schachbrett fix angebracht. Der Kupferstich aus [Win83] in Abbildung 1.1 zeigt die Vorderseite des Schachspielers.

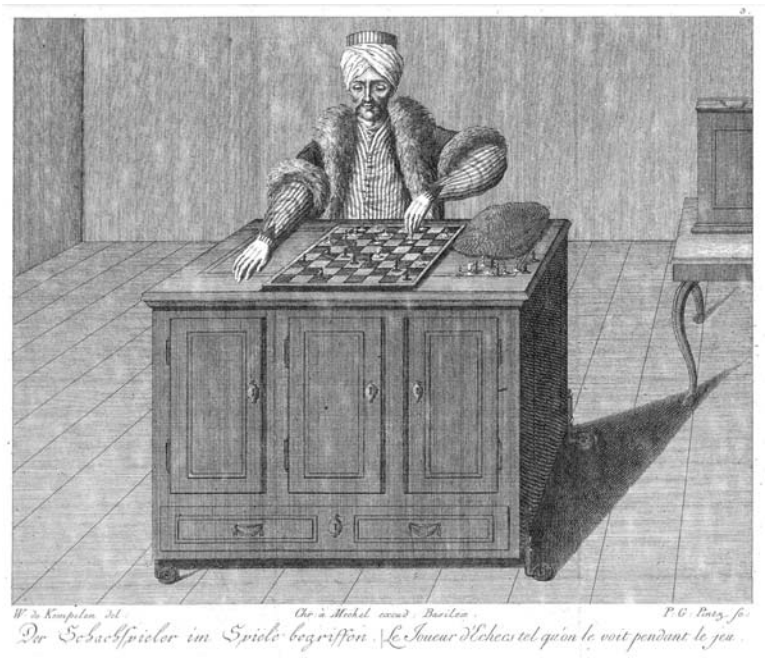


Bild 1.1: Vorderansicht des Schachspielers, mitten im Spiel mit erhobenem Arm.

Vor dem Beginn einer Vorstellung wurde das Innere des Kastens in einer immer genau wiederholten Reihenfolge vorgeführt. Dazu aus einem Artikel von Edgar Allen Poe, der bei einigen Vorstellungen in Richmond, Virginia zugegen war, in [Lev00]:

> *„It will be necessary for a proper understanding of the subject, that we repeat here in a few words, the routine adopted by the exhibitor in disclosing the interior of the box – a routine from which he never deviates in any material particular.“*

Der Kasten besaß zu diesem Zweck an allen vier Ecken Messingrollen, auf denen er leicht verschoben und gedreht werden konnte. Die Türen auf der Vorder- und Rückseite wurden geöffnet und mit einer Kerze das Innere durchleuchtet. Hinter den drei Türen des Kastens erblickten die Zuseher ein Gewirr von Walzen, Zahnrädern und Gestängen von verschiedensten Größen. Die Abbildung 1.2 aus [Win83] zeigt die Kommode von vorne mit geöffneten Türen und herausgezogener Schublade.

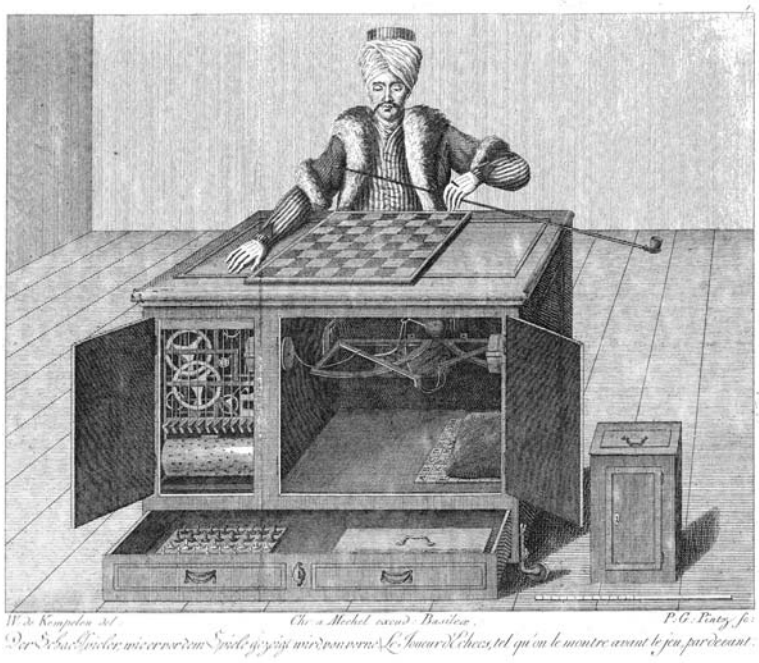


Bild 1.2: Diese Zeichnung stellt den Schachspieler so vor, wie er, ehe das Spiel beginnt, von vorne mit offenen Türen und herausgezogener Schublade gezeigt wird.

In der Lade unterhalb der Türen befanden sich die Schachfiguren. Dazu schrieb Karl Gottlieb von Windisch in [Win83]:

> *„In der Schublade desselben stehen die Spielsteine von weissem, und rothgefärbten Elfenbein auf einem Brette, die sammt demselben herausgenommen, und zu dem Schachbrette hingestellt werden.“*

Vor dem Spiel wurden alle Türen auf der Vorder- und Rückseite wieder geschlossen. Die Schublade blieb auch während des Spiels herausgezogen. Weiters wurden vor der Schachpartie die Regeln für das Spiel mit dem Automaten vereinbart. Diese zählte Joseph Friedrich Freyherr zu Racknitz in [Rac89] auf:

1) Daß der Türke den ersten Zug habe.
2) Daß man bey dem Ziehen die Steine recht in die Mitte des Feldes setze.
3) Daß man den gezogenen und schon auf das Feld gesezten Stein, wenn man auch einen besseren Zug thun könnte, nicht wieder zurücknehme; und
4) daß, wenn entweder aus Uebereilung, oder in der Absicht, den Türken auf die Probe zu stellen, einen falschen, den Regeln des Spieles widersprechenden Zug thue, (z.B. wenn man den Laufer als Springer, oder den Rochen als Laufer gehen lassen) der Zug verloren sey.

Nun einige Einzelheiten zu den Bewegungen welche die Figur ausführen konnte. Der Automat führte die Züge mit der linken Hand der Puppe aus. Windisch schrieb dazu in [Win83]:

„Nun, wann diese Maschine spielt – hier muß ich Ihnen, damit Sie ja alles genau wissen, sagen, daß sie mit der linken Hand spielt.
Ich fragte um die Ursache und erfuhr, daß der Erfinder darauf nicht Acht hatte, und diesen kleinen Fehler erst bemerkte, als er mit seiner Arbeit schon zu weit gekommen war, als daß er ihn hätte verbessern können. – Doch das gehört zur Sache nicht; denn was liegt uns daran, ob Titan sein Bild mit der rechten, oder linken Hand gemalt hat! –“

Folgende Bewegungen konnten mit der linken Hand ausgeführt werden und wurden in [Rac89] dargestellt:

„Den Arm beweget der Türke
1) vertikal, oder auf- und niederwärts;
2) horizontal, rechts und links, und
3) winkelförmig.“

Zusätzlich zu den Bewegungen der linken Hand, welche dazu dienten, die Schachfiguren zu bewegen, konnte der Automat auch Bewegungen mit dem Kopf machen. Die Bewegungen des Kopfes und der linken Hand werden von Racknitz [Rac89] wie folgt beschrieben:

„Er hatte seine Augen auf das Schachbret gerichtet, neigte, ehe er einen Stein zog, den Kopf, drehete denselben rechts und links, als ob er das Spiel besähe, richtete ihn wieder in die erste Stellung, erhob alsdann den linken Arm vom Polster gerade in die Höhe, machte damit die zu Erreichung des zu ziehenden Steines nöthige horizontale und winkelförmige Bewegung, ließ den Arm sinken, wenn er diesen erreichet hatte, öfnete die Hand, ergrif mit der sich wieder schließenden Hand den Stein, brachte ihn auf das Feld, wo er hinkommen sollte, öfnete die Finger und brachte den Arm wieder durch eine horizontale und winkelförmige Bewegung über das Polster, auf welches derselbe sich sodann senkte.“

Beim Schlagen einer gegnerischen Figur stellte der Türke die geschlagene Schachfigur vor seinem Zug neben das Seidenpolster, auf dem die linke Hand des Türken normalerweise ruhte. Danach machte er seinen Zug. Gab der Türke Schach dem gegnerischen *König*, so nickte er dreimal mit dem Kopf, gab er Schach der *Königin*, so nickte er nur zweimal. Wollte man eine Partie fortsetzen, welche bereits beendet war,

so nahm der Türke das Spiel nicht an und schüttelte den Kopf. Zur Bewegung des Kopfes findet sich in [Rac89] folgender Text:

> *„Mit dem Kopfe machet er zweyerley Bewegungen, indem er denselben*
> *1) vorwärts neiget, und*
> *2) rechts und links wendet.*
> *Beydes thut er, wenn er das Spiel zu besehen scheinet, und das erstere auch wenn er Schach biethet.“*

Bei einem den Schachregeln widersprechenden Zug des Gegners schüttelte der Türke den Kopf, zum Zeichen, daß er den Zug nicht annimmt, und setzte die gegnerische Schachfigur wieder auf ihr ursprüngliches Feld zurück. Darauf folgend tat er seinen eigenen Zug und der Schachzug war für den Gegner verloren.

1.2.3 Funktionsweise

Der türkische Schachspieler wurde bei den Vorführungen als vollständig mechanischer Automat vorgestellt. Dies war mit den technischen Mitteln des 18. Jahrhunderts aber noch nicht zu realisieren. Es gab zu diesem Zeitpunkt schon mechanische Automaten, wie z.B. den Flötenspieler von Jacques de Vaucansons oder die zeichnenden und schreibenden Androiden von Henri-Louis Jaquet-Droz [Sta02] [Str96]. Für einen Mechanismus, der ein beliebiges Schachspiel durchführen konnte, war die technische Entwicklung aber noch nicht weit genug fortgeschritten. Im Inneren des Automaten verbarg sich ein meistens sehr guter Schachspieler, welcher das Spiel von innen lenkte.

Einige der Bewunderer und Skeptiker stellten sich die legitime Frage, ob es möglich war, eine „intelligente“ Maschine zu bauen. Um diese Frage beantworten zu können, versuchten sie hinter das „Geheimnis“ des Türken zu kommen. Einer, der dieses Problem zu beantworten versuchte, war Joseph Friedrich Freyherr zu Racknitz. Er sah den Automaten bei mehreren Vorstellungen in Deutschland und versuchte durch Aufstellung von verschiedenen Erklärungsmodellen schlüssig zu beweisen, daß sich im Inneren der Kommode ein Mensch befindet. Er baute mehrere Modelle, um die Richtigkeit seiner Behauptungen zu belegen. Abbildung 1.3 aus [Rac89] zeigt den Automaten während des Spieles, mit dem in ihm befindlichen, vor dem Publikum verborgenen Schachspieler.

Es gab für Racknitz zwei größere Probleme zu lösen: Wie kann der verborgene Schachspieler im Inneren des Automaten die Züge auf dem Schachbrett verfolgen und wie wird der linke Arm des Türken von innen gesteuert. Zur Lösung des ersten Problems bedient sich Racknitz der magnetischen Eigenschaften von Eisen und schreibt dazu in [Rac89]:

> *„Hier schien mir nun die magnetische Kraft das Mittel, wodurch der innere Spieler jeden Zug der auf dem oberen Schachbrete geschieht, bemerken, und indem er solchen, auf einem vor sich habenden kleinen Schachbrete nach thue, die oben stehende Partie immer übersehen könne.“*

Dazu wird jede Schachfigur mit einem Magnetkern, unsichtbar für den menschlichen Spieler, versehen. Das Schachbrett selbst besteht nur aus dünnem Holz und ist

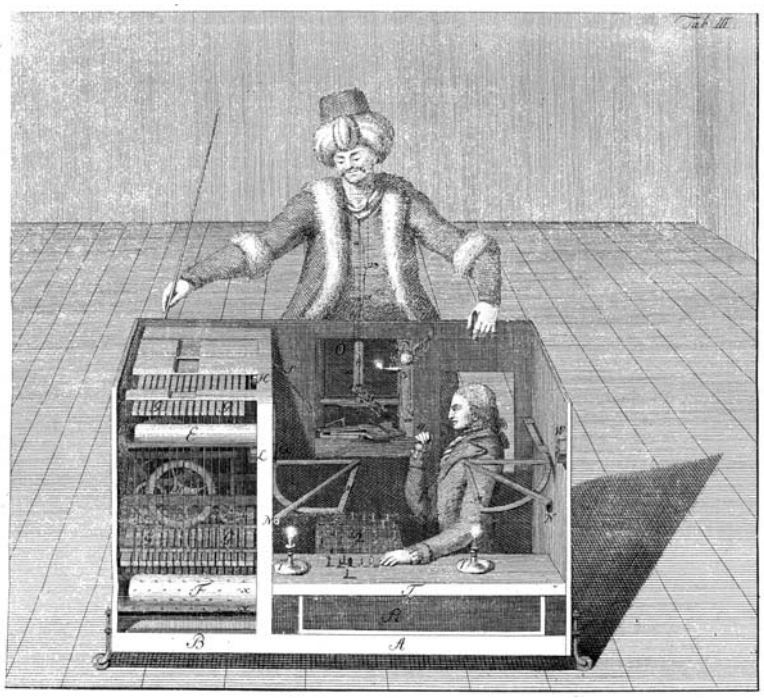


Bild 1.3: Vorderansicht des Automaten mit in ihm verborgenem Schachspieler.

bündig in den Tisch eingelassen. Unter jedem Feld des Schachbrettes befindet sich innerhalb der Kommode, nur für den verborgenen Spieler sichtbar, eine Metallnadel, welche durch das Hinstellen oder Wegnehmen einer Schachfigur ihre Position ändert. Kerzen lieferten dem Spieler im Innern des Automaten das benötigte Licht.

Die Lenkung des Türken erfolgte mit großer Präzision. Mittels einer Storchenschnabelmechanik war es dem verborgenen Spieler möglich, die gewünschten Bewegungen auf einem verkleinerten Schachbrett im Inneren durchzuführen. Diese wurden über einen Seilzug auf den linken Arm des Türken am großen Schachbrett übertragen. Zusammenfassend schreibt [Rac89] wie folgt:

> *Im Ganzen ist die Idee, nach welcher ich gearbeitet habe, diese: Ein in dem Kasten verborgener Mensch bemerkt an gewissen magnetischen Nadeln, was für Züge auf dem obern Schachbrete geschehen, thut diese auf einem kleinen vor ihm liegenden Schachbrete nach, übersiehet auf diese Art das Spiel, und lässet durch den Türken die nöthigen Gegenzüge auf dem obern Schachbrete verrichten.*

Seine Annahme, daß sich ein menschlicher Schachspieler im Innenraum des Automaten verborgen hielt, sah er durch verschiedene Gegebenheiten bestätigt. So soll Wolfgang von Kempelen Folgendes zu seinen Freunden gesagt haben (aus [Rac89]):

> *„bey seiner Maschine komme Täuschung vor, sie werde nicht durch bloßen Mechanismus in Bewegung gesezet; er halte sich aber nicht verbunden, solches dem ganzen Publicum zu eröfnen.“*

Nach diesem Überblick über die Geschichte und die Funktionsweise des historischen Schachspielers wollen wir uns mit den technischen Grundlagen für die neue Version des Automaten beschäftigen.

1.3 Relevante Literatur

Ein Großteil der diese Arbeit betreffenden Publikationen kann zwei Gruppen zugeordnet werden. Die erste Gruppe an Veröffentlichungen beschäftigt sich mit der Lösung von Problemen im Bereich der *inversen Kinematik*, im Speziellen mit der Bewegung der menschlichen Gliedmaßen. Eine Zusammenfassung dieser Arbeiten findet sich in Kapitel 3.2. Die zweite Gruppe befaßt sich mit der *Interaktion* von realen und virtuellen Menschen und wird im Folgenden beschrieben.

BALCISOY und THALMANN beschreiben in [BT97] das Design und die Implementierung eines AR-Systems, welches die Interaktion einer Person mit einer virtuellen Figur erlaubt. An Hand einer Beispielanwendung, dem interaktiven Bühnenspiel mit einem realen und einem virtuellen Schauspieler, werden die Probleme und Möglichkeiten für die Interaktion der Akteure miteinander und mit realen und virtuellen Objekten dargestellt. In [TT97] befassen sich THALMANN und THALMANN mit dem Zusammenspiel einer virtuellen Figur und der realen Umgebung. Im Speziellen wird auf Verdeckungen zwischen der virtuellen Figur und der realen Szene eingegangen.

TORRE, FUA, BALCISOY, PONDER und THALMANN schildern in [BTP+00] und [TBF+00] ein System, welches die Interaktion zwischen einem realen Menschen und einer virtuellen Figur in einem AR-Kontext erlaubt. Dies wird durch die Kombination eines nicht–invasiven Bilderkennungssystems, einer Echtzeitsimulation für die virtuelle Figur und einer geeigneten Rendering Plattform erreicht. Die beschriebene Anwendung erlaubt es, das Spiel *Dame* gegen einen virtuellen Kontrahenten zu spielen.

1.4 Augmented Reality

Augmented Reality ist ein wachsender Forschungsbereich. Die uns umgebende Welt enthält eine Fülle von Informationen, welche im Computer nur sehr schwer dargestellt werden kann. Ein Augmented Reality-System bietet dem Benutzer eine kombinierte Ansicht der realen Welt und der vom Computer generierten Bilder. Diese synthetischen Bilder liefern zusätzliche Informationen zur realen Umgebung. Die virtuellen Objekte in diesen Bildern können unter anderem spezifische Informationen sein, welche der Benutzer mit seinen Sinnen nicht oder nur sehr schwer erfassen kann. Augmented Reality Anwendungen versuchen dem Anwender die Wahrnehmung von Gegebenheiten in der realen Welt und die Durchführung von Tätigkeiten in ihr zu erleichtern.

1.4.1 Augmented vs. Virtual Reality

Der Begriff „Virtual Reality“ umfaßt ein breites Spektrum an Ideen und Technologien. Unter diesem Begriff kann man eine „computer-generierte, interaktive, dreidimensionale Umgebung, in die eine Person eintaucht“ [AB92] verstehen. Wie man sieht, besteht diese Definition aus drei wichtigen Punkten. Der erste Punkt besagt, daß die virtuelle Welt von einem Computer generiert wird. Dafür benötigt man die entsprechende leistungsfähige Hardware, um ein den Anforderungen entsprechendes Bild zu bekommen. Der zweite Punkt befaßt sich mit der Interaktivität. Es soll dem Anwender

möglich sein, mit dem Inhalt der dargestellten Szene in Realzeit interagieren zu können. Der letzte Punkt besagt, daß die Person, die das System benutzt, in die virtuelle Umgebung eintauchen soll.

Einer der augenfälligsten Unterschiede zwischen einem Virtual Reality- und einem Augmented Reality-System ist der Grad dessen wie viel der Anwender von der ihn umgebenden Realität noch wahrnimmt. Bei einem Virtual Reality-System ist man bestrebt, den Benutzer vollständig mit einer künstlichen Realität zu umgeben. Ein Augmented Reality-System liefert zusätzlich zur gegenständlichen Welt weitere Informationen. Dafür ist es aber notwendig, daß sich der Anwender weiterhin seiner Präsenz in der Realität bewußt ist.

Um eine konsistente Definition für diese Begriffe zu haben, beschrieb MILGRAM et al. in [MTU+94] das so genannte „Reality-Virtuality (RV) Continuum“, welches in Abbildung 1.4 zu sehen ist. An der linken Seite des RV-Continuums ist die reale Welt angesiedelt. Auf der rechten Seite findet man eine Welt, in der ausschließlich virtuelle Objekte existieren.

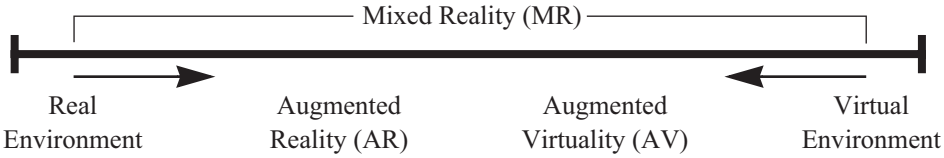


Bild 1.4: Reality-Virtuality (RV) Continuum von Milgram.

Die Mischung aus realen und virtuellen Objekten wird als „Mixed Reality“ bezeichnet. Es wird auch zwischen Augmented Reality und Augmented Virtuality unterschieden. Hierbei kommt es darauf an, wie hoch der Anteil an realen und virtuellen Objekten ist.

AZUMA definiert Augmented Reality-Systeme in [Azu97] mit Hilfe von drei charakteristischen Merkmalen:

1) Combines real and virtual
2) Interactive in real time
3) Registered in 3-Dimensions

Diese Definition kann auch auf andere Sinne, wie z.B. Gehör oder Haptik, ausgeweitet werden.

1.4.2 Augmented Reality Techniken

Die meisten Augmented Reality-Systeme bestehen aus den gleichen Typen von Komponenten:

- Sensoren, welche die Daten der realen Welt liefern (z.B. Videokamera)
- Computer, welche die Bilder der virtuellen Szene berechnen
- Sensoren (Tracker) für die Ermittlung der Positions- und Lage-Daten
- eine Vorrichtung, welche die errechneten mit den realen Bildern kombiniert
- und die Anzeige, auf welcher die Bilder für den Betrachter dargestellt werden.

Laut obiger Definition erzeugt ein Augmented Reality-System eine zusammengesetzte Sicht für den Benutzer. Es kombiniert die reale Umgebung mit einer vom Computer erzeugten Information.

Man kann Augmented Reality-Systeme anhand der verwendeten Technik in vier Hauptklassen einteilen. *Optical-See-Through* Augmented Reality (AR) verwendet transparente Head Mounted Displays (HMD), um die virtuelle Information über das Bild der realen Welt anzuzeigen. *Video-See-Through* AR verwendet ein undurchsichtiges HMD, um die reale Welt, welche über Videokameras aufgenommen wird, mit dem Video der virtuellen Umgebung zu verschmelzen. *Projektor* basierende AR verwendet reale Objekte als Projektionsfläche für die virtuelle Umgebung. Die *Monitor* basierende AR verwendet, ähnlich wie die Video-See-Through AR, kombinierte Videoquellen, zeigt diese aber auf einem herkömmlichen Arbeitsplatzbildschirm oder transportablen Gerät (Hand Held, PDA, etc.) an. Jede dieser Techniken hat ihre Vor- und Nachteile. Eine detaillierte Beschreibung ist u.a. in [Azu97] [RL01] zu finden.

Optisches und Video See-Through Head Mounted Display

Ein See-Through Head Mounted Display ist ein Gerät, welches die Bilder der realen und virtuellen Welt miteinander vereinigen kann. Um die Mischung der realen und virtueller Bilder vorzunehmen gibt es zwei grundlegende Möglichkeiten – optisch oder auf Video basierend. Bei geschlossenen HMDs ist es nicht möglich, die reale Umgebung in das Blickfeld mit einzubeziehen. See-Through HMDs ermöglichen es dem Anwender, die virtuellen Objekte mit der realen Szene gemeinsam zu betrachten. In Augmented Reality-Systemen kommen zwei unterschiedliche Typen von See-Through HMDs zum Einsatz. Es wird zwischen Optical-See-Through- (OST-HMD) und Video-See-Through-Systemen (VST-HMD) unterschieden.

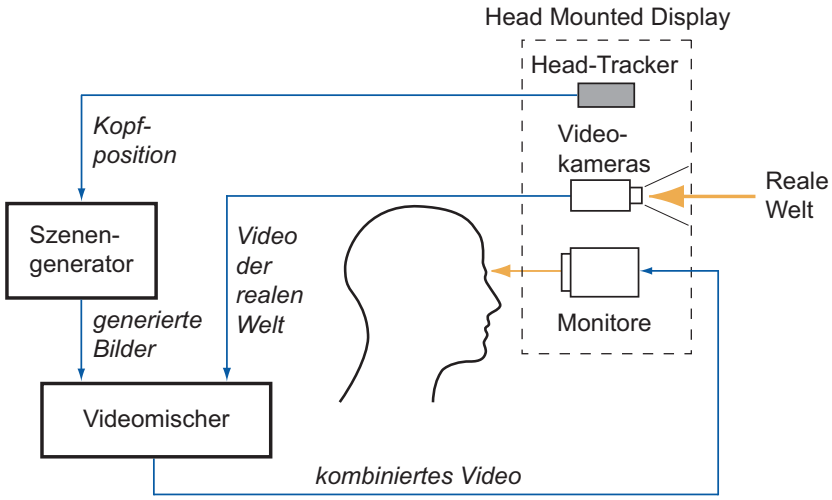


Bild 1.5: Konzept des Video-See-Through Head Mounted Display nach [Azu97].

Beim Video-See-Through HMD wird die reale Welt mit einer oder mehreren Videokameras, die im HMD eingebaut sind, aufgenommen (siehe Abbildung 1.5). Die Videokameras nehmen für den Benutzer die reale Umgebung auf. Die Mischung von Real- und computergeneriertem Bild kann auf verschiedene Arten erfolgen. Eine ein-

fache Möglichkeit ist das Chroma-Keying. Der Hintergrund der im Rechner generierten Bilder ist mit einer bestimmten Farbe, z.B. Grün, ausgefüllt. Diese spezielle Farbe sollte natürlich von keinem der virtuellen Objekte verwendet werden. Der Kombinationsschritt ersetzt nun im computergenerierten Bild alle in der speziellen Farbe eingefärbten Bildpunkte durch die entsprechenden Bildpunkte des Realbildes. Die virtuellen Objekte werden über die reale Szene gelegt. Es sind auch komplexere Verschmelzungen, bei denen z.B. die Tiefeninformationen im virtuellen Bild miteinbezogen werden, möglich. Damit sind Überdeckungen in beide Richtungen machbar. Das daraus resultierende Bild wird auf den Monitoren im HMD, welche sich vor den Augen des Benutzers befinden, gezeigt. Abbildung 1.6 zeigt ein Video-See-Through HMD, welches auf dem COASTAR (Co-optical Axis See-Through for Augmented Reality) basiert [TYS+00]. Dabei haben die Videokameras und die Monitore dieselben optischen Achsen. Das ermöglicht dem Betrachter einen natürlichen optischen Eindruck und reduziert das Unbehagen (Balanceprobleme, etc.), welches bei nicht übereinstimmenden Achsen auftritt.

Bild 1.6: Canon VH-2002 Video-See-Through Head Mounted Display aus [Tam02].

Beim Optical-See-Through HMD erfolgt die Mischung der Bilder über halbtransparente Spiegel vor dem Auge des Benutzers (siehe Abbildung 1.7). Diese zum Teil durchlässigen Spiegel gestatten dem Betrachter einen Blick auf die reale Szene.

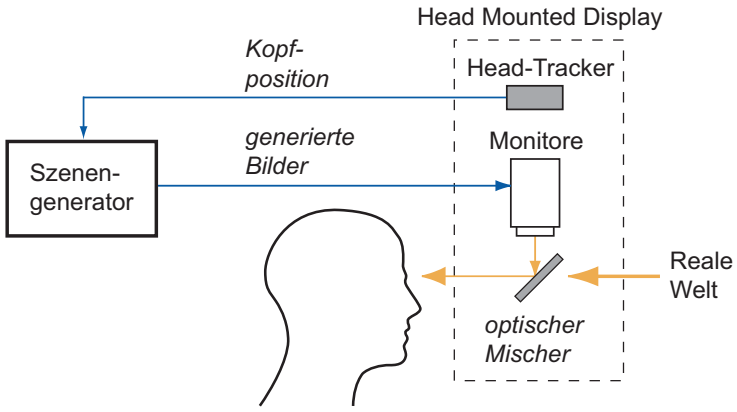


Bild 1.7: Konzept des Optical-See-Through Head Mounted Display nach [Azu97].

Gleichzeitig reflektieren sie aber auch die Bilder der im HMD befindlichen Monitore in die Augen. Die halbtransparenten Spiegel verringern normalerweise den Anteil des Lichtes, welches der Anwender aus der realen Szene sieht. Dies ist erforderlich, damit auch das Bild der Monitore wahrgenommen werden kann.

Abbildung 1.8 zeigt zwei optische See-Through HMDs. Das SONY GLASTRON wird für AR sehr gerne verwendet. Leider wird die hochauflösende Version nicht mehr erzeugt.

(a) Sony Glastron. (b) Virtual Vision VCAP.

Bild 1.8: Zwei Optical-See-Through Head Mounted Displays aus [Sch03].

Es ist nicht generell möglich einer der beiden HMD-Varianten den Vorzug zu geben. Vielmehr hängt es von den Parametern der Anwendung ab, für welchen Typ man sich entscheiden sollte. Allgemein ergeben sich für See-Through HMDs folgende Eigenschaften:

Video-See-Through HMD:
- Überdeckung von realem und virtuellem Bild möglich
- Verminderung der Bildqualität der realen Szene, aber mehr Flexibilität um z.B. Zeitverzögerungen auszugleichen
- bei technischen Problemen kann der Anwender unter Umständen nichts mehr sehen

Optical-See-Through HMD:
- Einfacher und daher auch billiger
- Direkte Sicht auf die reale Szene und daher keine Zeitverzögerung und weniger Verzerrung für das reale Bild
- es ist keine gegenseitige Überdeckung möglich

Monitor basierende Augmented Reality

Es ist auch ein Monitor basiertes AR-System möglich. Dabei trägt der Benutzer die Monitore oder optischen Mischer nicht am Kopf. Typischerweise ist das Anzeigesystem an einer festen Position im Raum installiert und der Benutzer betrachtet aus einer fixen Position den Monitor. Abbildung 1.9 zeigt den konzeptionellen Aufbau eines solchen Systems. Die Bilder einer oder mehrerer Videokameras, welche eine reale Szene aufnehmen, werden mit dem vom Computer generierten Material kombiniert und auf einem Monitor ausgegeben. Eine in [DGM+93] vorgestellte Anwendung ist

ARGOS (Augmented Reality through Graphic Overlays on Stereovideo) und ist in Abbildung 1.10 zu sehen.

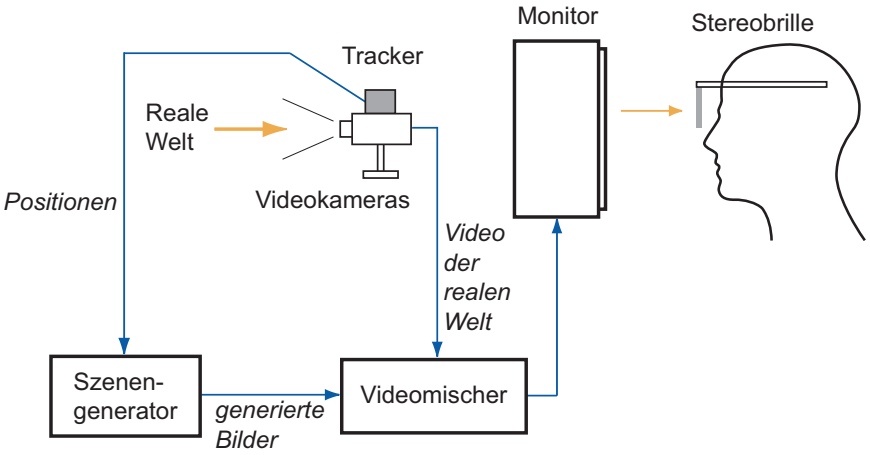
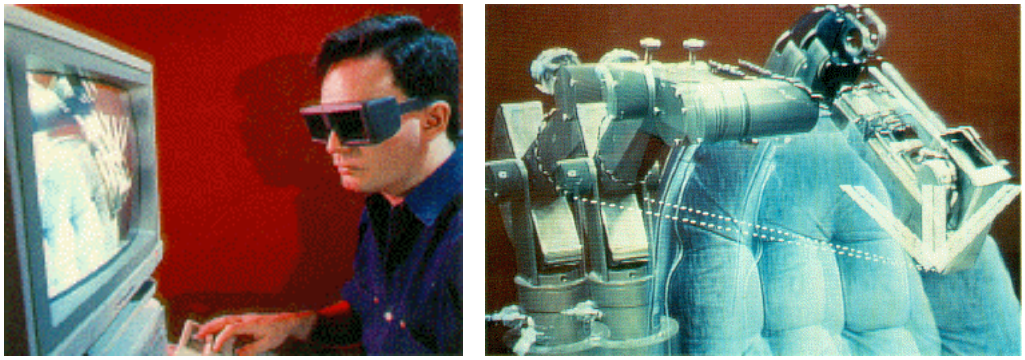


Bild 1.9: Konzept der Monitor basierenden Augmented Reality nach [Azu97].



(a) Externe Sicht des Anwenders. (b) Am Monitor angezeigtes Stereobild.

Bild 1.10: ARGOS, Monitor basierendes AR-System aus [DGM+93].

Projektor basierende Augmented Reality

Bei Projektor basierter AR wird der computergenerierte Bildinhalt mit Hilfe eines oder mehrerer Videoprojektoren auf reale Objekte abgebildet. Die realen Objekte besitzen dafür entweder eine retroreflektierende Oberfläche oder haben eine helle (weiße) Farbe. Abbildung 1.11 zeigt den konzeptionellen Aufbau eines Projektor basierten AR-Systems. In Bild 1.12 ist ein typischer Aufbau zu sehen. In der Mitte ist das reale Gebäudemodell mit heller Oberfläche zu sehen. Dieses reflektiert die virtuellen Bilder, welche die Videoprojektoren wiedergeben. Bei Bestimmung der Kopfposition des Benutzers kann der Bildinhalt entsprechend angepaßt werden. Weiterführende Information findet sich in [RL01].

Registrierung und Tracking

Für ein gut funktionierendes Augmented Reality-System ist es wichtig, daß die Kalibrierung der Positions- und Orientierungs-Sensoren möglichst genau ist. Ist dies nicht der Fall, gibt es eine Abweichung zwischen realer und virtueller Welt.

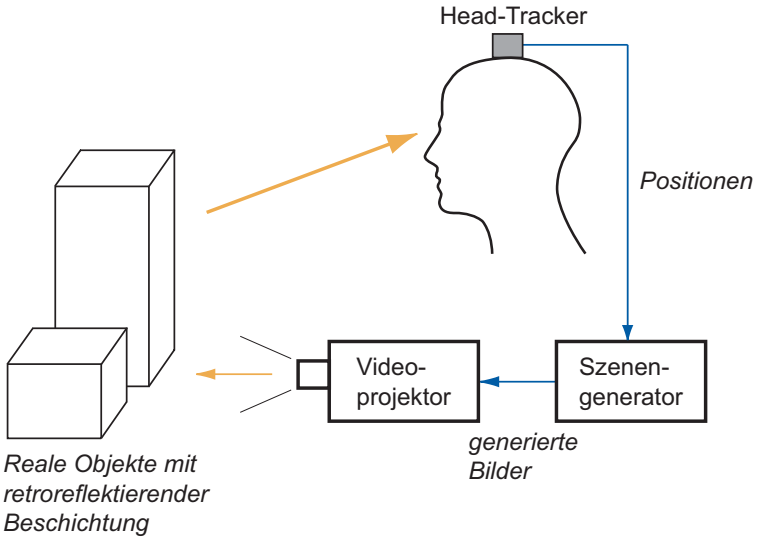


Bild 1.11: Konzept der Projektor basierenden Augmented Reality nach [Sch03].

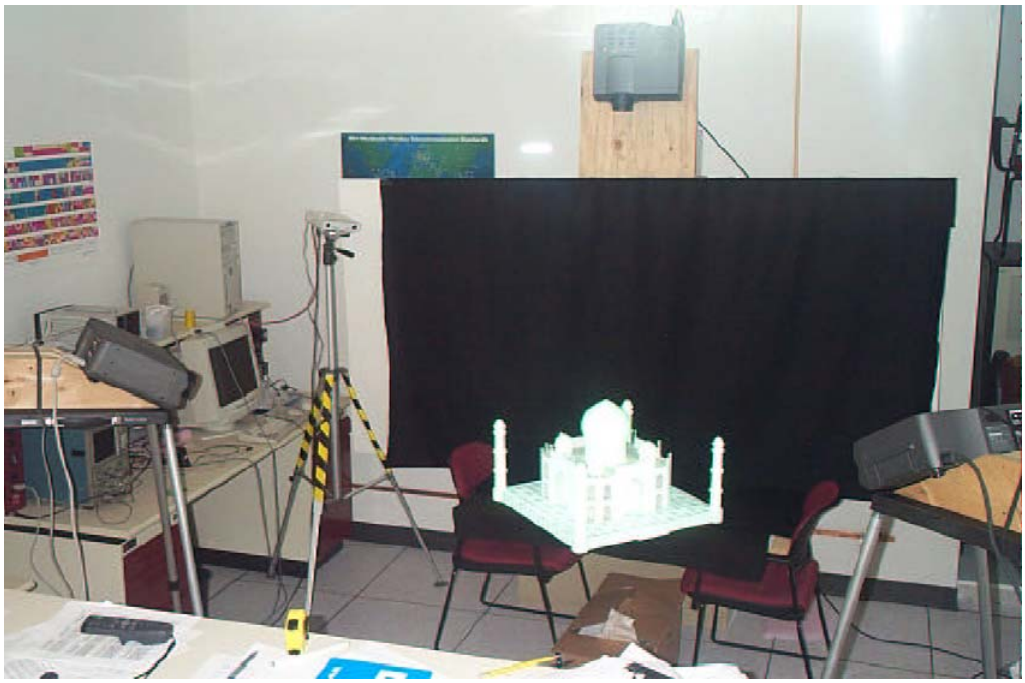


Bild 1.12: Ein Modell des Taj Mahal wird mit mehreren Videoprojektoren beleuchtet. Der optische Kopf-Tracker steht auf dem Stativ [RL01].

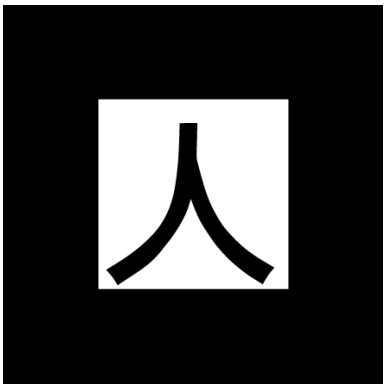
Unter Tracking versteht man das Messen von Körpern, welche sich im Raum bewegen. Meßgrößen sind dabei z.B. die Ortskoordinaten und/oder die Orientierungsdaten (Winkel) des Körpers. Werden gleichzeitig die Ortskoordinaten (drei Zahlenwerte) und die Orientierungswinkel (drei Zahlenwerte) für einen Körper gemessen, so spricht man von einer „6 Freiheitsgrade“-Messung (engl. degree of freedom).

Es gibt verschiedene Techniken, die in Trackingsystemen eine Verwendung finden. Dazu gehören z.B. magnetische Systeme, optische Systeme im visuellen oder Infrarotbereich, mechanische Wegaufnehmer, Ultraschallsysteme und auf Beschleunigungs- und Gyroskop-Sensoren aufbauende Tracker. Von den „berüh-

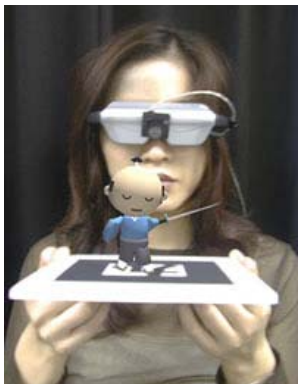
rungslosen“ Trackingsystemen bieten derzeit die optischen Tracker die höchste Genauigkeit und sind auch sehr robust in Bezug auf Interferenzen.

Eine weitere Möglichkeit ist das Tracking mit Hilfe von Videokameras. Beim Marker Tracking (engl. fiducial tracking) werden die Marker direkt am zu verfolgenden Objekt angebracht. Die Position und Orientierung des Objektes (Marker) werden in Echtzeit berechnet. Die Marker können unterschiedliches Aussehen haben, z.B. aus Strichcodes bestehen [RN95], aus verschiedenfarbigen Kreisen, die skalierbar sind [NYC+99] oder ein beliebiges Muster aufweisen wie beim ARTOOLKIT [KB99].

Das ARTOOLKIT verwendet zur Positionsbestimmung Passpunktmarker. Dies sind quadratische Marker von bekannter Größe. Zur Identifizierung dienen beliebige Namen oder Bilder im Marker (Abbildung 1.13). Bild (a) zeigt einen der verwendeten Marker. In Bild (b) wird mit Hilfe eines AR-Systems dem Marker ein generiertes Objekt zugeordnet.



(a) Marker mit Kanji Zeichen.



(b) Marker mit überlagerter Geometrie.

Bild 1.13: ARToolkit des Human Interface Technology Laboratory, University of Washington.

Von einem AR-System wird auch verlangt, daß der Benutzer in Echtzeit mit dem System interagieren kann. Dafür müssen vom bildgebenden Rechnersystem genügend Bilder pro Sekunde errechnet und vom Trackingsystem ausreichend Daten zur Verfügung gestellt werden. Geschieht dies zu langsam, erhält der Betrachter „springende“ Bilder oder es stimmt die Registrierung zwischen virtueller und realer Welt nicht überein. Als Untergrenze sollte 10 Bilder pro Sekunde angenommen werden. Mit der heute zur Verfügung stehenden Rechner- und Graphikleistung ist dieser Wert ohne weiteres zu erreichen.

Damit virtuelle Objekte für den Benutzer zu einem Teil der realen Szene werden, müssen sie einen gewissen Grad an Fotorealismus aufweisen (z.B. Texturierung). Eine weitere Möglichkeit, weshalb es zu Problemen mit der Registrierung zwischen realer und künstlicher Welt kommt, ist die im Gesamtsystem vorhandene Verzögerung. Diese muß in und zwischen den Einzelsystemen, wie z.B. Videokamera, Positionssensoren, etc., möglichst klein gehalten werden und untereinander abgestimmt sein. So kann es Sinn machen, das von einer Videokamera aufgenommene Realbild zu verzögern und erst zu einem späteren Zeitpunkt, wenn das dazu passende virtuelle Bild errechnet ist, diese miteinander zu kombinieren und auszugeben. Eine ausführliche Abhandlung zu diesen Themen findet sich in [Azu97].

1.4.3 Anwendungsgebiete

Der Einsatz von Augmented Reality kann in einigen Anwendungsbereichen die Effizienz steigern oder neue Möglichkeiten der Interaktion schaffen. AZUMA führt in [Azu97] folgende Anwendungsgebiete, beim Forschungsstand von 1997, an:

1. Medizinische Visualisierung
2. Fertigung, Wartung und Instandsetzung
3. Kommentieren und Visualisieren
4. Robotics und Telerobotics
5. Unterhaltung
6. Militärisches Training

Darüber hinaus sind die Bereiche industrielles Design, Produktdesign und computerunterstütztes kooperatives Arbeiten (engl. computer supported cooperative work, CSCW) weitere Anwendungsgebiete von Augmented Reality-Systemen. Im Folgenden werden einige Forschungsprojekte exemplarisch beschrieben.

Im Rahmen des „Virtual Showcase“-Projektes wurde für den Fachbereich Paläontologie eine Anwendung realisiert. Der reale Schädelknochen eines Sauriers wird mit zusätzlicher Information kombiniert (Abbildung 1.14). Es können u.a. einzelne Muskelgruppen im Bereich des Schädels und auch die Hautoberfläche visualisiert werden.

(a) Realer Schädel eines Deinonychus im Inneren der Vitrine.

(b) Verschiedene Muskelgruppen werden im realen Schädel angezeigt.

Bild 1.14: Visualisierung von Muskelsträngen in einem Saurierschädel mit dem Virtual Showcase aus [BGW+02].

Dem Betrachter des Ausstellungsstückes wird durch die Kombination von im Computer generierten Bildern und dem realem Fossil zusätzliche Information zugänglich gemacht. Abbildung 1.15 zeigt einen Kiosk, bei dem das Ausstellungsstück zwei Betrachtern zur gleichen Zeit präsentiert werden kann. Bei dieser Vitrine wird das auf den Monitoren angezeigte Bild mit Hilfe von halbtransparenten Spiegeln mit dem Bild des realen Objektes kombiniert. Der Betrachter trägt Stereobrillen (hier LCD-Shutter Glasses), um einen räumlichen Eindruck zu erhalten. Die Kopfposition der Betrachter wird über optische Tracker erfaßt. Details dazu finden sich in [BGW+02] [BES03].

Das zweite Anwendungsbeispiel ist aus dem Bereich der Lehre. CONSTRUCT3D [KS03] ist ein dreidimensionales Geometriekonstruktionswerkzeug, welches speziell für den Mathematik- und Geometrie-Unterricht entwickelt wurde. Es basiert auf dem

Bild 1.15: Monitor basierender Virtual Showcase Prototyp mit Saurierschädel aus [BES03].

Augmented Reality System STUDIERSTUBE [SFH+00] (siehe "Studierstube" auf Seite 38). CONSTRUCT3D ist ein Mehrbenutzer-AR-System und unterstützt verschiedene Lehrer-Schüler-Szenarien (Abbildung 1.16). Das System ist einfach zu bedienen, fördert das Experimentieren mit geometrischen Strukturen und erhöht die räumliche Vorstellungskraft. Durch die Verwendung von See-Through HMDs können sich die interagierenden Personen sehen. Das Tracking der HMDs und der Interaktionswerkzeuge erfolgt mit magnetischen Sensoren.

(a) Ein Student fügt eine Kugel in einen Kegel ein.

(b) Ein Tutor hilft dem Studenten beim Lösen eines Problems.

Bild 1.16: Gemeinsames Arbeiten mit dem Augmented Reality Programm Construct3D aus [KS03].

In Kooperation zwischen DENSO Corporation, Japan und MR Systems Laboratory, CANON, Inc. entstand eine AR-Applikation zur Evaluierung von Design-Entwürfen im Automobilbau [Tam02]. DENSO Corp. erzeugt für die Automobilindustrie Einzelteile für den Fahrzeugbau. Für die Überprüfung der Funktionen und des Designs von Fahr-

zeugarmaturenbrettern benötigte die Design-Abteilung von DENSO eine Applikation, welche dies bereits im frühen Entwurfsstadium ermöglicht.

(a) Ohne Augmentation. (b) Mit Augmentation.

Bild 1.17: Design-Evaluation des Armaturenbrettes eines Personenwagens aus [Tam02].

Abbildung 1.17 (a) zeigt den Simulator des AR-Systems. Der Fahrersitz, das Lenkrad und die beiden Bedieneinheiten (für Audio, Klimaanlage und das Navigationssystem) in der Mittelkonsole sind reale Objekte. Das Instrumentenpanel in der Fahrerkabine ist nur eine Rahmenkonstruktion. Durch ein See-Through HMD wird das Armaturenbrett (Abbildung 1.17 (b)) für den Fahrer mit den realen Objekten kombiniert. Es können so die Ergonomie der Instrumentenanordnung und deren Funktion getestet werden. Auch hier ist es wichtig, daß die Registrierung von realen Objekten mit den virtuellen Daten sehr gut übereinstimmt.

Kapitel 2

Konzept und Aufgabenstellung

> *„Aber, wird man fragen, eine leblose Figur, die größtentheils ganz mechanisch spielt, gegen einen denkenden ganz frey handelnden Gegenspieler? - Freyhandelnden? Gewiß nicht so frey, als man sich wohl vorstellt, wenn man die Sache nicht in der Nähe betrachtet. Ich behaupte vielmehr, so paradox das auch anfänglich scheinen mag: Der Gegenspieler handle bey seiner Vertheidigung desto weniger frey, je besser er spielt.“*
>
> – Karl Friedrich Hindenburg, aus dem Buch *Ueber den Schachspieler des Herrn von Kempelen. Nebst einer Abbildung und Beschreibung seiner Sprachmaschine*, 1784

2.1 Aufgabenstellung im „Virtual Showcase“-Projekt

Der folgende Absatz beschreibt die generelle Aufgabenstellung für den „Türkischen Schachspieler“ im Rahmen des „Virtual Showcase“-Projektes.

Für das Technische Museum Wien soll eine moderne und zeitgemäße Reproduktion des historischen Automaten entwickelt werden. Den Besuchern soll es möglich sein, gegen den Automaten Schach zu spielen und Einblicke in die Funktionsweise des historischen Mechanismus zu bekommen. Weiters soll diese Installation die Haltung der heutigen Gesellschaft in Bezug auf die Verwendung von Technologie widerspiegeln, so wie dies auch beim geschichtlich überlieferten Schachspieler der Fall war. Auch soll es möglich sein, das neue Exponat in anderen Städten auszustellen, und so die über achtzig Jahre andauernde Reisetätigkeit der historischen Attraktion fortzuführen.

2.2 Vorhandenes Konzept

Eine ausführliche Beschreibung des ursprünglichen Konzeptes für den „Türkischen Schachspieler“ im Rahmen des „Virtual Showcase“-Projektes findet sich in [MVR+02]. Im Folgenden findet sich eine Zusammenfassung, die sich auf dieses Dokument stützt.

2.2.1 Präsentation des Exponates

Der Besucher nimmt den „Türkischen Schachspieler“ als großen Tisch mit einem auf ihm angebrachten Spiegel und diversen Hilfsaufbauten für eine Projektionsfläche wahr. Nachdem der Betrachter eine in antikem Stil gehaltene Brille aufgesetzt hat, sieht er einen dreidimensionalen „türkischen“ Mann. Dieser sitzt hinter dem Tisch und ladet ihn zu einer Partie Schach ein. Zwischen Tischplatte und Spiegel befindet sich eine Öffnung, in welche der Betrachter seine Hand stecken kann. Durch die Bewegung der Hand in dieser Öffnung wird eine virtuelle Hand gesteuert, die sich über einem virtuellen Schachfeld befindet. Auf einfache Weise kann der Besucher dadurch die gegenstandslosen Schachfiguren bewegen und seine Züge machen. Andere Besucher können die Installation während des Spieles von der Seite betrachten, wo die Szene aus einer entsprechenden Sicht gezeigt wird.

Nach einigen Zügen werden der mechanische Schachspieler, das Schachbrett, die Figuren und der Tisch transparent. Die mechanischen Vorrichtungen innerhalb der Figur und des Tisches, wie auch der im Tisch versteckte menschliche Schachspieler werden sichtbar. Schließlich löst sich die gesamte Szene in Zahlen und Zeichen auf. Abbildung 2.1 zeigt einen konzeptionellen Entwurf des Automaten.

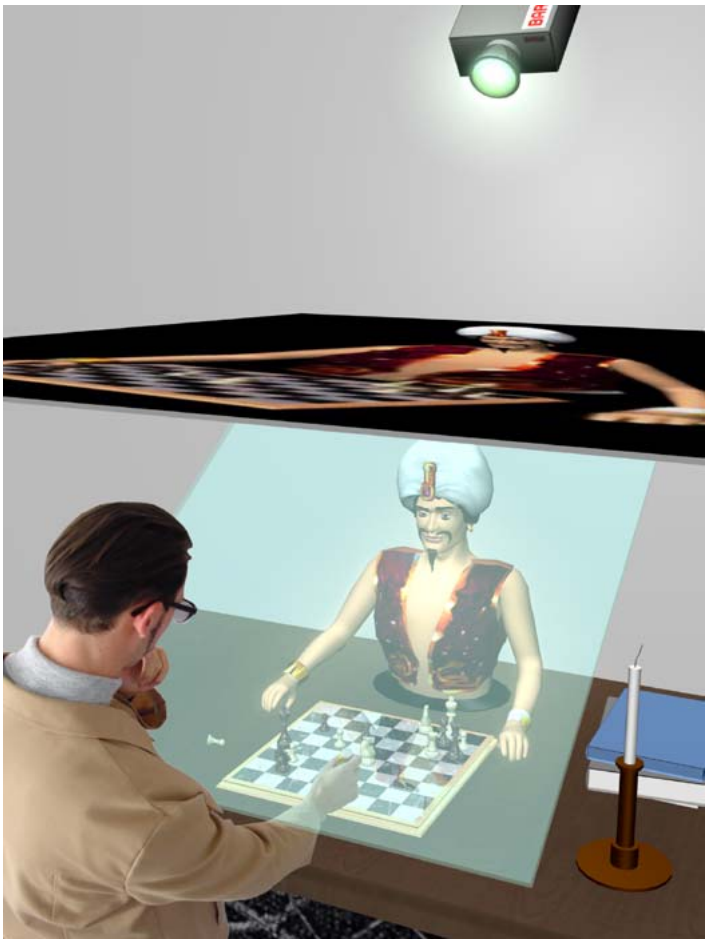


Bild 2.1: Entwurf des „modernen“ Schachspielers [MVR+02].

2.2.2 Technische Beschreibung

Die Installation muß zwei Aufgaben erfüllen. Es kann nur jeweils ein Besucher die aktive Rolle übernehmen und gegen den Automaten eine Partie Schach spielen. Dieser Besucher soll einen hohen Grad an Interaktion und ein überzeugendes 3D-Bild erleben. Den anderen Besuchern, die auf die Möglichkeit warten, mit dem Schachspieler zu spielen oder einfach nur dem Geschehen zusehen, muß auch ein Mittel zur Unterhaltung geboten werden. Basierend auf diesen zwei Notwendigkeiten gibt es für beide Kategorien von Besuchern entsprechende Lösungen.

2.2.2.1 Projektion für den Spieler

Für den interagierenden Besucher wurde die Verwendung eines geneigten halbtransparenten Spiegels (siehe Abbildung 2.1) vorgeschlagen. Die Vorderseite des Spiegels schließt nicht mit der Tischplatte vollständig ab, sodaß der menschliche Spieler mit seiner Hand unter den Spiegel greifen kann. Die geneigte Ebene des Spiegels approximiert die projizierte 3D-Szene am besten. Wenn die geneigte Ebene als Reflektor ausgeführt und eine horizontale Projektionsfläche verwendet wird, dann liegt die Schärfenebene des visualisierten 3D-Inhaltes in etwa dort, wo er physikalisch sein sollte. Auch kann sich der Besucher über den Tisch lehnen, ohne mit dem Kopf gegen eine Glasscheibe zu stoßen. Der spielende Besucher sieht den Schachspieler, das Brett und die Figuren als stereoskopische Projektion. Für die Interaktionen des Spielers ist es wichtig, daß der reale und virtuelle Inhalt der Szene aufeinander abgestimmt sind. Aus diesem Grund wird für diesen Besucher Head-Tracking verwendet.

In der Projektion ist neben dem virtuellen türkischen Schachspieler auch das Schachbrett, die Schachfiguren und die virtuelle Hand, welche die Hand des menschlichen Spielers im virtuellen Schachbrett repräsentiert, zu sehen. Ein sitzender Besucher ist einem stehenden Besucher vorzuziehen, da dadurch Größenunterschiede (z.B. Kind, Erwachsener) besser ausgeglichen werden können.

2.2.2.2 Projektion für die Zuschauer

Für die anderen Zuschauer, welche dem aktiven Spieler nur zusehen wollen, wird eine zweite nicht transparente Projektionsfläche das Spiel von der Seite zeigen. Die Seitenszene wird im Großen und Ganzen den gleichen Inhalt wie die Projektion für den aktiven Spieler darstellen. Der Blickpunkt wird für einen durchschnittlichen Besucher, der seitlich am Tisch steht, berechnet. Für einen Betrachter, welcher nicht mit der Szene interagiert, ist ein exakter Blickpunkt nicht unbedingt notwendig. Die Seitenprojektion dient auch dazu, vorbeigehende Besucher auf das Exponat aufmerksam zu machen.

Weiters kann angenommen werden, daß noch immer viele Personen mit Virtual Reality-Installationen nicht vertraut sind und daher lieber passiv das Geschehen verfolgen. Mit dem vorliegenden Konzept kann auch dieser Personengruppe die Geschichte des Schachspielers näher gebracht werden.

2.2.3 Interaktionsmöglichkeiten des Benutzers

Die intuitivste Möglichkeit für den Benutzer Schach zu spielen ist mit Hilfe seiner eigenen Hand. Neben den Wechselwirkungen zwischen den realen Elementen der Szene (z.B. dem greifbaren Tisch) und den virtuellen Objekten gibt es eine Wechselwirkung zwischen dem Körper des Betrachters und den Teilen des virtuellen Inhalts. Dieser Sachverhalt wirft einige Fragestellungen auf, welche im folgenden Abschnitt behandelt werden.

2.2.3.1 Virtuelle Schachfiguren

Die Schachfiguren am Spielfeld müssen sowohl vom virtuellen Schachspieler als auch vom realen Besucher verwendet werden können. Sie stellen die Verbindung zwischen der realen und virtuellen Welt dar. Die Verwendung von realen Schachfiguren wäre eine Möglichkeit, die Züge des „Schachtürken“ mittels Magnetismus durchzuführen. Der menschlichen Spieler könnte die materiellen Figuren auf natürliche Weise bewegen. Eine weitere Variante wäre, wenn der Mensch mit realen Figuren und der Automat mit virtuellen Schachfiguren spielen würde. In beiden Fällen gibt es aber ein Problem, wenn eine Figur im Zuge des Spiels geschlagen wird.

Außerdem ist die magnetische Bewegung der Schachfiguren eine Abweichung zur historischen Vorlage. Dort hob der Automat die Schachfiguren mit seiner Hand vom Feld, bewegte sie zum neuen Feld und setzte sie darauf ab. Ein weiteres Problem ist, daß die realen Schachfiguren in einem Museum leicht abhanden kommen.

Diese und weitere Überlegungen führten zur Entscheidung, die Schachfiguren als virtuelle Objekte auszuführen.

2.2.3.2 Die Interaktionsöffnung

Im Vorderbereich des Spiegels befindet sich eine Öffnung, welche wir künftig als „Interaktionsöffnung“ (engl. interaction hole) bezeichnen werden. Der Besucher kann gegen den „Türkischen Schachspieler“ spielen, indem er die nötigen Interaktionen mit seiner Hand innerhalb dieser Öffnung ausübt.

Innerhalb des virtuellen Schachbrettes werden die horizontalen Bewegungen (links-rechts, vor-zurück) der Spielerhand durch eine korrespondierende virtuelle Hand, ähnlich einem zweidimensionalen Bildschirmcursor, dargestellt. Um dies zu ermöglichen, muß die Hand des Besuchers in geeigneter Weise verfolgt werden.

2.2.3.3 Verfolgung der Hand des Spielers

Nachfolgend werden einige Möglichkeiten der Interaktion des Besuchers im Rahmen des Schachspiels aufgezählt.

Optisches Tracking

Für den oben beschriebenen Anwendungsfall kann die Hand des Besuchers über eine Videokamera verfolgt und ihre Position bestimmt werden. Die Kamera wird innerhalb der Vitrine montiert. Da der Bereich des virtuellen Schachbrettes nicht beleuchtet

wird, ist es notwendig, eine Infrarotbeleuchtung vorzusehen, um die Bildverarbeitung zu ermöglichen. Der statische Hintergrund der Tischoberfläche, vor dem sich die Hand des menschlichen Spielers befindet, erleichtert die Bildanalyse.

Ein kritischer Punkt ist das Erkennen, ob der Spieler eine Schachfigur aufnehmen oder absetzen will. Es ist logisch, daß dem Aufnehmen einer Schachfigur auch das Absetzen und umgekehrt folgt. Es kann also für das Aufnehmen und Absetzen einer Figur der gleiche Erkennungsalgorithmus angewendet werden.

Ein Weg dies zu erreichen ist die Auswertung der Silhouette der Hand. Die Geste für das Nehmen einer Figur, wenn Daumen und Zeigefinger nahe beieinander sind, ist unterschiedlich zum Umriß einer offenen Hand.

Wenn die Videokamera oberhalb der Hand angebracht ist, muß die Orientierung der Hand so sein, daß die relativen Bewegungen zwischen Daumen und Zeigefinger aus dem Blickwinkel der Kamera erfaßbar sind. Die virtuelle Hand in der Szene wird die geeignete Orientierung veranschaulichen und die Nachahmung durch den Besucher garantiert die gewünschte Interaktion.

Die Hauptvorteile dieses Lösungsansatzes sind, daß er berührungslos arbeitet und die Gesten für das Ergreifen und Loslassen einer Figur bekannt und intuitiv sind.

Touch Pad

Der Einsatz eines Touch Pad, das in die Tischoberfläche eingelassen ist, bietet eine einfache und kostengünstige Alternative für die Interaktion des Besuchers. Durch die Berührung des Touch Pad mit der Hand legt der Besucher das Start- und Endfeld für einen Schachzug fest.

Diese Lösung ist nicht so intuitiv wie das optische Tracking, aber das Erkennen des Aufnehmens und Abstellens einer Schachfigur gestaltet sich wesentlich einfacher. Ein Nachteil dieser Methode ist, daß der Benutzer keine Rückmeldung in der virtuellen Szene bekommt, solange er das Touch Pad nicht mit seinen Fingern berührt. Falls der Besucher seine reale Hand am Touch Pad nicht sehen kann, hat er Schwierigkeiten, die mit seinen Fingern korrespondierende Position einer virtuellen Schachfigur zu bestimmen.

Hybride Lösung

Eine weitere Lösungsmöglichkeit ist die Kombination der zwei bereits erwähnten Technologien. Die horizontale Bewegung der Spielerhand wird optisch über die Videokamera erfaßt. Das Ergreifen und Loslassen einer virtuellen Figur wird über das Touch Pad ausgeführt. Die Rückmeldung der Interaktion erfolgt wieder über die virtuelle Hand des Spielers in der Szene.

Eine Schwierigkeit bei dieser Lösung besteht darin, dem Besucher die Funktion des Touch Pad mitzuteilen. Hier kann eine visuelle Unterstützung durch die Änderung der Farbe des Schachfeldes und der Schachfigur und/oder eine Änderung in der Geometrie erfolgen.

2.2.4 Bewegungen der virtuellen Hand

Das Ziehen einer virtuellen Schachfigur durch den Besucher wird über dessen virtuelle Hand in der Szene dargestellt. Die Schachfigur wird von der virtuellen Hand angehoben und folgt den Bewegungen der realen Hand bis zu dem Feld, auf dem sie abgesetzt wird. Da nur die horizontalen Bewegungen einen Einfluß auf diese Operation haben, ist es nicht notwendig, die vertikale Position der Spielerhand zu bestimmen.

Die zu bewegende Schachfigur und das Zielfeld auf dem Schachbrett sind durch die horizontale Position der virtuellen Hand, welche mit der horizontalen Position der realen Hand korrespondiert, bestimmt. Ist eine bestimmte Schachfigur oder ein bestimmtes Feld durch die horizontale Position der Hand des Besuchers und seine „Ergreifen/Loslassen"-Interaktion bestimmt, wird das Aufnehmen, Bewegen und Absetzen der Figur durch die virtuelle Hand mit Hilfe des Animationssystems ausgeführt.

Da die vertikale Position der realen Hand, wie bereits oben erwähnt, nicht ausgewertet wird, gibt es auch keine Verdeckung zwischen virtueller Hand und den virtuellen Schachfiguren.

2.2.5 Überlagerung der realen und virtuellen Objekte

Reale Hand und virtuelle Schachszene

Für eine glaubwürdige Überlagerung der realen Hand mit der virtuellen Szene des Schachspiels ist es notwendig, die dreidimensionale Geometrie der Hand des Spielers zu ermitteln und im AR-System zu verarbeiten. Da die Lösung dieser Aufgabe relativ komplex wäre, wird im Bereich des virtuellen Schachbrettes die Hand des Besuchers durch eine virtuelle Hand repräsentiert. Außerhalb des Schachfeldes sieht der Besucher seine reale Hand. Dieser Umstand hilft dem Besucher auch, seine Scheu zu überwinden und mit seiner Hand in die Interaktionsöffnung zu greifen.

Auch hier ist die Genauigkeit der Positionsbestimmung der Hand sehr wichtig, um den „Abstand" zwischen realer und künstlicher Welt möglichst klein zu halten.

Realer Tisch und virtueller Schachspieler

Ein weiterer Aspekt ist das Zusammenspiel zwischen dem realen Tisch und dem virtuellen Schachspieler. Es ist wichtig, daß der Betrachter den „Türkischen Schachspieler" als hinter dem Tisch sitzend wahrnimmt. Weiters legt die virtuelle Figur des Schachspielers beide Arme auf der Tischoberfläche ab.

2.2.6 Beleuchtung

Die Vitrine des „Türkischen Schachspielers" wird derart beleuchtet, daß der Bereich des realen Tisches, in dem sich das virtuelle Schachbrett und die darauf stehenden Schachfiguren befinden, im Dunkeln liegt. Dadurch wird gewährleistet, daß der virtuelle Inhalt genügend hell erscheint, und im Bereich des Schachbrettes die virtuelle Hand des Besuchers und nicht seine reale Hand zu sehen ist. Die Bereiche neben dem

Schachbrett und der Vordergrund sind beleuchtet. Der Hintergrund, vor dem der virtuelle Schachspieler erscheint, ist hingegen unbeleuchtet und dunkel.

2.3 Vorhandener Prototyp

Als Teil des „Virtual Showcase“-Projektes wurde ein erster Prototyp am Institut für Softwaretechnik und Interaktive Systeme, Arbeitsgruppe für Interaktive Multimediale Systeme, an der Technischen Universität Wien gebaut. Eine ausführliche Beschreibung findet sich in [MPB02].

2.3.1 Technische Beschreibung

2.3.1.1 Projektion

Ein DLP-Projektor (INFOCUS Lp 350), welcher in Abbildung 2.2 links oben zu sehen ist, projiziert indirekt auf eine horizontal angebrachte Rückprojektionsscheibe. Ein Spiegel, der im Winkel von 45 Grad montiert ist, reflektiert den horizontalen Lichtstrahl des Videoprojektors auf die Rückprojektionsscheibe. Diese Projektionsscheibe besteht aus zwei Glasplatten, in deren Mitte sich ein dünner Film befindet. Die Rückprojektionsscheibe (52cm x 68cm) ist horizontal über einem halbtransparenten Spiegel befestigt. Der halbtransparente Spiegel hat einen Neigungswinkel von 33 Grad zur Horizontalen und reflektiert das Bild von der Rückprojektionsscheibe zum Betrachter. Über fünf Halogenlampen, die hinter dem Spiegel angebracht sind, wird die Szene beleuchtet. Die Abmaße des gesamten Prototypen sind: Länge=73cm, Breite=56cm, Höhe=175cm.

2.3.1.2 Stereo-Darstellung

Eine aktive Stereobrille ist mit der Videokarte des Computers synchronisiert und ermöglicht die Darstellung eines dreidimensionalen Bildes.

2.3.1.3 Tracking

Für das Tracking wird eine optische Methode verwendet. Eine über FIREWIRE an den Computer angeschlossene Videokamera, die am oberen Ende des Prototypen montiert ist, nimmt die Umgebung vor dem halbtransparenten Spiegel auf. An der Stereobrille ist ein optischer Marker (wie in Abbildung 1.13 (a)) befestigt. Die Position des Markers wird mit Hilfe der Software ARTOOLKIT [KB99] berechnet. Dies erlaubt die Ermittlung der Kopfposition des Betrachters.

2.3.1.4 Rechnersystem

Als Rechnersystem kommt ein handelsüblicher PC (FUJITSU-SIEMENS Celsius 600 Workstation, 512MB Hauptspeicher) mit MICROSOFT Windows 2000 als Betriebssystem zum Einsatz. Als Graphikkarte wird eine NVIDIA Quadro4 900/XGL verwendet. Die Visualisierung der Daten geschieht über das AR-System STUDIERSTUBE [SFH+00].

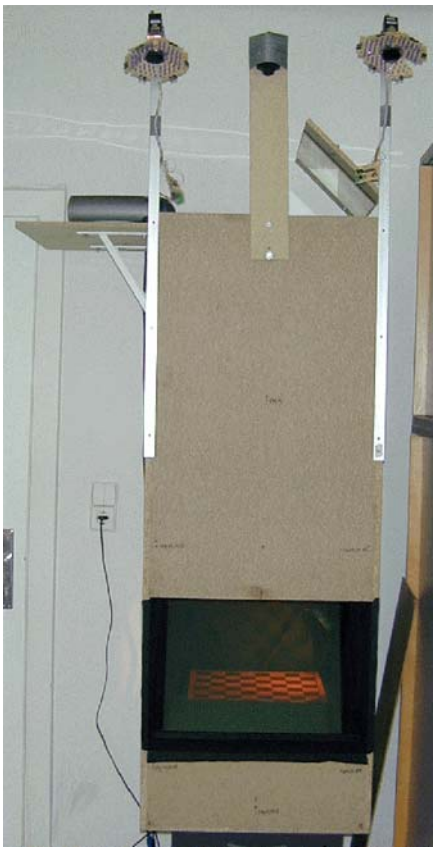


Bild 2.2: Virtual Showcase Prototyp für den „Türkischen Schachspieler“ aus [MPB02].

2.3.2 Interaktionsmöglichkeiten

Die Interaktion mit dem Prototypen ist auf zwei Arten möglich. Über ein Touch Pad, welches auf der Vorderseite des Kastens angebracht ist, und über einen „virtuellen Laserpointer“. Der „virtuelle Laserpointer“ besteht aus einem einfachen Stab mit zwei ARTOOLKIT-Markern. Aus den Markerpositionen kann die Richtung des virtuellen Laserstrahls ermittelt werden. Dieser Laserstrahl kann zur Interaktion mit dem Inhalt verwendet werden.

2.3.3 Ergebnisse

Dieser Prototyp stellte einen ersten Versuch dar, das Szenario für den „Türkischen Schachspieler“ zu realisieren. Er bietet wie im ursprünglichen Konzept vorgesehen (siehe “Vorhandenes Konzept” auf Seite 19) einem Benutzer die Möglichkeit zur Interaktion mit der dargestellten Szene. Weiters wird bereits die Kopfposition des Betrachters ermittelt und die virtuelle Szene entsprechend dargestellt.

Der Bereich hinter dem halbtransparenten Spiegel bietet Platz für ein reales Exponat und hat die Größe 50cm x 50cm. Der relativ flache Winkel des halbtransparenten Spiegels von 33 Grad zum Vitrinenboden limitiert die maximale Höhe des realen Exponats auf ca. 20cm. Im Fall des Prototypen ist das reale Exponat das Schachbrett.

Der Prototyp hat gezeigt, daß der gewünschte Aufbau möglich ist und die Qualität des Bildes auch in einer hellen Umgebung den Anforderungen entspricht. Weiters lie-

ferte der Prototyp wichtige Daten, was die Größe der Rückprojektionsfläche, des halbtransparenten Spiegels und die Länge des Strahlenganges anbelangt.

Ein Problem war die Synchronisierung des DLP-Videoprojektors mit der aktiven Stereobrille. Obwohl die LCD-Brille mit der Videokarte des bildgebenden Rechners synchron lief, konnte das vom DLP-Projektor erzeugte Stereobild nicht überzeugen.

2.4 Aufgabenstellung der Diplomarbeit

Die folgende Liste gibt einen Überblick über die Aufgabenstellung dieser Diplomarbeit.

- Für die Entwicklung der Software des „Türkischen Schachspielers“ soll ein bestehendes Augmented Reality-System verwendet werden. Es soll ermittelt werden, ob das AR-System STUDIERSTUBE die Anforderungen dieser Anwendung erfüllt.
- In das ausgewählte AR-System soll eine bestehende Handtracking-Software eingebunden werden. Die Software soll mit den von ihr benötigten Hardwarekomponenten auf einem Rechner installiert und getestet werden. Für die Kommunikation zwischen Handtracking-Software und dem AR-System sind gegebenenfalls die entsprechenden Änderungen in der Handtracking-Anwendung durchzuführen.
- Für die Bestimmung der Kopfposition des Benutzers soll ein geeignetes Gerät ausgewählt werden. Falls erforderlich, ist die Software für die Einbindung des Gerätes in das verwendete AR-System zu entwickeln.
- Die Konfiguration des Rechners, auf dem das AR-System mit der Applikation des „Türkischen Schachspielers“ installiert wird, ist festzulegen. Dies beinhaltet auch die Wahl des Betriebssystems und der benötigten Graphikkarte.
- Die Auswahl der Hardwarekomponenten für die stereoskopische Ausgabe der vom AR-System generierten Bilder soll in Zusammenarbeit mit der Firma BARCO erfolgen. Dazu sind die Prüfung und Evaluierung von verschiedenen Hardwareprototypen für die Videoprojektion erforderlich.
- Für die Stereo-Projektion sollen die verschiedenen Materialien und Produkte für den halbtransparenten Spiegel und die Rückprojektionsscheibe ermittelt und getestet werden.
- Für die Animation der Figur des „Türkischen Schachspielers“ soll aus einem anderen Projekt eine bereits existierende Softwarekomponente wieder verwendet werden. Der Einsatz im Rahmen dieses Projektes ist zu überprüfen und die nötigen Anpassungen sind durchzuführen.
- Ein Schachprogramm soll für die Berechnung und die Auswertung der Schachzüge in die Anwendung integriert werden.

Kapitel 3

Architektur des Schachspielers

> *Automat (griech.), im weitern Sinn jede sich selbst bewegende mechanische Vorrichtung, die durch im Innern verborgene Kraftmittel (Federn, Gewichte etc.) in Bewegung gesetzt wird, z. B. Uhren, Planetarien u. dgl.; im engern Sinn ein mechanisches Kunstwerk, welches vermittelst eines innern Mechanismus die Thätigkeit lebender Wesen, der Menschen (Android) oder Tiere, nachahmt und meist auch an Gestalt diesen nachgebildet ist.*
>
> *– Meyers Konversations-Lexikon. Eine Encyklopädie des allgemeinen Wissens.* Vierte, gänzlich umgearbeitete Auflage, Leipzig 1885

Einführung

Aufbauend auf das Konzept und dem vorhandenen Prototyp aus Kapitel 2 werden im Folgenden die verwendeten Werkzeuge, welche für die Entwicklung notwendig waren, beschrieben. Dabei unterteilt sich das Kapitel in vier Schwerpunkte. Im Abschnitt *Visualisierung* (siehe Kapitel 3.1) werden die Werkzeuge vorgestellt, die nötig sind, um die virtuelle Szene für den Betrachter darzustellen. Dazu gehören unter anderem das AR-System STUDIERSTUBE und die Hardwarekomponenten für die Videoprojektion. Der Textabschnitt über *Animation* (siehe Kapitel 3.2) beschreibt die Softwarekomponenten, welche für die Bewegungen des „Türkischen Schachspielers“ genutzt werden. Zu diesen Softwarekomponenten gehören die inverse Kinematik-Bibliothek IKAN und die OPEN INVENTOR-Erweiterung MESHDEFORMER. Im Kapitel 3.3 mit dem Titel *Interaktion* werden die für die Ermittlung der Positions- und Orientierungsdaten verwendeten Hard- und Software-Systeme beschrieben. Es wird gezeigt, wie in das OPENTRACKER-System der optische Tracker und die Handtracking-Software eingebunden werden. Der Abschnitt über die *Simulation* (siehe Kapitel 3.4) erläutert das benutzte Schachprogramm. In Abbildung 3.1 ist ein zusammenfassendes Systemdiagramm des „Türkischen Schachspielers“ zu sehen.

3.1 Visualisierung

Für die Realisierung des Software-Prototypen des „Türkischen Schachspielers“ wurde das AR-System STUDIERSTUBE gewählt. Ein Grund dafür war, daß zu Beginn der Softwareentwicklung noch nicht feststand, welche Ein- und Ausgabegeräte eingesetzt wer-

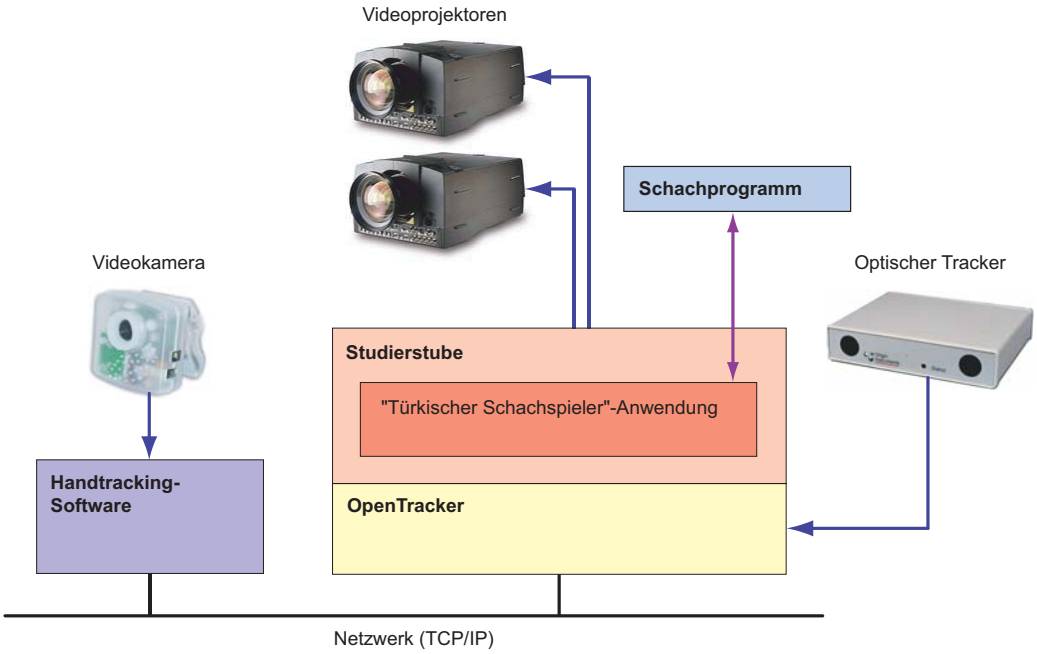


Bild 3.1: Systemdiagramm des „Türkischen Schachspielers“.

den. Die STUDIERSTUBE bietet die gewünschte Modularität und Flexibilität. STUDIERSTUBE basiert auf der Graphikumgebung OPEN INVENTOR. Eine Übersicht über die Architektur von OPEN INVENTOR bietet das Kapitel 3.1.1. Eine Beschreibung der STUDIERSTUBE folgt in Kapitel 3.1.2. Für die Evaluierung verschiedener Anzeigesysteme wurden zwei weitere Hardwareprototypen getestet. Diese werden in Kapitel 3.1.3 vorgestellt.

3.1.1 Open Inventor

3.1.1.1 Einführung

OPEN INVENTOR [SC92] wurde ursprünglich von SILICON GRAPHICS (SGI) entwickelt und ist ein weitverbreitetes Werkzeug, um interaktive 3D-Graphikapplikationen zu erstellen. Es handelt sich dabei um eine umfangreiche C++ Klassenbibliothek, welche die Entwicklung von objektorientierten Graphikprogrammen erlaubt. Der folgende Überblick über OPEN INVENTOR basiert auf [Wer93].

OPEN INVENTOR (OIV) stellt dem Programmierer verschiedene Bausteine zur Verfügung, um die mächtigen Fähigkeiten moderner Graphikhardware zu nutzen, ohne direkt in OPENGL programmieren zu müssen. OPEN INVENTOR basiert auf OPENGL und der Funktionsumfang kann den Anforderungen entsprechend modifiziert und erweitert werden.

Die Bibliothek beinhaltet database primitives wie shape, property, transformation, group und engine Objekte. Für die Interaktion gibt es manipulators wie z.B. trackball und handle box und weitere Komponenten wie Material- und Licht-Editoren. Um den Datenaustausch mit anderen Programmen zu gewährleisten, besitzt Inventor auch ein eigenes 3D-Dateiformat.

Die Benutzung von OPEN INVENTOR ist weitgehend unabhängig von der verwendeten Betriebssystemumgebung. Es gibt Bibliotheken für verschiedene Betriebssysteme und Hardwareplattformen. Seit dem Jahr 2000 ist OPEN INVENTOR unter einer Open Source-Lizenz verfügbar. Von SYSTEM IN MOTION gibt es seit Mitte 2001 eine unter der GNU General Public License stehende Version von OPEN INVENTOR mit dem Namen COIN. Sie ist unter anderem für folgende Betriebssysteme verfügbar: UNIX, Linux, verschiedene BSD-Varianten, MICROSOFT Win32-Plattform und Mac OS X.

Abbildung 3.2 zeigt die Softwarearchitektur von OPEN INVENTOR. Inventor basiert auf der Graphikbibliothek OPENGL und dem Betriebssystem. Der für den Softwareentwickler interessanteste Teil der Architektur ist die Szenendatenbank, welche die Szenengraphen verwaltet. Node Kits sind ein Konzept zur Strukturierung der Knoten innerhalb der Szenendatenbank. Manipulatoren sind graphische Komponenten die zur interaktiven Veränderung der Szene verwendet werden. Die Komponenten-Bibliothek (engl. component library) beinhaltet plattformspezifische Routinen für die Initialisierung, das Fenstermanagement und Rendering. Für einfache Programme, welche die erzeugte Graphik in ein Fenster ausgeben, ist dadurch eine einfache Portierung auf unterschiedliche Betriebssysteme möglich.

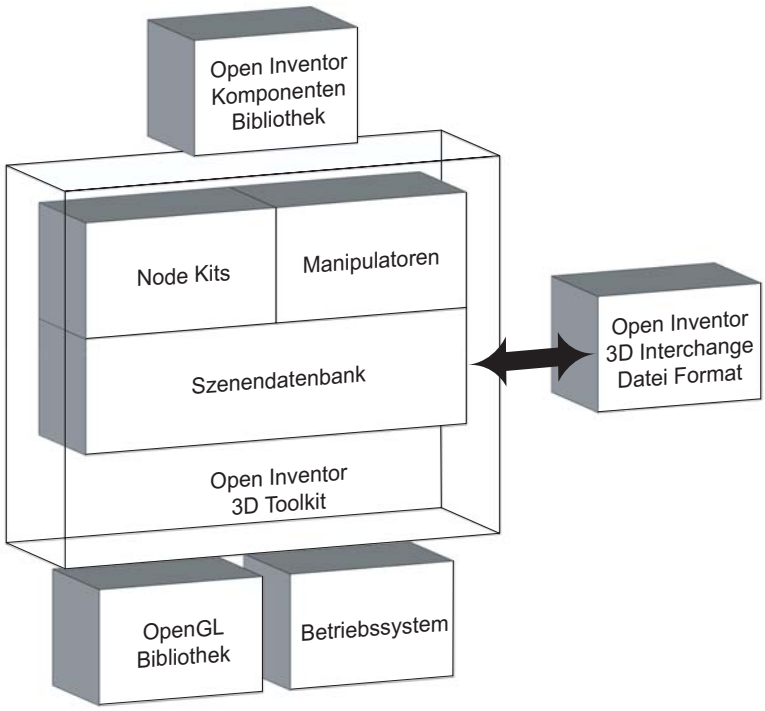


Bild 3.2: Inventor Architektur.

Durch den hohen Grad an Abstraktion erlaubt OPEN INVENTOR dem Entwickler sich auf die Kernbereiche seiner 3D-Applikation zu konzentrieren. Die Objekte einer Szene werden in der so genannten *Szenendatenbank* (engl. scene database) verwaltet. Auf diesen Objekten können verschiedene Aktionen ausgeführt werden. Dazu zählen z.B. das Berechnen der Bounding Box, das Bewegen eines Objektes mit der Maus oder das Zeichnen der 3D-Objekte. Die Ausgabe des *Szenengraphen* als Bild erfolgt über

OPENGL-Aufrufe. Während bei OPENGL das Rendering explizit erfolgt, beinhalten bei Inventor die Objekte die dafür notwendigen Methoden.

OPEN INVENTOR beinhaltet eine Vielzahl unterschiedlicher Klassen, welche einen großen Teil der Anforderungen erfüllen. Es ist auch möglich, den Funktionsumfang von OPEN INVENTOR mit neuen Klassen und Methoden zu erweitern. Dies wird in [Wer94] beschrieben.

3.1.1.2 Konzepte

OPEN INVENTOR verwendet 3D-Objekte und keine einfachen Geometrieelemente wie Punkte, Linien und Polygone, um eine Szene aufzubauen. Alle Informationen über diese Objekte, wie z.B. ihre Form, Größe, Oberflächentextur, Farbe und die Position und Orientierung im Raum, werden in der Szenendatenbank gespeichert. Die Informationen in der *Szenendatenbank* können auf verschiedene Art und Weise genutzt werden. Eine der häufigsten Verwendungen ist das Anzeigen der 3D-Objekte am Bildschirm.

OPEN INVENTOR wurde speziell für interaktive Anwendungen entwickelt und enthält daher neben Klassen, die Graphik-Primitiven entsprechen, auch noch Werkzeuge, die vor allem das interaktive Arbeiten mit 3D-Szenen erleichtern. Für den Programmierer bietet der Inventor-Toolkit folgende Werkzeuge:

- Eine *Szenendatenbank*. Diese beinhaltet shape, property, group, engine und sensor Objekte, welche zum Aufbau einer hierarchischen 3D-Szene verwendet werden.
- Eine Reihe von *Node Kits*. Diese sind ein komfortabler Mechanismus zum Bilden von vorgefertigten Gruppen von Inventor Knoten.
- Eine Menge von *Manipulatoren* (engl. manipulators). Dazu gehören u.a. die handle box und der trackball. Das sind Objekte in der Szenendatenbank mit denen der Benutzer direkt interagieren kann.
- Die *Inventor-Komponenten-Bibliothek* für die jeweilige Plattform. Diese beinhaltet u.a. das Fenster für die Bildschirmausgabe, den Material-Editor und Hilfsfunktionen, welche Dienste für höhere Interaktionsaufgaben zur Verfügung stellen.

3.1.1.3 Szenendatenbank

Eines der grundlegenden Konzepte in OPEN INVENTOR ist der *Szenengraph*. Darunter versteht man eine geordnete Sammlung von Knoten (engl. nodes). Die Knoten werden dabei in einer baumartigen Struktur angeordnet – genauer gesagt, in einem gerichteten azyklischen Graphen (engl. directed acyclic graph). Die *Szenendatenbank* kann einen oder mehrere Szenengraphen enthalten.

Der grundlegende Baustein in der Szenendatenbank ist ein *Knoten*. Ein Knoten wird als Teil des Szenengraphen von einem oder mehreren Knoten referenziert und kann selbst einen oder mehrere Knoten referenzieren. Jeder Knoten stellt eine bestimmte Information dar, z.B. das Oberflächenmaterial, eine Beschreibung der Form eines 3D-Körpers (Quader, Kugel, Kegel, etc.), eine geometrische Transformation, eine Lichtquelle oder eine Kamera. Alle 3D-Objekte, Attribute, Kameras und Licht-

quellen in einer Szene sind als Knoten repräsentiert. Auch interaktive Elemente (*Manipulatoren*) und einfache Animationen können mit Hilfe solcher Knoten abgebildet werden.

Ein Szenengraph besteht aus einem oder mehreren Knoten. Wenn man vom Wurzel-Knoten (engl. root node) ausgeht, kann man einen einzelnen Knoten unter Umständen über mehr als einen Weg erreichen. Die Suchreihenfolge im Graphen ist von oben nach unten (engl. depth-first) und von links nach rechts. Schleifen sind im Pfad nicht erlaubt. Abbildung 3.3 zeigt einen einfachen Szenengraphen mit sechs Knoten. Die Beschriftung der einzelnen Knoten gibt den jeweiligen Typ an.

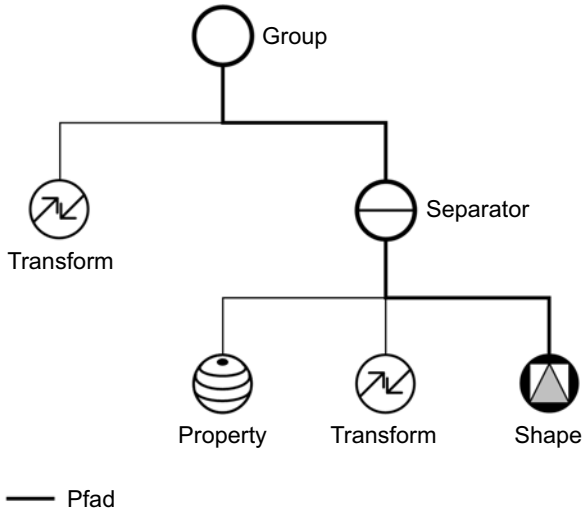


Bild 3.3: Beispiel eines einfachen Szenengraphen.

Alle Knoten-Klassen haben ihrem Namen die Buchstaben `So` (für scene object) vorangestellt, wie z.B. `SoCube` und `SoMaterial`. Die Abbildungen 3.4, 3.5 und 3.6 zeigen eine grafische Repräsentation der Klassenhierarchie [Wer93].

Knoten-Typen

Die Knoten-Klassen kann man in drei grundlegende Kategorien einteilen:

- *Form-Knoten* (engl. shape nodes) definieren die Form eines geometrischen 3D-Objektes (wie z.B. `SoSphere`, `SoCube`, `SoNurbsSurface`, `SoText3`) und verwenden dazu möglicherweise Daten, die vorher im Szenengraphen definiert wurden (z.B. Koordinaten, Texturen).
- *Eigenschaftsknoten* (engl. property nodes) definieren das Aussehen und andere qualitative Merkmale der Szene (z.B. `SoMaterial` legt die Farbe und Transparenz fest, `SoDrawStyle` bestimmt Linienstärke und das Muster, `SoTransformation` gibt eine geometrische Operation an).
- *Gruppen-Knoten* (engl. group nodes) sind Container, die Knoten in Untergraphen zusammenfassen (z.B. `SoSeparator`, `SoSwitch`).

Jeder Knoten besteht aus einer Menge von Feldern. Diese Felder beschreiben die Parameter eines Knoten und jedem Feld ist ein *Name* zugeordnet. Werte des geeigneten Typs (ganze Zahlen, Zeichenketten, etc.) können in diese Felder geschrieben oder

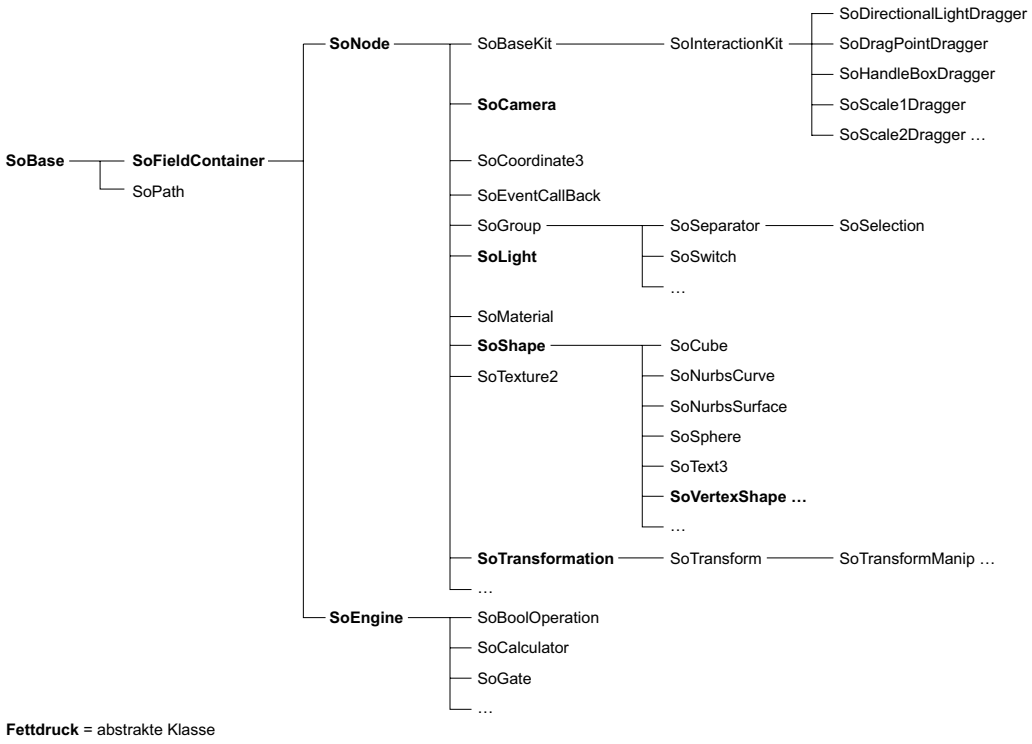


Bild 3.4: Open Inventor Klassenhierarchie Zusammenfassung (1).

von ihnen ausgelesen werden. Beim Erzeugen eines Knoten werden seine Felder mit *Standardwerten* belegt. Zum Beispiel enthält der Knoten für einen Würfel `SoCube` drei Felder mit den Namen `width`, `height` und `depth`. Alle drei Felder können eine Fließkommazahl vom Typ `SoSFFloat` größer als Null beinhalten und geben damit die Ausmaße des geometrischen Objektes an. Der *Standardwert* für alle drei Felder ist die Zahl zwei. Es ist auch möglich, einzelne Felder miteinander zu *verbinden*. Ändert sich der Wert im ursprünglichen Feld, bekommt das verbundene Feld den gleichen Wert zugewiesen.

Aktionen

Auf dem Szenengraphen werden Aktionen (engl. actions) ausgeführt. Die Aktionen durchlaufen den Szenengraphen im Allgemeinen von oben nach unten und von links nach rechts. Die Szenendatenbank verwaltet dabei den Traversierungsstatus, der beim Durchlaufen der Knoten verändert werden kann. Jeder Knoten hat ein eigenes Verhalten in Bezug auf die Unterklassen von `SoAction`. Häufig verwendete Aktionen sind z.B. `SoGLRenderAction` und `SoGetBoundingBoxAction`.

Für die Ausgabe eines Bildes wird der Szenengraph von einer `SoGLRenderAction` durchlaufen. Das Passieren eines Eigenschaftsknotens kann den aktuellen OPENGL-Zustand verändern, eine geometrische Transformation ausführen oder Werte, wie z.B. Textur-Koordinaten (`SoTextureCoordinate3`), für den nachfolgenden Knoten bereitstellen.

Mit der `SoSearchAction` können Knoten im Szenengraph nach ihrem Namen oder ihrem Typ gesucht werden. Als Ergebnis liefert diese Aktion einen exakten Weg vom Wurzel-Knoten zum gefunden Knoten mit dem Typ `SoPath`. Auch eine vom Be-

nutzer mit einem Eingabegerät ausgelöste `SoPickAction`, z.B. durch Anklicken eines Objektes mit der Maus, liefert als Resultat einen Weg vom Typ `SoPath`.

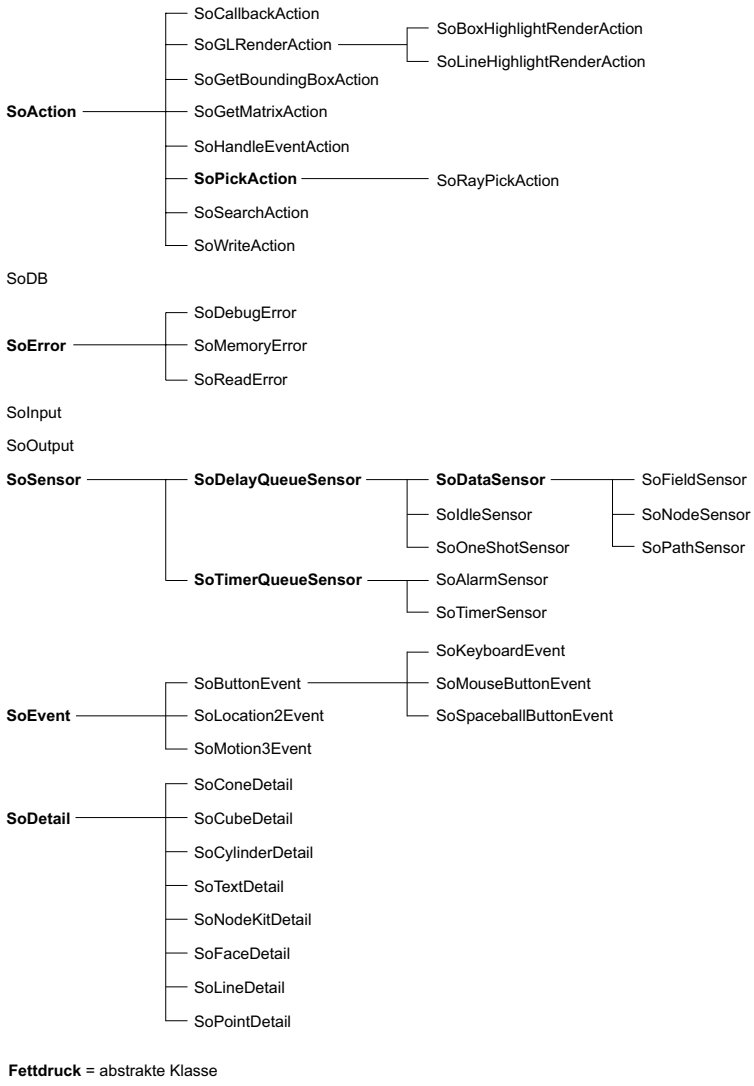


Bild 3.5: Open Inventor Klassenhierarchie Zusammenfassung (2).

Die Knoten vom Typ `SoSeparator` (abgeleitet von `SoGroup`) werden dazu benutzt, um Effekte in einer Gruppe zu isolieren. Bevor die Kinder eines `SoSeparator` durchlaufen werden, wird der momentane Zustand abgespeichert. Nach dem Durchlaufen aller Kinder wird der ursprüngliche Zustand wiederhergestellt. Knoten, die rechts vom `SoSeparator` stehen, sind nicht von Änderungen, wie z.B. Transformationen oder Farbänderungen, die innerhalb des Knotens ausgeführt werden, betroffen. Ein `SoSeparator` kann alle Eigenschaften für seine Unterknoten kapseln und als eigenständiger Baum betrachtet werden.

Referenzen und Löschung

Die Erzeugung von Knoten erfolgt auf die gleiche Weise wie in C++ [Str00] mit Hilfe von `new SoNode`. Das Löschen eines Knotens weicht aber vom Vorgehen in C++ ab. Um die Szenendatenbank konsistent zu halten, dürfen die Knoten nicht einfach mit der

Anweisung `delete` gelöscht werden. Würde man so vorgehen, hätten OPEN INVENTOR und die Szenendatenbank keine Kenntnis über die Änderung. Deshalb bietet OPEN INVENTOR eine andere Methode, um Knoten zu löschen.

Jeder Knoten speichert die Anzahl der Referenzen, die innerhalb der Datenbank vorkommen, ab. Das heißt, jeder Knoten besitzt einen *Referenz-Zähler* (engl. reference count). Wird ein Knoten `K` als Kind einer Gruppe `G` hinzugefügt, so erhöht sich der *Referenz-Zähler* von `K` um eins. Wird `K` aus `G` wieder entfernt, so vermindert sich der Zähler von `K` um eins. Wird der *Referenz-Zähler* eines Knotens gleich Null, so wird der Knoten gelöscht. Ein Knoten wird nur dann gelöscht, wenn der *Referenz-Zähler* auf Null vermindert wird, d.h. er muß vorher eine Zahl größer als Null gewesen sein. Dadurch werden neu erstellte Knoten nicht sofort gelöscht.

Wird eine Aktion auf einem Knoten ausgeführt, erfolgt die automatische Erzeugung eines Pfades, in dem der Knoten referenziert wird. Nach der Beendigung der Aktion wird der dazugehörende Pfad automatisch gelöscht und damit auch der *Referenz-Zähler* des Knotens vermindert. Der Entwickler kann durch manuelles Referenzieren und Dereferenzieren der Knoten ein ungewolltes Löschen verhindern.

Sensoren

Sensoren sind eine spezielle Klasse von Objekten, die mit der Datenbank verbunden sind. Sie können auf Änderungen in der Datenbank oder auf verschiedene zeit-basierende Ereignisse mit dem Aufruf einer vom Benutzer definierten Rückruf-Funktion (engl. callback function) reagieren.

- *Zeit-Sensoren* (engl. timer sensors) verständigen die Applikation, wenn bestimmte Typen von Zeit-Ereignissen auftreten, z.B. wird die Rückruf-Funktion aufgerufen, wenn eine bestimmte Zeitspanne vergangen ist.
- *Daten-Sensoren* (engl. data sensors) überwachen einen bestimmten Teil der Datenbank und benachrichtigen die Applikation, wenn sich dieser ändert. Die Änderung in einem Feld kann z.B. über einen `SoFieldSensor` erfaßt werden.

Engines

Für Berechnungen, das Verknüpfen von Feldwerten und einfache Animationen stellt Inventor das Konzept der *Engine* (dt. Maschine) zur Verfügung. Es gibt vier verschiedene Typen von Maschinen:

- *Arithmetische Maschine* (engl. arithmetic engine)
- *Animationsmaschine* (engl. animation engine)
- *ausgelöste Maschine* (engl. triggered engine)
- *Feldmanipulationsmaschine* (engl. array manipulation engine)

OPEN INVENTOR bietet eine Vielzahl von vordefinierten *arithmetischen Maschinen*, um z.B. Vektoren und Matrizen aus skalaren Werten zu erzeugen oder Rotationen und Vektoren linear zu interpolieren. Zusätzlich gibt es eine allgemeine Maschine `SoCalculator`, die es erlaubt, beliebige einfache Berechnungen durchzuführen. Die Rechenvorschrift wird in Form einer Zeichenkette der Maschine übergeben.

Mit *Animationsmaschinen* kann ein Feldwert entsprechend der Echtzeit geändert werden, z.B. funktioniert `SoElapsedTime` wie eine Stoppuhr; sie gibt die seit ihrem Start verstrichene Zeit aus.

Normalerweise werden Maschinen dazu verwendet, um Knoten miteinander zu verbinden. Es gibt aber einige vordefinierte Knoten mit eingebauten Maschinen. Dazu zählen `SoRotor`, `SoPendulum`, `SoShuttle` und `SoBlinker`. So ist `SoRotor` eine Kombination eines `SoRotation` Knotens mit einer `SoElapsedTime` Maschine. Er ändert den Rotationswinkel mit einer bestimmten Geschwindigkeit und behält dabei die Rotationsachse konstant. Dieser Knoten kann z.B. verwendet werden, um die Segel einer Windmühle zu drehen.

Die Felder einer Maschine sind in Eingabe- und Ausgabe-Felder unterteilt. Ein Feld eines Knotens kann mit dem zum Typ passenden Ausgabe-Feld einer Maschine verbunden werden. Stimmen die zwei Feld-Typen nicht überein, versucht Inventor durch Feldkonverter (engl. field converter) eine Übereinstimmung zu erzielen. Mehrfachkonvertierungen werden nicht unterstützt.

Das Ausgabe-Feld einer Maschine kann mit beliebig vielen Feldern verbunden werden. Ein Feld darf mit genau einem Ausgabe-Feld einer Maschine oder einem anderen Feld gekoppelt werden.

Das Zählen der Referenzen bei Maschinen funktioniert ähnlich wie bei anderen Knoten. Feld-zu-Feld-Verbindungen, einschließlich der Verbindung von der Maschinen-Eingabe zu einem Feld, erhöhen nicht den Referenz-Zähler. Jede Verbindung von der Maschinen-Ausgabe zu einem Feld erhöht den Referenz-Zähler um eins. Eine nicht referenzierte Maschine, deren Ausgabefelder nicht verbunden sind, wird gelöscht.

Die Gesamtheit aller Maschinen mit ihren Feldverbindungen in der Szenendatenbank wird *Maschinennetzwerk* (engl. engine network) genannt. Man könnte annehmen, daß in einem Maschinennetzwerk die Änderung eines Wertes die Neuberechnung aller anderen Werte nach sich ziehen würde. Aus Gründen der Effizienz werden aber nur die Felder und Maschinen-Eingaben als veraltet markiert. Der Wert eines so markierten Feldes wird nur bei einer Abfrage neu berechnet.

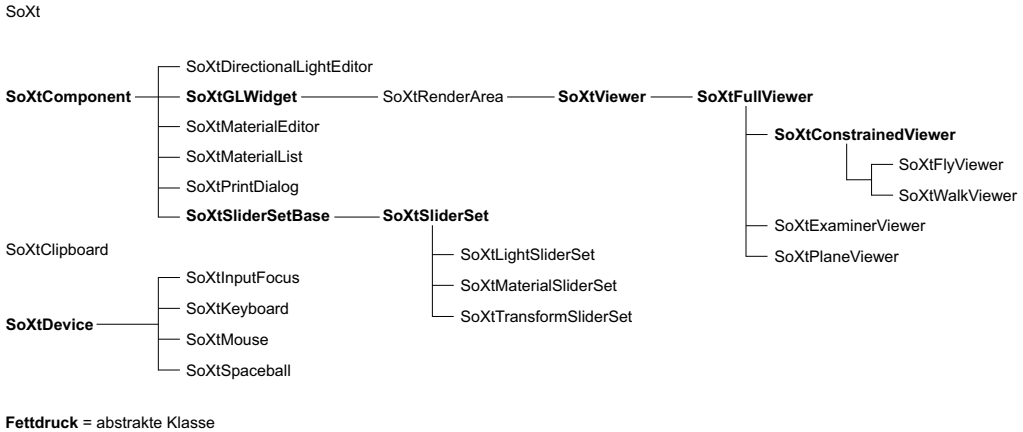


Bild 3.6: Open Inventor Klassenhierarchie Zusammenfassung (3).

3.1.1.4 Node Kits

Node Kits (dt. Knoten-Bausätze) unterstützen die Erzeugung einer konsistenten und strukturierten Datenbank. Mit Node Kits kann man eine Menge von Knoten zu einem Knoten zusammenfassen. Ein Node Kit kann als normaler Knoten betrachtet werden, welcher einen ganzen Untergraphen verbirgt.

Für jeden Node Kit existiert eine Schablone. Diese legt fest, welche Knoten bei Bedarf hinzugefügt werden und wo sich diese im Baum befinden. Die Struktur eines Node Kit ist im Klassen-Konstruktor definiert. Ein Node Kit kann weitere Node Kits enthalten. Damit sind auch hierarchische Strukturen mit Node Kits möglich. Node Kits bieten Methoden, die es auf einfache Weise erlauben, neue Knoten innerhalb eines Node Kit zu erzeugen.

3.1.1.5 Manipulatoren

OPEN INVENTOR wurde speziell im Hinblick auf die Unterstützung von interaktivem Arbeiten entworfen. Ein bedeutendes Mittel sind dabei die *Manipulatoren*.

Ein Manipulator ist ein spezieller Knotentyp. Er reagiert auf vom Benutzer ausgelöste Ereignisse und kann direkt bearbeitet werden. Manipulatoren haben typischerweise Teile, die in der Szene angezeigt werden, z.B. eine greifbare Box. Der Anwender kann durch die entsprechende Interaktion mit dem Manipulator ein Objekt verschieben, rotieren oder skalieren. Die Änderungen an einem Manipulator werden in die passenden Änderungen in der Szenendatenbank umgesetzt. Infolge der Interaktion werden die geänderten Teile am Bildschirm neu gerendert und sind für den Benutzer sofort sichtbar. Abbildung 3.7 zeigt verschiedene Typen von Manipulatoren (im Bild von links nach rechts: `SoTrackBallManip`, `SoHandleBoxManip` und `SoTransformBoxManip`).

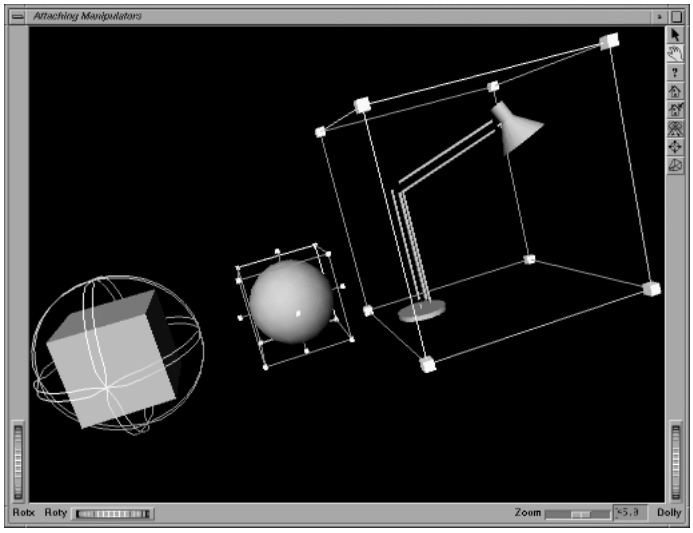


Bild 3.7: Verschiedene Typen von Manipulatoren aus [Wer93].

3.1.1.6 Inventor-Komponenten-Bibliothek

Um eine Szene ausgeben zu können, ist die Integration von OPEN INVENTOR in ein Fenstersystem (z.B. X, MacOS X, MICROSOFT Windows, etc.) notwendig. Diese Verbindung wird durch die *Inventor-Komponenten-Bibliothek* realisiert. Die Bibliothek hat folgende Bestandteile:

- Ein Fensterobjekt für das Ausgeben der Szene
- Hauptschleife und Initialisierungsroutinen
- Eine Ereignis-Verarbeitung
- Komponenten: Editoren für Material und Licht, Betrachter wie examiner viewer und fly viewer

Die Bibliothek kann erweitert werden, um zusätzliche Fenstersysteme zu unterstützen oder um weitere Editoren für Objekteigenschaften hinzuzufügen.

3.1.2 Studierstube

Seit 1996 entwickelt die Gruppe für Virtual Reality des Instituts für Computergraphik und Algorithmen an der Technischen Universität Wien eine Entwicklungsumgebung für Augmented Reality-Anwendungen mit dem Namen STUDIERSTUBE [SFH+00]. Die STUDIERSTUBE hat, neben den Eigenschaften eines AR-Systems, noch folgende Merkmale:

- es können *mehrere* Personen *gleichzeitig* mit dem System interagieren
- *mehrere* Benutzer haben die Möglichkeit, *gemeinsam* zu arbeiten
- die Darstellung der Szene kann für jeden Benutzer *eigenständig* definiert werden
- es ist möglich, *mehrere* Anwendungen zur *gleichen* Zeit auszuführen; ein Datenaustausch zwischen den Anwendungen ist möglich
- das AR-System kann auf *mehreren unterschiedlichen* Rechnern *verteilt* betrieben werden
- es können *mehrere unterschiedliche* Anzeigesysteme zur *gleichen* Zeit verwendet werden

Während die meisten VR- und AR-Systeme für eine spezielle Anwendung maßgeschneidert sind, bietet die STUDIERSTUBE eine Umgebung, in der mehrere Anwendungen zur gleichen Zeit ablaufen. Jede Anwendung hat dabei ihr eigenes 3D-Fenster. Dabei verwendet die STUDIERSTUBE eine ähnliche Metapher, wie sie uns von den herkömmlichen 2D-Fenster-basierten Benutzerschnittstellen bekannt ist.

Die STUDIERSTUBE gestattet mehreren Benutzern gleichzeitig, die virtuelle Szene zu betrachten und mit ihr zu interagieren. Dabei kann jeder Benutzer seinen Blickpunkt frei wählen. Abbildung 3.8 zeigt zwei Benutzer, die mit derselben Anwendung arbeiten. Diese Art der Zusammenarbeit wird auch als *Collaborative Augmented Reality* [SFH00] bezeichnet. Weiters sieht man auf diesem Bild zwei 3D-Fenster. Das mittlere Fenster mit dem blau hervorgehobenen Rahmen ist aktiv, das Fenster rechts vorne ist inaktiv. Für jeden Benutzer kann es genau ein aktives 3D-Fenster, aber beliebig viele inaktive Fenster geben. Ein 3D-Fenster wird aktiviert, indem man seinen Rahmen aus-

wählt. Größe und Position der 3D-Fenster können vom Benutzer oder der Anwendung den Anforderungen entsprechend geändert werden.

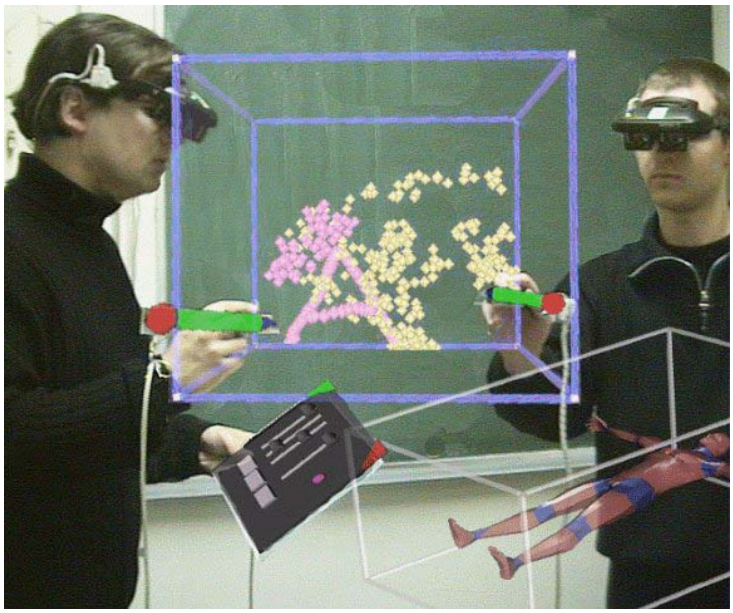


Bild 3.8: Collaborative Augmented Reality ermöglicht die Zusammenarbeit mehrerer Benutzer.

Wie bereits oben erwähnt, kann jeder Benutzer eine eigene Ansicht der 3D-Szene haben. Dies kann z.B. für Spiele verwendet werden. Abbildung 3.9 zeigt zwei Benutzer, die gegeneinander das Spiel Mah-Jongg spielen [SEG98]. Dabei sitzen die Spieler um einen gemeinsamen Tisch. Jeder der Mitspieler ist mit einem See-Through HMD und den Eingabegeräten, dem *Personal Interaction Panel* [SG97] und einem Stift, ausgestattet. Das AR-System stellt die Steine auf dem Tisch und den Eingabegeräten dar und erlaubt die Interaktion mit ihnen. Jeder Spieler sieht während des Spieles nur die für ihn bestimmten Steine.

Bild 3.9: Mehrere Benutzer spielen gemeinsam Mah-Jongg.

Die STUDIERSTUBE unterstützt auch den gleichzeitigen Einsatz von verschiedenen Typen von Anzeigesystemen. So verwendet die Drehbuch-Anwendung [SFH+00] in

Abbildung 3.10 zwei unterschiedliche Systeme zum Anzeigen der Bildinhalte. Zum einen werden die 3D-Inhalte für die beiden Anwender über die See-Through HMDs als Stereobilder angezeigt. Gleichzeitig liefert das AR-System auch ein Monobild für eine Videoprojektion.

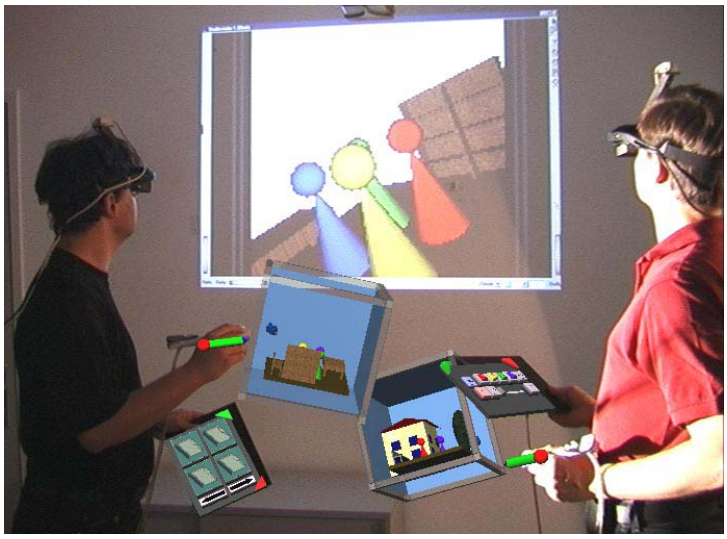


Bild 3.10: Eine Drehbuch-Anwendung mit verschiedenen Anzeigesystemen.

Um STUDIERSTUBE-Applikationen auf mehrere Rechner verteilt zu betreiben, ist eine Erweiterung von OPEN INVENTOR nötig. Diese heißt *Distributed Open Inventor* (DIV) und wird in [HSF+99] beschrieben. Der DIV-Toolkit erweitert OIV um das Konzept eines verteilten, gemeinsam genutzten *Szenengraphen*. Aus der Sicht des Anwendungsentwicklers teilen sich die verschiedenen Rechner einen gemeinsamen Szenengraphen. Jede auf einem Teil des Szenengraphen ausgeführte Operation wird an alle beteiligten Rechner weitergegeben. Das Management von verteilten STUDIERSTUBE-Anwendungen wird in [SRH03] beschrieben.

3.1.2.1 Software-Architektur

Die STUDIERSTUBE-Entwicklungsumgebung basiert auf OPEN INVENTOR und einer Sammlung darauf aufbauender C++ Klassen. Diese befassen sich hauptsächlich mit der Verarbeitung von 3D-Interaktionen und -Ereignissen. Weiters stellen sie die nötige Ablaufumgebung und Infrastruktur für die Ausführung der Anwendungen zur Verfügung. Diese Komponenten bieten dem Entwickler ein komfortables Softwareentwicklungsmodell mit einer wohldefinierten Schnittstelle. Die Anwendungen sind shared objects und werden von der Ablaufumgebung dynamisch geladen.

3.1.2.2 Tracking

STUDIERSTUBE-Anwendungen können gleichzeitig auf verschiedenen Rechnern in einer heterogenen und verteilten Umgebung ablaufen. Um die Trackerdaten den Anwendungen zur Verfügung zu stellen, wurde ein abstraktes Schnittstellenmodell gewählt. Diese Vorgehensweise hat, gegenüber einem auf Treiber basierenden Modell, folgende Vorteile:

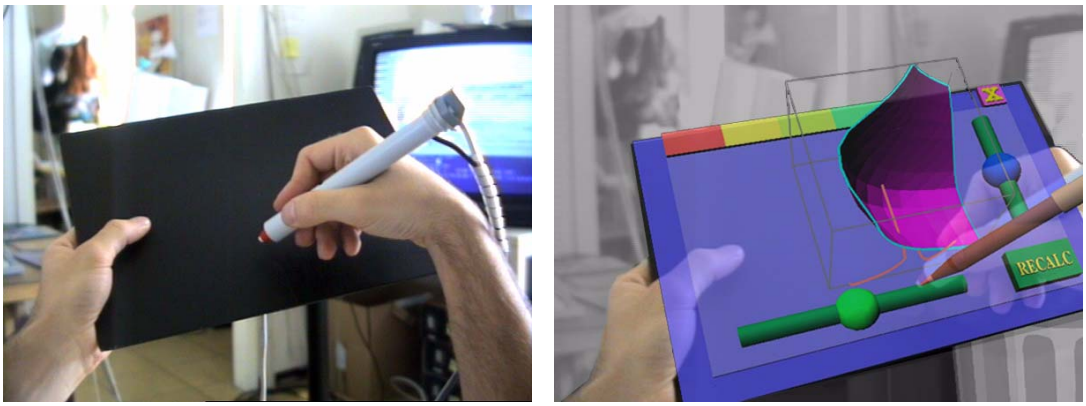
- *Konfigurierbarkeit*: Die typischen Konfigurationen für AR-Umgebungen haben immer ähnliche Basiskomponenten. Sie unterscheiden sich meistens nur in kleinen Details wie z.B. der Positionierung der Tracker-Sensoren und deren Anzahl. Die gewählte Architektur erlaubt es auf einfache Weise, diese Parameter über eine Konfigurationsdatei zu beschreiben.
- *Filterung*: Die meisten Konfigurationsoptionen können als Filter dargestellt werden. Darunter versteht man die Änderung der ursprünglichen Trackerdaten. Beispiele sind Transformation, Verzerrungskompensation und das Zusammenführen der Daten von mehreren Sensoren.
- *Verteilte Ausführung*: Das Verarbeiten der Trackerdaten kann unter Umständen sehr rechenintensiv sein und muß über mehrere Rechner verteilt werden. Gleichfalls kann es notwendig werden, daß die Daten eines Sensors für mehrere Maschinen im Netzwerk zugänglich sind. Das wird erreicht, indem man das Trackingsystem als lose gekoppelte Kommunikationsprozesse implementiert. Der Datenaustausch zwischen den beteiligten Prozessen erfolgt über Unicast- und Multicast-Netzwerkprotokolle.
- *Erweiterbarkeit*: Eine modulare und objektorientierte Architektur des Trackingsystems erlaubt die einfache Erweiterbarkeit und das Hinzufügen von neuen Eigenschaften.

Die für das Projekt benutzte Version der STUDIERSTUBE verwendet ein solches System mit dem Namen OPENTRACKER, welches in Kapitel 3.3.1 genauer beschrieben wird.

3.1.2.3 Eingabegeräte

Die Daten im 3D-Fenster einer STUDIERSTUBE-Anwendung können entweder direkt manipuliert oder über das *Personal Interaction Panel* (PIP) verändert werden. Das *Personal Interaction Panel* [SG97] ist eine Implementierung der Pen & Tablet-Methode und ist ein Eingabegerät, welches mit beiden Händen bedient wird. Die beiden Teile des PIP sind ein flaches Tablett, welches aus einem leichten und nicht magnetischen Material (wie Holz, Plastik oder Acrylglas) besteht, und ein Stift mit ein oder zwei Knöpfen. Die Position des Stiftes und des Tabletts wird durch optisches oder magnetisches Tracking bestimmt. Im AR-System werden der Stift und das Tablett durch ähnliche 3D-Objekte repräsentiert. Dabei werden die reale und die virtuelle Geometrie miteinander verschmolzen. Abbildung 3.11 (a) zeigt die Einzelteile des PIP ohne Augmentierung. Bild (b) zeigt das PIP mit den virtuellen Interaktionselementen aus der Sicht eines Benutzers mit einem HMD.

Auf dem Tablett befinden sich üblicherweise 2D- und 3D-Interaktionselemente (engl. widgets), die eine Interaktion mit der Szene erlauben. Dazu zählen Knöpfe, Schieberegler, Drehregler und neuartige 3D-Elemente. Eine Anzahl dieser Elemente am PIP wird auch als *PIP-Sheet* bezeichnet. Ein *PIP-Sheet* ist als OIV-*Szenengraph* implementiert und besteht zum großen Teil aus STUDIERSTUBE-Interaktionselementen. Der PIP-*Szenengraph* kann auch Geometrie enthalten, die den Zustand einer Interaktion anzeigt oder rein dekorativen Zwecken dient. Der Inhalt des *PIP-Sheet* ist von der

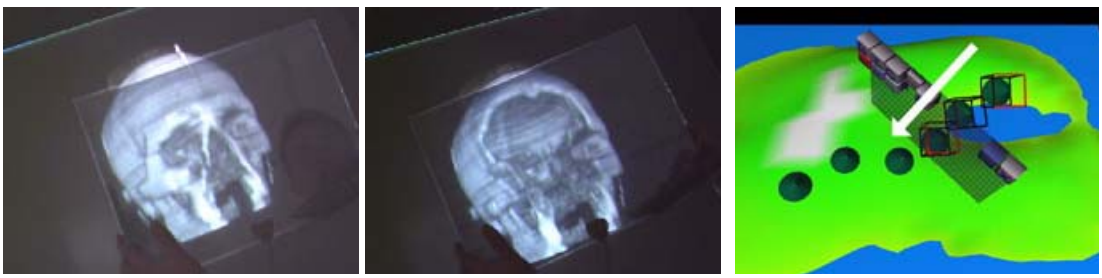


(a) Ohne überlagerte Geometrie. (b) Mit überlagerter virtueller Geometrie.

Bild 3.11: Personal Interaction Panel.

jeweiligen Applikation abhängig. Die Interaktion mit den Elementen auf dem PIP geschieht mit dem Stift.

Die haptische Rückmeldung zwischen Tablett und Stift erleichtert die Interaktion des Benutzers. Die Manipulation eines Interaktionselementes auf einem Tablett, welches man in einer Hand hält, ist einfacher, als ein frei im Raum schwebendes Objekt zu bedienen. Neben den Elementen am PIP kann auch das Tablett selbst zur Interaktion mit der virtuellen Szene verwendet werden. Abbildung 3.12 zeigt zwei Beispiele. Das Bild (a) stellt zwei Aufnahmen, die im Rahmen des STUDYDESK-Systems [WES00] entstanden sind, dar. Hier wird mit Hilfe eines transparenten Tabletts eine Schnittebene durch die 3D-Daten eines menschlichen Schädels gelegt. An der unteren Seitenkante des PIP ist der magnetische Tracker zu sehen. In Bild (b) wird das Tablett als „Fischernetz“ verwendet [SES99]. Dabei werden alle Objekte in der 3D-Szene, die mit dem Tablett in „Berührung“ kommen, ausgewählt.

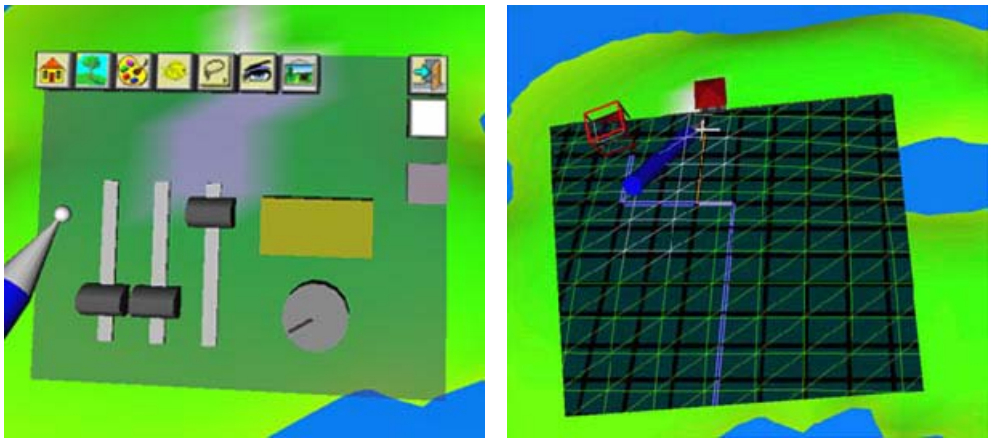


(a) Das Tablett definiert eine Schnittebene in den Volumsdaten. (b) “Fischernetz“-Metapher.

Bild 3.12: Anwendungsbeispiele, bei denen das Tablett als Interaktionsmittel verwendet wird.

Weiters kann die Vorder- und Rückseite des PIP nicht nur die für die Interaktion notwendigen Elemente anzeigen, sondern auch als tragbare Anzeige verwendet werden. In Bild 3.9 wurde bereits die Anwendung im Rahmen eines Spieles gezeigt. Die Steine des Spielers befinden sich auf der Oberfläche des PIP und können mit dem Stift manipuliert werden. In [SES99] wird die Rückseite des PIP in einer Landschaftsplanungsapplikation verwendet. Durch das Umdrehen des PIP wird eine „Röntgen“-Ansicht auf die unter der Oberfläche befindlichen Leitungen und Kabeln ermöglicht (siehe Bild 3.13). Dabei wird die Rückseite des PIP zum Darstellen des „Röntgenbildes“ der Landschaft benutzt. Der Anwender kann nun auf einfache und intuitive Weise

mit dem Stift die gewünschten Leitungsverbindungen herstellen. Durch den Einsatz von beiden Seiten des PIP ist die „Röntgen“-Ansicht des Terrains immer verfügbar.



(a) Die Vorderseite des PIP zeigt hier ein RGB-Farbauswahlwerkzeug.

(b) Die Rückseite des PIP bietet eine „Röntgen“-Ansicht.

Bild 3.13: Die Verwendung der Vorder- und Rückseite des PIP im Rahmen eines Landschaftsplanungsprogrammes.

Es gibt auch die Möglichkeit, die virtuelle Repräsentation von Tablett und Stift über die Tastatur und die Maus zu bedienen. Hierbei lösen bestimmte Tastenkombinationen die gewünschten Translationen und Rotationen aus. Diese Variante wird hauptsächlich während der Softwareentwicklung verwendet.

3.1.2.4 Ausgabegeräte

Die STUDIERSTUBE ist als Umgebung für verschiedene Anwendungen ausgelegt und unterstützt daher auch eine Vielfalt an Anzeigesystemen. Dazu gehören unter anderem auch HMDs und Geräte, die auf Videoprojektoren basieren. Da es unterschiedliche Techniken gibt, um z.B. die Kameraposition festzulegen und Stereobilder zu erzeugen, besitzt die STUDIERSTUBE einen OIV-kompatiblen Betrachter (engl. viewer), der über eine Plug-In-Architektur erweiterbar ist. Derzeit werden folgende Anzeigemodi unterstützt:

- *Field Sequential Stereo*: Die Bilder für das linke und das rechte Auge werden in aufeinanderfolgenden Frames ausgegeben.
- *Line Interleaved Stereo*: Die Bilder für das linke und das rechte Auge belegen die ungeraden und geraden Zeilen eines einzelnen Frames.
- *Dual-Screen Stereo*: Die Bilder für das linke und das rechte Auge werden über zwei unterschiedliche Kanäle (engl. channel) ausgegeben.
- *Anaglyph Stereo*: Die überlagerten Bilder für das linke und das rechte Auge sind in den Farben Rot und Grün oder Rot und Blau kodiert.
- *Mono*: Das gleiche Bild wird für beide Augen verwendet.

Weiters werden dem Entwickler folgende Kamerakontrollmodi angeboten:

- *Tracked Display*: Der Blickpunkt (engl. viewpoint) und die Anzeigefläche werden gemeinsam bewegt und ihre Positionsdaten erfaßt (üblicherweise ein HMD).

- *Tracked Head*: Der Blickpunkt des Benutzers wird erfaßt (normalerweise die Position und Orientierung des Kopfes) und die Anzeige hat eine fixe Position im Raum (z.B. Videowand, Monitor).
- *Desktop*: Der Blickpunkt ist entweder stationär oder kann mit Hilfe der Maus verändert werden.

Die Kombination der oben angeführten Kamera- und Anzeigearten mit einer allgemeinen Off-Axis-Kamera (implementiert als OIV-Knoten `SoOffAxisCamera`, eine Beschreibung findet sich in [SF03]) erlaubt den Einsatz nahezu aller verfügbaren Anzeigesysteme.

3.1.3 Geräte für die Stereo-Projektion

Wenn wir ein Objekt betrachten, entsteht das Bild des Objektes in unserem Geist. Durch den Abstand zwischen unseren Augen sieht das linke Auge ein geringfügig anderes Bild als das rechte Auge. Der Abstand zwischen unseren Augen wird auch „Parallaxe“ oder „binokulare Disparität“ genannt. Die von unseren Augen aufgenommenen Bilder werden an das Gehirn übermittelt, welches daraus ein Stereobild erzeugt.

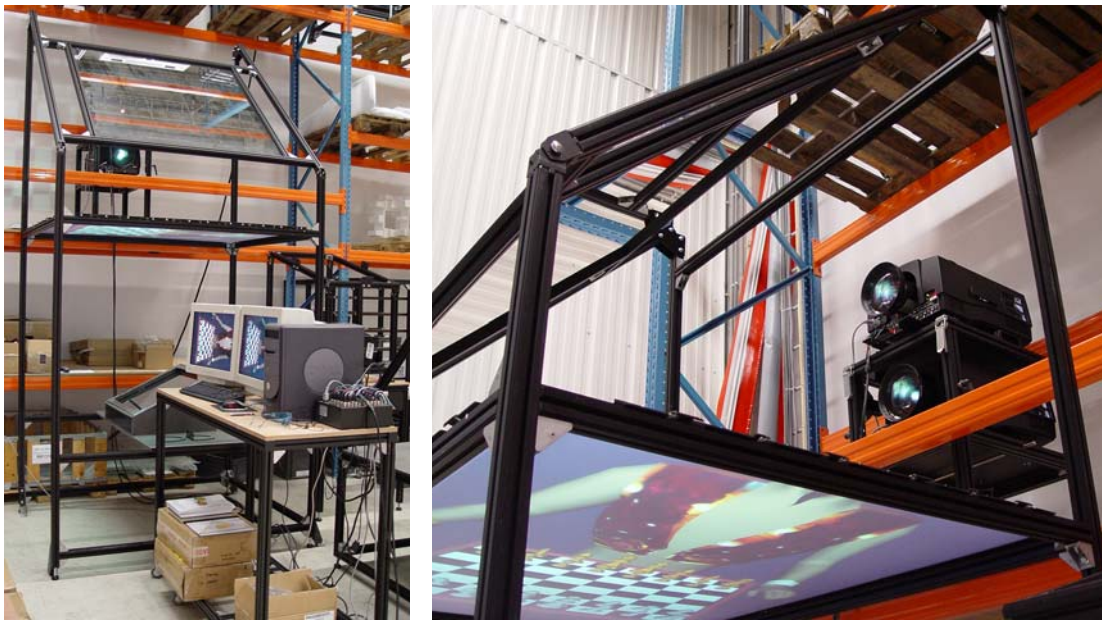
Um mit Hilfe eines Computers Stereobilder zu erzeugen, muß man zwei unterschiedliche Ansichten für das linke und das rechte Auge generieren. Weiters muß man gewährleisten, daß die jeweilige Ansicht nur für das entsprechende Auge sichtbar ist. Das ist auf verschiedene Arten zu erreichen. Abhängig von der Art der verwendeten Stereobrille kann man die Stereo-Systeme in *aktive* und *passive* Systeme einteilen. Bei beiden Methoden kommen Brillen, die vom Betrachter getragen werden, zur Anwendung. Bei *aktiven* Systemen werden Brillen mit elektronischen Komponenten verwendet. *Passive* Systeme enthalten keine elektronischen Bauteile in den Brillen.

3.1.3.1 Passive Stereo-Projektion

Die *passiven* Stereo-Systeme basieren auf dem Prinzip der Polarisation von Licht. Abbildung 3.14 zeigt den Aufbau des getesteten *passiven* Stereo-Projektionssystems, welches zwei Videoprojektoren verwendet. In jedem der beiden Projektoren befindet sich im Linsensystem ein Polarisationsfilter. Die Brille besitzt die dazu passenden Filter als Brillengläser. Dadurch wird ein farbiges dreidimensionales Stereobild möglich. Beispiele von Stereobrillen mit Polarisationsfiltern finden sich in Abbildung 3.17.

Theorie

Den *passiven* Systemen liegt folgende Theorie zugrunde: Eine Lichtwelle rotiert in eine beliebige Richtung. Die spezifische Orientierung für jeden vorgegebenen Zeitpunkt bestimmt die Polarisation der Lichtwelle. Wird unpolarisiertes Licht durch einen Polarisationsfilter geschickt, so erhält man Lichtwellen mit nur einer Orientierung als Ergebnis. Ist die Polarisation für das linke und rechte Auge unterschiedlich ausgeführt, so kann man jedem Auge ein unterschiedliches Bild zuführen. Das menschliche Auge ist sehr unempfindlich in Bezug auf polarisiertes Licht. Das wahrgenommene Bild wird durch die Änderung der Polarisation im Allgemeinen nicht beeinflußt.



(a) Gesamter Hardwareaufbau.

(b) Zu sehen sind die beiden Videoprojektoren, der große Spiegel und die Rückprojektionsfläche.

Bild 3.14: Passive Stereo-Projektion mit Umlenkung des Bildes über zwei Spiegel.

Wird das Licht in nur eine Richtung polarisiert, so spricht man von *linearer* Polarisierung. Ändert der Träger von *linear* polarisierten Brillen seine Kopfhaltung, indem er z.B. den Kopf zur Seite neigt, stimmt die Polarisation zwischen Brille und Videoprojektoren nicht mehr überein. Dadurch kommt es zum Verlust der Stereoinformation beim Betrachter. Trotzdem ist die *lineare* Polarisation eine kostengünstige Technologie für stereoskopische Anwendungen und liefert eine sehr gute Bildtrennung zwischen dem linken und dem rechten Auge.

Abbildung 3.15 veranschaulicht die Funktion von Polarisationsfiltern. Die beiden Filter für das linke und das rechte Auge haben im Allgemeinen eine um 90 Grad versetzte Ausrichtung. Die Ausrichtung der beiden Polarisationsebenen ist meistens horizontal und vertikal oder dazu um 45 Grad gedreht.

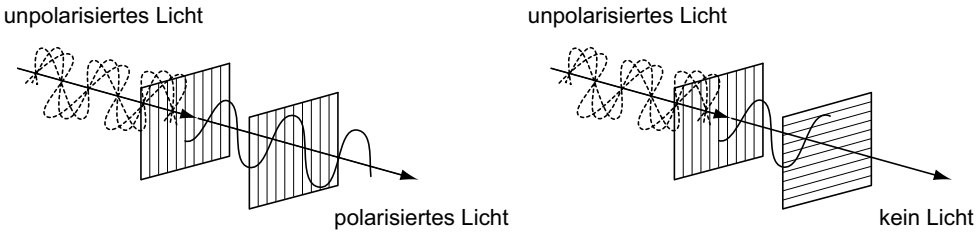


Bild 3.15: Links: Beide Polarisationsfilter haben die gleiche Ausrichtung, das Licht wird durchgelassen. Rechts: Die Filter haben eine um 90 Grad gedrehte Ausrichtung, kein Licht wird durchgelassen.

Eine weitere Methode für *passive* Stereo-Systeme ist die *kreisförmige* (engl. circular) Polarisation. Hier kann der Betrachter seinen Kopf neigen und seinen Blickwinkel relativ zur Projektionsfläche ändern, ohne daß der stereoskopische Eindruck verloren geht. Dies ist möglich, da die Polarisation des Lichtes nicht nur in eine einzige Rich-

tung erfolgt. Diese Technologie erfordert aber höhere Präzision und eine feinere Abstimmung zwischen den Polarisationsfiltern in den Projektoren und der Brille. Für die hier beschriebenen Tests sind nur *lineare* Polarisationsfilter zum Einsatz gekommen.

Aufbau des Systems

Zwei Videoprojektoren vom Typ BARCO SIM6 Ultra MM [Bar02a], mit Polarisationsfiltern im Strahlengang, liefern die Bilder für das linke und rechte Auge. Das Licht wird über einen großen Spiegel auf eine milchige Rückprojektionsscheibe reflektiert (siehe Bild 3.14 (b)). Der Betrachter sieht das Bild von der Rückprojektionsscheibe mit Hilfe eines halbtransparenten Spiegels (siehe Bild 3.16).

Bild 3.16: Das Stereobild auf der Rückprojektionsfläche wird über einen halbtransparenten Spiegel zum Betrachter reflektiert. Rechts neben dem Spiegel liegen die Polarisationsbrillen.

Es ist wichtig, daß der große Spiegel, die Rückprojektionsscheibe und der halbtransparente Spiegel die Polarisation des Lichtes unverändert lassen. Abbildung 3.18 zeigt den Strahlengang von den Projektoren zum Benutzer. Für den halbtransparenten Spiegel wurden die in Tabelle 3.1 angegebenen Folien überprüft. Eine weitere Auswahl an Materialien findet sich in [Mad02].

		Sichtbares Licht (%)	
Filmtyp	**Firma**	**Reflektion**	**Transmission**
Scotchtint RE15SIXL	3M	63,0	16,0
Scotchtint RE20SIARL	3M	58,0	19,0
Silber 15	G&G Folienwelt	57,2	17,3
Silber 35	G&G Folienwelt	37,0	31,9

Tabelle 3.1: Überprüfte halbtransparente Spiegelfolien.

Da die meisten in der Tabelle 3.1 genannten Folien Polyesterbestandteile verwenden, sind sie für den Einsatz bei polarisationsbasierten passiven Stereo-Systemen ungeeignet. Als Einzige lieferte die Folie „Silber 15“ ein befriedigendes Ergebnis in Bezug auf die Polarisationserhaltung.

Die Bilder für die beiden Videoprojektoren wurden von einem PC (DELL Dimension 8250; 2.4GHz, 512MB RAM, MICROSOFT Windows XP Professional SP1) geliefert. Die beiden analogen Ausgänge der Videokarte (Quadro4 900 XGL; 128MB) waren mit den beiden Videoprojektoren verbunden. Dabei war der Ausgang der Video-

(a) Normale Ausführung aus Karton. (b) Exklusive Ausführung.

Bild 3.17: Stereobrillen mit polarisierten Gläsern von BARCO.

karte, auf dem das Bild für das linke Auge dargestellt wird, mit dem Videoprojektor, der den Polarisationsfilter für das linke Auge enthält, verbunden. Der zweite Videoausgang der Videokarte wurde an den anderen Projektor angeschlossen. Als Software wurde die STUDIERSTUBE mit einer Vorversion des „Türkischen Schachspielers“ verwendet.

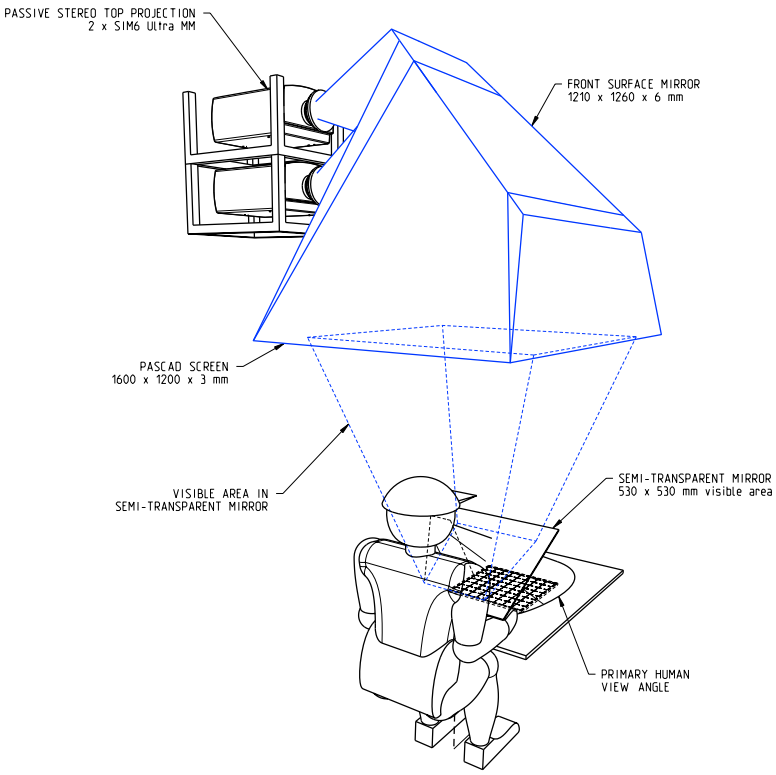


Bild 3.18: Projektionspyramide der passiven Stereo-Projektion - Teilansicht aus der Konstruktionszeichnung (mit freundlicher Unterstützung von BARCO).

Auf ein Tracking der Kopfposition des Betrachters wurde bei diesem Testaufbau verzichtet. Es sollten primär die Qualität des erzeugten Stereobildes und die Ausmaße der mechanischen Konstruktion beurteilt werden.

Der Testaufbau lieferte ein gutes Stereobild mit einer sehr guten Trennung des linken und rechten Videobildes durch die Polarisationsfilter. Der Aufbau bestätigte aber die Überlegungen in Bezug auf die Größe der mechanischen Konstruktion. Bei der maßgetreuen Darstellung des virtuellen „Türkischen Schachspielers“ würden die Abmaße für die beiden Spiegel und die Rückprojektionsscheibe zu groß werden.

3.1.3.2 Aktive Stereo-Projektion

Bei *aktiven* Stereo-Systemen werden Brillen mit eingebauten elektronischen Komponenten verwendet. Abbildung 3.20 zeigt eine LCD-Stereobrille (Liquid Crystal Display) der Firma StereoGraphics. Bei *aktiven* Stereo-Systemen wird am Ausgabegerät abwechselnd das Bild für das linke und rechte Auge gezeigt. Synchron dazu wird in der Brille das jeweilige andere Auge mit Hilfe eines LCD-Verschlusses abgedeckt.

Ist die Frequenz, mit welcher zwischen dem linken und rechten Auge umgeschalten wird, zu niedrig, dann nimmt der Betrachter dies als „Flimmern“ wahr. Die Bildwiederholfrequenz für jedes Auge sollte mindestens 40Hz sein. Das Ausgabegerät und die verwendete Videokarte müssen also Bilder mit einer Frequenz von mindestens 80Hz liefern können.

Für ein gutes Stereobild ist auch die Synchronisation zwischen der Videokarte, dem Ausgabegerät (z.B. Videoprojektor, Monitor) und der Stereobrille wichtig. Normale LCD- und DLP-Videoprojektoren sind meistens für eine synchrone Ausgabe des Videobildes nicht geeignet. Es kommen deshalb spezielle Röhren- und DLP-Videoprojektoren zum Einsatz. Ein Teil dieser Geräte erlaubt über einen speziellen Eingang (Stereo Sync Input) die explizite Synchronisation mit der Videokarte.

Die Stereobrille wird über ein Kabel oder über einen Infrarot-Sender mit dem Videosignal synchronisiert. Die Abbildungen 3.19 zeigen das getestete *aktive* Stereo-Projektionssystem.

Bild 3.19: Aktive Stereo-Projektion auf eine holographische Rückprojektionsscheibe.

Aufbau des Systems

Das Bild eines Videoprojektors vom Typ Barco Galaxy WARP [Bar02b] wird auf eine holographische Rückprojektionsscheibe abgebildet. Der vor der Projektionsscheibe sitzende Betrachter nimmt durch die LCD-Stereobrille die Bilder als dreidimensionale Szene wahr. In Abbildung 3.22 wird der Strahlengang vom Projektor zum

Betrachter dargestellt. Als PC-System wurde die bereits in Kapitel 3.1.3.1 beschriebene Konfiguration verwendet.

Eine Besonderheit stellt die im Testaufbau verwendete Rückprojektionsscheibe dar. Eine der Projektvorgaben ist die Transparenz der Scheibe. Auf Grund dieser Anforderung kann kein normales durchsichtiges Material verwendet werden. Der Betrachter würde beim Verwenden einer herkömmlichen Rückprojektionsscheibe immer direkt in das Objektiv des Videoprojektors (engl. hot spot) schauen und dadurch geblendet werden. Um dies zu vermeiden, wird eine sogenannte holographische Rückprojektionsscheibe eingesetzt. Bei diesen Scheiben ist auf einem Trägermaterial (z.B. Glas, Acryl, etc.) eine Folie aufgebracht, welche spezielle optische Eigenschaften hat. Die Folie kann mit Hilfe von holographisch-optischen Elementen das auf sie eintreffende Licht lenken.

Die Funktion der lichtlenkenden Folie hängt von der Richtung des einfallenden Lichtes ab. Nur Licht, welches von einem bestimmten Punkt ausgeht, wird so durch die Folie hindurchgelenkt, daß es der Betrachter als Bild auf der Projektionsscheibe wahrnehmen kann. Licht aus anderen Richtungen passiert die Projektionsscheibe nahezu unbeeinflußt.

Bild 3.20: StereoGraphics CrystalEyes LCD-Stereobrille.

Der Videoprojektor muß in einem bestimmten Winkel zur Rückprojektionsscheibe ausgerichtet sein. Dieser Winkel wird für die meisten Scheiben mit ca. 35 Grad angegeben. Bild 3.21 zeigt den Projektionswinkel bei unterschiedlicher Anordnung des Videoprojektors.

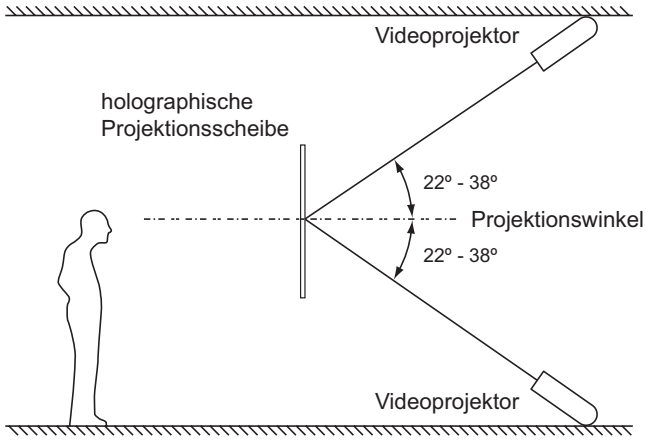


Bild 3.21: Projektionswinkel zur holographischen Rückprojektionsscheibe.

Durch diese Form der Projektion kommt es zu zwei Problemen beim resultierenden Bild. Durch die schräg auftreffende Projektion entsteht eine *Verzerrung* des Bildes in der Form eines *Trapezes* (zu sehen in Abbildung 3.22). Weiters stimmt die *Schärfeebene* nicht mehr mit der *Projektionsebene* überein. Die Verzerrung des Bildes kann im Projektor durch einen *Trapezausgleich* (engl. keystone correction) kompensiert werden. Die beiden Schärfeebenen können durch eine *Scheimpflug-Korrektur* wieder angeglichen werden. Der *Trapezausgleich* wird durch die digitale Bearbeitung des Bildes im Projektor durchgeführt. Die *Scheimpflug-Korrektur* erfolgt durch Hardware im Linsensystem des Videoprojektors.

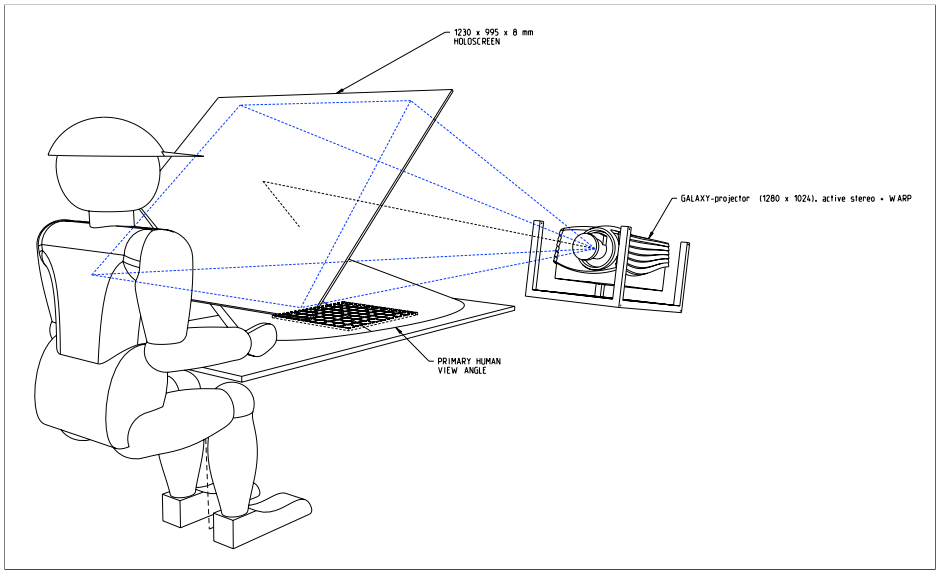


Bild 3.22: Projektionspyramide der aktiven Stereo-Projektion - Teilansicht aus der Konstruktionszeichnung (mit freundlicher Unterstützung von BARCO).

Die in der getesteten Konfiguration verwendete Rückprojektionsscheibe ist vom Typ HOLO SCREEN der Firma DNP. In der Tabelle 3.2 sind weitere transparente Rückprojektionsscheiben angeführt. Scheiben, die eine sichtbare Struktur besitzen, wie z.B. das Modell HOLOPRO der Firma G+B PRONOVA, sind nicht Bestandteil der Tabelle. Durch das technisch bedingte Muster in der holographischen Folie sind sie auf Grund der Anforderungen für diese Anwendung nicht geeignet.

Bezeichnung	Firma	Projektionswinkel	Transparenz	Peak-Gain
HOPS	sax3d.com gmbh	38º	85%	>3,1
tranScreen	Lumin Visual Technologies AG	22° – 35°	60%	3,8
Holo Screen	dnp denmark as	27° – 37°	60%	3,2

Tabelle 3.2: Überprüfte holographische Rückprojektionsfolien.

3.1.3.3 Virtueller Tisch

Der „Virtuelle Tisch“ ist ein *aktives* Stereo-System und zugleich ein weiteres Beispiel für die vielfältigen Möglichkeiten in der STUDIERSTUBE. Auf die schwenkbare Rückprojektionsfläche, die in der Form eines Tisches ausgeführt ist, wird mit Hilfe eines

Röhrenprojektors die Bildinformation projiziert. Abbildung 3.23 zeigt den verwendeten BARCO BARON Tisch [Bar02c] des TECHGATE. In Bild (a) sind der Tisch und die beiden auf der Rahmenkonstruktion angebrachten optischen Tracker zu sehen. Als optisches Tracking-System wird der ARTTRACK (siehe Abbildung 3.24 (a)) verwendet.

Neben den Positions- und Orientierungsdaten des Stiftes, der LCD-Stereobrille und des PIP werden auch die Daten des Tisches über reflektierende Markierungen bestimmt. Durch den im Betrieb schwenkbaren und optisch verfolgten Tisch wird das zu betrachtende Volumen vergrößert [SF03]. Bild 3.23 (b) zeigt einen Schach spielenden Benutzer. Die Interaktion des Benutzers der mit virtuellen Szene erfolgt mit dem Stift.

(a) BARCO Baron Virtual Table mit aktiver Rückprojektion.

(b) Benutzer spielt Schach mit optisch getracktem Stift und LCD-Stereobrille.

Bild 3.23: Virtueller Tisch mit optischem Tracking-System.

Die kugelförmigen Markierungen am Stift und an der Tischumrahmung sind im Bild gut zu erkennen, da sie das Blitzlicht des Fotoapparates reflektieren. Die Marker auf der LCD-Stereobrille sind in Bild 3.24 (b) zu sehen. Für den „Virtuellen Tisch“ ist das verwendete PIP aus einem transparenten Material gefertigt.

(a) Optischer Tracker.

(b) LCD-Stereobrille mit optischen Markern.

Bild 3.24: Optisches Tracker-System der Firma Advanced Realtime Tracking GmbH.

3.2 Animation

Der historische Automat führte die Schachzüge mit der linken Hand aus, welche von einem Menschen durch mechanische Einrichtungen (Seilzüge, Gestänge, etc.) gesteu-

ert wurden. Die Abbildung 3.25 zeigt eine Nachbildung der Figur des Türken wie ihn RACKNITZ in [Rac89] beschrieben hat. GLAESER und STROUHAL nehmen diese Beschreibung als Grundlage für eine Computersimulation der mechanischen Komponenten des Türken. Dazu verwenden sie das OPEN GEOMETRY-System. Sie gelangen in [GS00] zum Ergebnis, daß der von RACKNITZ beschriebene Automat die Aufgaben des „Türkischen Schachspielers“ von Kempelen ausführen konnte.

Für die Realisierung des Schachspielers im Rahmen dieser Arbeit ist es nicht erforderlich, die mechanischen Details im Inneren des Automaten nachzubilden. Vielmehr liegt hier der Schwerpunkt auf der optisch ansprechenden Darstellung der Figur des Türken und der überzeugenden mechanischen Bewegung seines Armes. Dafür werden zwei Hilfsmittel benötigt. Für die Bewegung des Armes wird das inverse Kinematiksystem IKAN verwendet (siehe Kapitel 3.2.2). Zur Animation der Figur wird ein OPEN INVENTOR *Node Kit* mit dem Namen MESHDEFORMER verwendet. Dieser *Node Kit* wird in Kapitel 3.2.3 beschrieben.

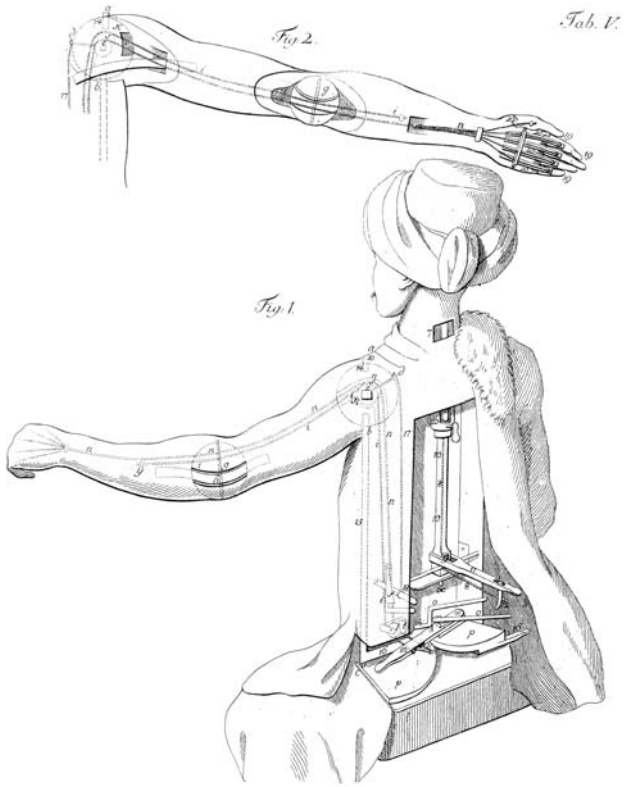


Bild 3.25: Mechanische Konstruktion des historischen Schachspielers nach Racknitz.

Die im Anschluß beschriebenen Veröffentlichungen beschäftigen sich mit der Lösung von Problemen im Bereich der *inversen Kinematik*, im Speziellen mit der Bewegung der menschlichen Gliedmaßen.

ZHAO und BADLER beschreiben in [ZB94] ein System zur Simulation einer menschlichen Figur. Das System mit dem Namen JACK verwendet für die Simulation der Bewegungen Methoden aus der inversen Kinematik und der nichtlinearen Programmierung. In [Bad97] beschreibt BADLER eine Erweiterung des JACK-Systems, die der virtuellen Figur auch komplexe Tätigkeiten ermöglicht, ohne daß die Vorgabe aller

einzelnen Zwischenschritte nötig ist. DIANE CHI et al. [CCZ+00] beschreibt EMOTE (Expressive MOTion Engine), ein 3D-Animationssystem für Figuren, welches die Bewegung der Arme und des Rumpfes durch die Angabe von *Effort*- und *Shape*-Parametern erlaubt. Dadurch können bei der Animation der Figur natürlich aussehende Gesten erzeugt werden. Die Gebärden werden durch high-level Parameter festgelegt, welche die qualitativen Aspekte der Bewegung beschreiben. LIU und BADLER beschreiben in [LB03] die Verwendung von Graphikhardware für die Planung der Armbewegungen einer virtuellen menschlichen Figur. Die Berechnungen erfolgen in Echtzeit mit Hilfe eines auf Heuristik basierenden Algorithmus. Bei gegebener Anfangs- und Endposition wird ein kollisionsfreier Pfad durch den 3D-Arbeitsraum für den Endeffektor berechnet.

In [Kon94] beschreibt KONDO einen inverse Kinematik-Algorithmus für die realistische Bewegung eines menschlichen Arms. Der Algorithmus basiert auf experimentellen Resultaten in der Neurophysiologie (Sensorimotor Transformation Model). Das zugrunde liegende Modell beschreibt, wie im Gehirn die visuelle Information des Zielpunktes in eine Armhaltung, welche zum Berühren des Zielpunktes durch eine Fingerspitze notwendig ist, umgesetzt wird. Aufbauend auf [Kon94] beschreiben KOGA, KONDO, KUFFNER und LATOMBE in [KKK+94] einen Algorithmus für die kollisionsfreie Planung von Bewegungsbahnen beim Zusammenspiel von mehreren Armen. Der Bewegungsplaner ist auch fähig, komplexere Bewegungsabläufe, welche das Abstellen und Wiederaufgreifen eines Objektes erfordern, zu lösen.

In [KW00] befassen sich KOPP und WACHSMUTH mit der automatischen Erzeugung von naturgetreuen Gesten für einen anthropomorphen virtuellen Agenten. Dem Modell, welches für die Erzeugung der Bewegungen verwendet wird, liegen unter anderem Methoden aus der Biologie zugrunde. In dem Artikel [WK02] beschreiben WACHSMUTH und KOPP ein Modell, welches aus einer symbolischen Beschreibung synthetisch naturgetreue Gesten für eine virtuelle Figur erzeugt. Dabei sind die Natürlichkeit der Bewegung und ihre zeitliche Regulierung zwei wichtige Aspekte. Ein weiterer Schwerpunkt ist die zeitliche Koordination mit externen Ereignissen, wie z.B. das Audiosignal aus einem Sprachausgabesystem.

3.2.1 Kinematik

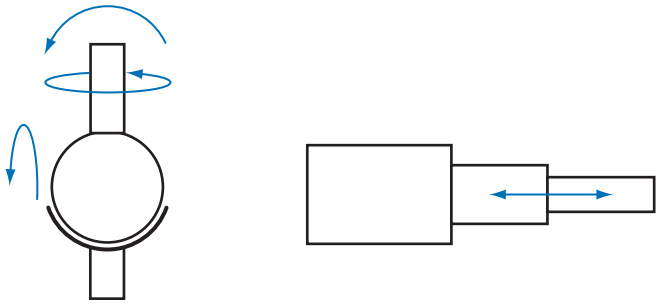
Die Kinematik ist ein Teilgebiet der Mechanik. Sie beschäftigt sich mit der geometrischen Beschreibung von Bewegungen und berücksichtigt dabei nicht die Ursache der Bewegung. Die Kinematik wird verwendet, um die Objekte eines Animationssystems, sogenannte *kinematische Ketten*, zu modellieren.

Eine *kinematische Kette* ist die Anordnung von mehreren *starren Körpern*, welche mittels *Bedingungen* miteinander oder mit der Umgebung verknüpft sind. Unter einem *starren Körper* versteht man ein geometrisches Objekt, das nicht deformiert werden kann. Eine *Bedingung* kann z.B. ein Gelenk sein. Gelenke lassen sich in zwei Gruppen einteilen. Es gibt zum einen *Drehgelenke* (siehe Abbildung 3.26 (a)), zum anderen *Gleitgelenke* (siehe Abbildung 3.26 (b)). Im Skelett des Menschen ist das Hüftgelenk ein Beispiel für ein Drehgelenk. In weiterer Folge befassen wir uns nur mit Drehgelen-

ken, da Gleitgelenke für die menschliche Anatomie keine Bedeutung haben. Gleitgelenke erlauben Verschiebungen auf einer oder mehreren Achsen und finden vor allem in der Simulation von Maschinen und Robotern Anwendung.

Ein Drehgelenk erlaubt die Rotation des zugeordneten *starren Körpers* um eine oder mehrere Bewegungsachsen. In diesem Zusammenhang spricht man auch von *Freiheitsgraden*. Die Freiheitsgrade eines Drehgelenks legen fest, wie weit man das Drehgelenk um jede der drei Achsen drehen kann. Weiters können gewisse Achsen für ein Drehgelenk inaktiv sein. So ist in den meisten Animationen für das menschliche Knie- und Ellbogengelenk nur eine Rotationsachse aktiv.

Die in diesem Projekt verwendete *kinematische Kette* ist eine hierarchische Ordnung mit einer Eltern-Kind-Beziehung. Sie besteht aus den *starren Körpern* Oberarm–Unterarm–Hand und aus den Drehgelenken Schulter–Ellbogen–Hand. In dieser Objekthierarchie sind die Körper so miteinander verknüpft, daß ein Objekt dem folgenden eindeutig über- oder untergeordnet ist.



(a) Drehgelenk. (b) Gleitgelenk.

Bild 3.26: Gelenktypen aus [Wei02].

Skelettmodell

Ein *Skelettmodell* entspricht einer *kinematischen Kette*. Hierbei werden den Knochen jeweils *starre Körper* zugeordnet. Die Gelenke entsprechen *Bedingungen* mit den zugehörigen *Freiheitsgraden*. Das Skelettmodell ist einfach zu begreifen, da es eine ähnliche Struktur wie das reale Skelett aufweist. Ein Skelettmodell kann meistens in mehrere kinematische Ketten zerlegt werden. So kann man z.B. Arme und Beine als eigenständige kinematische Ketten betrachten. Ein Ende einer kinematischen Kette ist meistens fest verankert, das andere läßt sich frei bewegen. Als Beispiel sei hier wieder der Arm gewählt. Die Schulter ist der Ausgangspunkt und wird als fest angenommen. Die Hand ist das frei bewegbare Ende und wird auch als *Endeffektor* bezeichnet.

Ein Vorteil des Skelettmodells ist die Möglichkeit, eine Verbindung zwischen der kinematischen Kette und der verwendeten Geometrie (z.B. der Haut) herzustellen. Dafür werden Koordinaten in der Geometrie (Haut) mit einem oder mehreren starren Körpern (Knochen) assoziiert. Durch Angabe von Gewichten kann der Einfluß einzelner Teile der kinematischen Kette gesteuert werden. Dadurch wird die Geometrie bei einer Bewegungen des Skeletts entsprechend deformiert. Dies ist eine Grundlage für realistische Animationen.

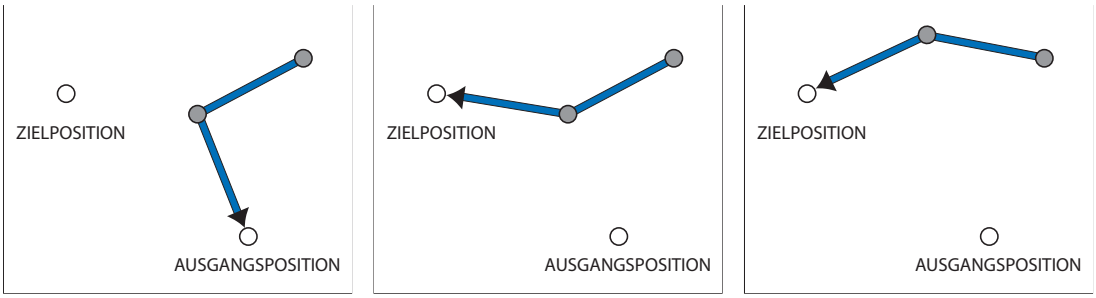
Vorwärtskinematik

Bei der *Vorwärtskinematik* sind die Winkel in den Gelenken, welche die *starren Körper* miteinander verbinden, bekannt. Der aus den bekannten Winkeln resultierende Endpunkt der *kinematischen Kette* ist unbekannt und soll ermittelt werden. Dies soll an einem Beispiel erläutert werden. Die einzelnen Glieder des Armes (Oberarm, Unterarm und Hand) sind hierarchisch verknüpft. Der Oberarm ist das ranghöchste Objekt in dieser Hierarchie. Durch die Verknüpfung sind die Glieder miteinander verbunden, aber der Winkel in den Gelenken kann verändert werden. Wird z.B. der Oberarm im Schultergelenk bewegt, hat dies einen Einfluß auf die Position des Unterarmes und der Hand. Wird die Hand im Handgelenk bewegt, hat dies keinen Einfluß auf die Position des Ober- und Unterarms. Allgemein kann man sagen, daß die Bewegung eines Gliedes in der Objekthierarchie einen Einfluß auf alle untergeordneten Glieder, aber keine Auswirkung auf übergeordnete Glieder hat.

Inverse Kinematik

Die *inverse Kinematik* basiert auf dem vorgestellten *Skelettmodell*. Sie ist wie die *Vorwärtskinematik* ein rein mathematisches Modell und findet in der Animation von Bewegungsabläufen ihre Anwendung. Anders als bei der Vorwärtskinematik beginnt bei der inversen Kinematik die Transformation beim *Endeffektor* der *kinematischen Kette*. Wird der Endeffektor bewegt, folgen alle übergeordneten Glieder entsprechend ihren *Bedingungen*. Bei dieser Methode ist die Position und Orientierung des Endeffektors bereits bekannt. Gesucht sind die Winkel in den Gelenken, um die gewünschte Stellung des Endeffektors zu erreichen.

Ursprünglich wurde die inverse Kinematik in der *Robotik* verwendet. Der Greifarm eines Roboters oder das Werkzeug in einer Maschine sollte an eine bestimmte Zielposition gebracht werden. Für die Positionierung werden die Winkel in den Gelenken mit Hilfe der inversen Kinematik berechnet. Es ist relativ einfach, diese Methode für die Computeranimation zu adaptieren. Dies soll wieder am Beispiel des menschlichen Arms gezeigt werden. Es wird wieder angenommen, daß die einzelnen Glieder des Armes hierarchisch verknüpft sind und als Skelettmodell vorliegen. Die Position des Schultergelenks ist fixiert. Die Hand als Endeffektor soll an eine Position gebracht werden. Mit Hilfe der inversen Kinematik erfolgt die Berechnung der benötigten Winkel im Hand-, Ellbogen- und Schultergelenk.



(a) Ausgangsposition. (b) Variante 1. (c) Variante 2.

Bild 3.27: Für eine gegebene Zielposition kann es mehrere Lösungen geben [Wei02].

Der Einsatz der inversen Kinematik bringt auch Probleme mit sich. Da die Software die Gelenkstellungen für die gewünschte Position und Orientierung des Endeffektors berechnet, hat der Anwender kaum Einfluß auf die Bewegung. Abbildung 3.27 zeigt ein solches Beispiel. Bild (a) zeigt die Gliederkette in ihrer Ausgangssituation. Die Bilder (b) und (c) zeigen zwei unterschiedliche Varianten, in denen die Zielposition erreicht wurde. Der Anwender hat meistens die Möglichkeit, über zusätzliche Parameter und Nebenbedingungen das Resultat der Berechnungen zu beeinflussen.

3.2.2 IKAN

Für die Animationsberechnungen des Armes wird die *inverse Kinematik*-Software IKAN (Inverse Kinematics using ANalytical Methods) der *University of Pennsylvania* eingesetzt. Sie kombiniert analytische und numerische Methoden, um verallgemeinerte inverse Kinematik-Probleme zu lösen. Dabei können Randbedingungen für die Position, die Orientierung und das Zielen mit dem *Endeffektor* berücksichtigt werden. Die Kombination von analytischen und numerischen Methoden erlaubt schnellere und zuverlässigere Algorithmen als konventionelle Lösungen, welche meistens auf der inversen Jacobimatrix oder optimierungsbasierten Techniken beruhen. Dadurch eignet sich diese Software auch gut für die Berechnung von Echtzeitanimationen. Eine Beschreibung von IKAN findet sich in [TGB00].

IKAN wurde speziell für die Verwendung mit *anthropomorphen* Gliedern wie den menschlichen Armen und Beinen entworfen. Dabei wird eine kinematische Kette mit sieben Freiheitsgraden angenommen. Für das Schulter- und Armgelenk werden jeweils drei Freiheitsgrade und für das Ellbogengelenk ein Freiheitsgrad angenommen. Wie in [TGB00] erläutert, ist ein Freiheitsgrad redundant und kann wie folgt physikalisch interpretiert werden. Wenn die Positionen des Handgelenks und der Schulter als fix angenommen werden, kann der Ellbogen weiterhin auf einem Kreis rotiert werden. Der Normalvektor dieses Kreises ist parallel zum Vektor zwischen Schulter- und Handgelenk. Abbildung 3.28 veranschaulicht diesen Sachverhalt. `L1` und `L2` bezeichnen die Länge des Ober- und Unterarms. Der Normalvektor wird mit `n` bezeichnet. Das Zentrum des Kreises ist mit `c`, sein Radius mit `R` beschriftet. Die Vektoren `u` und `v` formen das lokale Koordinatensystem der Ebene, in welcher der Kreis liegt.

Der Rotationswinkel des Ellbogens stellt für den Anwender eine sinnvolle physikalische Interpretation dar und kann als Vorgabe für die Berechnungen im IKAN-System angegeben werden (siehe `elbowSwivelAngel` auf Seite 93**)**.

3.2.2.1 IKAN-Bibliothek

IKAN ist in C++ geschrieben und kann einfach in jedes Softwaresystem integriert werden. Das IKAN-System beinhaltet den Quellcode für die inverse Kinematik-Bibliothek, ausführliche Dokumentation und einige Beispielapplikationen. Weiters kann IKAN einfach auf verschiedene UNIX- und Windows-Plattformen portiert werden. Die Klassen-Bibliothek ist hierarchisch aufgebaut und kann in mehrere Bereiche unterteilt werden.

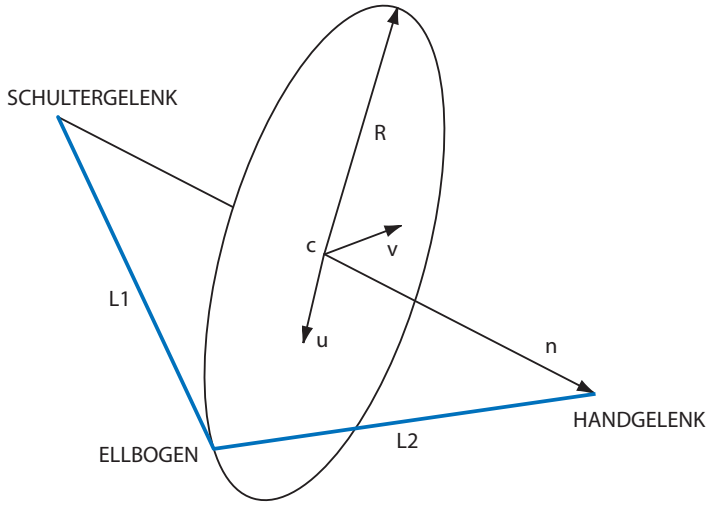


Bild 3.28: Bei einem vorgegebenen Ziel kann der Ellbogen entlang eines Kreises rotiert werden.

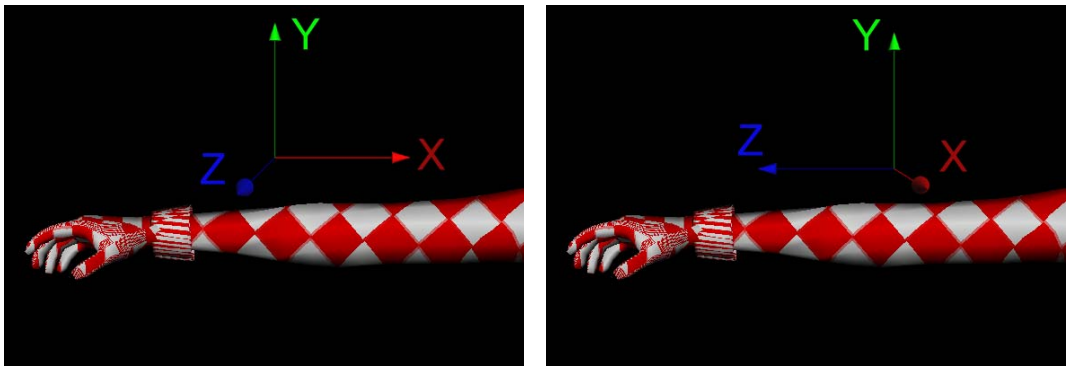
- Die Klassen `JackArm` und `JackLeg` sind höhere Klassen, welche für die Kontrolle der Arme und Beine des „EAI Jack Human“ Modells verwendet werden.
- Die Klasse `Limb` ist eine Hilfsklasse zwischen den *inverse Kinematik*-Klassen und den `JackArm`- und `JackLeg`-Klassen.
- Die Klasse `SRS` löst bei den gegebenen Matrizen G, S und T die Gleichung $G = R_2 * S * R_y * T * R_1$ für R_1, R_2 und R_y. Dabei sind R_1 und R_2 generelle Rotationsmatrizen und R_y eine Rotation um die Y-Achse. Im Falle des Armes ist R_1 das Schultergelenk, R_y das Ellbogengelenk, R_2 das Handgelenk, S die Transformation von der Hand zum Ellbogen, T die Transformation vom Ellbogen zur Schulter und G die gewünschte Lösungsmatrix.
- Jede Rotationsmatrix kann durch drei aufeinanderfolgende Rotationen durch die richtige Auswahl der Kombinationen von X-, Y- und Z-Achse erzeugt werden. Die Klasse `EulerPsiSolver` führt dies für ein System mit drei Gelenken durch, wobei ein Gelenk einfach und zwei Gelenke komplex sind.
- Um die Wertebereiche für die erlaubten Gelenkswinkel anzugeben und abzuspeichern, wird die Klasse `AngleInt` mit ihren Unterklassen `AngleIntList`, `AngleIntIterator` und `AngleIntListIterator` verwendet.
- Um plausible Lösungen mit Hilfe des *inversen Kinematik*-Systems zu berechnen, ist es notwendig, die Wertebereiche für die Gelenke auszuwerten. Zur Berechnung aller möglichen Winkel, welche den vorgegebenen Bedingungen genügen, wird ein analytischer Algorithmus verwendet. Dies erfolgt in den Klassen `SimpleJtLimit` und `ComplexJtLimit`. Wenn es zu einer gewünschten Vorgabe (z.B. Position des Endeffektors) keine Lösung gibt, wird auch dies vom zugrundeliegenden Algorithmus ermittelt.

3.2.2.2 Koordinatensystem

Das von IKAN verwendete Koordinatensystem unterscheidet sich vom OPEN INVENTOR-Koordinatensystem. Abbildung 3.29 zeigt das von OIV (a) und

IKAN (b) benutzte Koordinatensystem. Im IKAN-System ist das Koordinatensystem für den Ellbogen folgendermaßen aufgebaut:

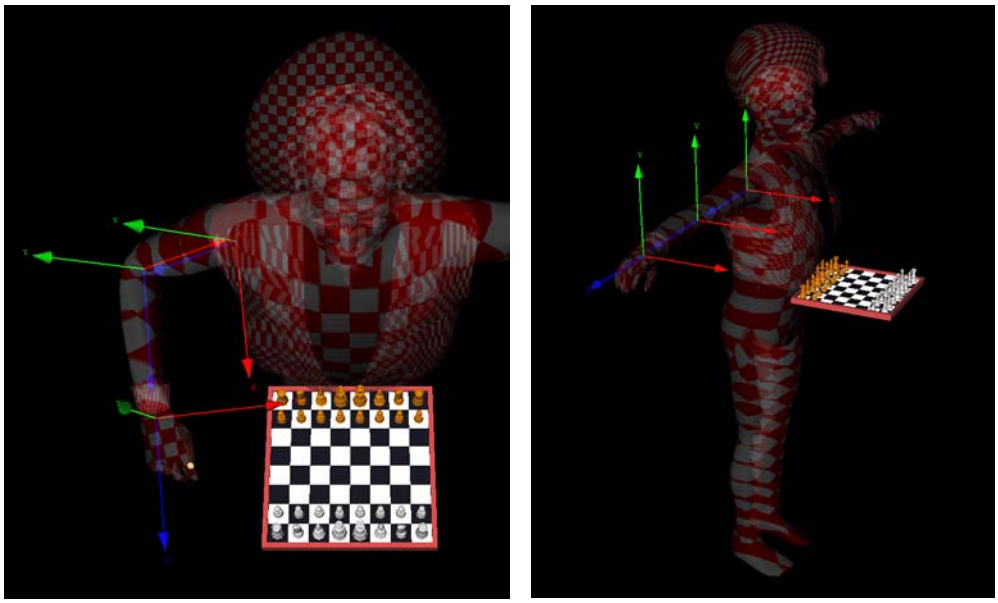
- Die Z-Achse verläuft vom Ellbogen zur Hand.
- Die Y-Achse ist senkrecht zu Z-Achse und ist die Rotationsachse des Gelenks.
- Die X-Achse zeigt vom Körper weg, sodaß die X-, Y- und Z-Achsen ein *rechtshändiges* Koordinatensystem bilden.



(a) OIV-Koordinatensystem. (b) IKAN-Koordinatensystem.

Bild 3.29: Gegenüberstellung der beiden Koordinatensysteme.

Gleichartige Koordinatensysteme werden für das Schulter- und das Handgelenk angenommen. Wie man sieht, lassen sich beide Koordinatensysteme einfach ineinander überführen. Durch eine 90 Grad-Rotation um die Y-Achse im Uhrzeigersinn kann das OIV- in das IKAN-Koordinatensystem transformiert werden.



(a) Abgewinkelter Arm. (b) Ausgestreckter Arm.

Bild 3.30: IKAN-Koordinatensystem im transparenten Modell des Schachspielers.

In Abbildung 3.30 sind die drei lokalen IKAN-Koordinatensysteme für Schulter, Ellbogen und Hand in einem abgewinkelten Arm (a) und in einem ausgestreckten Arm (b) zu sehen.

3.2.3 MeshDeformer

Zur Animation der Figur des „Türkischen Schachspielers“ wird das bereits in Kapitel 3.2.1 beschriebene *Skelettmodell* verwendet. Über das Skelett wird eine „Haut“ gelegt (siehe Abbildung 3.31) und mit Hilfe von Kontrollpunkten mit dem Skelett verbunden. Dies ermöglicht, daß bei einer Bewegung des Skeletts durch das *inverse Kinematik*-System die „Haut“ entsprechend den vorgegebenen Bedingungen den animierten Teilen des Skeletts folgt.

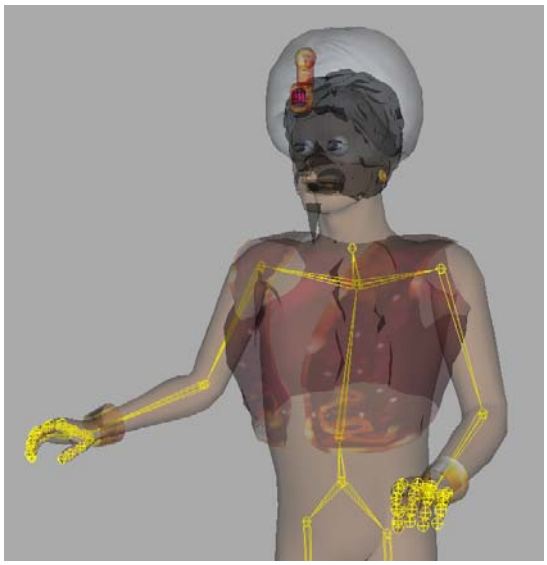


Bild 3.31: Transparentes Modell mit dem zugrundeliegenden Skelett.

Das Skelettmodell und die dazugehörende Geometrie wird mit dem Programm MAYA von ALIAS|WAVEFRONT erstellt. Für den Export wird eine MAYA-Erweiterung mit dem Namen CHARACTER EXPORTER verwendet. Diese Software wandelt die Daten aus dem internen MAYA-Format in ein für den OIV-Node Kit MESHDEFORMER geeignetes Datenformat um. Die Softwares CHARACTER EXPORTER und MESHDEFORMER wurden von der Firma IMAGINATION entwickelt und konnten für diese Arbeit verwendet werden. Abbildung 3.32 zeigt einen Ausschnitt des Bildschirms im Programm MAYA mit der Exporterweiterung CHARACTER EXPORTER.

Das Ergebnis des Exports ist eine OIV-Datei, welche u.a. die Geometrie der Haut, die Daten für das Skelett und die Verbindung zwischen Haut und Skelett mit den Gewichten enthält. Der folgende Ausschnitt aus der erzeugten Datei zeigt die *kinematische Kette* und die Rotationen für die Gelenke Schulter, Ellbogen und Hand. Dabei

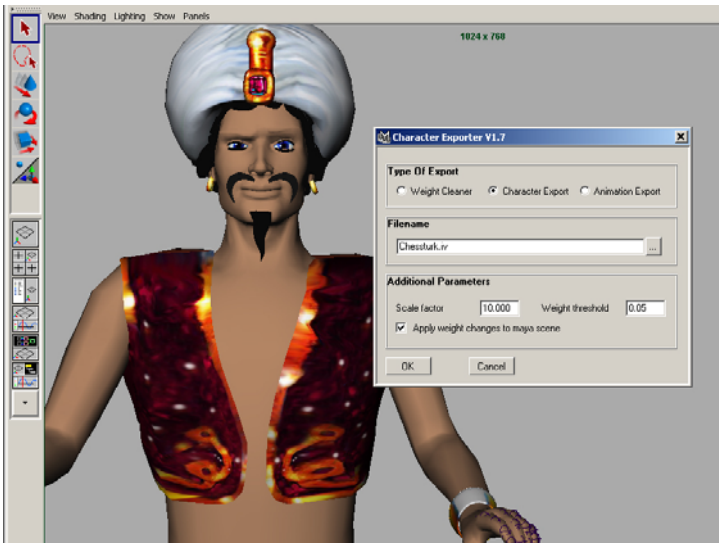


Bild 3.32: Export der virtuellen Figur aus der Modellierungssoftware Maya.

sind die Rotationen der drei Gelenke JRECHTSSCHULTER1, JRECHTSELLBOGEN1 und JRECHTSHAND1 bereits mit der OIV-Engine ANIMATIONENGINE verbunden.

```
# |Tuerke|jRumpf1|jHuefte1|jOberkoerper1|jRechtsSchulter1
Translation {  translation 0.130831747 0.02354414632 -0.02443981122 }
DEF JRECHTSSCHULTER1 Rotation {  rotation = USE
    ANIMATIONENGINE.rightShoulderRotation }

# |Tuerke|jRumpf1|jHuefte1|jOberkoerper1|jRechtsSchulter1|jRechtsEllbogen1
Translation {  translation 0.1783335694 -0.007303818856 0.01158487358 }
DEF JRECHTSELLBOGEN1 Rotation {  rotation = USE
    ANIMATIONENGINE.rightElbowRotation }

# |Tuerke|jRumpf1|jHuefte1|jOberkoerper1|jRechtsSchulter1|jRechtsEllbogen1|
# jRechtsHand1
Translation {  translation 0.1885710472 -0.003339511362 0.001063490849 }
DEF JRECHTSHAND1 Rotation {  rotation = USE
    ANIMATIONENGINE.rightWristRotation }
```

Um auf einfache Weise dem Node ANIMATIONENGINE Daten zur Verfügung zu stellen, wird eine Anpassung im Node Kit MESHDEFORMER vorgenommen. Er stellt nun die Transformation vom lokalen Nullpunkt des Skeletts zur rechten Schulter über ein Feld zur Verfügung.

```
shoulderTranslation = USE TURKMESH.rightShoulderTranslation
```

Die Erstellung des Modells des „Türkischen Schachspielers“ und der Export aus der 3D-Software MAYA wurden von Mitarbeitern der Firma IMAGINATION ausgeführt.

3.3 Interaktion

Für die *Interaktion* des Benutzers mit dem „Türkischen Schachspieler“ müssen die Positionen der Hand und des Kopfes des Betrachters ermittelt werden (siehe auch Kapitel 2.2). Das Kapitel 3.3.1 befaßt sich mit der verwendeten Software für die Verarbeitung der ermittelten Positions- und Orientierungsdaten. Im Abschnitt 3.3.2 wird auf das benutzte Programm für die optische Erkennung der Hand eingegangen. Die

verwendete Hard- und Software zur Bestimmung der Kopfposition des Benutzers wird in Kapitel 3.3.3 beschrieben.

3.3.1 OpenTracker

Im verwendeten AR-System STUDIERSTUBE wird das Tracking nicht von der Applikation, sondern vom objektorientierten Framework OPENTRACKER [RS01a] [RS01b] durchgeführt (siehe Kapitel 3.1.2.2). OPENTRACKER basiert auf einem *Datenflußkonzept*. Dabei wird die Manipulation der Daten in einzelne Schritte zerlegt. Die Daten der Tracker werden von OPENTRACKER empfangen, transformiert und an das verarbeitende Programm, in diesem Fall STUDIERSTUBE, weitergeleitet.

3.3.1.1 Konzept

Die Erzeugung und Verarbeitung von Daten geschieht in OPENTRACKER in sogenannten *Knoten* (engl. nodes). Diese *Knoten* sind in der Form einer *Baumstruktur* angeordnet und miteinander verbunden. Es wird zwischen drei verschiedenen Knotentypen unterschieden:

- *Source Nodes* sind die Blätter in der Baumstruktur und erhalten ihre Daten von externen Quellen (z.B. Geräten).
- *Sink Nodes* geben die Daten, welche sie von anderen Knoten empfangen haben, an externe Empfänger (z.B. an die STUDIERSTUBE) weiter.
- *Filter Nodes* sind dazwischenliegende Knoten, welche die von anderen Knoten empfangenen Daten verändern.

Die meisten *Source Nodes* beinhalten Treiber für externe Geräte, wie z.B. den POLHEMUS-Tracker. Diese externen Geräte sind häufig über eine Schnittstelle mit dem Computersystem verbunden. Die Source Nodes stellen deshalb auch Hilfsmittel für die Konfiguration und die Initialisierung dieser Geräte zur Verfügung. Andere Gruppen von Source Nodes erlauben es, komplexe eigenständige Systeme wie z.B. den ARTOOLKIT in OPENTRACKER einzubinden, Trackerdaten mit Hilfe der Tastatur zu erzeugen oder auf Daten über ein Netzwerk zuzugreifen. OPENTRACKER ermöglicht den Source Nodes das Ausführen ihres Codes im Rahmen eines multi-threading Modells. Dadurch können auch Geräte mit blockierter Ein- und Ausgabe effizient eingebunden werden.

Die *Sink Nodes* sind ähnlich wie die Source Nodes aufgebaut. Anders als Source Nodes liefern sie aber keine Daten, sondern geben die empfangenen Daten an eine Multicast-Gruppe im Netzwerk weiter oder erlauben die Integration von OPENTRACKER in andere Programme durch ein Shared Memory.

Die *Filter Nodes* empfangen Daten von einem oder mehreren Kindknoten. Aufgrund der empfangenen Information berechnet der Filter-Knoten seinen neuen Zustand. Die folgende Liste gibt einen Überblick über die gebräuchlichsten Filter:

- *Transformation Filter* führen geometrische Transformationen wie Translation, Rotation und Skalierung auf den Daten aus.

- *Prediction Filter* erlauben die teilweise Kompensation von verzögerten Trackerdaten, welche durch die Messung oder die Verarbeitung entstanden sind.
- *Noise Filter* gestatten die Korrektur von Meßungenauigkeiten, die beim Erfassen der Daten im Tracker entstanden sind.
- *Merge Filter* erlauben die Kombination von Daten, die von unterschiedlichen Kindknoten stammen. So kann z.B. die Orientierung eines Objektes durch einen Trägheits-Tracker (engl. inertial) und seine Position durch einen optischen Tracker ermittelt werden.
- *Conversion Filter* ermöglichen die Konvertierung von einem Datentyp in einen anderen. Durch Hinzufügen eines konstanten Wertes können z.B. die 2D-Daten einer Maus in eine 3D-Position umgewandelt werden.
- *Clamp Filter* sind nichtlineare Transformationsfilter und gestatten es, einen Wertebereich auf ein vom Benutzer vorgegebenes Minimum und Maximum zu beschränken.
- *Store-and-Forward Filter* sind dann nützlich, wenn mit dem vorübergehenden Ausfall der Trackingdaten gerechnet werden kann. Diese Filter wiederholen den zuletzt gültigen Wert, bis wieder zulässige Daten vorliegen.
- *Confidence Filter* wählen aus den Daten von verschiedenen Kindern jene mit einem bestimmten *Konfidenzwert* aus. Der *Konfidenzwert* gestattet im Allgemeinen eine Aussage über die Genauigkeit der Daten.

Durch das Hinzufügen von neuen Knoten kann das System leicht erweitert und angepaßt werden.

3.3.1.2 Konfiguration

Die Beschreibung einer OpenTracker-Konfiguration erfolgt durch das Erstellen einer XML-Datei. XML (eXtensible Markup Language) ist eine Beschreibungssprache, die es erlaubt, durch die Angabe einer *Document Type Definition* (DTD) wiederum eine hierarchische Beschreibungssprache zu definieren [HM01].

Durch die Verwendung einer geeigneten DTD können Standardwerkzeuge für das Editieren, Verifizieren und die Syntaxanalyse von beliebigen XML-Dateien benutzt werden. Die Bereitstellung einer DTD, welche die *Datenflußgraphen* der OpenTracker-Knoten auf eine XML-Struktur abbildet, erleichtert die Erstellung einer Konfiguration.

Die folgende XML-Datei zeigt eine Konfiguration, die während der Entwicklung der Software verwendet wurde. Abbildung 3.33 zeigt den aus der Datei resultierenden Graphen.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OpenTracker SYSTEM "../studierstube/opentracker.dtd">
<OpenTracker>
    <configuration>
        <ConsoleConfig headerline="Showcase" interval="20" display="off"/>
    </configuration>
    <StbSink station="0">
        <ConsoleSink comment="Pen">
            <EventTransform translation="0.0 0.05 0.0" rotation=
             "1 0 0 -1.57" rotationtype="axisangle" scale="0.75 1 -0.7">
                <EventVirtualTransform rotation="1 0 0 -1.57"
                 rotationtype="axisangle" translation="-0.5 -0.5 0">
                    <StbMouseSource window="1" mode="relative"/>
                </EventVirtualTransform>
            </EventTransform>
        </ConsoleSink>
    </StbSink>
    <StbSink station="1">
        <ConsoleSink comment="Pip">
            <StbKeyboardSource number="1"/>
        </ConsoleSink>
    </StbSink>
</OpenTracker>
```

Als *Source Nodes* werden `StbKeyboardSource` und `StbMouseSource` verwendet. Diese beiden Knoten erhalten ihre Daten von der Tastatur und der Maus. Die *Filter Nodes* `EventVirtualTransform` und `EventTransform` führen die erforderlichen Transformationen auf den Daten aus. Die *Sink Nodes* mit dem Namen `ConsoleSink` geben die empfangenen Daten in einem Textfenster z.B. für Diagnosezwecke aus. Über die *Sink Nodes* `StbSink` gelangen die Daten von OPENTRACKER in die STUDIERSTUBE-Anwendung.

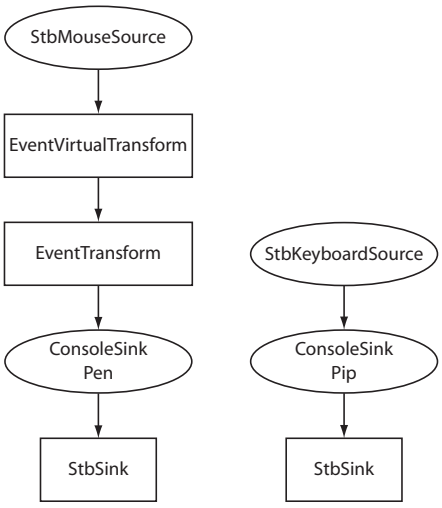


Bild 3.33: Graph des OpenTracker-Konfigurationsbeispiels.

3.3.1.3 Verteiltes Tracking

OPENTRACKER ermöglicht es mehreren Sendern und Empfängern von Tracker-Daten, über ein Netzwerk miteinander zu kommunizieren. Dabei werden die Daten asynchron mit Hilfe des IP-Multicast-Protokolls ausgetauscht. Diese Vorgehensweise hat unter anderem folgende Vorteile:

- Die Trackerdaten können auf mehreren Rechnersystemen verwendet werden. Das ist vor allem in *verteilten* virtuellen Umgebungen von Nutzen, z.B. bei einer STUDIERSTUBE-Anwendung, die auf mehreren Rechnersystemen läuft.
- Durch diesen Ansatz können auch kostengünstige Computersysteme im Rahmen eines Multi-Processing-Konzeptes verwendet werden. Die zu lösende Aufgabe wird auf mehrere Rechner verteilt (engl. load balancing), wodurch eine bessere Berechnung von vor allem rechenintensiven Funktionen (wie z.B. komplexe Filter) erreicht wird.
- Der Einsatz eines Netzwerkes erleichtert den Aufbau einer heterogenen Rechnerumgebung mit verschiedenen Betriebssystemen.

Die Abbildung 3.34 zeigt eine Beispielskonfiguration. Zwei getrennte Tracker-

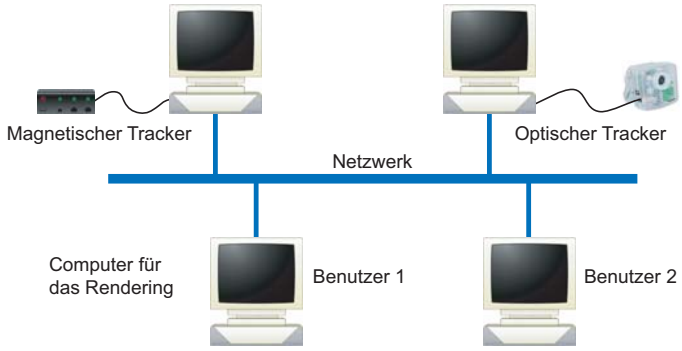


Bild 3.34: OpenTracker ermöglicht verteiltes Tracking.

Systeme liefern die Daten von einem magnetischen und einem optischen Sensor. Die ermittelten Tracker-Daten werden an die beiden für das Rendering zuständigen Computer über ein Netzwerk übermittelt.

Neben dem Multicast-Protokoll stehen auch weitere IP-Protokolle für eine Übertragung der Daten zur Verfügung. Für dieses Projekt wurde das Multicast-Protokoll zur Anbindung der *Handtracking*-Software (siehe Kapitel 3.3.2) verwendet.

3.3.2 Handtracking-Software

Für die Ermittlung der Position und Haltung der Hand des Benutzers wird eine im Rahmen dieses Projektes von der Firma IMAGINATION entwickelte Software verwendet. Die Software ist als eigenständiges Programm ausgeführt und übermittelt die errechneten Daten über ein Multicast-Protokoll an OPENTRACKER.

Für die Aufnahme des Videosignals wird eine *FireWire-Videokamera* der Firma UNIBRAIN eingesetzt. Diese ist über eine FIREWIRE-PCI-Karte an den Computer ange-

schlossen. Die Abbildung 3.35 (a) zeigt die Videokamera. Das Bild (b) zeigt einen Testaufbau der Kamera.

(a) FireWire-Videokamera. (b) Testaufbau mit der Kamera auf einem Stativ.

Bild 3.35: Videokamera für die Handtracking-Software.

Das Videobild wird von der Handtracking-Applikation mit Hilfe von Bildverarbeitungsverfahren analysiert. Die Software liefert neben der Position und der Orientierung der Hand auch die Information, ob sie offen oder geschlossen ist. Die Position der Hand wird verwendet, um die gewünschte Schachfigur am Schachbrett auszuwählen. Der ermittelte Zustand der Hand (offen/geschlossen) wird verwendet, um eine Schachfigur aufzunehmen oder abzustellen. Der folgende Ausschnitt aus der OPENTRACKER Konfigurationsdatei zeigt die Einbindung der Handtracking-Software in das Gesamtsystem. Die Daten vom Handtracking-Programm werden über den `NetworkSource` *Node* von OPENTRACKER empfangen. Danach werden die erforderlichen Filter und Transformationen auf den Daten ausgeführt.

```
<StbSink station="0">
    <ConsoleSink comment="Pen">
        <EventTransform translation="0.0 0.04 0.16" scale="0.8 1.0 1.2">
            <EventOrientationTransform rotationtype="quaternion"
             rotation="-1 -1 -1 1">
                <EventMatrixTransform matrix="1  0  0  0
                                              0  0  1  0
                                              0 -1  0  0
                                              0  0  0  1">
                    <Filter weight="0.25 0.25 0.25 0.25" type="all">
                        <EventQueue length="4">
                            <NetworkSource number="0"
                             multicast-address="224.0.0.3" port="3333"/>
                        </EventQueue>
                    </Filter>
                </EventMatrixTransform>
            </EventOrientationTransform>
        </EventTransform>
    </ConsoleSink>
</StbSink>
```

Da die Handtracking-Software eine eigene Applikation ist, welche die Daten über ein Netzwerk übermittelt, kann sie auf dem gleichen Rechner wie das AR-System

STUDIERSTUBE laufen oder auf einem anderen Computer zur besseren Lastverteilung betrieben werden.Während der Entwicklung des „Türkischen Schachspielers“ wurden alle Softwarekomponenten auf dem gleichen Rechner ausgeführt.

3.3.3 Head Tracking

Für die realistische Darstellung der virtuellen Szene ist es wichtig, die Kopfposition des Betrachters festzustellen. Bei einer Veränderung der Kopfposition soll der Bildinhalt entsprechend dem neuen Blickpunkt berechnet und ausgegeben werden. Für die Bestimmung der Kopfposition wird der optische Tracker DYNASIGHT der Firma ORIGIN verwendet. Die Abbildung 3.36 zeigt ein Bild des Gerätes. Mit Hilfe eines *retroreflektierenden Markers* wird die Kopfposition gemessen. Der Marker ist auf der vom Betrachter benutzten Stereobrille angebracht.

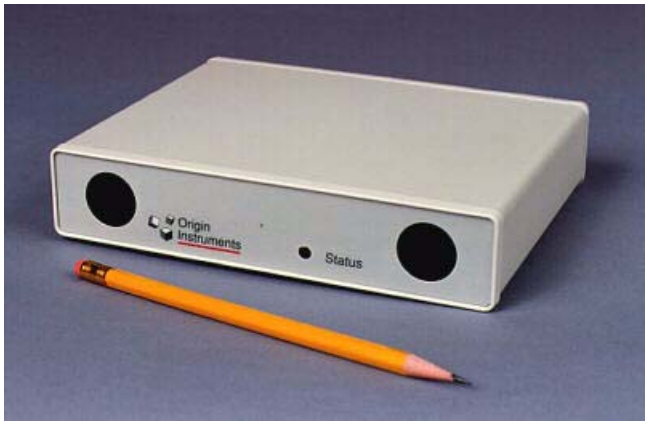


Bild 3.36: Optischer Tracker DynaSight der Firma Origin.

Vom Tracker werden die X-, Y- und Z-Positionen des Markers im lokalen Koordinatensystem des Gerätes gemessen. Das Koordinatensystem hat die gleiche Ausrichtung wie das von OIV verwendete Koordinatensystem. Der Nullpunkt des Trackerkoordinatensystems ist durch eine Markierung auf der Vorderseite des Gerätes gekennzeichnet. Der Tracker liefert seine Daten über eine serielle Schnittstelle an den Computer.

Die Einbindung des DYNASIGHT-Trackers in OPENTRACKER erfolgt durch einen für dieses Projekt entwickelten *Source Node* mit dem Namen `DynaSightSource`.

Der folgende Ausschnitt aus einer OPENTRACKER Konfigurationsdatei zeigt den Einsatz des `DynaSightSource` Knoten und seine Konfiguration.

```
<configuration>
    <ConsoleConfig headerline="Showcase" interval="20" display="on"/>
    <DynaSightConfig device="com1"/>
</configuration>
<StbSink station="2">
    <ConsoleSink comment="Camera">
        <Merge>
            <MergeDefault>
                <Filter weight="0.75 0.12 0.08 0.05" type="all">
                    <EventQueue length="4">
                        <EventTransform translation="-0.023 0.545 -0.07">
                            <EventTransform rotation="1 0 0 0.66322511"
                             rotationtype="axisangle">
                                <DynaSightSource target="0"/>
                            </EventTransform>
                        </EventTransform>
                    </EventQueue>
                </Filter>
            </MergeDefault>
            <MergeOrientation>
                <TestSource orientation="0 0 0 1"/>
            </MergeOrientation>
        </Merge>
    </ConsoleSink>
</StbSink>
```

In der Konfigurationsbeschreibung `DynaSightConfig` wird die serielle Schnittstelle, über die der Tracker an den Rechner angeschlossen ist, angegeben. Da der Tracker beim Einsatz von aktiven Markern zwischen mehreren Markern unterscheiden kann, muß eine Auswahl des gewünschten Markers mit dem Parameter `target="0"` erfolgen. Dabei bestimmt die Zahl den auszuwertenden Marker. Aktive Marker sind kabelgebunden und mit Leuchtdioden ausgestattet.

Um die Genauigkeit der vom optischen Tracker ermittelten Positionsdaten zu bestimmen, wurden zwei Meßreihen durchgeführt. Dafür wurde der retroreflektierende Marker auf einer Position fixiert und diese mit einem Maßband vermessen. Anschließend wurde die Position mit dem optischen Tracker gemessen und der Wert abgelesen. Die Tabellen 3.3 und 3.4 zeigen die erhaltenen Werte für die XZ-Achsen und die YZ-Achsen. Die Spalte „Maßband“ enthält den mit dem Maßband gemessenen Wert. In der Spalte „DynaSight“ findet sich der vom optischen Tracker ermittelte Wert. Die Spalte „Differenz“ gibt die Differenz der beiden Werte an. Die Werte wurden im lokalen Koordinatensystem des DYNASIGHT-Trackers gemessen. Die Messung wurde so durchgeführt, daß für den Wert der „fehlenden“ Achse vom Tracker der Wert Null gemessen wurde; z.B. für die XZ-Messung war der Y-Wert gleich Null. Abbildung 3.37 zeigt das lokale Koordinatensystem des DYNASIGHT-Tracker und die beiden Ebenen in denen die Messungen durchgeführt wurden.

Die Abbildungen 3.38 bis 3.41 veranschaulichen die Größe des Fehlers zwischen dem von DYNASIGHT ermittelten Ergebnis und dem händisch gemessenen Wert. Der

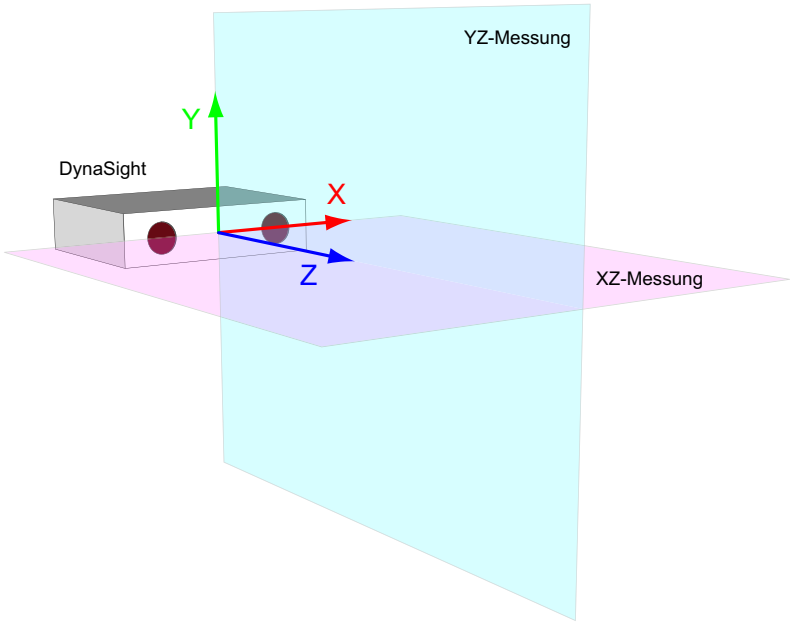


Bild 3.37: Lokales Koordinatensystem des DynaSight-Tracker.

Durchmesser des Kreises im Diagramm korrespondiert mit der Größe des Fehlers. Die Farbe eines Kreises gibt an, ob der Fehlerwert positiv (violett) oder negativ (weiß) ist.

Abstand Z-Achse [mm]			**Abstand X-Achse [mm]**		
Maßband	**DynaSight**	**Differenz**	**Maßband**	**DynaSight**	**Differenz**
800,00	790,00	10,00	-232,00	-230,00	-2,00
800,00	793,00	7,00	19,00	17,30	1,70
800,00	827,00	-27,00	247,00	252,00	-5,00
800,00	809,00	-9,00	513,00	518,00	-5,00
800,00	721,00	79,00	589,00	523,00	66,00
1200,00	1197,00	3,00	-232,00	-232,00	0,00
1200,00	1163,00	37,00	0,00	-2,50	2,50
1200,00	1181,00	19,00	19,00	17,60	1,40
1200,00	1197,00	3,00	589,00	584,00	5,00
1200,00	1202,00	-2,00	247,00	246,00	1,00
1200,00	1170,00	30,00	513,00	497,00	16,00
1200,00	1205,00	-5,00	779,00	779,00	0,00
2000,00	1979,00	21,00	-232,00	-231,10	-0,90
2000,00	1960,00	40,00	0,00	-1,70	1,70
2000,00	1956,00	44,00	19,00	17,50	1,50
2000,00	1972,00	28,00	247,00	242,00	5,00
2000,00	1971,00	29,00	513,00	504,00	9,00
2000,00	1972,00	28,00	779,00	768,00	11,00

Tabelle 3.3: Differenz der ermittelten Positionsdaten auf der X- und Z-Achse.

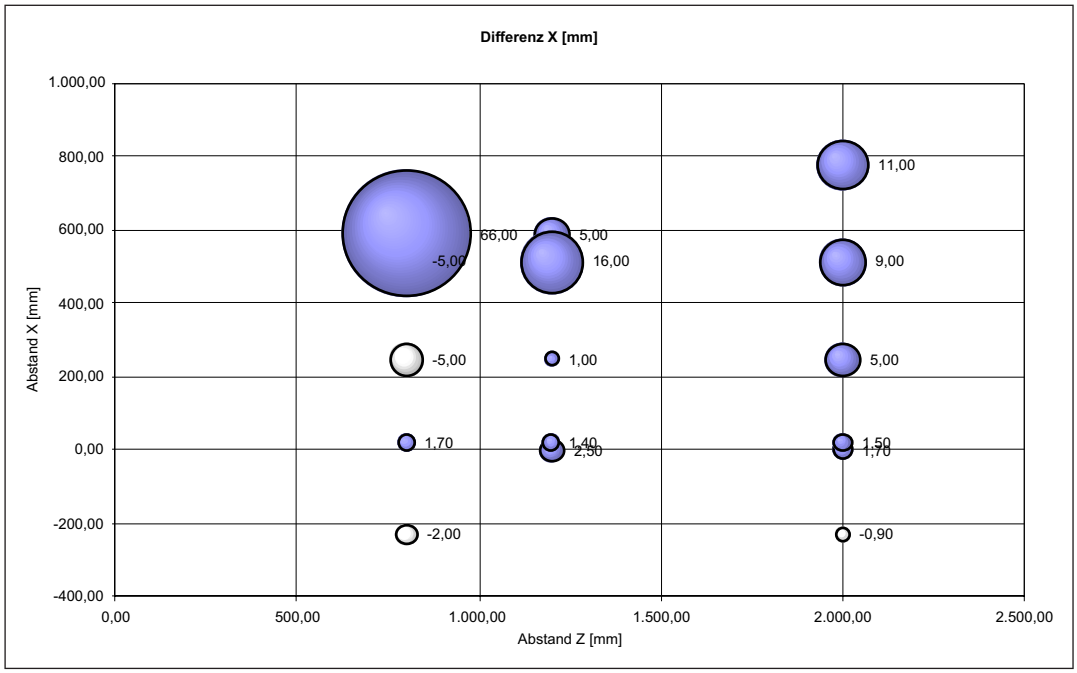


Bild 3.38: Differenz der X-Werte bei einem gegebenen X- und Z-Wert.

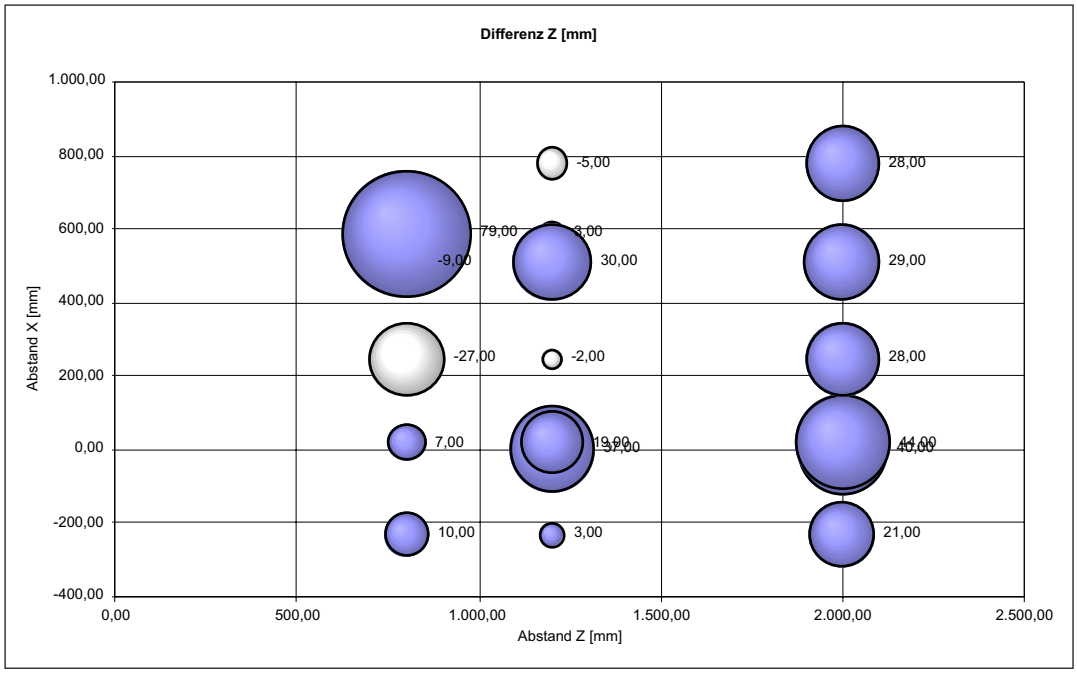


Bild 3.39: Differenz der Z-Werte bei einem gegebenen X- und Z-Wert.

Abstand Z-Achse [mm]			Abstand Y-Achse [mm]		
Maßband	**DynaSight**	**Differenz**	**Maßband**	**DynaSight**	**Differenz**
450,00	446,90	3,10	-303,00	-305,50	2,50
450,00	449,00	1,00	-171,00	-172,60	1,60
450,00	452,00	-2,00	-19,00	-21,00	2,00
450,00	451,60	-1,60	13,00	7,90	5,10
450,00	449,90	0,10	149,00	144,80	4,20
450,00	449,60	0,40	263,00	257,90	5,10
750,00	737,00	13,00	-571,00	-559,90	-11,10
750,00	742,40	7,60	-437,00	-438,50	1,50
750,00	745,00	5,00	-305,00	-307,30	2,30
750,00	744,00	6,00	-171,00	-173,80	2,80
750,00	744,00	6,00	-19,00	-21,70	2,70
800,00	793,60	6,40	-438,00	-440,20	2,20
800,00	796,00	4,00	-305,00	-307,70	2,70
800,00	794,50	5,50	-171,00	-173,90	2,90
800,00	794,10	5,90	-20,00	-22,00	2,00
800,00	797,70	2,30	12,00	5,50	6,50
800,00	799,40	0,60	148,00	141,90	6,10
800,00	800,60	-0,60	281,00	274,80	6,20
800,00	800,30	-0,30	414,00	406,50	7,50
800,00	766,50	33,50	510,00	479,10	30,90
1200,00	1191,70	8,30	-571,00	-574,00	3,00
1200,00	1194,20	5,80	-437,00	-440,60	3,60
1200,00	1194,20	5,80	-304,00	-307,90	3,90
1200,00	1194,70	5,30	-172,00	-175,40	3,40
1200,00	1189,10	10,90	-20,00	-24,40	4,40
1200,00	1193,70	6,30	11,00	1,90	9,10
1200,00	1192,60	7,40	149,00	139,10	9,90
1200,00	1189,00	11,00	282,00	270,20	11,80
1200,00	1193,10	6,90	415,00	403,50	11,50
1200,00	1197,80	2,20	548,00	538,40	9,60
1200,00	1182,10	17,90	700,00	681,40	18,60
2000,00	1962,00	38,00	-840,00	-837,50	-2,50
2000,00	1968,40	31,60	-706,00	-708,90	2,90
2000,00	1968,60	31,40	-570,00	-573,40	3,40
2000,00	1975,70	24,30	-437,00	-441,80	4,80
2000,00	1967,60	32,40	-304,00	-307,60	3,60
2000,00	1954,90	45,10	-171,00	-175,30	4,30
2000,00	1956,30	43,70	-19,00	-26,70	7,70
2000,00	1978,00	22,00	11,00	-4,68	15,68
2000,00	1983,50	16,50	149,00	131,60	17,40
2000,00	1986,50	13,50	282,00	264,10	17,90
2000,00	1991,40	8,60	416,00	397,20	18,80
2000,00	1969,30	30,70	548,00	522,60	25,40
2000,00	1992,30	7,70	700,00	678,30	21,70

Tabelle 3.4: Differenz der ermittelten Positionsdaten auf der Y- und Z-Achse.

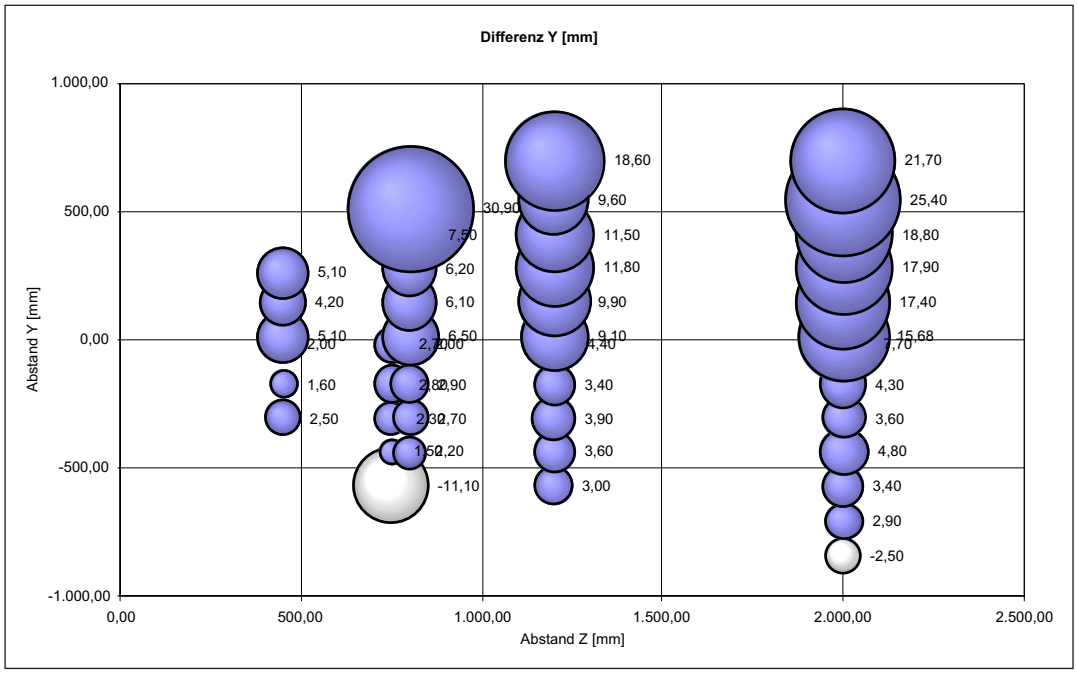


Bild 3.40: Differenz der Y-Werte bei einem gegebenen Y- und Z-Wert.

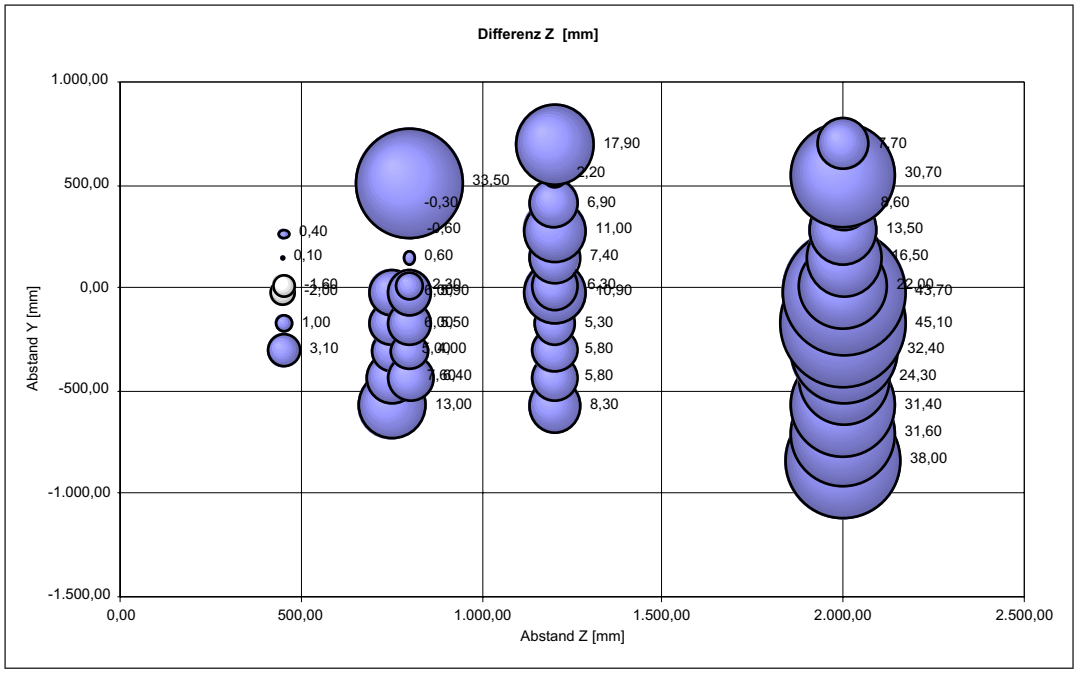


Bild 3.41: Differenz der Z-Werte bei einem gegebenen Y- und Z-Wert.

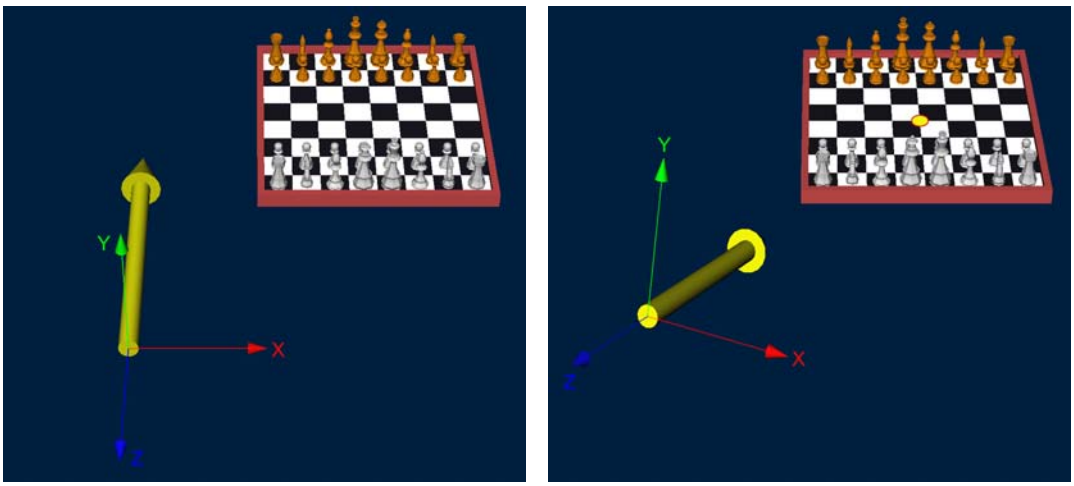
Die Messungen haben gezeigt, daß die vom DYNASIGHT-Tracker gelieferten Daten teilweise mit systematischen Meßfehlern behaftet sind. So kann man z.B. in Abbildung 3.40 erkennen, daß der Meßfehler im positiven Abschnitt der Y-Achse größer ist als im negativen Abschnitt. Weiters sieht man, daß an der Grenze des vom optischen Tracker abgedeckten Meßbereiches die Meßfehler größer werden. Auf Grund der in Tabelle 3.3 und 3.4 ermittelten Werte wäre eine Verbesserung der Daten möglich. Den Ausgangspunkt für eine Korrektur sollten mehrere Meßreihen mit präziseren Daten bilden. Die Veränderung der Daten könnte durch einen neuen *Filter Node* in

OPENTRACKER erfolgen. Für die Messung wurde ein retroreflektierender Marker mit einem Durchmesser von 25mm verwendet.

Wie bereits oben beschrieben, liefert der DYNASIGHT-Tracker nur die Positionsdaten des Markers zurück. Der im Rahmen dieses Projektes verwendete OIV-Knoten `SoOffAxisCamera` [SF03] aus dem STUDIERSTUBE AR-System nutzt aber auch die Orientierungsinformation in den OPENTRACKER-Daten zur Berechnung des Bildes. Eine weitere Konfigurationsoption in `DynaSightConfig` erlaubt nun die zusätzliche Berechnung von Orientierungsdaten. Dazu wird die Konfigurationsbeschreibung um den Text `lookat="0.0 0.0 0.0"` erweitert. Die drei Zahlen geben eine Position im lokalen Koordinatensystem des Trackers an. Die Reihenfolge der Achsen ist X, Y und Z. Folgender Auszug aus einer OPENTRACKER-Konfigurationsdatei soll den Einsatz verdeutlichen.

```
<configuration>
    <ConsoleConfig headerline="Showcase" interval="20" display="on"/>
    <DynaSightConfig device="com1" lookat="-0.5 2.2 3.5"/>
</configuration>
```

Wird die `lookat`-Option nicht verwendet, ist die vom `DynaSightSource` Knoten gelieferte Orientierung immer parallel zur Z-Achse des Trackers (siehe Abbildung 3.42 (a)). Bei Verwendung der Option wird die Orientierung so berechnet, daß die lokale Z-Achse durch die gemessene Position und die vorgegebene `lookat`-Position geht. Dies soll Abbildung 3.42 (b) zeigen. Die vorgegebene `lookat`-Position befindet sich hier in der Mitte des Schachbrettes. Diese Art der Berechnung der Orientierungsdaten ist keine Lösung, die alle Anwendungsfälle abdecken kann. Für die vorliegende Anwendung ist sie aber ausreichend, da angenommen werden kann, daß der Betrachter seine Aufmerksamkeit hauptsächlich dem Schachbrett mit den darauf befindlichen Figuren widmet.



(a) Ohne LookAt-Parameter. (b) Mit LookAt-Parameter.

Bild 3.42: Darstellung der vom DynaSight Node errechneten Orientierung.

Um die Unterschiede in den berechneten Bildern zu verdeutlichen, wurde das Differenzbild einer Stereo-Darstellung des Schachbretts mit und ohne `lookat`-Option gebildet. Die Abbildung 3.43 zeigt das Differenzbild für das linke Auge. Für die Er-

stellung des Differenzbildes wurden die beiden Bilder in einem Bildbearbeitungsprogramm voneinandern subtrahiert. Die unterschiedlichen Grauwerte geben die Größe der Abweichung zwischen den beiden Bildern an.

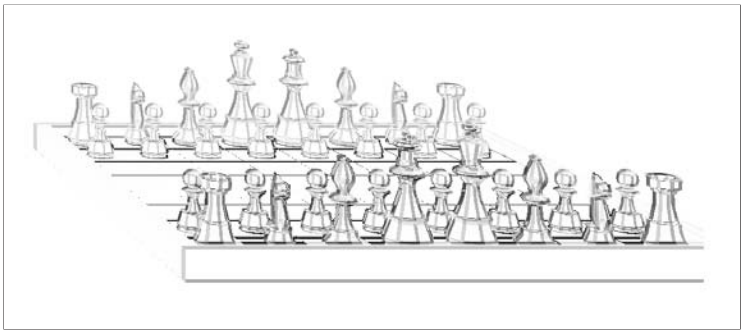


Bild 3.43: Differenz der Bilder (mit und ohne LookAt-Parameter) für das linke Auge.

3.4 Simulation

Für die Realisierung des „Türkischen Schachspielers“ wird auch ein *Schachprogramm* benötigt, welches es der virtuellen Figur erlaubt, gegen einen realen Spieler eine Partie Schach zu spielen. Als Schachprogramm wird das frei verfügbare GNU CHESS verwendet. Es ist als Sourcecode oder bereits übersetztes Programm für verschiedene Rechnersysteme erhältlich (siehe Link-Sammlung). Für dieses Projekt wird die Version 5.06 des Programmes für die INTEL/MICROSOFT Windows-Plattform verwendet.

Das Programm GNU CHESS hält sich an die Regeln für das Schachspiel, welche unter anderem in [Fid03] zu finden sind. GNU CHESS verwendet Textein- und Ausgabe zur Kommunikation mit dem Spieler oder einem anderen Programm. Wird beim Starten des Schachprogrammes der Parameter „`-x`“ angegeben, dann verwendet die Anwendung das *Chess Engine Communication Protocol* zur Kommunikation. Dieses Protokoll wird in [Man03] detailliert beschrieben. Es dient vor allem zur Kommunikation des Schachprogramms mit einem grafischen Anwenderprogramm wie z.B. der MICROSOFT Windows-Applikation WINBOARD.

Das folgende Protokoll zeigt die Interaktion eines Benutzers mit dem Schachprogramm im Textmodus. Die in Fettschrift hervorgehobenen Textstellen stellen die Eingaben des Benutzers dar.

```
Z:\gnuchess>gnuchess506 -x
Chess
f2f4
1. f2f4
1. ... e7e5
My move is: e7e5
g2g4
2. g2g4
2. ... d8h4
My move is: d8h4
0-1 {computer wins as black}
show board

white  KQkq
r n b . k b n r
p p p p . p p p
. . . . . . . .
. . . . p . . .
. . . . . P P q
. . . . . . . .
P P P P P . . P
R N B Q K B N R

quit

Z:\gnuchess>
```

Die Schachzüge werden in *Coordinate Algebraic Notation* angegeben. Dabei ist `f2f4` ein normaler Zug einer Figur vom Feld `f2` auf das Feld `f4`. Die Promotion eines *Bauern* zu einer *Dame* wird als `e7e8q` geschrieben. Die verschiedenen Varianten der Rochaden sind `e1g1`, `e1c1`, `e8g8` und `e8c8`.

Tätigt der Anwender einen ungültigen Schachzug, wird dies durch die Meldung ”Illegal move: e2e4“ vom Schachprogramm angezeigt. Dabei ist `e2e4` der ungültige Schachzug. Der von GNU CHESS berechnete Zug wird als ”My move is: e7e5“ ausgegeben. Das Ende einer Schachpartie wird durch einen der folgenden vier Texte angezeigt: ”0-1 {computer wins as black}“, ”1-0 {computer loses as black}“, ”1/2-1/2 {draw}“ und ”1/2-1/2 {stalemate}“.

Zur Einbindung des Schachprogramms in die STUDIERSTUBE-Anwendung wurde ein OIV-Knoten vom Typ *Engine* entwickelt (siehe Kapitel 4.2.5). Dieser OIV-Knoten kommuniziert mittels Ein-/Ausgabeumlenkung mit dem Schachprogramm und über Felder mit den anderen OIV-Knoten.

Im Rahmen dieses Projektes spielt der menschliche Gegner des „Türkischen Schachspielers“ immer mit den weißen Figuren und hat bei jeder Schachpartie den ersten Zug.

Kapitel 4

Implementierung

„Der Schachspieler des Herrn von Kempelen war ein Kunstwerk, durch dessen Vorzeigen das Publicum angenehm unterhalten ward, ohne daß die Geheimhaltung seiner inneren Structur nachteilig war.“

– Joseph Friedrich Freyherr zu Racknitz, aus dem Buch *Über den Schachspieler des Herrn von Kempelen und dessen Nachbildung*, 1789

4.1 Hardwarekomponenten

Der Hardwareaufbau für den „Türkischen Schachspieler“ besteht aus sechs Hauptkomponenten: den beiden Videoprojektoren, der Projektionsfläche, dem Rechnersystem, dem optischen Tracker, der Videokamera und der Stereobrille für den Benutzer. Die Abbildung 4.1 zeigt in einem Übersichtsdiagramm die verwendeten Hardwarekomponenten. In Bild 4.2 ist der Hardwareaufbau des Prototypen zu sehen.

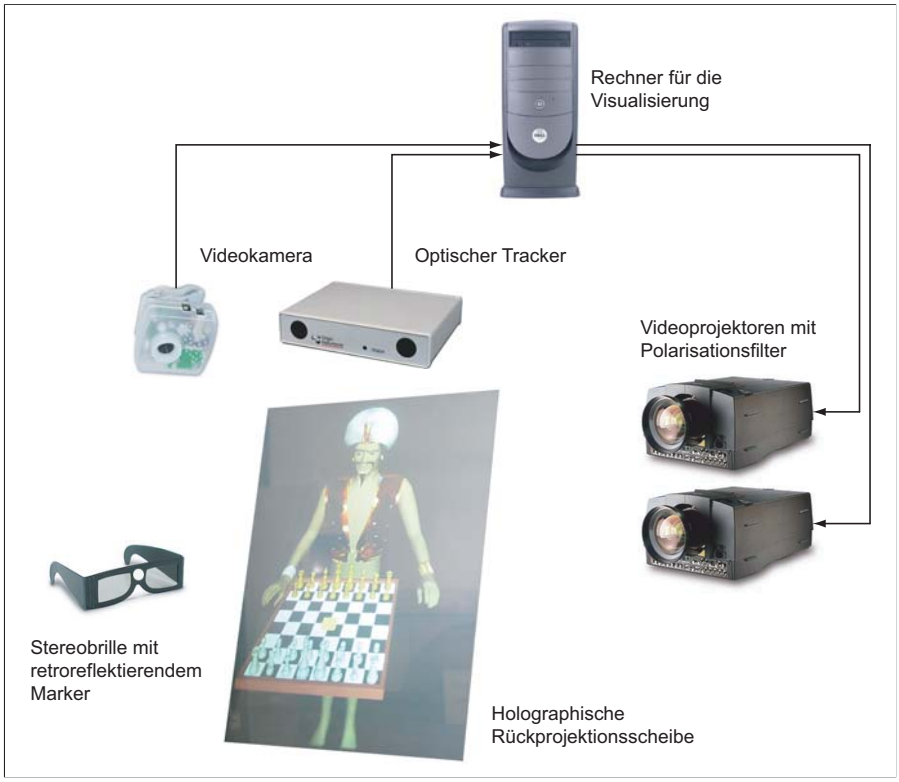


Bild 4.1: Übersicht der Hardwarekomponenten.

Als *Videoprojektoren* werden zwei BARCO BARCOREALITY SIM 6 ULTRA MM mit *Trapezausgleich* (engl. keystone correction) und *Scheimpflug-Korrektur* verwendet (siehe auch Kapitel 3.1.3.2). Die technischen Informationen zu diesen Geräten finden sich in [Bar02a]. Beide Projektoren haben *lineare* Polarisationsfilter in ihrem Linsensystem installiert (siehe auch Kapitel 3.1.3.1). Die Polarisationsebenen der beiden Filter sind *vertikal* und *horizontal* ausgerichtet. Die beiden Videoprojektoren sind in Bild 4.4 (a) im Detail zu sehen.

Bild 4.2: Hardwareaufbau des erweiterten Prototypen.

Für die *Projektionsfläche* wird das Produkt HOPS der Firma SAX3D.COM GMBH verwendet. Diese holographische Rückprojektionsfolie ist auf einem Trägermaterial aufgebracht (z.B. Acrylglas). Details zu den optischen Eigenschaften finden sich in Tabelle 3.2, "Überprüfte holographische Rückprojektionsfolien." auf Seite 50. Abbildung 4.3 veranschaulicht die Anordnung der beiden Videoprojektoren relativ zur Rückprojektionsfläche.

Die Bilder für die beiden Videoprojektoren werden von einem PC (DELL Dimension 8250; 2.4GHz, 512MB RAM, MICROSOFT Windows XP Professional SP1) geliefert. Die beiden analogen Ausgänge der Videokarte (Quadro4 900 XGL; 128MB) sind mit den beiden Videoprojektoren verbunden.

Der *optische Tracker* und die *FireWire-Videokamera* sind oberhalb der Rückprojektionsscheibe montiert und mit dem Rechnersystem über ein serielles und ein FIREWIRE-Kabel verbunden. Die Abbildung 4.4 (b) zeigt den optischen DYNASIGHT-Tracker und die UNIBRAIN FIREWIRE-Videokamera.

Die vom Anwender getragene *Stereobrille* mit den beiden Polarisationsfiltern und dem retroreflektierenden Marker ist in Bild 4.5 (b) zu sehen. Auf Grund der schlechten Beleuchtungsverhältnisse beim Test der Anwendung wird in Abbildung 4.5 ein weißer Handschuh zur Erhöhung des Kontrastes für die Handtracking-Software verwendet.

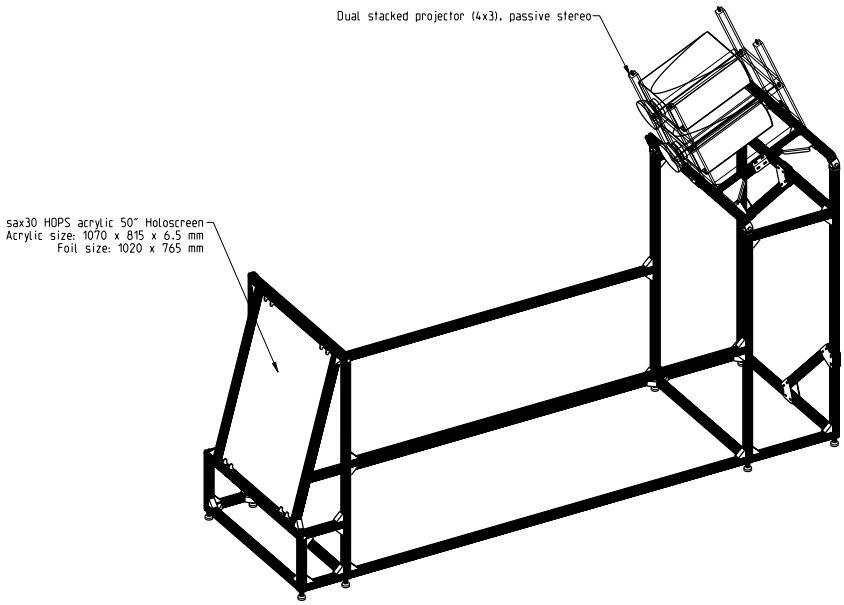


Bild 4.3: Teilansicht des erweiterten Prototypen aus der Konstruktionszeichnung (mit freundlicher Unterstützung von BARCO).

(a) BARCO Stereo-Videoprojektoren.

(b) DynaSight-Tracker und FireWire-Videokamera.

Bild 4.4: Detailansichten der Hardwareinstallation.

Die Rahmenkonstruktion zur Unterbringung der beiden Videoprojektoren und der Rückprojektionsscheibe wurde von der Firma BARCO gefertigt. Weiters wurde die Installation und Einrichtung der Videoprojektion von einem BARCO-Mitarbeiter durchgeführt.

4.2 Softwarekomponenten

Eine Übersicht der Softwarekomponenten ist in Abbildung 4.6 zu sehen. Eine zentrale Rolle nimmt dabei das AR-System STUDIERSTUBE ein. Die Daten aus dem Handtracking und die Kopfposition werden dem AR-System über OPENTRACKER zur Verfügung gestellt. Die Handtracking-Software übermittelt ihre Daten über eine Netzwerkverbindung direkt an OPENTRACKER (siehe Kapitel 3.3.2). Die Daten für die Kopfposition, welche vom optischen Tracker ermittelt werden, gelangen über den

(a) Der Anwender bewegt eine Schachfigur. (b) Stereobrille mit Marker.

Bild 4.5: Interaktion des Benutzers mit dem „Türkischen Schachspieler“.

DYNASIGHT-*Source Node* in OPENTRACKER (siehe Kapitel 3.3.3). Die Abhängigkeiten der STUDIERSTUBE-Komponenten zueinander werden in der Abbildung durch Pfeile dargestellt.

Der „Türkische Schachspieler“ ist die einzige interaktive Komponente dieser STUDIERSTUBE-Anwendung. Es ist daher nicht notwendig, die Möglichkeiten des dynamischen Ladens von verschiedenen Programmen innerhalb der STUDIERSTUBE-Umgebung auszunutzen. Der „Türkische Schachspieler“ wurde als eigenständiges Programm konzipiert, welches auf den verschiedenen Elementen der STUDIERSTUBE aufbaut.

Die Szene des Programmes besteht aus drei Hauptteilen: der *Figur* des „Türkischen Schachspielers“, dem *Schachbrett* und den 32 *Schachfiguren*. Diese Objekte sind als *Node Kits* implementiert. Diese Node Kits sind mit *Engines* für die Berechnung der Schachzüge und die Animation verbunden. Im Folgenden werden die einzelnen Komponenten beschrieben.

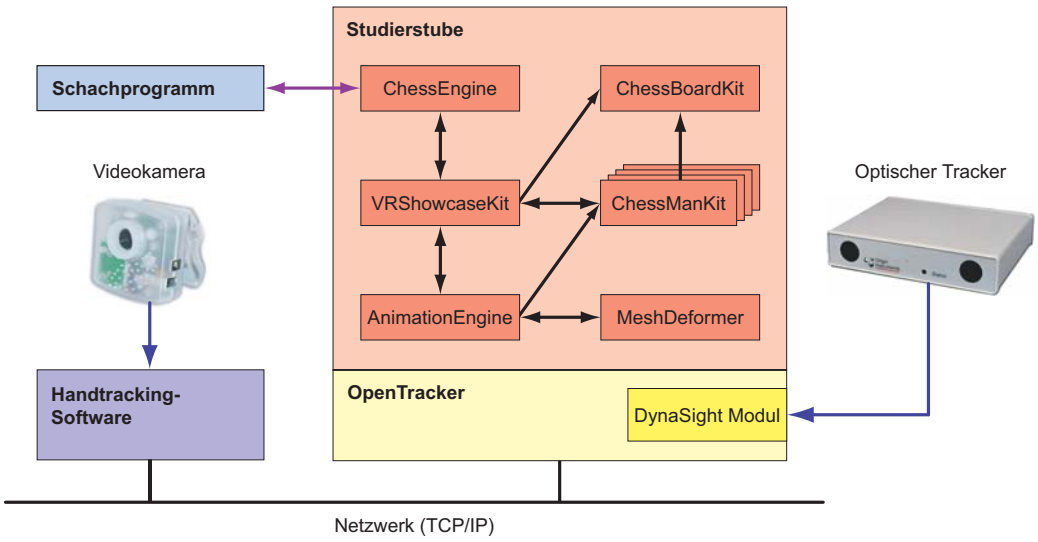


Bild 4.6: Übersicht der Softwarekomponenten.

4.2.1 DynaSight OpenTracker Modul

Für die Einbindung des DYNASIGHT-Trackers in OPENTRACKER wurde ein *Source Node* mit dem Namen `DynaSightSource` entwickelt (siehe Kapitel 3.3.3). Er liest die Daten des Trackers über die serielle Schnittstelle des Rechners innerhalb eines Thread aus. Diese Daten werden an eine State Machine übergeben und ausgewertet. Liegt ein vollständiger Satz an neuen Koordinaten vor, werden diese vom internen Format des Trackingsystems in Meter umgerechnet.

Ist in der OPENTRACKER-Konfigurationsdatei die `lookat`-Option angegeben, wird zu jedem ermittelten Koordinatenpunkt die entsprechende Orientierung berechnet. Diese wird als *Quaternion* im OPENTRACKER-State gespeichert.

Der optische Tracker kann feststellen, wann er den retroreflektierenden Marker nicht mehr erfassen kann. Diese Information wird in der Variable `confidence` des OPENTRACKER-State kodiert. Dabei gibt ein Wert von 0,5 an, daß ein Verlust der Trackerdaten kurz bevor steht. Ansonst wird der Wert 1,0 zurückgeliefert.

4.2.2 Das Hauptprogramm

Das Kernstück der Applikation ist die Klasse `VRShowcaseKit`, welche von `SoContextKit` abgeleitet ist. Der `VRShowcaseKit` übernimmt die Initialisierung und Koordinierung aller verwendeten *Node Kits* und *Engines*, besitzt aber außer dem PIP und Pen keine eigene Geometrie. Der `VRShowcaseKit` ist über mehrere Felder mit einer `ChessEngine` und einer `AnimationEngine` verbunden. Weiters enthält er Referenzen auf die Geometrie der Schachfiguren, auf die 32 Node Kits der Schachfiguren und auf das Node Kit des Schachbrettes.

Bei der Initialisierung weist der `VRShowcaseKit` den einzelnen Schachfiguren ihre Geometrie zu. Es ist dadurch auf einfache Weise möglich, bei der Promotion eines *Bauern* die Geometrie der Schachfigur z.B. auf eine *Dame* zu ändern.

Führt der Benutzer einen Schachzug aus, wird dieser über die `ChessEngine` vom `VRShowcaseKit` überprüft. Entspricht der Schachzug nicht den Regeln, wird der Zug rückgängig gemacht. Bei einem gültigen Schachzug wird von der `ChessEngine` der Gegenzug berechnet und über die `AnimationEngine` von der Geometrie des „Türkischen Schachspielers“ ausgeführt. Wird das Ende einer Schachpartie z.B. durch ein Schachmatt erreicht, veranlaßt der `VRShowcaseKit` die Ausgabe einer Meldung über dem Schachbrett.

Der folgende Text stellt einen Auszug aus der OPEN INVENTOR-Datei für die Anwendung dar. Er zeigt die Verwendung der einzelnen Felder des `VRShowcaseKit`-Knotens.

```
contextKit DEF CONTEXT VRShowcaseKit {
  isValidMoveIn = DEF CHESS_ENGINE ChessEngine {
  …
  }.isValidMoveOut
  boardPositionIn = USE CHESS_ENGINE.boardPositionOut
  isGameOverIn = USE CHESS_ENGINE.isGameOverOut
  gameOverMessageIn = USE CHESS_ENGINE.gameOverMessageOut
  readyForNewGameIn = USE CHESS_ENGINE.readyForNewGameOut
  handRestPosition -0.5 0.0 0.0
  handYPositionUp 0.15
  handYPositionDown 0.05

  pawn_content Separator {
    DEF CHESS_MAN_BB File { name "../../../geometry/chessman_bb.iv" }
    DEF PAWN File { name "../../../geometry/pawn.iv" }
  }
  …
  animationEngineStartupFinishedIn = USE
    ANIMATIONENGINE.readyWithInitialization
  animationFinishedIn = USE ANIMATIONENGINE.animationFinished
  animationTouchDownIn = USE ANIMATIONENGINE.animationTouchDown
  …
}
```

Es folgt eine Beschreibung der einzelnen Felder dieses Node Kits. Feldbezeichnungen, die mit der Wortendung „In“ abschließen, sind Eingabe-Felder, die im Normalfall mit einer Engine verbunden sind und von dieser einen Wert zugewiesen bekommen. Die Feldbezeichnungen, die mit der Endung „Out“ aufhören, sind Ausgabe-Felder und stellen einen Wert für eine Engine bereit.

isValidMoveIn `SoSFBool` War der letzte Schachzug gültig, erhält dieses Feld den Wert `TRUE`. Im anderen Fall den Wert `FALSE`. Dieses Feld ist ein Eingabe-Feld und wird mit dem entsprechenden Ausgabe-Feld der `ChessEngine` verbunden.

isGameOverIn `SoSFBool` Ist die Schachpartie zu Ende, wird dieses Eingabe-Feld von der `ChessEngine` auf `TRUE` gesetzt.

startNewGameOut `SoSFTrigger` dient zum *Zurücksetzen* der `ChessEngine` und damit zum Beginn einer neuen Schachpartie. Dieses Feld ist ein Ausgabe-Feld und mit der `ChessEngine` verbunden (siehe Kapitel 4.2.5).

readyForNewGameIn `SoSFTrigger` Dieses Eingabe-Feld wird von der verbundenen `ChessEngine` gesetzt, wenn sie die Vorbereitungen für ein neues Spiel abgeschlossen hat.

gameOverMessageIn `SoSFString` ist ein Eingabe-Feld und enthält den von der `ChessEngine` zum Spielschluß übermittelten *Ergebnistext*.

boardPositionIn `SoSFString` Das Feld erhält den von der `ChessEngine` errechneten Schachzug in der *Coordinate Algebraic Notation* (siehe Kapitel 3.4).

boardPositionOut `SoSFString` Das Ausgabe-Feld enthält den vom Benutzer getätigten Schachzug in der *Coordinate Algebraic Notation* (siehe Kapitel 3.4).

handRestPosition `SoSFVec3f` legt die *Position* der Hand des „Türkischen Schachspielers“ in der Ruheposition fest. Diese Position liegt normalerweise neben dem Schachfeld (siehe Abbildung, erstes Bild) und ist die erste und letzte Stützstelle der vom `VRShowcaseKit` errechneten Animation der Hand. Die Position wird im globalen Koordinatensystem der OIV-Szene angegeben.

handYPositionUp `SoSFFloat` legt die „Up“-Y-Koordinate der Animation fest. Während sich die Hand des „Türkischen Schachspielers“ über das Schachfeld bewegt, wird für die Stützstellen des Endeffektors die Y-Koordinate auf diesen Wert gesetzt. Dieser Wert wird im globalen Koordinatensystem der OIV-Szene angegeben.

handYPositionDown `SoSFFloat` legt die „Down“-Y-Koordinate der Animation fest. Vor dem Ergreifen einer Schachfigur bewegt sich die Hand des „Türkischen Schachspielers“ von der Y-Koordinate `handYPositionUp` auf die Y-Koordinate `handYPositionDown`. Dieser Wert wird im globalen Koordinatensystem der OIV-Szene angegeben.

animationEngineStartupFinishedIn `SoSFBool` Ist die Initialisierung der `AnimationEngine` abgeschlossen, wird dieses Eingabe-Feld von der *Engine* auf den Wert `TRUE` gesetzt.

startAnimationOut `SoSFTrigger` Über dieses Ausgabe-Feld teilt der `VRShowcaseKit` der `AnimationEngine` mit, daß die für die Animation benötigten Werte in den Ausgabe-Feldern `animationCoordinatesOut`, `animationHandPostureOut` und `animationChessMenOut` vorliegen und mit der Animation begonnen werden kann (siehe Kapitel 4.2.6).

animationFinishedIn `SoSFBool` Dieses Eingabe-Feld wird von der `AnimationEngine` auf den Wert `TRUE` gesetzt, wenn die Animation abgeschlossen ist.

animationTouchDownIn `SoSFBool` Dieses Eingabe-Feld wird von der `AnimationEngine` auf den Wert `TRUE` gesetzt, wenn die vom „Türkischen Schachspieler“ gezogene Schachfigur abgesetzt wird. Dieses Ereignis wird vom `VRShowcaseKit` unter anderem dazu verwendet, um geschlagene Schachfiguren vom Schachbrett zu entfernen oder bei einer Promotion die Geometrie des *Bauern* zu verändern.

animationCoordinatesOut `SoMFVec3f` Dieses Ausgabe-Feld enthält alle *Stützstellen*, die während der Animation des Hand-Endeffektors von der `AnimationEngine` angefahren werden sollen. Diese Werte werden im globalen Koordinatensystem der OIV-Szene angegeben.

animationHandPostureOut `SoMFEnum` Dieses Ausgabe-Feld legt die *Stellung* der Hand zwischen zwei Stützstellen der Animation fest. Die Stellung der Hand

kann entweder *offen* (entspricht dem Wert `OPEN`) oder *geschlossen* (Wert `CLOSED`) sein.

animationChessMenOut `SoMFNode` Dieses Feld enthält einen oder mehrere `ChessManKit`-Knoten und wird mit der `AnimationEngine` verbunden. Enthält dieses Feld nur einen `ChessManKit`, so ist dieser die bei der Animation zu ziehende Schachfigur. Bei einer Rochade enthält dieses Feld neben der ursprünglichen Schachfigur (in diesem Fall dem *König*) auch den von der Rochade betroffenen *Turm*.

messageVisibleOut `SoSFBool` Dieses Ausgabe-Feld wird vom `VRShowcaseKit` auf den Wert `TRUE` gesetzt, wenn die im Feld `messageTextOut` vorliegende Textnachricht vom `ChessBoardKit` gezeigt werden soll (siehe Kapitel 4.2.4).

messageTextOut `SoMFString` enthält den mehrzeiligen *Nachrichtentext*, der vom `ChessBoardKit` angezeigt werden soll (siehe Kapitel 4.2.4).

pawn_content, **knight_content**, **bishop_content**, **rook_content**, **queen_content**, **king_content** `SoSFNode` Diese Felder enthalten die unterschiedlichen OIV-*Geometrien*, die den einzelnen Schachfiguren gemäß ihrem Typ (z.B. *Bauer*) zugewiesen werden.

4.2.3 Die Schachfiguren

Der *Node Kit* `ChessManKit` ist von der Klasse `SoDragKit` abgeleitet. Jede *Instanz* der Klasse `ChessManKit` entspricht einer virtuellen Schachfigur. Die Klasse `SoDragKit` ist Teil des STUDIERSTUBE-Systems und erlaubt eine Interaktion des Benutzers mit Objekten dieser Klasse. Die OIV-Knoten dieses Typs können vom Anwender mit dem STUDIERSTUBE-Pen ausgewählt und in der Szene bewegt werden. Dieser Umstand wird für die Bewegung der virtuellen Schachfiguren innerhalb der OIV-Szene ausgenutzt.

Der folgende Text stellt einen Auszug aus der OPEN INVENTOR-Datei für die Anwendung dar. Er zeigt die Verwendung der einzelnen Felder des `ChessManKit`-Knotens.

```
DEF WHITE_PAWN1 ChessManKit {
  yTranslationOn FALSE
  defaultPosition 0 6
  kind PAWN
  isWhite TRUE
  chessBoardKit USE BOARD
  vrShowcaseKit USE CONTEXT
}
```

Das *Koordinatensystem* des Schachbrettes, welches für diese Anwendung verwendet wird, ist in Abbildung 4.7 zu sehen. Die *schwarze* Beschriftung stellt die herkömmliche Beschriftung eines Schachfeldes dar. Die *rote* Beschriftung wird zur Angabe eines Schachfeldes im Rahmen dieses Projektes verwendet. Bei einem Koordinatentupel wird als Erstes die Spalte und nachfolgend die Zeile angegeben. Das soll an einem Beispiel gezeigt werden. Der weiße *Bauer* auf Feld `a2` wird als das Tupel (0, 6)

geschrieben. Die Werte für das Feld `defaultPosition` werden in diesem Koordinatensystem angegeben.

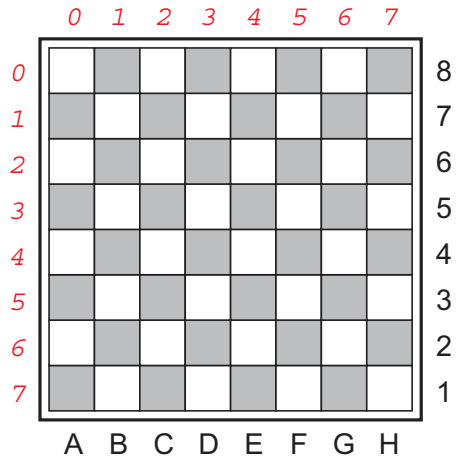


Bild 4.7: Koordinatensysteme für das Schachbrett.

Damit die Schachfiguren vom Anwender *einfach* ausgewählt und bewegt werden können, wird jeder Schachfigur eine Bounding Box zugeordnet. Diese nicht sichtbare Bounding Box wird in der OIV-Datei der Anwendung definiert und ist in ihrer Grundfläche etwas kleiner als ein einzelnes Schachfeld. Dadurch überschneiden sich die Bounding Boxes von nebeneinander stehenden Schachfiguren nicht. Der folgende Text zeigt die Definition der OIV-Geometrie für den *Bauern* mit der Bounding Box in der OIV-Datei `chessman_bb.iv`.

```
pawn_content Separator {
  DEF CHESS_MAN_BB File {  name "../../../geometry/chessman_bb.iv" }
  DEF PAWN File {  name "../../../geometry/pawn.iv" }
}
```

Die Abbildung 4.8 zeigt das Schachbrett mit allen Schachfiguren. Dabei wurden für alle Schachfiguren die Umrisse der Bounding Boxes sichtbar gemacht. Die größeren Bounding Boxes *erleichtern* vor allem das Auswählen der einzelnen Schachfiguren in Verbindung mit dem optischen *Handtracking* (siehe Kapitel 3.3.2).

Es folgt eine Beschreibung der einzelnen Felder des `ChessManKit`.

kind `SoSFEnum` Dieses Feld legt den *Typ* der Schachfigur fest. Dem Feld kann ein Wert aus folgender Liste zugewiesen werden: `PAWN`, `ROOK`, `BISHOP`, `KNIGHT`, `KING` und `QUEEN`. Dieser Wert wird auch verwendet, um der Schachfigur die korrekte Geometrie zuzuordnen.

defaultPosition `SoSFVec2f` Dieses Feld legt die *Anfangsposition* der Schachfigur am Schachbrett fest. Der Wert wird als *Zahlentupel* im internen Koordinatensystem angegeben (siehe Abbildung 4.7).

currentPosition `SoSFVec2f` Dieses Feld legt die *aktuelle Position* der Schachfigur am Schachbrett fest. Der Wert wird als *Zahlentupel* im internen Koordinatensystem angegeben (siehe `defaultPosition`).

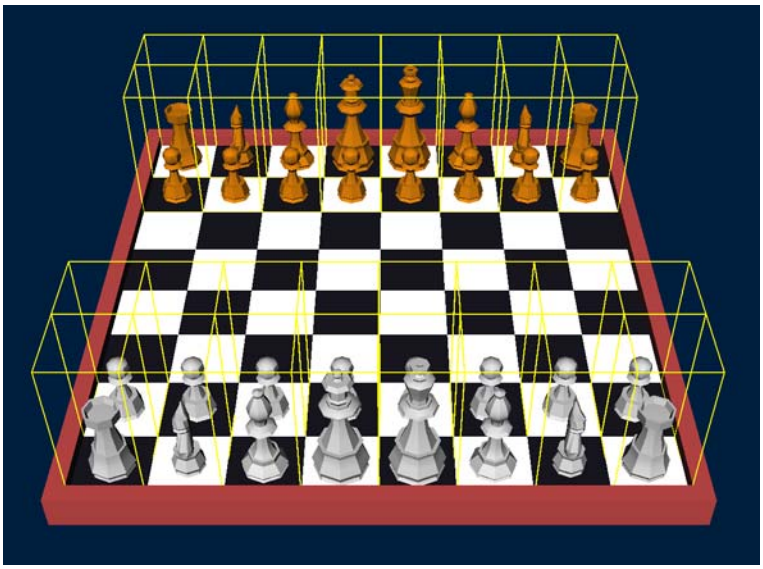


Bild 4.8: Schachfiguren mit ihrer Bounding Box.

isWhite `SoSFBool` Diesem Feld wird der Wert `TRUE` zugewiesen, wenn es sich um eine *weiße* Schachfigur handelt. Bei einer *schwarzen* Schachfigur ist der Wert `FALSE`.

doHighlight `SoSFBool` Wird dieser Wert auf `TRUE` gesetzt, dann erfolgt ein Highlight der Schachfigur, sobald sich der STUDIERSTUBE-Pen *ohne* gedrücktem Knopf innerhalb der OIV-Geometrie der Schachfigur befindet.

whiteColor, **blackColor** `SoSFColor` Diese beiden Felder legen die *Farbwerte* für die *weißen* und *schwarzen* Schachfiguren fest.

highliteColor `SoSFColor` Dieses Feld definiert den *Farbwert* für das Highlight der Schachfigur (siehe `doHighlight`). Als Standardwert ist die Farbe *Gelb* vorgegeben.

pickColor `SoSFColor` Dieses Feld bestimmt den *Farbwert* für eine *ausgewählte* Schachfigur. Beim Auswählen einer Schachfigur befindet sich der STUDIERSTUBE-Pen *mit* gedrücktem Knopf innerhalb der OIV-Geometrie der Schachfigur. Als Standardwert ist die Farbe *Rot* vorgegeben.

chessBoardKit `SoSFNode` Dieses Feld enthält eine Referenz auf das `ChessBoardKit` der OIV-Szene.

vrShowcaseKit `SoSFNode` Dieses Feld enthält eine Referenz auf das `VRShowcaseKit` der OIV-Szene.

Zusätzlich zu den oben beschriebenen Feldern des `ChessManKit` gibt es noch die Felder des Node Kit `SoDragKit`. Diese ermöglichen es unter anderem, die Freiheitsgrade für die Bewegungen der Schachfiguren zu beschränken. So unterbindet zum Beispiel `yTranslationOn FALSE` die Verschiebung der Schachfigur mit Hilfe des STUDIERSTUBE-Pen entlang der Y-Achse.

4.2.4 Das Schachbrett

Der *Node Kit* `ChessBoardKit` ist ebenfalls von der Klasse `SoDragKit` abgeleitet. Seine Aufgabe ist die *Darstellung* des Schachbrettes, das Hervorheben des Schachfeldes, über dem sich der STUDIERSTUBE-Pen befindet und die Ausgabe von *Nachrichten* über dem Schachfeld. Weiters stellt er Hilfsroutinen für die Positionierung der Schachfiguren am Schachbrett zur Verfügung. Die Abbildung 4.9 zeigt den Szenengraph des `ChessBoardKit`.

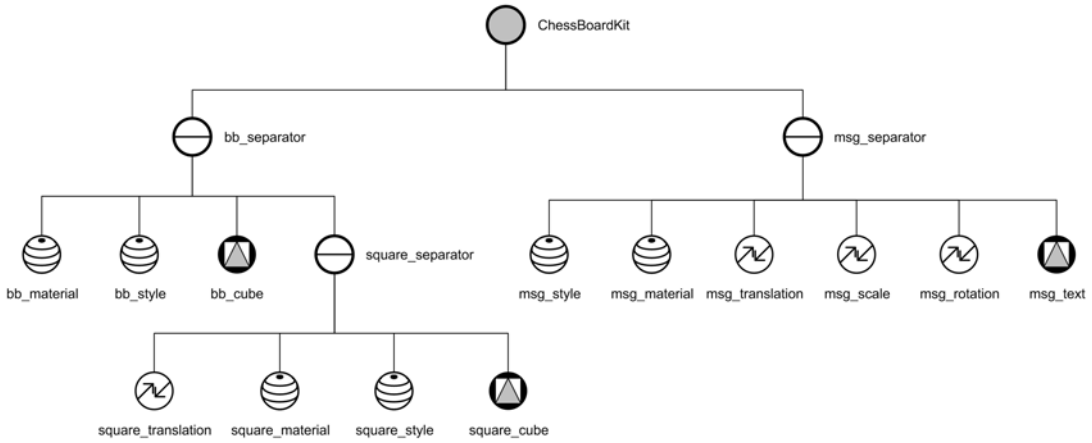


Bild 4.9: Szenengraph des ChessBoardKit.

Der *Szenengraph* des Node Kits besteht aus den beiden Separatoren `bb_separator` und `msg_separator`. Unter dem Separator `bb_separator` befindet sich der Subgraph zum Anzeigen des Highlight eines Schachfeldes und der Bounding Box `bb_cube`, welche alle Schachfelder umschließt.

Der Subgraph, der für das *Hervorheben* eines Schachfeldes verantwortlich ist, befindet sich unterhalb des Separators `square_separator`. Die Knoten `square_translation`, `square_material` und `square_style` legen die Position und die Eigenschaften für die Geometrie `square_cube` fest.

Damit das aktuelle Schachfeld, über dem sich der STUDIERSTUBE-Pen befindet, auch ohne ausgewählte Schachfigur hervorgehoben werden kann, muß der `ChessBoardKit` die dafür notwendigen *Ereignisse* (engl. events) empfangen können. Um dies zu ermöglichen, wird eine Bounding Box erzeugt, welche alle Schachfelder und die auf dem Schachbrett befindlichen Schachfiguren inklusive ihrer Bounding Boxes umschließt. Der Mittelpunkt dieser Bounding Box ist das Zentrum des Schachbrettes. Die Knoten `bb_material` und `bb_style` legen die Eigenschaften der nicht sichtbaren Bounding Box fest. Der Knoten `bb_cube` ist die Geometrie der Bounding Box.

Der Subgraph unter dem Knoten `msg_separator` ermöglicht das Anzeigen einer mehrzeiligen *Textnachricht* über dem Schachbrett. Ein Beispiel dazu ist in Abbildung 4.10 zu sehen. Die OIV-Knoten `msg_style` und `msg_material` bestimmen das Aussehen der durch den Knoten `msg_text` dargestellten Geometrie. Die Knoten `msg_translation`, `msg_scale` und `msg_rotation` bestimmen Position, Größe und Lage des Textes in der Szene.

Bild 4.10: Meldung zum Ende einer Schachpartie.

Der folgende Text stellt einen Auszug aus der OPEN INVENTOR-Datei für die Anwendung dar. Er zeigt die Verwendung der einzelnen Felder des `ChessBoardKit`-Knotens.

```
DEF BOARD ChessBoardKit {
  eventStation 0
  boxHeight 0.35
  boxVisible FALSE
  squareHeight 0.005
  squareYTranslation 0.001
  squareDiffuseColor 1 1 0
  squareEmissiveColor 0.4 0.4 0.0
  messageRotation -1.0
  messageVisible = USE CONTEXT.messageVisibleOut
  messageText = USE CONTEXT.messageTextOut
  tileLength 0.0625

  content So3DSeparator {
    File { name "../../../geometry/chessboard.iv" }
  }
}
```

Es folgt eine Beschreibung der einzelnen Felder des `ChessBoardKit`.

eventStation `SoSFShort` Dieses Feld enthält die Nummer jener OPENTRACKER-Station, deren *Ereignisse* vom `ChessBoardKit` ausgewertet werden. Im Normalfall ist dies die OPENTRACKER-Stationsnummer des STUDIERSTUBE-Pen und dient zum *Hervorheben* des aktuellen Schachfeldes.

tileLength `SoSFFloat` Der Wert in diesem Feld gibt die *Seitenlänge* eines quadratischen Schachfeldes an. Es wird davon ausgegangen, daß alle Schachfelder die gleichen Abmessungen haben.

boxHeight `SoSFFloat` Der Wert dieses Feldes definiert die *Höhe* der Bounding Box `bb_cube`, welche das Schachbrett und die am Brett befindlichen Schachfiguren umschließt.

boxVisible `SoSFBool` Ist der Wert dieses Feldes auf `TRUE` gesetzt, dann ist die Geometrie der Bounding Box `bb_cube` sichtbar. Diese Einstellung ist bei der Fehlersuche und Konfiguration nützlich. Der Standardwert dieses Feldes ist `FALSE`.

squareHeight `SoSFFloat` Die Höhe des Würfels `square_cube`, welcher zum *Hervorheben* des aktuellen Schachfeldes dient, wird durch diesen Wert festgelegt. Der Wert wird im lokalen Koordinatensystem des `ChessBoardKit` angegeben.

squareYTranslation `SoSFFloat` Dieser Wert erlaubt das *Verschieben* des Würfels `square_cube` entlang der Y-Achse. Der Wert wird im lokalen Koordinatensystem des `ChessBoardKit` angegeben.

squareDiffuseColor, **squareSpecularColor**, **squareEmissiveColor** `SoSFColor` Diese drei Felder legen die verschiedenen *Farbwerte* für den Würfel `square_cube` fest. Als Standardwert ist für alle drei Felder der Farbwert *schwarz* festgelegt.

squareTransparency `SoSFFloat` Dieses Feld bestimmt die *Transparenz* des Würfels `square_cube`. Bei Verwendung des Standardwertes ist die Geometrie *nicht* transparent.

messageRotation `SoSFFloat` Dieses Feld enthält den Winkel für die *Rotation* der Textnachricht um die X-Achse. Der Wert wird im lokalen Koordinatensystem des `ChessBoardKit` angegeben.

messageVisible `SoSFBool` Wird dieses Feld auf den Wert `TRUE` gesetzt, erfolgt das `Anzeigen` der Textnachricht im Feld `messageText` über dem Schachbrett. Zum *Ausblenden* der Nachricht wird diesem Feld der Wert `FALSE` zugewiesen. Dieses Feld ist üblicherweise mit dem Ausgabe-Feld `messageVisibleOut` des `VRShowcaseKit` verbunden.

messageText `SoMFString` Diesem Feld wird der Text der mehrzeiligen Nachricht, welche über dem Schachbrett angezeigt werden soll, zugewiesen. Dieses Feld ist üblicherweise mit dem Ausgabe-Feld `messageTextOut` des `VRShowcaseKit` verbunden.

Zusätzlich zu den oben beschriebenen Feldern enthält der `ChessBoardKit` auch die Felder des Node Kit `SoDragKit`. Die Geometrie des Schachbrettes wird zum Beispiel dem Feld `content` aus dem `SoDragKit` zugewiesen.

4.2.5 Das Schachprogramm

Die Klasse `ChessEngine` ist von der Klasse `SoEngine` abgeleitet. Die Aufgabe dieser *Engine* ist die *Validierung* der vom Benutzer durchgeführten Schachzüge und die *Berechnung* der vom „Türkischen Schachspieler“ gespielten Züge.

Zu diesem Zweck wird ein bereits bestehendes *Schachprogramm* eingesetzt (siehe Kapitel 3.4). Dieses Programm wird mit Hilfe des Funktionsaufrufes `CreateProcess` aus der MICROSOFT Windows-API als eigener *Prozeß* gestartet.

Die Kommunikation zwischen der OIV-*Engine* und dem *Prozeß* erfolgt über Pipes. Ein Thread innerhalb der `ChessEngine`-Instanz übernimmt den Datenaustausch über diese Verbindung. Im Thread ist eine State Machine implementiert, welche auf die verschiedenen Meldungen des Schachprozesses entsprechend reagiert. Die Verständigung zwischen Thread und `ChessEngine`-Instanz erfolgt über Shared Memory und *Semaphoren*. Das Diagramm in Abbildung 4.11 stellt die Zusammenhänge zwischen der `ChessEngine` und dem Schachprogramm dar.

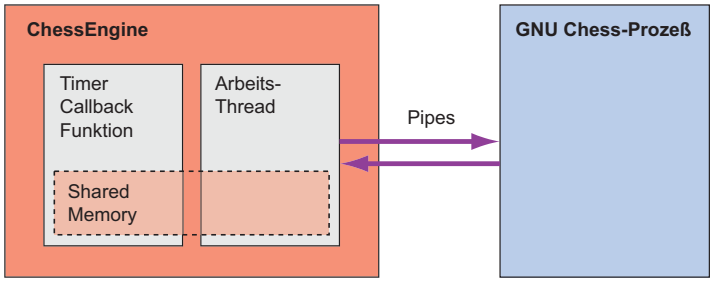


Bild 4.11: Aufbau des ChessEngine-Knotens.

Da es Probleme mit der *Terminierung* des Schachprozesses beim Beenden der STUDIERSTUBE-Anwendung gab, wird der Funktionsaufruf `atexit` zur Beendigung des Schachprogrammprozesses verwendet.

Da sowohl die STUDIERSTUBE-Anwendung als auch das Schachprogramm auf dem gleichen Rechner laufen, kann es zu einem Engpaß an Rechnerleistung kommen. Dies ist vor allem dann der Fall, wenn das Schachprogramm die möglichen Zugbewertungen für das weitere Spiel vorausberechnet. Damit die Qualität der Graphikausgabe nicht zu stark von diesen Berechnungen betroffen ist, wird die *Priorität* des Schachprogrammes, abhängig vom Zustand des Spieles, verändert. Für die Änderung der *Priorität* wird der Funktionsaufruf `SetPriorityClass` aus der MICROSOFT Windows-API verwendet. Ist der Anwender am Zug, dann wird die Priorität des Schachprozesses auf den Wert `IDLE_PRIORITY_CLASS` gesetzt. Soll die `ChessEngine` einen Zug des Anwenders validieren und einen Gegenzug berechnen, dann wird die Priorität auf den im Feld `chessEnginePriority` angegebenen Wert gesetzt.

Die Verwendung der Felder des `ChessEngine`-Knotens wird im folgenden Auszug aus der OPEN INVENTOR-Datei für die Anwendung und den Text aus Kapitel 4.2.2 gezeigt.

```
isValidMoveIn = DEF CHESS_ENGINE ChessEngine {
  chessEnginePath "../../../vrshowcase/gnuchess/"
  chessEngineName "gnuchess506.exe -x"
  chessEnginePriority NORMAL
  startNewGameIn = USE CONTEXT.startNewGameOut
  boardPositionIn = USE CONTEXT.boardPositionOut } .isValidMoveOut
```

Es folgt eine Beschreibung der einzelnen Felder der `ChessEngine`. Feldbezeichnungen, die mit der Wortendung „In“ abschließen, sind Eingabe-Felder, die im Normalfall über eine Feldverbindung einen Wert zugewiesen bekommen. Die Feldbezeichnungen, die mit „Out“ enden, sind Ausgabe-Felder und stellen einen Wert bereit.

chessEnginePath `SoSFString` Dieses Feld enthält den absoluten oder relativen *Pfad* zum ausführbaren *Schachprogramm* im Dateisystem des Rechners. Der Wert dieses Feldes wird mit dem Wert des Feldes `chessEngineName` verknüpft und dient zum Starten des Schachprozesses.

chessEngineName `SoSFString` Dieses Feld enthält den *Dateinamen* des *Schachprogrammes*, gefolgt von den benötigten *Kommandozeilenparametern*. Der Wert dieses Feldes wird mit dem Wert des Feldes `chessEnginePath` verknüpft und dient zum Starten des Schachprozesses.

chessEnginePriority `SoSFEnum` Dieses Feld legt die *Priorität* des Schachprozesses für den Zeitraum, in dem der Benutzer einen Schachzug durchführt, fest. Dem Feld können die folgenden drei Werte zugewiesen werden: `IDLE`, `LOW` und `NORMAL`.

boardPositionIn `SoSFString` Dieses Eingabe-Feld enthält den vom Anwender getätigten *Schachzug* in der *Coordinate Algebraic Notation* (siehe Kapitel 3.4). Ein Beispiel für den Inhalt dieses Feldes ist ”b2b4“.

startNewGameIn `SoSFTrigger` Dieses Eingabe-Feld dient zum *Zurücksetzen* der `ChessEngine` und damit zum Beginn einer neuen Schachpartie.

isValidMoveOut `SoEngineOutput` (`SoSFBool`) Dieses Ausgabe-Feld wird auf den Wert `TRUE` gesetzt, wenn der letzte vom Anwender getätigte Schachzug *gültig* war. Im anderen Fall wird dem Feld der Wert `FALSE` zugewiesen.

isGameOverOut `SoEngineOutput` (`SoSFBool`) Ist die Schachpartie zu *Ende*, dann wird dieses Ausgabe-Feld auf den Wert `TRUE` gesetzt.

readyForNewGameOut `SoEngineOutput` (`SoSFTrigger`) Über dieses Ausgabe-Feld gibt die `ChessEngine` bekannt, daß sie die *Vorbereitungen* für eine *neue* Schachpartie abgeschlossen hat.

gameOverMessageOut `SoEngineOutput` (`SoSFString`) Diesem Ausgabe-Feld wird der vom Schachprozeß übermittelte *Ergebnistext* zugewiesen. Dieser Text enthält den Spielausgang (siehe Kapitel 3.4).

boardPositionOut `SoEngineOutput` (`SoSFString`) Das Feld erhält den von der `ChessEngine` errechneten *Schachzug* in der *Coordinate Algebraic Notation* (siehe Kapitel 3.4). Zwei Beispiele für den Inhalt dieses Feldes sind ”d7d6“ und ”d2d1q“.

4.2.6 Die Animation der Figur

Die Klasse `AnimationEngine` ist von der Klasse `SoEngine` abgeleitet. Die Aufgabe dieser *Engine* ist die *Animation* der Geometrie des „Türkischen Schachspielers“.

Der OIV-Knoten `AnimationEngine` erhält über das Eingabe-Feld `animationPathCoordinates` die Stützstellen der Animation. Mittels linearer Interpolation werden die Zwischenpunkte für die Animation des Armes berechnet. Für die Berechnung der Rotationen der drei Gelenke (Schulter, Ellbogen und Hand) des

rechten Armes wird die IKAN-Bibliothek verwendet (siehe Kapitel 3.2.2). Die `AnimationEngine` bewegt auch die Schachfiguren des „Türkischen Schachspielers“. Die Liste der bei einem Zug bewegten Schachfiguren wird im Feld `animationChessMen` an die *Engine* übergeben. Zur Durchführung dieser Tätigkeiten wurde eine einfache State Machine implementiert.

Da der OIV-Knoten `MeshDeformer` beim Starten etwas Zeit für die Initialisierung seiner Datenstruktur benötigt, meldet die `AnimationEngine` über das Feld `readyWithInitialization` den Abschluß dieser Vorbereitungen (siehe Kapitel 3.2.3).

Es war nicht auf einfache Weise möglich, die im Szenengraphen auf dem Knoten `MeshDeformer` angewandten Transformationen automatisch zu bestimmen. Diese Werte werden nun über einzelne Felder der `AnimationEngine` zugeführt. Weiters werden auch die von der IKAN-Bibliothek benötigen Daten, wie z.B. Abmaße des Ober- und Unterarms, über Eingabe-Felder der Engine bekannt gegeben.

Die folgenden Texte stellen einen Auszug aus der OPEN INVENTOR-Datei für die Anwendung dar. Sie zeigen die Verwendung der einzelnen Felder des `AnimationEngine`-Knotens.

```
DEF TURKMESH MeshDeformer {
  rootTranslation 0.0 0.0 0.0
  rootRotation = DEF ANIMATIONENGINE AnimationEngine {
    baseRootRotation 0.0 1.0 0.0 3.1415926
    baseRootTranslation 0.0 -0.1 -0.45
    rightOpenedHandThumb1 0.0 0.0 1.0 -0.1
    rightOpenedHandIndex1 0.0 0.0 1.0 0.5
    rightOpenedHandMiddle1 0.0 0.0 1.0 0.4
    rightOpenedHandRing1 0.0 0.0 1.0 0.6
    rightOpenedHandPinky1 0.0 0.0 1.0 0.6
    rightClosedHandThumb1 0.0 0.0 1.0 -0.3
    rightClosedHandIndex1 0.0 0.0 1.0 0.7
    rightClosedHandMiddle1 0.0 0.0 1.0 0.6
    rightClosedHandRing1 0.0 0.0 1.0 0.6
    rightClosedHandPinky1 0.0 0.0 1.0 0.6
    geometryScaleFactor 2.5
    shoulderTranslation = USE TURKMESH.rightShoulderTranslation
    animationIncrement 200.0
    animationPickDropDelay 150
    elbowSwivelAngel -1.1
    enableJointLimits FALSE
    positionOnlyWristRotation 1.0 0.0 0.0 -0.65
    # matrix T - in mesh coordinate system
    upperArmTranslation 0.1783335694 -0.007303818856 0.01158487358
    # matrix S - in mesh coordinate system
    lowerArmTranslation 0.1885710472 -0.003339511362 0.001063490849
    # matrix EE - in IKAN coordinate system
    endEffectorTranslation 0.0 -0.05 0.06 # pos
    animationPathCoordinates = USE CONTEXT.animationCoordinatesOut
    animationPathHandPosture = USE CONTEXT.animationHandPostureOut
    animationChessMen = USE CONTEXT.animationChessMenOut
    startAnimation = USE CONTEXT.startAnimationOut
  }.rootRotation
  …
}
```

Die von der `AnimationEngine` errechnete Rotation für das rechte Schultergelenk wird dem entsprechenden Rotationsknoten des `MeshDeformer` zugewiesen. Mit den anderen Feldern wird äquivalent verfahren.

```
# --- |Tuerke|jRumpf1|jHuefte1|jOberkoerper1|jRechtsSchulter1 ---
Separator {
  Translation {  translation 0.130831747 0.02354414632 -0.02443981122 }
    DEF JRECHTSSCHULTER1 Rotation {
      rotation = USE ANIMATIONENGINE.rightShoulderRotation }
}
```

Die Bilder in Abbildung 4.12 zeigen eine volle Animationssequenz der Figur. Das erste Bild (links oben) und das letzte Bild (rechts unten) zeigen die Hand in der Ruheposition (siehe `handRestPosition` auf Seite 81). Die zeitliche Anordnung der Bilder ist von links nach rechts und von oben nach unten.

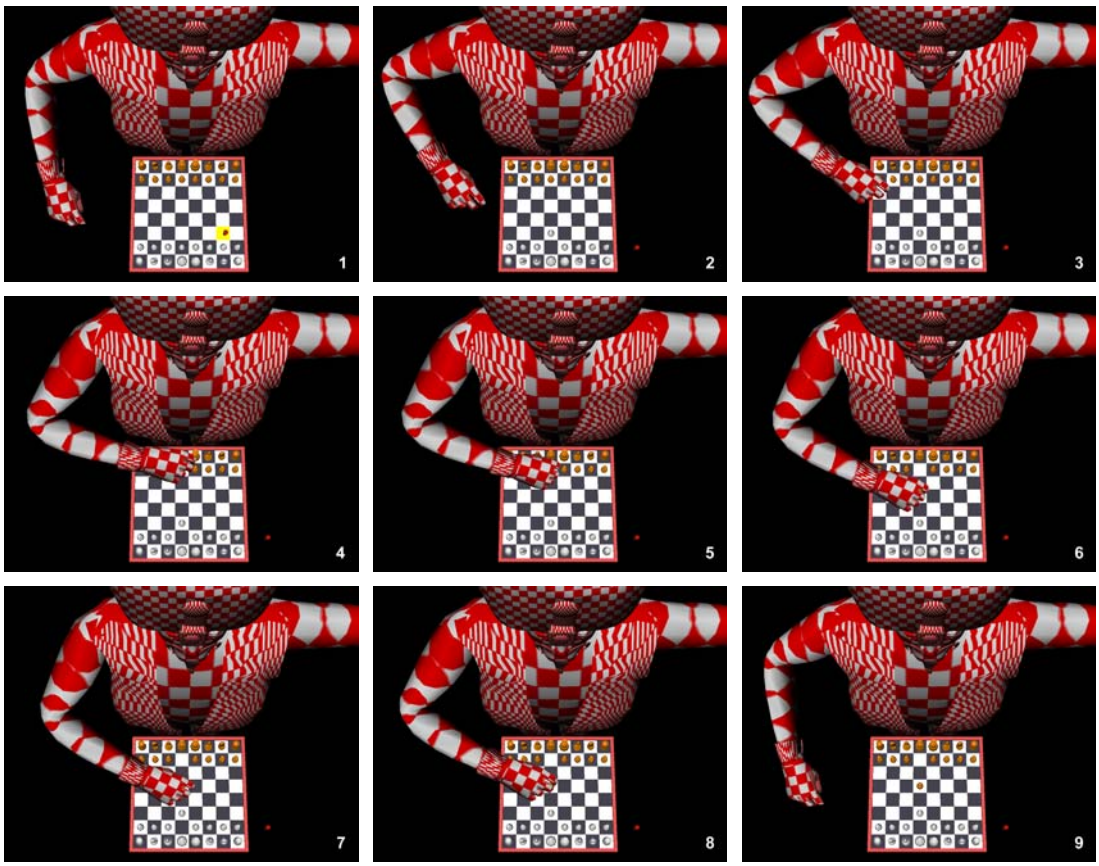


Bild 4.12: Animation der Hand des „Türkischen Schachspielers“.

Es folgt eine Beschreibung der einzelnen Felder der `AnimationEngine`.

timeIn `SoSFTime` Dieses Eingabe-Feld wird von der `AnimationEngine` mit dem globalen Feld `realTime` von OPEN INVENTOR verbunden. Die einzelnen Schritte für die Animation werden von dieser Zeit abgeleitet.

upperArmTranslation `SoSFVec3f` Dieses Eingabe-Feld legt die Ausmaße des rechten *Oberarms* der Figur für die IKAN-Bibliothek fest und entspricht der Matrix T (siehe Kapitel 3.2.2.1). Der Wert wird im `MeshDeformer`-Koordinatensystem angegeben.

lowerArmTranslation `SoSFVec3f` Dieses Eingabe-Feld legt die Ausmaße des rechten *Unterarms* der Figur für die IKAN-Bibliothek fest und entspricht der Matrix *S* (siehe Kapitel 3.2.2.1). Der Wert wird im `MeshDeformer`-Koordinatensystem angegeben.

endEffectorTranslation `SoSFVec3f` Der Wert dieses Eingabe-Feldes bestimmt die Verschiebung des *Endeffektors*, ausgehend vom Handgelenk der rechten Hand. Er entspricht der Matrix *EE* für die IKAN-Bibliothek und wird im IKAN-Koordinatensystem angegeben.

baseRootRotation `SoSFRotation` Der Wert dieses Eingabe-Feldes legt die *Grundrotation* der Figur in der Szene fest. Damit kann z.B. ein fehlerhafter Export aus MAYA, bei dem die Figur um den Winkel von 180º um die X-Achse verdreht ist, einfach korrigiert werden.

baseLeftShoulderRotation, **baseLeftElbowRotation**, **baseLeftWristRotation** `SoSFRotation` Die Werte dieser drei Eingabe-Felder werden ohne Änderung dem linken Schulter-, Ellbogen- und Handgelenk im `MeshDeformer` zugewiesen und bestimmen die *Haltung* des linken Arms. Dabei dienen die Felder `leftShoulderRotation`, `leftElbowRotation` und `leftWristRotation` als Ausgabe-Felder.

rightOpenedHandThumb1, **rightOpenedHandPinky1**, **rightOpenedHandIndex1**, **rightOpenedHandMiddle1**, **rightOpenedHandRing1**, `SoSFRotation` Diese fünf Eingabe-Felder legen die Rotation des ersten Gelenks der fünf Finger der rechten Hand in *geöffnetem* Zustand fest.

rightClosedHandThumb1, **rightClosedHandPinky1**, **rightClosedHandIndex1**, **rightClosedHandMiddle1**, **rightClosedHandRing1**, `SoSFRotation` Diese fünf Eingabe-Felder legen die Rotation des ersten Gelenks der fünf Finger der rechten Hand in *geschlossenem* Zustand fest.

baseRootTranslation `SoSFVec3f` Die *Translation* der Geometrie des „Türkischen Schachspielers“ wird durch diesen Wert beschrieben. Der Wert dieses Eingabe-Feldes wird im globalen OIV-Koordinatensystem angegeben. Dieser Wert wird von der `AnimationEngine` zur Berechnung der Animation herangezogen.

shoulderTranslation `SoSFVec3f` Der Wert dieses Eingabe-Feldes beschreibt die Translation vom *lokalen Nullpunk*t der Figur zum rechten *Schultergelenk*.

geometryScaleFactor `SoSFFloat` Der *Skalierungsfaktor* der Geometrie im globalen OIV-Koordinatensystem wird durch den Wert in diesem Eingabe-Feld beschrieben.

animationPathCoordinates `SoMFVec3f` Dieses Eingabe-Feld enthält die Koordinaten der *Stützpunkte* für die Animation des rechten Armes (siehe `animationCoordinatesOut` auf Seite 81).

animationPathHandPosture `SoMFEnum` Dieses Eingabe-Feld legt die *Stellung* der Hand zwischen zwei Stützstellen der Animation fest. Die Stellung der Hand

kann entweder *offen* (entspricht dem Wert `OPEN`) oder *geschlossen* (Wert `CLOSED`) sein (siehe `animationHandPostureOut` auf Seite 81).

animationChessMen `SoMFNode` Dieses Eingabe-Feld enthält einen oder mehrere `ChessManKit`-Knoten. Es sind dies die *Schachfiguren*, welche von der `AnimationEngine` während der Animation bewegt werden müssen (siehe `animationChessMenOut` auf Seite 82).

startAnimation `SoSFTrigger` Über dieses Eingabe-Feld wird der `AnimationEngine` mitgeteilt, daß die für die Animation benötigten Werte in den Eingabe-Feldern vorliegen und mit der Animation *begonnen* werden kann.

meshToIKANMatrix `SoSFMatrix` Dieses Eingabe-Feld enthält die Matrix für die *Umrechnung* der Koordinaten vom `MeshDeformer`- in das IKAN-Koordinatensystem.

inventorToMeshMatrix `SoSFMatrix` Dieses Eingabe-Feld enthält die Matrix für die *Umrechnung* der Koordinaten vom IKAN- in das `MeshDeformer`-Koordinatensystem.

animationIncrement `SoSFFloat` Der Wert in diesem Eingabe-Feld legt fest, wie viele *Zwischenwerte* für die Interpolation zwischen zwei Stützstellen berechnet werden. Da die `AnimationEngine` an die Zeit gekoppelt ist, bestimmt dieser Wert auch die Geschwindigkeit, mit der die Animation ausgeführt wird.

animationPickDropDelay `SoSFLong` Der Wert dieses Eingabe-Feldes bestimmt die Länge einer *Pause*, welche während der Animation nach dem Aufnehmen und Abstellen einer Schachfigur gemacht wird. Der Wert wird in Millisekunden angegeben.

elbowSwivelAngel `SoSFFloat` Dieses Eingabe-Feld bestimmt den *Rotationswinkel* des *Ellbogens* für die inverse Kinematik-Berechnungen. Eine Erklärung inklusive Abbildung findet sich in Kapitel 3.2.2 und in [TGB00] ab Seite 363.

enableJointLimits `SoSFBool` Dieses Eingabe-Feld legt fest, ob für die Berechnungen im inverse Kinematik-System die *Beschränkungen* für die Gelenkswinkel ein- oder ausgeschaltet sind. Bei einem Wert von `TRUE` sind die Beschränkungen aktiviert. Eine detaillierte Erklärung findet sich in [TGB00] ab Seite 365.

ikanGoalType `SoSFEnum` Dieses Eingabe-Feld bestimmt, ob für die Berechnungen im inverse Kinematik-System nur die *Position* des Endeffektors oder auch dessen *Ausrichtung* herangezogen wird. Dieses Feld kann die Werte `POSITION` oder `ORIENTATIONPOSITION` annehmen.

positionOnlyWristRotation `SoSFRotation` Dieses Eingabe-Feld wird verwendet, wenn das Feld `ikanGoalType` auf den Wert `POSITION` gesetzt ist. Es ermöglicht die *Rotation* der rechten Hand im Handgelenk, um einen optisch besseren Eindruck zu erreichen.

orientationWristAngel `SoSFFloat` Dieses Eingabe-Feld wird verwendet, wenn das Feld `ikanGoalType` auf den Wert `ORIENTATIONPOSITION` gesetzt ist. Dieser Wert entspricht dem Parameter ψ in [TGB00] ab Seite 372.

endEffectorAxis `SoSFVec3f` Diesem Eingabe-Feld wird jene Achse des Endeffektors (Hand) zugewiesen, welche entlang der durch das Feld `orientationGoalAlignmentAxis` vorgegebenen Achse ausgerichtet werden soll.

orientationGoalAlignmentAxis `SoSFVec3f` Dieses Eingabe-Feld legt fest, entlang welcher *Hauptachse* (X-, Y- oder Z-Achse) die `endEffectorAxis`-Achse ausgerichtet werden soll.

rootRotation `SoEngineOutput` (`SoSFRotation`) Dieses Ausgabe-Feld enthält die Rotation für die gesamte Geometrie der Figur. Eine *Modifikation* des Wertes veranlaßt den mit diesem Feld verbundenen `MeshDeformer`-Knoten, die an ihn übergebenen Feldwerte auszuwerten. Diesem Feld wird der Wert des Feldes `baseRootRotation` zugewiesen.

leftShoulderRotation, **leftElbowRotation**, **leftWristRotation** `SoEngineOutput` (`SoSFRotation`) Diese drei Ausgabe-Felder geben die Werte der Felder `baseLeftShoulderRotation`, `baseLeftElbowRotation` und `baseLeftWristRotation` unverändert weiter.

rightShoulderRotation, **rightElbowRotation**, **rightWristRotation** `SoEngineOutput` (`SoSFRotation`) Diese drei Ausgabe-Felder enthalten die Ergebnisse der Berechnungen, die mit Hilfe der IKAN-Bibliothek durchgeführt werden. Diese drei Werte sind die für die *Animation* notwendigen Rotationen für das Schulter-, Ellbogen- und Armgelenk der Figur.

rightHandThumb1, **rightHandIndex1**, **rightHandMiddle1**, **rightHandRing1**, **rightHandPinky1** `SoEngineOutput` (`SoSFRotation`) Die Werte dieser fünf Ausgabe-Felder bestimmen, ob die *Hand* der Figur geöffnet oder geschlossen ist.

readyWithInitialization `SoEngineOutput` (`SoSFBool`) Dieses Ausgabe-Feld wird auf den Wert `TRUE` gesetzt, wenn der Knoten `AnimationEngine` seine *Initialisierung* beendet hat.

animationFinished `SoEngineOutput` (`SoSFBool`) Der Wert dieses Ausgabe-Feldes wird auf `TRUE` gesetzt, wenn die *Animation* des Armes für einen Schachzug *beendet* wurde.

animationTouchDown `SoEngineOutput` (`SoSFBool`) Der Wert dieses Ausgabe-Feldes wird auf den Wert `TRUE` gesetzt, wenn während der Animation eine Schachfigur von der `AnimationEngine` auf dem Schachbrett *abgesetzt* wird. Details dazu in der Feldbeschreibung von `animationTouchDownIn` auf Seite 81.

Kapitel 5

Ergebnisse

„Jeder wünschte zu wissen, wie es zugehe, daß eine Maschine Schach spiele? Eine denkende Maschine? – War der Schachspieler dergleichen? Lässet sich überhaupt dergleichen denken? – Und, wenn sie möglich wäre, könnte sie in der kurzen Zeit von sechs Monaten, in welcher dieser Automat entstand, zu Stande gebracht werden? – Schon die beyden letzten Fragen zeigen dem, der sie beantworten will, ein so weites Feld voll Bedenklichkeiten, daß man mir hoffentlich verzeihen wird, wenn ich mich auf die sich dahin beziehende, von einigen Personen angenommene Hypothese, nicht weiter einlasse.“

– Joseph Friedrich Freyherr zu Racknitz, aus dem Buch *Über den Schachspieler des Herrn von Kempelen und dessen Nachbildung*, 1789

5.1 Zusammenfassung

Ausgehend von einem vorliegenden Konzept sollte der Prototyp einer zeitgemäßen Reproduktion des historischen Automaten entwickelt werden. Zur Durchführung dieser Aufgabe war neben der Entwicklung der Software auch eine Auswahl der benötigten Hardware erforderlich.

Als *Augmented Reality*-Plattform wurde das auf OPEN INVENTOR basierende STUDIERSTUBE-System gewählt. Eine breite Palette an Varianten für die Videoausgabe und die Unterstützung von verschiedenen Betriebssystemen ermöglichte eine flexible Wahl der Geräte. So konnte die Anwendung im Laufe ihrer Entwicklung auf verschiedenen passiven und aktiven Stereoprojektionssystemen ohne Änderung in den Quelldateien erprobt werden. Auch der Einsatz von OPENTRACKER für die Bereitstellung der Daten für das Hand- und Kopf-Tracking, erleichterte die Entwicklung des Prototypen. Der modulare Aufbau und das von OPENTRACKER verwendete *Datenflußkonzept* vereinfachte das Hinzufügen der neuen Softwarekomponente für den optischen DYNASIGHT-Tracker. Das OPENTRACKER-Modul für den DYNASIGHT-Tracker wird bereits im Rahmen eines anderen Projektes wiederverwendet.

Die Auswahl der für die Videoprojektion benötigten Komponenten wurde in Zusammenarbeit mit BARCO durchgeführt. Es wurden unterschiedliche Stereoprojektionssysteme und Rückprojektionsmaterialien untersucht. Im Rahmen dieser Arbeit konnte ermittelt werden, daß eine passive Stereoprojektion der Videosignale auf eine

holographische Rückprojektionsscheibe ein Stereobild mit ausreichender Qualität erzeugen kann. Für die passive Stereoprojektion kommen speziell ausgerichtete Polarisationsfilter im Linsensystem der beiden Videoprojektoren und der Stereobrille für den Anwender zum Einsatz.

Die während dieses Projektes entwickelte Software ermöglicht es einem Benutzer, mit einem virtuellen Opponenten – dem „Türkischen Schachspieler“ – eine Partie Schach zu spielen. Auf eine *einfache* und *intuitive* Interaktion mit der Installation wurde ein Hauptaugenmerk gelegt. Die Interaktion zwischen dem menschlichen Spieler und der virtuellen Szene geschieht über die virtuellen Schachfiguren. Wie bei einem realen Schachspiel bewegt der Benutzer mit der bloßen Hand eine Schachfigur vom Ausgangsfeld zur gewünschten Zielposition. Optische Anhaltspunkte, wie z.B. das farbliche Hervorheben des aktuellen Feldes und der ausgewählten Schachfigur, erleichtern die Interaktion.

Eine gute Abstimmung zwischen realem und virtuellem Raum ist für das positive Erfolgserlebnis des Anwenders sehr wichtig. Zu diesem Zweck müssen die Daten, welche von der Handtracking-Software geliefert werden, möglichst genau mit dem Inhalt der virtuellen Szene korrespondieren.

5.2 Mögliche Erweiterungen

Wie bei allen Projekten gibt es immer Möglichkeiten für Verbesserungen. Einige dieser Ideen sollen im folgenden Abschnitt beschrieben werden.

5.2.1 Verteilte Berechnung

Der Prozeß des Schachprogrammes und die STUDIERSTUBE-Anwendung werden auf demselben Rechner abgearbeitet. Es kann daher zu einem Engpaß in der Rechnerleistung kommen. Eine Lösung für dieses Problem wurde in Kapitel 4.2.5 vorgestellt. Durch diesen Lösungsansatz wird aber die Möglichkeit der Vorausberechnung weiterer Züge durch das Schachprogramm nicht genützt. Durch die Aufteilung dieser beiden Programmteile auf zwei Rechner stünde wieder genügend Rechnerleistung zur Verfügung. Die Kommunikation zwischen der `AnimationEngine` und dem Schachprozeß könnte über ein Datennetzwerk zwischen den beiden Rechnern erfolgen. Auch der Einsatz eines Mehrprozessorsystems ist denkbar. Einem der Prozessoren könnte exklusiv der Prozeß für die Berechnung der Schachzüge zugeordnet werden.

5.2.2 Schatten

OPEN INVENTOR bietet keine direkte Unterstützung für Schatten, welche von den Objekten in der Szene verursacht werden. Es hat sich aber gezeigt, daß Schatten den Realismus einer animierten Szene erhöhen.

5.2.3 Historischer Schachspieler

Aus den historischen Beschreibungen des „Türkischen Schachspielers“ in Kapitel 1.2.2 kann man entnehmen, daß es Unterschiede zu der hier beschriebenen An-

wendung gibt. So spielte der historische „Türke“ mit der linken Hand, die virtuelle Figur des Schachspielers aber mit der rechten Hand. Der historische Automat machte bei jeder Schachpartie den ersten Zug. Bei einem Spiel mit dem virtuellen „Türken“ hat das Vorrecht auf den ersten Schachzug immer der menschliche Spieler.

Weiters führte die historische Figur einige Bewegungen mit dem Kopf aus, welche noch nicht bei der Realisierung der virtuellen Figur berücksichtigt wurden.

5.3 Die Museumsinstallation

Aufbauend auf dem in dieser Arbeit entstandenen Softwareprototypen wird von der Firma IMAGINATION der „Türkische Schachspieler“ für das Technische Museum Wien weiterentwickelt. Die Abbildung 5.1 zeigt die endgültige Geometrie und Texturierung für die Figur, das Schachbrett und die Schachfiguren.

Bild 5.1: Endgültige Geometrie für den „Türkischen Schachspieler“.

In einer Seitenansicht ist in der Abbildung 5.2 der Aufbau der Stereoprojektion dargestellt. Im Gegensatz zu dem in dieser Arbeit verwendeten Hardwareaufbau ist die Kamera für das Handtracking in der Nähe des Bodens angebracht. Dies hat den Vorteil, daß bei der im Museum bestehenden Beleuchtungssituation die Kontur der Hand des menschlichen Spielers besser ermittelt werden kann.

Neben der Stereoprojektion kommt in der Museumsinstallation auch eine normale Videoprojektion mit einem Monobild zum Einsatz. Dieses Videobild wird auf eine zweite, milchige Rückprojektionsfläche abgebildet. Der Inhalt dieser Projektion ist eine Seitenansicht der in der Stereoprojektion dargestellten 3D-Szene. Mittels dieser Projektion können weitere Museumsbesucher dem Schachspiel folgen. Das Videosignal für die zweite Videoprojektion wird von einem weiteren Rechner geliefert. Die STUDIERSTUBE-Anwendungen auf beiden Rechnern werden über ein Datennetzwerk miteinander synchronisiert.

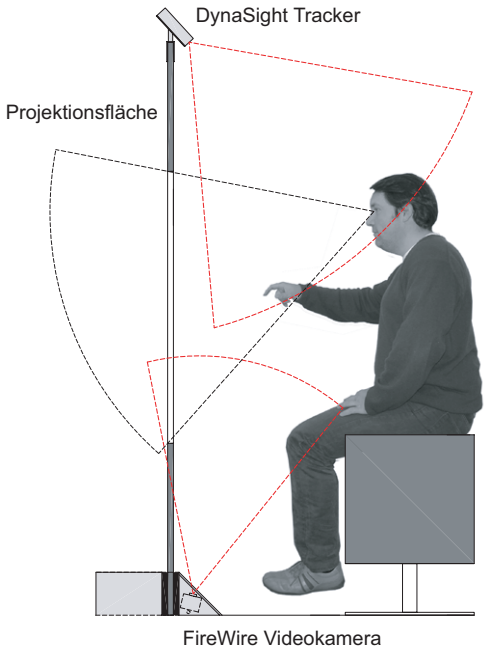


Bild 5.2: Seitenansicht der Stereoprojektion der Museumsinstallation
(mit freundlicher Unterstützung von DI Gernot Humer und DI Peter Hanisch).

Neben der Möglichkeit, mit dem „Türkischen Schachspieler“ eine Partie Schach zu spielen, erlaubt es die Museumsinstallation dem Besucher auch, etwas über die Geschichte des historischen Automaten zu erfahren. Für diesen Zweck wurde ein alter kolorierter Kupferstich in ein 3D-Modell umgewandelt. Mit Hilfe dieses 3D-Modells wird es dem Besucher auf einfache Weise ermöglicht, die Funktionsweise und Bedienung des überlieferten Schachautomaten zu ergründen.

Literaturverzeichnis

[AB92] Steve Aukstakalnis, David Blatner. *Silicon Mirage; The Art and Science of Virtual Reality*. Peachpit Press, Berkeley, CA, USA, 1992.

[Azu97] Ronald T. Azuma. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, Vol. 6, No. 4, pp. 355–385, 1997.

[Bad97] Norman I. Badler. Real-time virtual humans. *Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, pp. 4, 1997.

[Bar02a] Barco BarcoReality SIM 6 Ultra MM. Datenblatt. `http://www.barco.com/projection_systems/downloads/BR SIM 6 Ultra MM.pdf`. Belgium, July 2002.

[Bar02b] Barco Galaxy WARP. Datenblatt. `http://www.barco.com/projection_systems/downloads/Barco_Galaxy_Warp.pdf`. Belgium, July 2002.

[Bar02c] Barco Baron High-Performance Projection Table. Datenblatt. `http://www.barco.com/projection_systems/downloads/Baron_5_03.pdf`. Belgium, July 2002.

[BES03] Oliver Bimber, L. Miguel Encarnação, Dieter Schmalstieg. The virtual showcase as a new platform for augmented reality digital storytelling. *Proceedings of the workshop on Virtual environments 2003*, pp. 87–95. Zurich, Switzerland, 2003.

[BFS+01a] Oliver Bimber, Bernd Fröhlich, Dieter Schmalstieg, L. Miguel Encarnação. The Virtual Showcase. *IEEE Computer Graphics & Applications*, Vol. 21, No. 6, pp. 48–55, 2001.

[BFS+01b] Oliver Bimber, Bernd Fröhlich, Dieter Schmalstieg, L. Miguel Encarnação. Article in CG Topics. Virtual Showcases Today and Tomorrow, *CG Topics 4/2001*, Vol. 13, 2001.

[BGW+02] Oliver Bimber, Stephen M. Gatesy, Lawrence M. Witmer, Ramesh Raskar, L. Miguel Encarnação. Merging Fossil Specimens with Computer-Generated Information. *IEEE Computer Society Press*, Vol. 35, No. 9, pp. 25–30, 2002.

[BT97] Selim Balcisoy, Daniel Thalmann. Interaction between Real and Virtual Humans in Augmented Reality. *Proceedings of the Computer Animation '97*, pp. 31, 1997.

[BTP+00] Selim Balcisoy, Rémy Torre, Michal Ponder, Pascal Fua, Daniel Thalmann. Augmented Reality for Real and Virtual Humans. *Proceedings of the International Conference on Computer Graphics*, pp. 303–308, 2000.

[CCZ+00] Diane Chi, Monica Costa, Liwei Zhao, Norman Badler. The EMOTE model for effort and shape. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 173–182, 2000.

[DGM+93] David Drascic, Julius J. Grodski, Paul Milgram, Ken Ruffo, Peter Wong, Shumin Zhai. ARGOS: a display system for augmenting reality. *Proceedings of the ACM Conference on Human Factors in Computing Systems (INTERCHI'93)*, pp. 521. Amsterdam, The Netherlands, 1993.

[Fid03] Fédération Internationale des Échecs. FIDE Handbuch. `http://www.fide.com/official/handbook.asp`. 2003.

[GS00] Georg Glaeser, Ernst Strouhal. Von Kempelen's Chess Playing Pseudo-Automaton and zu Racknitz's Explanation of its Controls, in 1789. *Intern. Symposium on History of Machines – Proceedings HMM2000* (Editor Marco Ceccarelli), pp. 351–360. Kluwer Academic Publishers, 2000.

[Hin84] Karl Friedrich Hindenburg. *Ueber den Schachspieler des Herrn von Kempelen. Nebst einer Abbildung und Beschreibung seiner Sprechmaschine*. Leipzig, 1784.

[HM01] Elliotte Rusty Harold, W. Scott Means. *XML in a Nutshell*. O'Reilly & Associates, Inc.; 2nd edition (June, 2002), Sebastopol, CA, USA, 2001.

[HSF+99] Gerd Hesina, Dieter Schmalstieg, Anton Fuhrmann, Werner Purgathofer. Distributed Open Inventor: a practical approach to distributed 3D graphics. *Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 74–81. London, United Kingdom, 1999.

[KB99] Hirokazu Kato, Mark Billinghurst. Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pp. 85–94. San Francisco, CA, USA, 1999.

[KKK+94] Yoshihito Koga, Koichi Kondo, James Kuffner, Jean-Claude Latombe. Planning motions with intentions. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 395–408, 1994.

[Kon94] Koichi Kondo. Inverse Kinematics of a Human Arm. Rep. STAN-CS-TR-94-1508, `ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/94/1508/CS-TR-94-1508.pdf`, Department of Computer Science, Stanford University, CA, USA, 1994.

[KS03] Hannes Kaufmann, Dieter Schmalstieg. Mathematics And Geometry Education With Collaborative Augmented Reality. *Computers & Graphics*, Vol. 27, No. 3, pp. 339–345, 2003.

[KW00] Stefan Kopp, Ipke Wachsmuth. Planning and Motion Control in Lifelike Gesture: A Refined Approach. *Proceedings of the 2000 Conference on Computer Animation*, pp. 92, 2000.

[LB03] Ying Liu, Norman I. Badler. Real-Time Reach Planning for Animated Characters Using Hardware Acceleration. *Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, pp. 86, 2003.

[Lev00] Gerald M. Levitt. *The Turk, chess automaton*. McFarland & Company, Incorporated Publishers, 2000.

[Mad02] Steffen Mader. Kommerziell verfügbare Spiegelmaterialien für den Einsatz am Virtual Showcase. Virtual Showcases – Presenting hybrid Exhibits. IST-2000-28610, 12. Dezember 2002.

[Man03] Tim Mann. Chess Engine Communication Protocol. `http://www.tim-mann.org/xboard/engine-intf.html`. 2003.

[MPB02] Steffen Mader, Erich Pohn, Oliver Bimber. Report on Proof-of-Concept Prototypes. Virtual Showcases – Presenting hybrid Exhibits. Deliverable D3.2, Work Package WP3, IST-2000-28610, 2002.

[MTU+94] Paul Milgram, Haruo Takemura, Akira Utsumi, Fumio Kishino. Augmented Reality: A Class of Displays on the Reality-virtuality Continuum. SPIE Vol. 2351-34, *Telemanipulator and Telepresence Technologies*, pp. 282–292, 1994.

[MVR+02] Steffen Mader, Jörg Voskamp, Alexandra Reitelmann, Dieter Schmalstieg, Erich Pohn, Ottmar Moritsch, Michael Gervautz, Sigrid Brunner, Isabel Silva. Specification of System Architecture and Component Interfaces. Virtual Showcases – Presenting hybrid Exhibits. Deliverable D2, Work Package WP2, IST-2000-28610, 2002.

[NYC+99] Ulrich Neumann, Suya You, Youngkwan Cho, Jongweon Lee, Jun Park. Augmented reality tracking in natural environments. *In International Symposium on Mixed Realities (ISMR'99)*. Tokyo, Japan, 1999.

[Rac89] Joseph Friedrich Freyherr zu Racknitz. *Ueber den Schachspieler des Herrn von Kempelen und dessen Nachbildung*. Mit sieben Kupfertafeln. Leipzig und Dresden, 1789.

[RL01] Ramesh Raskar, Kok-Lim Low. Interacting with spatially augmented reality. *Proceedings of the 1st international conference on Computer graphics, virtual reality and visualisation*, pp. 101–108. Camps Bay, Cape Town, South Africa, 2001.

[RN95] Jun Rekimoto and Katashi Nagao. The World Through the Computer: Computer Augmented Interaction with Real World Environments. *Proceedings of the 8th annual ACM symposium on User interface and software technology (UIST'95)*, pp. 29–36. Pittsburgh, PA, USA, 1995.

[RS01a] Gerhard Reitmayr, Dieter Schmalstieg. An Open Software Architecture for Virtual Reality Interaction. *Proceedings of the ACM symposium on Virtual reality software and technology (VRST'01)*. Banff, Canada, Nov. 15-17, 2001.

[RS01b] Gerhard Reitmayr, Dieter Schmalstieg. OpenTracker-An Open Software Architecture for Reconfigurable Tracking based on XML. *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, pp. 285, 2001.

[SC92] Paul S. Strauss, Rikk Carey. An object-oriented 3D graphics toolkit. *Proceedings of the 19th annual conference on Computer graphics and interactive techniques (SIGGRAPH'92)*, pp. 341–349, 1992.

[Sch03] Dieter Schmalstieg. Unterlagen zur Vorlesung „Virtual Reality“. `http://www.ims.tuwien.ac.at/teaching/vr/folien/index.php`. Technische Universität, Wien, Austria, 2003.

[SEG98] Zsolt Szalavári, Erik Eckstein, Michael Gervautz. Collaborative Gaming in Augmented Reality. *Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST'98)*, pp. 195–204. Taipei, Taiwan, Nov. 2-5, 1998.

[SES99] Dieter Schmalstieg, L. Miguel Encarnação, Zsolt Szalavári. Using transparent props for interaction with the virtual table. *Proceedings of the ACM Symposium on Interactive 3D Graphics (I3DG'99)*, pp. 147–153. Atlanta, Georgia, USA, April 1999.

[SF03] Rainer Splechtna, Anton Fuhrmann. Extending the working volume of projection-based mixed reality systems. *Proceedings of the workshop on Virtual environments 2003*, pp. 287–292. Zurich, Switzerland, 2003.

[SFH+00] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L. Miguel Encarnação, Michael Gervautz, Werner Purgathofer. The studierstube augmented reality project. *Presence: Teleoperators and Virtual Environments*, Vol. 11, No. 1, pp. 33–54, 2002.

[SFH00] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina. Bridging Multiple User Interface Dimensions with Augmented Reality. *Proceedings of the 3rd International Symposium on Augmented Reality (ISAR 2000)*. Munich, Germany, October 2000.

[SG97] Zsolt Szalavári, Michael Gervautz. The Personal Interaction Panel: A two-handed interface for augmented reality. *Proceedings of Eurographics ’97*, 16(3) of Computer Graphics Forum, pp. 335–346. Budapest, Hungary, 1997.

[SRH03] Dieter Schmalstieg, Gerhard Reitmayr, Gerd Hesina. Distributed applications for collaborative three-dimensional workspaces. *Presence: Teleoperators and Virtual Environments*, Vol. 12, No. 1, pp. 52–67, 2003.

[Sta02] Tom Standage. *Der Türke. Die Geschichte des ersten Schachautomaten und seiner Abenteuerlichen Reise um die Welt*. Aus dem Englischen von Thomas Merk und Thomas Wollermann. Campus Verlag, Frankfurt/New York, 2002.

[Str00] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[Str96] Ernst Strouhal. *Acht x acht, zur Kunst des Schachspiels*. Springer, Wien, Austria, 1996.

[Tam02] Hideyuki Tamura. Steady Steps and Giant Leap Toward Practical Mixed Reality Systems and Applications. `http://informatiksysteme.pt-it.de/vr-ar-2/projekte/referat_TAMURA.pdf`, Internationale Statustagung „Virtuelle und Erweiterte Realität“, Leipzig, Germany, Nov. 5-6, 2002.

[TBF+00] Rémy Torre, Selim Balcisoy, Pascal Fua, Daniel Thalmann. Interaction Between Real and Virtual Humans: Playing Checkers. *Proceedings Eurographics Workshop On Virtual Environments*, 2000.

[TGB00] Deepak Tolani, Ambarish Goswami, Norman I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models and Image Processing*, Vol. 62, No. 5, pp. 353–388, 2000.

[TT97] Nadia Magnenat Thalmann, Daniel Thalmann. Animating Virtual Actors in Real Environments. *Multimedia Systems*, May 1997, pp. 113–125, 1997.

[TYS+00] Akinari Takagi, Shoichi Yamazaki, Yoshihiro Saito, Naosato Taniguchi. Development of a Stereo Video See-through HMD for AR Systems. *Proceedings of IEEE and ACM International Symposium on Augmented Reality (ISAR)*, pp. 68–77. Munich, Germany, 2000.

[Wei02] Claudia Weiß. Inverse Kinematik. `http://medien.informatik.uni-ulm.de/lehre/courses/ss02/Computergrafik/ClaudiaWeiss.pdf`. Universität Ulm, 2002.

[Wer93] Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3d Graphics with Open Inventor, Release 2*. Addison-Wesley Longman Publishing Co., Inc., 1993.

[Wer94] Josie Wernecke. *The Inventor Toolmaker: Extending Open Inventor, Release 2*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[WES00] Werner Wohlfahrter, L. Miguel Encarnação, Dieter Schmalstieg. Interactive Volume Exploration on the StudyDesk. *Proceedings of the 4th International Immersive Projection Technology Workshop (IPT'00)*. Iowa State University, Ames, Iowa, USA, June 2000.

[Win83] Karl Gottlieb von Windisch. *Briefe über den Schachspieler des Hrn. von Kempelen, nebst drey Kupferstichen die diese berühmte Maschine vorstellen*. Basel, Schweiz, 1783.

[WK02] Ipke Wachsmuth, Stefan Kopp. Lifelike Gesture Synthesis and Timing for Conversational Agents. *Revised Papers from the International Gesture Workshop on Gesture and Sign Languages in Human-Computer Interaction*, pp. 120–133, 2002.

[ZB94] Jianmin Zhao, Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics (TOG)*, Vol. 13, No. 4, pp. 313–336, 1994.

Link-Sammlung

3M	http://www.3m.com/
Advanced Realtime Tracking GmbH	http://www.ar-tracking.de/
Alias Systems	http://www.aliaswavefront.com/
ARToolKit	http://www.hitl.washington.edu/artoolkit/
Barco N.V.	http://www.barco.com/
Canon Inc.	http://www.canon.com/
coin	http://www.coin3d.org/
Dell Ges.m.b.H.	http://www.dell.at/
Denso Corporation	http://www.denso.co.jp/en/
dnp denmark AS	http://www.dnp.dk/
Fédération Internationale des Échecs	http://www.fide.com/
Fujitsu Siemens Computers GmbH	http://www.fujitsu-siemens.at/
G&G Folienwelt	http://www.folienwelt.at/
G+B pronova GmbH	http://www.holopro.com/
GNU Chess	http://www.gnu.org/software/chess/
Imagination Computer Services GesmbH	http://www.imagination.at/
InFocus Corporation	http://www.infocus.com/
Lumin Visual Technologies AG	http://www.lumin.de/
Microsoft Corporation	http://www.microsoft.com/
NVIDIA Corporation	http://www.nvidia.com/
OpenTracker	http://www.studierstube.org/opentracker/
Origin Instruments Corporation	http://www.orin.com/

sax3d.com gmbh	`http://www.sax3d.com/`
Silicon Graphics, Inc.	`http://www.silicongraphics.com/`
Sony Corporation	`http://www.sony.net/`
StereoGraphics Corporation	`http://www.stereographics.com/`
Studierstube	`http://www.studierstube.org/`
Systems in Motion AS	`http://www.sim.no/`
Tech Gate Vienna Wissenschafts- und Technologiepark GmbH	`http://www.techgate.at/`
Unibrain Inc.	`http://www.unibrain.com/`
Virtual Showcases	`http://www.virtualshowcases.com/`
Virtual Vision, Inc.	`http://www.virtualvision.com/`
VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH	`http://www.vrvis.at/`

Danksagung

Als Erstes möchte ich MICHAEL GERVAUTZ für die ausgezeichnete Betreuung während meiner Diplomarbeit danken. Durch seine Denkanstöße und Diskussionen konnten einfache und elegante Lösungen im Rahmen dieser Arbeit gefunden werden.

Ich möchte auch den Kollegen der Firma IMAGINATION für ihren Rat und ihre Hilfe während der Entwicklung der Software danken. Allen voran REINHARD SAINITZER und SIGRID BRUNNER. Vielen Dank auch an RAINER SPLECHTNA, aus dem VRVis Zentrum für Virtual Reality und Visualisierung, der mir ausdauernd mit Rat zur Seite stand.

Diese Arbeit wäre nicht möglich gewesen ohne die Unterstützung aus dem STUDIERSTUBE und OPENTRACKER Team. Ich bedanke mich hier speziell bei DIETER SCHMALSTIEG, FLORIAN LEDERMANN und GERHARD REITMAYR für ihre schnelle und kompetente Hilfe.

Mein Dank gilt auch dem Wiener VRCENTER, vornehmlich HERBERT BUCHEGGER, für die Möglichkeit die Anwendung auf der vorhandenen VR-Hardware zu testen und den Architekten GERNOT HUMER und PETER HANISCH für das zur Verfügung gestellte Bildmaterial.

Weiters möchte ich meinen Eltern danken, die mir dieses Studium ermöglicht haben.