Diplomarbeit

Repräsentation realer Objekte in Outdoor Augmented Reality Systemen

durchgeführt am

Fraunhofer-Institut für Graphische Datenverarbeitung

Darmstadt

von

Rebecca Krenzel

Fachhochschule Gießen-Friedberg
Bereich Friedberg
Fachbereiche IEM, MND, MNI
Studiengang Medieninformatik

Wintersemester 03/04

Referentin:
Prof. Dr.-Ing. Dipl.-Math. Monika Lutz

Korreferent:
Dipl.-Ing. Daniel Holweg

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass die vorliegende Diplomarbeit ohne unzulässige Hilfe und ausschließlich unter Verwendung der angegebenen Literatur angefertigt wurde.

Frankfurt, 11. Februar 2004

Rebecca Krenzel

Kurzfassung

Gegenstand der hier vorgestellten Arbeit ist die Konzeption und Entwicklung einer auf Java und Java 3D basierten Softwarekomponente, die dazu dient, den realistischen Eindruck der Darstellung von Augmented Reality Szenen zu verbessern. Mit Hilfe dieser Softwarekomponente können optische Verdeckungen zwischen virtuellen und realen Objekten in Outdoor Augmented Reality Systemen berücksichtigt und visualisiert werden können. Diese Komponente repräsentiert reale, verdeckende Objekte im Szenengraph der AR-Szene und visualisiert damit die Objektverdeckung. Hierfür wird analysiert, unter welchen Bedingungen eine optische Verdeckung auftreten kann, und darauf aufbauend ein Konzept zur Berechnung und Darstellung der verdeckenden Objekte entwickelt. Die konkrete Umsetzung verwendet dabei einen Algorithmus, der in mehreren Stufen die Menge der verdeckenden Objekte berechnet und im AR-System visualisiert. Um die Funktionalität des Algorithmus und damit der Objektverdeckungskomponente zu demonstrieren, wird eine graphische Benutzeroberfläche entwickelt, welche die Objektverdeckung implementiert. Der direkte Vergleich zeigt, dass sich virtuelle Objekte unter dem Einsatz der entwickelten Objektverdeckungskomponente realistischer in eine AR-Szene einbetten lassen, als bei Nichtverwendung der Software.

Abstract

The subject of this diploma thesis is the conception and development of a Java and Java 3D based software component, which serves to improve the realistic impression of augmented reality scenes. By this software component optical occlusion between virtual and real objects in outdoor augmented reality systems can be represented and visualized. By representing real, occluding objects in the scene-graph of the augmented reality scene this software component visualizes the occlusion between real and virtual objects. Therefore, it is analyzed under which conditions an optical oc-

clusion can occur. Based on this analysis a concept for the computation and representation of the occluding objects is developed. The concrete implementation uses thereby an algorithm, which computes the quantity of the occluding objects in several stages and visualizes them in the augmented reality system. In order to demonstrate the functionality of the algorithm and the software component, a graphical user interface is developed, which implements the occlusion component. The direct comparison shows that virtual objects can be embedded more realistically into an augmented reality scene under the employment of the developed software component as without using the software.

Danksagung

Ich möchte mich bedanken

bei Prof. Dr.-Ing. Dipl.-Math. Monika Lutz für ihr Engagement und die ausgezeichnete Betreuung,

bei Dipl.-Ing. Daniel Holweg für seine Unterstützung und wertvollen Tipps bei der Entwicklung und Implementierung der Software sowie seine Bereitschaft, diese Arbeit als Zweitgutachter zu betreuen,

bei den Mitarbeitern der Abteilung Graphische Informationssysteme des Fraunhofer-Instituts für Graphische Datenverarbeitung in Darmstadt für die äußerst angenehme Arbeitsumgebung und viele beantwortete Fragen

und nicht zuletzt bei meiner Familie für ihr Vertrauen, das Korrekturlesen und die konstruktive Kritik.

Des Weiteren möchte ich meiner Familie danken, die mir durch ihre anhaltende Unterstützung dieses Studium ermöglicht hat.

Frankfurt, 11. Februar 2004

Rebecca Krenzel

Inhaltsverzeichnis

1 Einleitung 1

1.1 Motivation und Zielsetzung 1

1.2 Inhaltliche Übersicht 3

2 Grundlagen 5

2.1 Einführung 5

2.2 Virtual und Augmented Reality 5

2.2.1 Definition von Virtual Reality 5

2.2.2 Definition von Augmented Reality 7

2.2.3 Head-Mounted Displays 11

2.2.4 Tracking-Systeme 13

2.3 Programmiertechnische Grundlagen 16

2.3.1 Einleitung 16

2.3.2 Java 16

2.3.3 Swing 17

2.3.4 Java 3D 18

2.4 Zusammenfassung 22

3 Stand der Forschung 24

3.1 Einleitung 24

3.2 Problematik der Objektverdeckung 24

3.3 Modellbasierter Ansatz 26

3.4 Tiefenbasierter Ansatz mit Realitätsmodell 29

3.5 Tiefenbasierter Ansatz ohne Realitätsmodell 30

3.6 Konturbasierter Ansatz 32

3.7 Bewertung der Lösungsansätze 34

4 Anforderungsanalyse 36

4.1 Übersicht 36

4.2 Reale Objekte 37

4.2.1 Begriffsbestimmung 37

4.2.2 Reale Objekte mit statischem Verhalten 37

4.2.3 Reale Objekte mit dynamischem Verhalten 38

4.3 Virtuelle Objekte 39

4.3.1 Begriffsbestimmung 39

4.3.2 Virtuelle Objekte mit statischem Verhalten 39

4.3.3 Virtuelle Objekte mit dynamischem Verhalten 40

4.4 Sichtbereich 41

4.4.1 Darstellung der AR-Szene 41

4.4.2 Allgemeines zum Sichtbereich 42

4.4.3 Position der abbildenden Komponenten 44

4.4.4 Orientierung der abbildenden Komponenten 47

4.4.5 Blickwinkel der abbildenden Komponenten 48

4.4.6 Abstand zwischen virtueller Kamera und Objekten 50

4.5 Anforderungen an die Geometrien und ihre Darstellung 51

4.6 Zusammenfassung der Anforderungen 53

5 Konzeption der Objektverdeckungskomponente 56

5.1 Übersicht 56

5.2 Anforderung und Zielsetzung 56

5.3 Konzeptinhalte 57

5.3.1 Übersicht 57

5.3.2 Verwaltung der 3D-Komponenten 58

5.3.3 Behandlung der Objektverdeckung 60

5.4 Zusammenfassung 66

6 Entwicklung der Objektverdeckungskomponente 67

6.1 Übersicht 67

6.2 Architektur und Paketstruktur 67

6.3 Aufbau der Szenengraphstruktur 69
6.3.1 Die Klassen des Pakets „scene“ 69
6.3.2 Aufbau des Szenengraphen 73
6.4 Behandlung der Objektverdeckung 74
6.4.1 Die Klassen des Pakets „occlusion“ 74
6.4.2 Datenhaltung 76
6.4.3 Schnittstelle des Objektverdeckungspakets 81
6.4.4 Berechnung und Visualisierung 83

7 Entwicklung der Testumgebung 90

7.1 Übersicht 90
7.2 Entwurf der Benutzeroberfläche 90
7.3 Benutzeroberfläche als Testumgebung 92
7.3.1 Die Klassen des Pakets „gui“ 92
7.3.2 Aufbau des Hauptfensters und der Menüleiste 94
7.3.3 Genereller Aufbau der internen Fenster 96
7.3.4 Aufbau des Visualisierungsfensters 97
7.3.5 Aufbau des Navigationsfensters 98
7.3.6 Aufbau des Objektfensters 99
7.3.7 Aufbau des Fensters zur Steuerung der Objektverdeckung .101
7.4 Die Klassen des Pakets „data“ 103
7.5 Testergebnisse 104

8 Zusammenfassung und Ausblick 109

9 Literaturverzeichnis 112

Anhang – Die CD zur Diplomarbeit 116

1 Einleitung

1.1 Motivation und Zielsetzung

Augmented Reality Systeme besitzen die Fähigkeit, gezielt virtuelle, künstliche Informationen und Objekte in die natürliche Umgebung einzublenden und damit visuell exakt zu überlagern. So können beispielsweise dreidimensionale, graphische Rekonstruktionen über reale, noch existierende Bauwerke abgebildet werden, um damit dem Betrachter einen besseren Eindruck von der Vergangenheit zu verschaffen. Genauso kann ein Architekt seine Vision eines geplanten Gebäudes durch den Einsatz von Augmented Reality (AR) anschaulich präsentieren, indem eine Graphik des Gebäudes visuell in die reale Landschaft eingebettet wird (Abbildung 1.1).

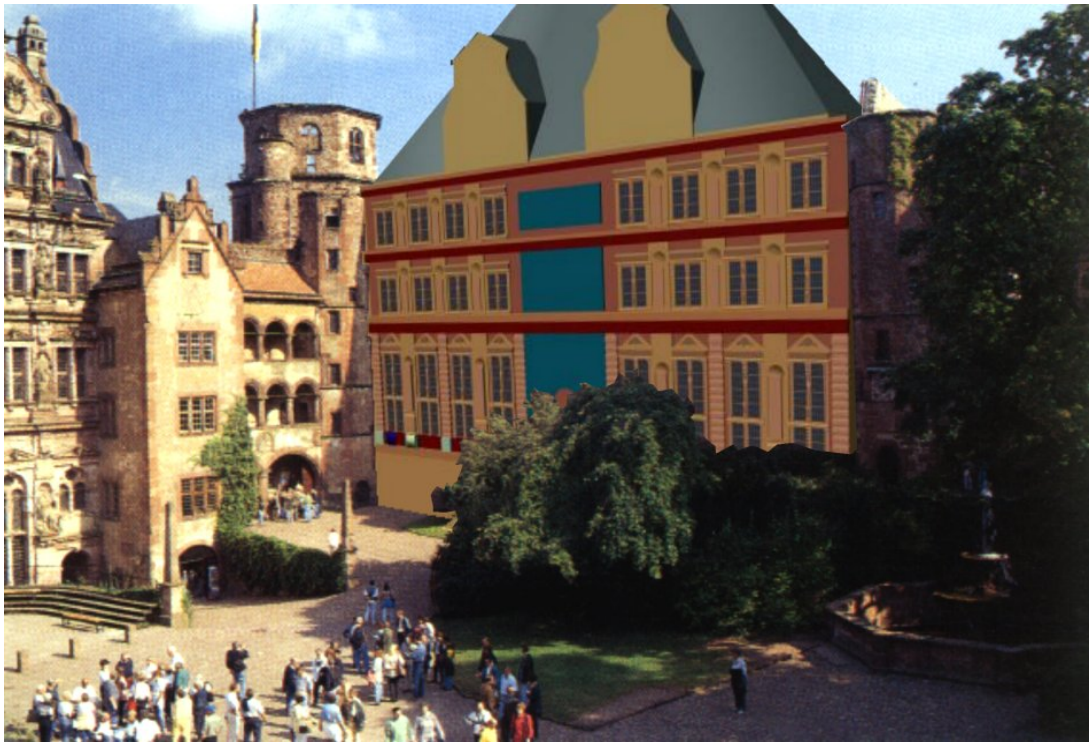


Abbildung 1.1: Virtuelle Rekonstruktion am Heidelberger Schloss

Die virtuellen Objekte werden im Raum jedoch nicht immer ihrer realen räumlichen Anordnung entsprechend dargestellt, da die virtuelle Information das Bild der Realität grundsätzlich überlagert. Dadurch werden auch virtuelle Objekte, die sich räum-

lich hinter einem realen Gegenstand befinden, immer vor der realen Umgebung abgebildet. Die fehlende Integration einer solchen Objektverdeckung wirkt sich oft störend auf den visuellen Eindruck des Betrachters aus. Um mit dem AR-System einen realistischen Eindruck zu erreichen, müssen auch diese realen Objekte aus der momentanen natürlichen Umgebung des Betrachters bei der Erstellung der virtuellen Szene berücksichtigt werden. Damit können Objektverdeckungen wirklichkeitsnah nachgebildet und der Eindruck der nahtlosen Verbindung von realer und virtueller Welt verbessert werden.

Das Ziel dieser Arbeit ist die Berücksichtigung solcher realen Objekte in einem Augmented Reality System, um dadurch Objektverdeckungen zu visualisieren. Hierfür sind die Berechnung der verdeckenden, realen Objekte sowie ihrer Repräsentation und Darstellung in der virtuellen Informationswelt erforderlich. Dabei soll das Konzept des Szenengraphen sinnvoll erweitert werden, um reale Objekte zu berücksichtigen und damit mögliche visuelle Verdeckungen durch diese Objekte darstellen zu können. Die Geometrieinformationen sollen dabei auf ein sinnvolles Maß beschränkt werden, um die graphische Leistungsfähigkeit des AR-Systems zu erhalten. Die Berechnung und Visualisierung dieser Objektverdeckung soll zudem als eine eigenständig arbeitende Einheit entwickelt werden, um den Einsatz in mehreren AR-Systemen zu gewährleisten. Um praktisch zu demonstrieren, wie die Objektverdeckung im AR-System umgesetzt werden kann, ist zusätzlich die Entwicklung einer graphischen Benutzeroberfläche erforderlich. Die Umsetzung dieses Ziels verteilt sich dabei auf mehrere Aufgabengebiete:

- Es ist zu analysieren, unter welchen Bedingungen eine Objektverdeckung in einem AR-System auftreten kann und welche Anforderungen sie an eine Repräsentation realer Objekte in diesem System stellt.
- Dazu soll ein Konzept entwickelt werden, über welches eine Objektverdeckung berechnet und dargestellt werden kann.
- Das Konzept soll zudem anhand einer graphischen Testumgebung praktisch umgesetzt werden.

Letztendlich soll eine Softwarekomponente entstehen, durch deren Einsatz visuelle Verdeckungen zwischen realen und virtuellen Objekten in Augmented Reality Systemen berücksichtigt und durch die Repräsentation der realen Objekte in der virtuellen Welt dargestellt werden können. Für die programmiertechnische Umsetzung dieser Aufgabe sollen Java und Java 3D verwendet werden.

1.2 Inhaltliche Übersicht

Im nachfolgenden zweiten Kapitel werden die Grundlagen, auf welchen die Umsetzung der oben beschriebenen Aufgabe basiert, erläutert. Dabei findet eine Klärung der Begriffe Virtual Reality und Augmented Reality statt und es wird erläutert, welche Hardware für den Einsatz dieser Techniken erforderlich ist. In einem weiteren Abschnitt der Grundlagen werden Technologien wie Java und Java 3D vorgestellt, da die praktische Realisierung der Objektverdeckung im Rahmen dieser Arbeit auf diesen aufbaut.

Das dritte Kapitel behandelt die allgemeine Problematik der Objektverdeckung in AR-Systemen und geht dabei vergleichend auf verschiedene, in der derzeitigen Forschung gängige Lösungsansätze ein.

Im vierten Kapitel findet aufbauend dem dritten Kapitel eine Analyse der Bedingungen statt, unter welchen eine Objektverdeckung zwischen realen und virtuellen Objekten auftreten kann. Hierfür werden zunächst die Eigenschaften der an einer Objektverdeckung beteiligten realen und virtuellen Objekte untersucht. Die konkreten Bedingungen zur Berechnung einer Objektverdeckung ergeben sich aus der Analyse der Umstände, unter welchen eine Verdeckung dargestellt werden muss. Die Anforderungen, die sich daraus an die Visualisierung der Objektverdeckung ergeben, sind ebenfalls Bestandteil des vierten Kapitels.

Aufbauend auf den im vorhergehenden Kapitel gewonnenen Erkenntnissen wird im fünften Kapitel das Konzept zur praktischen Umsetzung der Objektverdeckung vorgestellt. Im sechsten Kapitel wird beschrieben, wie dieses Konzept realisiert wird, indem auf konkrete Programmiermaßnahmen eingegangen wird. Für die Entwicklung der Testumgebung ist das siebte Kapitel vorgesehen.

Begriffe aus der Software-Programmierung wie Schlüsselworte aus Programmiersprachen, Programmfragmente, Klassen-, Objekt- oder Methodennamen werden in dieser Arbeit zur Verdeutlichung durch die Schriftart `Courier New` typografisch hervorgehoben.

2 Grundlagen

2.1 Einführung

Diese Arbeit behandelt die Problematik und Visualisierung der Objektverdeckung zwischen realen und virtuellen Objekten in einem Augmented Reality System. Um das Verständnis der nachfolgenden Kapitel und damit der Problematik der Objektverdeckung zu vereinfachen, soll in diesem Kapitel ein grundlegender Überblick über die in dieser Arbeit verwendeten Begriffe und Techniken gegeben werden.

Hierfür werden zuerst die Begriffe Virtual Reality und Augmented Reality geklärt und ein kleiner Einblick in die für diese Techniken notwendige Hardware gegeben. Ein weiterer Abschnitt beschäftigt sich mit Technologien, die konkret in dieser Arbeit angewendet werden. Er bildet die Grundlage für das Verständnis der technischen Konzeption und Realisierung der Objektverdeckung, da sie mit diesen Technologien berechnet und dargestellt wird. Beschrieben werden die Konzepte zu Java, Java 3D und Swing.

2.2 Virtual und Augmented Reality

2.2.1 Definition von Virtual Reality

Virtual Reality (VR, virtuelle Realität) ist ein Überbegriff für alle Konzepte und Techniken, die dem menschlichen Betrachter eine Welt vorspiegeln, indem sie die natürliche audiovisuelle und räumliche Wahrnehmung durch den Computer simulieren. Diese Welt existiert nicht real, sondern nur als eine Art virtuelles Abbild im Speicher eines Computers. Virtual Reality umfasst Methoden für die räumliche Darstellung von Objekten, für die räumliche Klangausbreitung und für eine Vielzahl an Interaktionsmöglichkeiten. Dabei wird die simulierte, virtuelle Welt erst durch Wiedergabe auf einem geeigneten Medium für den Betrachter wahrnehmbar. Besonders wichtig für VR-Anwendungen ist zudem, dass sie einen hohen Grad an Interaktivität des Benutzers mit dem VR-System bieten. Damit verbunden ist der an VR gestellte An-

spruch an eine möglichst realistisch wirkende Wiedergabe der dargestellten virtuellen Welt. Zusätzlich können künstliche Objekte in der virtuellen Welt über Sensoren wie in der Realität beeinflusst werden. Zum Beispiel kann ein virtuelles Objekt in einer virtuellen Welt von einem Ort zu einem anderen transportiert werden.

Die Schöpfung des Begriffs „Virtual Reality“ wird Jaron Lanier zugeschrieben und ist auf das Jahr 1988 datiert. Er definiert Virtual Reality als etwas, das „nur als elektronisches Bild existiert, aber sonst keine konkrete Gegenständlichkeit hat. Es ist, als wäre es da, aber es ist nicht“ [Rueg99]. Mittlerweile spricht man bei Virtual Reality von einer neuen Generation von Mensch-Maschine-Schnittstellen. Sie stellt gleichzeitig aber auch ein völlig neues Medium dar.

Um ein VR-System zu entwickeln und anzuwenden, müssen verschiedene Komponenten und Techniken zusammenarbeiten. Anfänglich waren diese VR-Techniken sehr kostenintensiv. Durch den immer breiter werdenden Einsatz in der Industrie und der enormen Weiterentwicklung in der Computertechnik, kann Virtual Reality immer umfassender eingesetzt werden. Ein Anwendungsbeispiel ist die Entwicklung virtueller Prototypen. Durch ihre Simulation kann viel Zeit und Geld bei der Entwicklung verschiedenster Produkte gespart werden. Für Heimanwender außerhalb der Industrie stellt Virtual Reality eine „konsequente Weiterentwicklung der bisherigen Multimedia-Systeme dar. Typische Anwendungsbeispiele sind 3D-Informationssysteme oder auch 3D-Teleshopping“ [Grau99].

Die Interaktion des Benutzers mit der virtuellen Welt erfolgt teilweise über neue 3D-Eingabegeräte und Ausgabemedien. Diese verhindern die Wahrnehmung der gewohnten Umwelt und lassen damit nur die kontrollierten computergesteuerten Eindrücke zu. Wenn es gelingt, den persönlichen Blickpunkt des Betrachters vollständig in die virtuelle Welt zu legen, handelt es sich um ein ideales VR-System. Man spricht vom Eintauchen (Immersion) in die virtuelle Welt. Daneben existiert eine non-immersive Variante der Virtual Reality. Hierzu gehören zum Beispiel interaktive Computerspiele. Sie bieten fotorealistische Darstellungen zusammen mit flexibler Navigation. Ein- und Ausgabemedium ist hierbei weiterhin der Bildschirm. Wenn ein normaler, flacher Computerbildschirm zur Darstellung der virtuellen Szene verwendet wird, wird von „Window on World“ gesprochen. Der Benutzer schaut

dabei von einem Blickpunkt von außerhalb wie durch ein Fenster in die virtuelle Welt [Rueg99].

Die Technik der Virtual Reality bedient sich neben dem normalen Computerbildschirm verschiedener weiterer Ein- und Ausgabemedien. Ein kleiner Überblick dazu ist in Abschnitt 2.2.3 zu finden.

2.2.2 Definition von Augmented Reality

Während bei der virtuellen Realität versucht wird, mit möglichst realistisch wirkenden Modellen die Wirklichkeit künstlich zu erzeugen, wird bei der Augmented Reality (AR, erweiterte Realität) der Informationsgehalt der Realität durch künstlich erzeugte, zusätzliche Informationen erweitert. Dies wird erreicht, indem die Informationen das reale Bild überlagernd dargestellt werden. Augmented Reality ist damit die Kombination einer real betrachteten Szene und einer virtuellen Szene, das heißt, die reale Szene wird von einer durch einen Computer generierten virtuellen Szene überlagert. Sie erweitert somit die menschliche Wahrnehmung, indem sie Informationen liefert, die normalerweise nicht durch menschliche Sinne wahrnehmbar und erkennbar sind.

Die Schwierigkeit liegt dabei in der nahtlosen Verschmelzung von virtueller und realer Welt. Denn unabhängig vom beliebigen Standpunkt des Benutzers müssen die virtuellen Objekte die reale Szene exakt überlagernd eingeblendet werden. Sie müssen dort erscheinen, wo sie vom Benutzer vermutet werden, ansonsten geht ein großer Teil des realistischen Eindrucks des Gesamtbildes verloren. Idealerweise existieren die realen und virtuellen Objekte nebeneinander, damit der Benutzer nicht fähig ist, zwischen der realen Welt und der virtuellen Erweiterung zu unterscheiden.

Ein weiterer Unterschied der Augmented Reality zur Virtual Reality liegt darin, dass es dem Benutzer der AR erlaubt ist, die reale Welt um ihn herum zu sehen. Er soll die reale Umgebung mit den überlagernden Informationen wahrnehmen. Dabei ergänzt AR die Realität, ersetzt sie jedoch nicht komplett [Azum97]. Abbildung 2.1 zeigt ein einfaches Beispiel für Augmented Reality:

Abbildung 2.1: Ein triviales Beispiel für Augmented Reality: Virtuelle jonglierte Bälle in einem realen Raum [Bree95]

Bei manchen Forschern wird AR in einer Weise definiert, welche die Verwendung von Head-Mounted Displays (HMDs) voraussetzt. Um Augmented Reality jedoch nicht an spezielle Technologien zu binden, definiert Ronald T. Azuma Augmented Reality als Systeme, die folgende drei Charakteristika aufweisen [Azum97]:

- Das System verknüpft virtuelle mit realen Informationen.
- Es ist in Echtzeit interaktiv anwendbar.
- Die Objekte der realen und der virtuellen Welt müssen einander richtig zugeordnet sein.

Diese Definition erlaubt auch den Einsatz anderer Technologien, ohne dass die grundlegenden Eigenschaften der AR geschmälert werden [Azum97]. Augmented Reality schließt dabei Filme wie „Jurassic Park“ nicht ein. Hier werden zwar fotorealistische, virtuelle Objekte in 3D nahtlos mit einer realen Umgebung verbunden, jedoch stellt dieser Film kein interaktives Medium dar. Nach Azumas Definition handelt es sich somit nicht um Augmented Reality.

Virtuelle Modelle müssen, wenn sie für den Einsatz in AR bestimmt sind, präziser modelliert werden als virtuelle Objekte, die in VR-Systemen Anwendung finden. Der Grund dafür liegt bei der direkten Vergleichsmöglichkeit der virtuellen mit der realen Welt. Während der Benutzer eines VR-Systems vollkommen von der realen Welt abgeschottet ist, kann der Benutzer eines AR-Systems die virtuellen Informationen direkt mit der realen Welt vergleichen [Klin98]. Der Eindruck der nahtlosen Integration der virtuellen Objekte in die reale Welt kann dadurch leicht zerstört werden. Gleiches geschieht, wenn sich die zwei Welten nicht exakt überlagern. Die menschliche Wahrnehmung ist in dieser Hinsicht sehr sensibel und erkennt kleine Fehler sehr schnell. Schon bei kleinsten Positionsunterschieden passen das Bild der realen Welt und die virtuellen Informationen nicht mehr zusammen.

Bis heute gibt es kein AR-System, das all diese Anforderungen erfüllen kann. Die größten Probleme bereitet dabei die exakte Positionierung und Überlagerung der zwei verschiedenen Welten. Im Gegensatz dazu kann der Anspruch an die fotorealistische Darstellung von virtuellen Objekten mit fortschreitender Entwicklung der Computertechnologien immer besser erfüllt werden. Nach Shahzad Malik wird eher ein evolutionäres als ein revolutionäres Herangehen an Augmented Reality benötigt, um AR nicht mehr nur in den Forschungslaboren sondern überall einsetzen zu können [Mali02].

Die Anwendungsmöglichkeiten von Augmented Reality sind breit gefächert. AR erweitert die menschliche Wahrnehmung und Interaktion mit der realen Welt. Durch die virtuellen Objekte werden Informationen dargestellt, die der Benutzer sonst mit seinen Sinnen nicht erfassen könnte. Augmented Reality kann den Anwender damit bei vielen Aufgaben in der realen Welt unterstützen. Durch den Einsatz von AR wird der Computer als Werkzeug dafür benutzt, eine Aufgabe für einen Menschen einfacher zu machen.

Es gibt verschiedene Einsatzgebiete für AR-Anwendungen. Dazu gehören Visualisierungen in der Medizin, Wartung und Reparatur, visuelle Zusatzinformation und Informationssysteme, Planung von Roboterbewegungen und Simulationsüberprüfung, Unterhaltung und Computerspiele sowie militärische Navigation und Zielpeilung

[Adam03]. Die folgenden Abbildungen zeigen zwei Beispiele für den Einsatz von AR in der Medizin und Industrie [Azum97].

Abbildung 2.2: Virtueller Fötus im Bauch einer Schwangeren [Azum97]

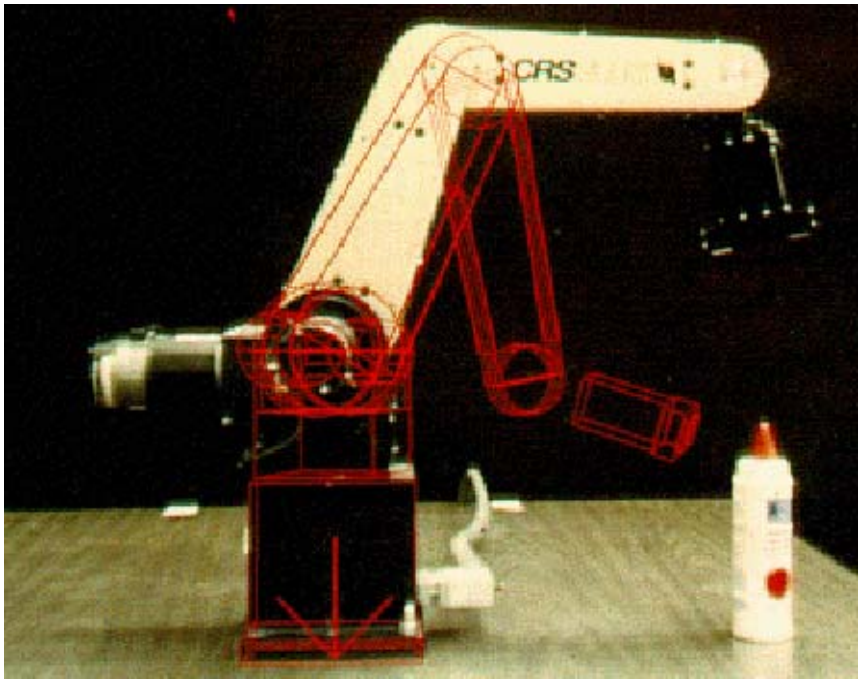


Abbildung 2.3: Virtuelle Linien zeigen eine geplante Bewegung des Roboterarms [Azum97]

2.2.3 Head-Mounted Displays

Die größte Herausforderung bei der Entwicklung eines AR-Systems ist, die Verbindung der realen mit der virtuellen Welt zu bewältigen. Dabei gibt es die althergkömmliche Art und Weise der Darstellung über einen Bildschirm, oder aber die Head-Mounted Displays (HMD). Die Wirkung von AR wird dabei durch HMDs besser unterstützt als von herkömmlichen Ausgabegeräten. Sie können dem Benutzer eher das Gefühl vermitteln, sich innerhalb einer erweiterten Welt zu befinden, anstatt sie wie auf einem Monitor von außen zu betrachten. Damit dem Benutzer auch ein räumlicher Eindruck vermittelt wird, erzeugen HMDs stereoskopische Bilder. Das heißt, den Augen werden zwei perspektivisch leicht verschobene Ansichten eines Bildes präsentiert. Die Verschiebung der beiden Ansichten entspricht dabei dem Abstand der Augen voneinander (Abbildung 2.4).

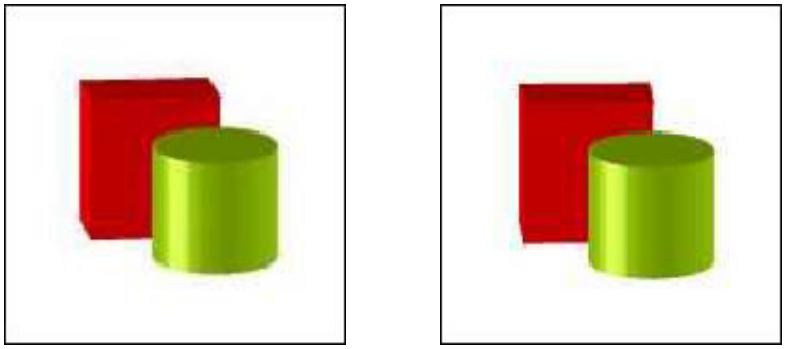


Abbildung 2.4: Zwei perspektivisch verschobene Ansichten eines Bildes [Ulbr02]

HMDs bestehen aus zwei kleinen Bildschirmen, wovon je einer vor dem Auge positioniert wird. Als Bildschirme können sowohl herkömmliche Kathodenstrahlröhren (Cathod Ray Tube - CRT) als auch Flüssigkristall-Bildschirme (Liquid Crystal Displays – LCD) eingesetzt werden. Von Nachteil bei Kathodenstrahlröhren sind ihre Größe und das damit verbundene große Gewicht. Im Allgemeinen ist die Bildqualität aber besser als die von Flüssigkristall-Bildschirmen. Ein weiterer Nachteil beim Einsatz dieser HMDs ist, dass sie blickdicht sind und dem Benutzer keine direkte Sicht auf die reale Umgebung erlauben. Für die Verwendung in der AR sind sie daher nicht optimal geeignet.

Durch die Position der Bildschirme, die sich direkt vor jedoch nicht seitlich der Augen befinden, füllen HMDs das Blickfeld des Benutzers nicht vollständig aus und können es dadurch stark einschränken. Das Blickfeld des menschlichen Auges wird in zwei Winkeln angegeben. Der horizontale Sichtwinkel beträgt beim menschlichen Auge etwa 120° bis etwa 180°, wobei in den Randbereichen nur noch Bewegungen wahrgenommen werden. Der vertikale Sichtwinkel beträgt höchstens 150°. Ein HMD dagegen verwendet einen horizontalen Sichtwinkel von 45° bis 110° und einen vertikalen Sichtwinkel von 25° bis 65°. Je kleiner diese Winkel sind, desto eher besteht für den Benutzer der Eindruck, durch ein Fernglas oder eine Röhre zu blicken [Ulbr02].

Für AR geeignet sind diese blickdichten HMDs nur, wenn sie mit ein oder zwei Videokameras kombiniert werden, da sonst kein Eindruck der erweiterten realen Umgebung dargestellt werden kann. Die Videokameras liefern die Sicht des Benutzers auf die reale Umgebung. Um das Bild der realen Welt mit virtuellen Informationen zu überlagern, werden die Bilder der Videokameras mit den generierten virtuellen Bildern überlagert. Dieses Gesamtbild wird wiederum auf den Bildschirmen des HMDs dargestellt. Die Abbildung 2.5 zeigt den schematischen Aufbau eines solchen videobasierten HMDs:

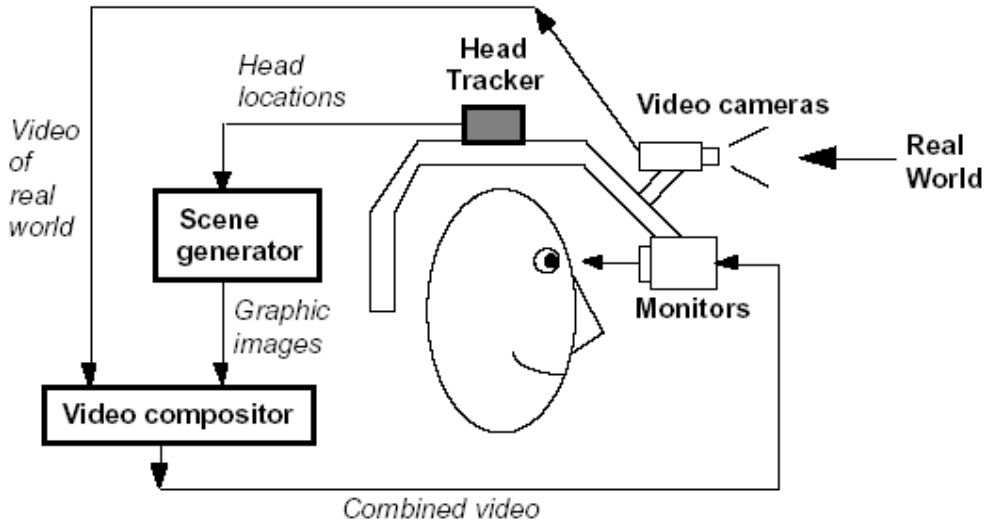


Abbildung 2.5: Schematischer Aufbau eines videobasierten HMDs [Azum97]

Neben dem videobasierten HMD gibt es auch optische HMD-Techniken. Der Hauptunterschied liegt hierbei darin, dass optische HMDs blickdurchlässig sind und somit dem Benutzer die Sicht auf die reale Umgebung erlauben. Bei optischen HMDs befindet sich vor den Augen des Benutzers eine Abbildungsfläche, auf welcher virtuelle Informationen abgebildet werden können. Diese Fläche ist in ihrer Sichtdurchlässigkeit variabel und gibt dem Benutzer damit die Möglichkeit, durch sie hindurch die reale Welt zu sehen. Er sieht damit durch das HMD hindurch die reale Welt, auf der direkt überlagernd die virtuellen Informationen dargestellt werden können. Blickdurchlässige HMDs eignen sich aus diesem Grund auch besser für den Einsatz in AR-Systemen, da die reale Umgebung nicht mit eingeblendet werden muss. Abbildung 2.6 zeigt den schematischen Aufbau eines solchen optischen HMDs.

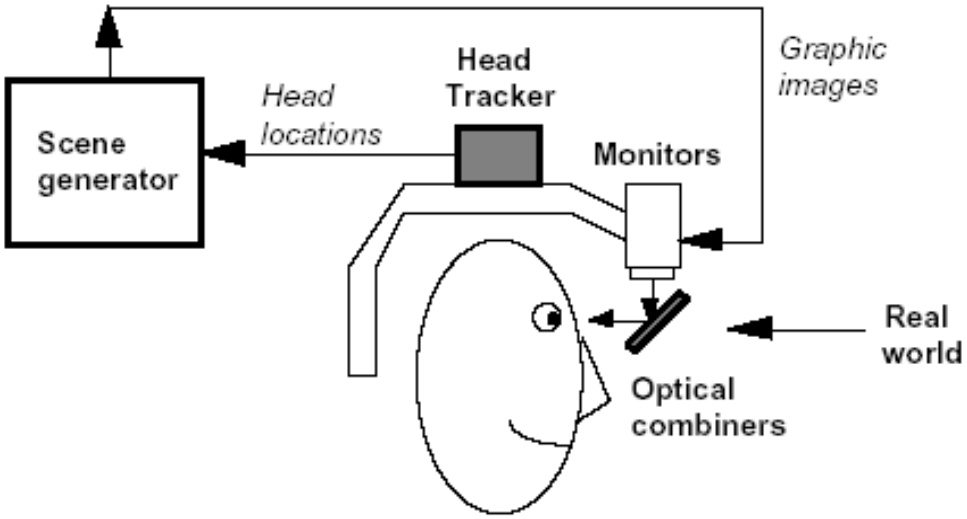


Abbildung 2.6: Schematischer Aufbau eines optischen, -halbdurchsichtigen HMDs [Azum97]

2.2.4 Tracking-Systeme

Damit das Bild der realen Umgebung und die überlagerten Informationen präzise zusammenpassen und sich ergänzen, sind neben dem HMD weitere Techniken notwendig. Die wichtigsten Techniken dienen dabei der Erfassung von Position und Blickrichtung des Benutzers. Ohne die Kenntnis dieser Parameter ist es nicht möglich, reale und virtuelle Welten miteinander zu verbinden.

Die ersten AR-Systeme unterstützten deswegen keine größeren Benutzerbewegungen. Stattdessen navigiert der Benutzer durch die virtuelle Umgebung, indem er beispielsweise in einer Tretmühle oder einem anderen Gefährt läuft. Dabei werden die Bewegungsdaten, die über die Mechanik des Gefährts gemessen werden, verwendet, um in der AR-Szene zu navigieren. Der Benutzer selbst bleibt jedoch an einem Ort in der realen Welt.

Es werden jedoch einige AR-Anwendungen entwickelt, die den Benutzer unterstützen, während er sich frei in seiner Umwelt bewegt. Die komplette AR-Ausrüstung muss dabei transportabel sein und – viel wichtiger – der Benutzer muss geortet werden können. Für diese mobilen Systeme werden so genannte Trackingsysteme benötigt, die die Position und Bewegung des Benutzers im dreidimensionalen Raum der realen Welt sensorisch erfassen. Für die Messung der aktuellen Position eines Benutzers kann zum Beispiel ein GPS-Gerät (Global Positioning System) verwendet werden. Es wird vor allem in Outdoor-AR-Anwendungen eingesetzt. Über ein Ortungsverfahren mit Hilfe von Satellitensignalen, wird die Position des Benutzers in Echtzeit bestimmt, ohne dass die Umgebung mit Markierungen verändert werden muss. Bei anderen Umgebungssensoren ist dies zur Ortung des Benutzers notwendig und damit nicht immer von Vorteil. Wegen der notwendigen Sichtbarkeit der Satelliten ist das GPS jedoch nicht für das Tracking in Gebäuden geeignet [Frei03].

Um die Kopfbewegungen und dadurch die Blickrichtung des Benutzers bestimmen zu können, werden verschiedene Arten von Tracking-Systemen verwendet. Sie lassen sich in absolute und relative Tracking-Systeme unterteilen. Zu den absoluten Systemen gehören mechanische, akustische, optische und magnetische Tracking-Systeme. Accelerameter beziehungsweise Beschleunigungssensoren gehören zu den relativen Tracking-Systemen. Hier wird nur kurz auf die optischen, magnetischen und relativen Systeme eingegangen, da die mechanischen und akustischen Systeme die Bewegungsfreiheit des Benutzers stark einschränken oder die Messwerte für eine korrekte Überlagerung zu ungenau sind. Für mobile AR-Systeme sind diese Systeme daher nicht geeignet [Frei03].

- Optische Tracking-Systeme:
  Hier werden mehrere Kameras um den Anwender herum positioniert und

messen die Positionsveränderungen des Benutzers anhand von verschiedenen Messpunkten, die auf seinem Körper angebracht sind. Diese Systeme bieten eine hohe Genauigkeit und gute Abtastraten, jedoch wird eine direkte Sichtlinie von möglichst jeder Kamera zu den Markierungen benötigt. Außerdem sind diese Systeme sehr kostenaufwendig und anfällig gegenüber Reflexionen.

- Magnetische Tracking-Systeme:
  Magnetische Tracking-Systeme basieren auf drei rechtwinklig angeordneten Feldern im Sender. Ein Empfänger kann dann die Änderungen des magnetischen Feldes messen und berechnet daraus die genaue Position und Ausrichtung des Senders beziehungsweise des Benutzers. Diese Systeme arbeiten schnell, genau und bieten hohe Abtastraten. Ein Nachteil ist jedoch ihre Anfälligkeit gegenüber ferromagnetischen Feldern. Da die Genauigkeit im Quadrat zur Entfernung von Sender zu Empfänger abnimmt, sind magnetische Tracking-Systeme nur bei einer Entfernung des Senders zum Empfänger unter 1,5 Meter sinnvoll einzusetzen.

- Accelerameter, Beschleunigungssensoren:
  Diese Tracker messen Translations- und Rotationskräfte, die auf eine beschleunigte Masse wirken. Daraus wird die Orientierung des Benutzers berechnet. Von Vorteil ist, dass sie keinen Referenzsender benötigen und unabhängig von der Raumgröße eingesetzt werden können. Einziger Nachteil ist die notwendige Kalibrierung des Trackers vor jeder Messung. Sie ist notwendig, um zu verhindern, dass sich Messfehler addieren.

Das gebräuchlichste Tracking-System besteht aus einem elektronischen Kompass und zwei Sensoren, die Neigungen messen. Diese Tracker sind klein und kostengünstig. Die besten unter ihnen arbeiten mit einer Messgenauigkeit von 0,5 Grad. Sie sind besonders für den Outdoor-AR-Bereich geeignet, da keine Anpassung der Umgebung beispielsweise durch Sensoren notwendig ist [Azum99].

Zusammenfassend lässt sich sagen, dass die Tracking-Technologien noch nicht ausgereift sind und weiterhin ein Forschungsgebiet für sich darstellen. Dadurch bleiben die Überlagerung der Welten und Sensorenfehler die zwei größten Probleme in AR-Systemen. Folglich macht die Integration des Benutzers in die virtuelle Welt und

umgekehrt weiterhin den größten Teil der Entwicklung eines AR-Systems aus. Ohne ein fehlerfreies Tracking-System ist es nicht möglich die generellen Bedingungen von AR zu erfüllen. Virtuelle Objekte können ohne eine exakte Positionierung des Benutzers nicht so dargestellt werden, dass sie die realen Objekte überlagern [Azum97].

In den vorhergehenden Abschnitten wurde ein kleiner Überblick über die in Augmented Reality Systemen verwendete Hardware gegeben. Eine ebenso wichtige Rolle bei der Entwicklung solcher Systeme spielt daneben die programmtechnische Programmierung und Visualisierung der virtuellen Informationen, welche die reale Umgebung überlagern sollen. Welche Programmiertechniken in dieser Arbeit verwendet werden, wird im nächsten Abschnitt beschrieben.

2.3 Programmiertechnische Grundlagen

2.3.1 Einleitung

Dieser Abschnitt soll einen grundlegenden Einblick in die in dieser Arbeit verwendeten Programmiertechniken geben. Hierfür ist für die einzelnen Techniken jeweils ein Abschnitt vorgesehen. Vorgestellt werden Java, Swing und Java 3D. Java bildet die Grundlage für die Entwicklung des Demonstrators und die Implementierung der Objektverdeckung in dieser Arbeit. Bei der Entwicklung der Benutzeroberfläche des Demonstrators wird Swing verwendet. Es ist leicht zu handhaben und bietet umfangreiche Funktionalitäten für den Aufbau einer graphischen Oberfläche. Java 3D wird für die Darstellung der dreidimensionalen, virtuellen Welt eingesetzt.

2.3.2 Java

Java ist eine moderne, objektorientierte Programmiersprache, die vollständig neu entworfen wurde. Dabei wurde die Syntax der Sprachen C, C++ und Smalltalk so weit wie möglich nachgeahmt, es wurde aber auf viele der fehlerträchtigen Merkmale dieser Sprachen verzichtet. Entwickelt wurde Java von SUN Microsystems. Die

erste Version von Java wurde im Januar 1999 durch die Version 1.2 ersetzt und in Java 2 Platform umbenannt.

Einer der Vorteile von Java ist seine Plattformunabhängigkeit. Sie wird erreicht, indem der Programmcode in Bytecode übersetzt wird, nicht wie bei anderen Programmiersprachen direkt in Binärcode. Im Gegensatz zum Binärcode ist dieser Bytecode unabhängig von Betriebssystem und Prozessor. Er wird während der Laufzeit durch die Java Virtual Machine, auch Java Runtime Environment genannt, interpretiert. Bei der Java Virtual Machine handelt es sich um eine Laufzeitumgebung, die sowohl die Prozessor- als auch die Betriebssystemschicht abstrahiert. Inzwischen liegen für praktisch alle Plattformen diese Ablaufumgebungen vor, so dass Anwendungen, die in Java entwickelt werden, unverändert auf jeder Plattform einsetzbar sind. Daneben ist Java auch eine robuste Sprache. Der Compiler überprüft den Quellcode bereits während des Kompiliervorgangs auf Typkorrektheit. Außerdem läuft ein Java-Programm in einem Interpreter ab, der Inkonsistenzen aufdeckt, die während der Laufzeit auftreten.

Im Gegensatz zu C++ unterstützt Java keine Mehrfachvererbung. Es ist nur eine Einfachvererbung möglich. Durch den Einsatz von Interfaces wird dies jedoch zu einem kleineren Nachteil. Zusätzlich wird in Java das Multithreading unterstützt. Das heißt, es können mehrere parallel ablaufende Aufgaben von einem Java-Programm bearbeitet werden. Damit des hierbei nicht zu gleichzeitigen Zugriffen auf Systemressourcen kommt, bietet Java Synchronisations-Mechanismen an. Weitere ausführlichere Informationen über Java finden sich auf der Homepage von SUN.[1]

2.3.3 Swing

Die Java-Laufzeitbibliothek bietet zusätzlich Klassen, mit deren Hilfe so genannte Graphical User Interfaces (GUI) erstellt werden können. Diese Klassen gehören zu den Java Foundation Classes (kurz JFC). Hierzu gehört auch Swing. Die Swing Komponenten erweitern die Funktionalität und den Umfang des AWT (Abstract Windo-

[1] http://java.sun.com

wing Toolkit) und stellen damit elementare Grafik- und Fensterfunktionen zur Verfügung. Swing ist vollständig in Java geschrieben und dadurch ebenfalls plattformunabhängig. Deshalb werden die Komponenten in Swing als leichtgewichtig bezeichnet, im Gegensatz zu den schwergewichtigen Komponenten des AWT. Diese werden vom umgebenden Betriebssystem bereitgestellt und greifen über Peer-Klassen auf Elemente des Betriebssystems zurück. Swing stellt für fast jede AWT-Komponente eine analoge Swing-Komponente dar, allerdings nicht zwangsläufig mit erweiterter Funktionalität. Die Swing-Komponenten bilden dabei eine Ergänzung zum AWT und bieten leistungsfähige Komponenten für graphische Benutzeroberflächen.

Durch die Beschränkung auf einen sehr kleinen und überschaubaren Umfang an Peer-basierten Funktionen, werden Abhängigkeiten von der zugrunde liegenden Plattform sehr stark minimiert. Daraus resultiert vor allem eine ernorme Robustheit im Hinblick auf plattformspezifische Fehler. Die höhere Flexibilität hat jedoch auch Nachteile und führt zu Geschwindigkeitsverlusten bei der Programmausführung. Sie resultieren hauptsächlich aus der eigenständigen Modellierung der Verhaltensweisen der Elemente. Durch den Einsatz leistungsfähiger Hardware, kann dieser Nachteil jedoch größtenteils wieder ausgeglichen werden [SUN-2].

2.3.4 Java 3D

Java 3D ist eine ebenfalls von SUN Microsystems entwickelte Klassenbibliothek. Sie ist jedoch nicht Teil des Standard Java API (Application Programming Interface). Das Java 3D API stellt Funktionalitäten zur Verfügung, um 3D-Geometrien zu erzeugen und deren Darstellung und Animation auf dem Bildschirm oder einem anderen unterstützten Ausgabegerät zu steuern. Darüber hinaus stellt es standardisierte Schnittstellen zur Interaktion bereit.

Spezifikation und Implementierung von Java 3D sind auf die Echtzeitdarstellung von großen 3D Szenen ausgerichtet. Da Java 3D in den derzeitigen Implementierungen über die Standardschnittstellen OpenGL oder DirectX auf die Grafikhardware zugreift, ist die Darstellungsqualität direkt von der Leistungsfähigkeit dieser Hardware abhängig. Java 3D wird zudem eine Schwäche bei seiner Performancefähigkeit

nachgesagt. Da sich die Hardware derzeit jedoch sehr schnell weiterentwickelt, ist zu erwarten, dass sich die Grafikleistung, die durch Java 3D zur Verfügung gestellt wird, in Zukunft weiter verbessern wird. Abhängig von den Grafikschnittstellen, stehen dem Entwickler momentan folgende Versionen der Java 3D Klassenbibliotheken zur Verfügung:

- Java 3D 1.3 OpenGL
- Java 3D 1.3 DirectX

Während die OpenGL-Version sowohl für Microsoft Windows als auch für die Sun Solaris Plattform verfügbar ist, wird die DirectX-Version nur von Windows unterstützt. Der Demonstrator wurde mit der Java 3D API Version 1.3 DirectX entwickelt. Für die korrekte Funktion empfiehlt sich daher die Verwendung der DirectXVersion.

Der große Vorteil von Java 3D ist, dass es auf Java basiert. Damit ist es möglich, Software im 3D-Bereich vollständig in Java zu entwickeln. Es ist auch sehr zweckmäßig, wenn der gesamte Quellcode einer Grafikanwendung und einer Benutzeroberfläche in einer einfach zu portierenden Sprache wie Java zur Verfügung steht. Ein weiterer Vorteil für den Entwickler ist, dass der Quellcode lesbarer, leichter zu warten, einfacher wieder verwendbar und leichter zu schreiben ist [SUN-3].

Szenengraph

Java 3D verwendet ein höherrangiges Modell zur Szenenbeschreibung. Es stellt dem Entwickler bei der Modellierung seiner 3D-Welt den Szenengraphen zur Verfügung. Der Szenengraph stellt eine Architektur dar, die es dem Entwickler erlaubt, Szenen leicht zu beschreiben, zu transformieren und wieder zu verwenden. Er bildet das zentrale Hierarchiemodell in Java 3D. Alle Elemente, die an der Darstellung der Java 3D-Szene beteiligt sind, werden in dieser Struktur, die die Form eines azyklisch gerichteten Graphen[2] hat, angeordnet und verwaltet. Der Szenengraph ist die hierarchische Repräsentation von Objekten, Verhaltensknoten, Licht- und Tonquellen der zu schaffenden Szene. Ein Graph besteht strukturell aus Kanten und Knoten,

[2] Ein azyklischer Graph erlaubt keine Zyklen und ist ein von Grund auf gerichteter Graph.

wobei ein Knoten ein Datenelement und eine Kante die Beziehung zwischen zwei Datenelementen beschreibt. Direkte Verbindungen zwischen zwei Knoten stellen dadurch immer eine Vater-Kind-Beziehung dar.

Der konkrete Java 3D Szenengraph besteht aus zwei Teilbäumen. Der eine Teilbaum ist der so genannte `ContentBranchGraph`, der zweite Teilbaum ist der `ViewBranchGraph`. Der `ContentBranchGraph` enthält alle Objekte, die in einer Szene vorkommen, das heißt, er enthält alle geometrischen Strukturen und ihr Verhalten. Der `ViewBranchGraph` hingegen beschreibt die Sicht des Benutzers auf den Inhalt der Szene. Er enthält die `ViewPlatform`, welche die in späteren Kapiteln erwähnte virtuelle Kamera repräsentiert und die 3D-Szene abbildet. Das `VirtualUniverse` dient als Wurzelknoten des Szenengraphen. An den Wurzelknoten wird ein `Locale`-Objekt gehängt. Dies ist ein Ort, der als Bezugssystem für die räumliche Platzierung von virtuellen Objekten im `VirtualUniverse` dient. Er repräsentiert ein rechtsdrehendes, kartesisches Koordinatensystem, wie es in Abbildung 2.7 dargestellt ist.

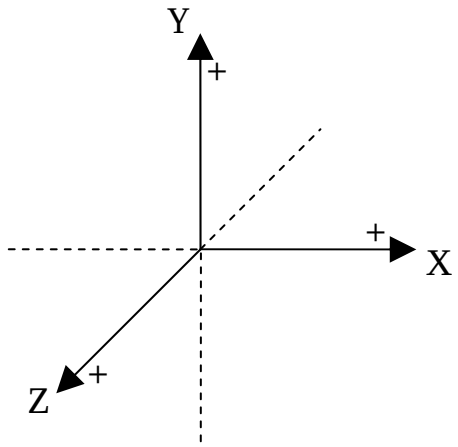


Abbildung 2.7: rechtsdrehendes, kartesisches Koordinatensystem

Einzelne Knoten werden im Java 3D Szenengraphen durch gruppierende Objekte wie der `TransformGroup` oder `BranchGroup` dargestellt. Eine `TransformGroup` enthält die Transformationen für seine Kindknoten. Wird zum Beispiel eine Translation definiert, so gilt diese für den gesamten abhängigen Subgraphen. Eine `BranchGroup` wiederum wird verwendet, um andere Knoten zu gruppieren. Am

Ende der Szenengraphstruktur befinden sich die `Leafs`. Sie stellen in Form von `Shape3D`-Objekten die eigentlichen Geometrien dar. Der Pfad, der vom `Locale`-Objekt bis zum `Leaf` entsteht, ist dabei immer eindeutig, da innerhalb eines Knotens stets mehrere Kinder, nie aber mehrere Elternknoten möglich sind. Abbildung 2.8 stellt den Aufbau des Java 3D Szenengraphen dar, Abbildung 2.9 enthält die dazugehörige Notation und Legende der einzelnen Symbole.

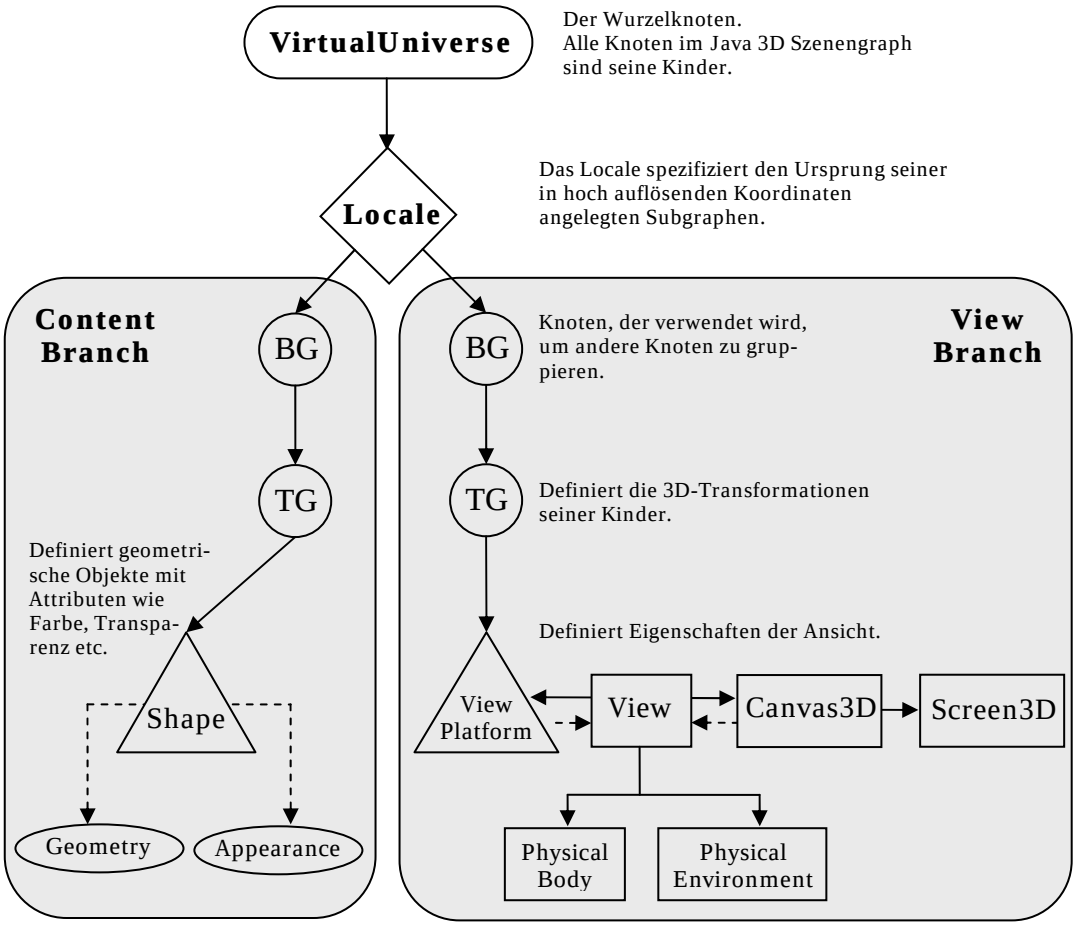


Abbildung 2.8: Aufbau des Java 3D Szenengraph[3]

[3] Hinweis zur Notation: Die Symbole und Bezeichnungen für den Szenengraphen sind die gleichen wie die von SUN. Siehe: http://java.sun.com/products/java-media/3D/collateral/

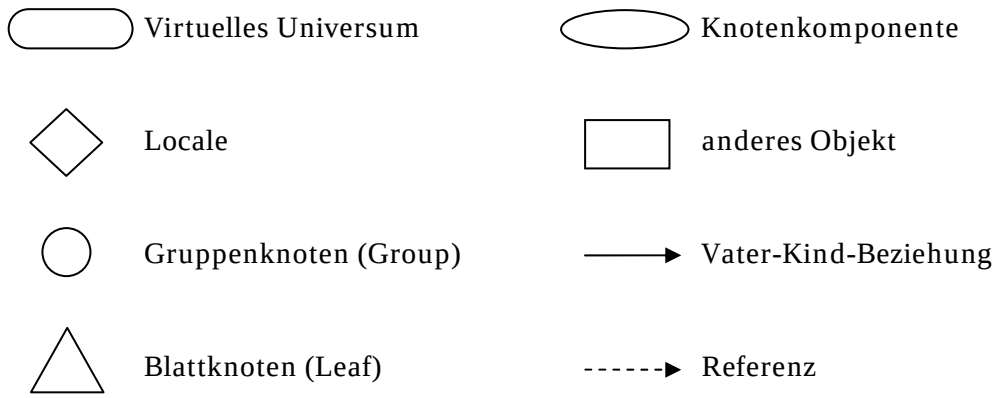


Abbildung 2.9: Notation Szenengraph

Diese Szenengraphstruktur erlaubt es, große komplexe Szenen aufzubauen und zu verwalten. Dabei können Abstraktionsebenen gebildet werden, die mehrere Objekte zusammenfassen und dadurch manipulierbar machen. Java 3D stellt eine komplett objektorientierte Abstraktion zur Verfügung, die den Umgang mit den komplexen Datenstrukturen erleichtert. Es existieren Abstraktionsklassen für einfache Körper wie Würfel oder Kugeln aber auch für komplexe Geometrien und Materialien. Dreidimensionale Transformationen werden in speziellen Klassen abgebildet und können dadurch einfach und anschaulich über Zugriffsmethoden manipuliert werden.

Statische Datenformate

Java 3D unterstützt durch ein einfaches Import-Konzept den Import von Modellen oder ganzen Szenen aus anderen Formaten. Ein häufig verwendetes Format ist dabei das VRML-Format (Virtual Reality Modeling Language). Für dieses und andere Formate ist mittlerweile eine Vielzahl von Import-Werkzeugen im Internet verfügbar. Dadurch muss das Parsen nicht mehr vom Entwickler übernommen werden. Eine Implementierung des entsprechenden Import-Werkzeugs ist ausreichend.

2.4 Zusammenfassung

Nach kurzen Einblicken in die Grundbegriffe und Techniken der Virtual Reality und der Augmented Reality und der Beschreibung der für diese Anwendungen erforder-

lichen Hardware und der Vorstellung der programmtechnischen Grundlagen, wird im folgenden Kapitel näher auf die Problematik der Objektverdeckung in Augmented Reality Systemen eingegangen.

3 Stand der Forschung

3.1 Einleitung

Dieser Abschnitt befasst sich mit der Problemstellung der Objektverdeckung in Augmented Reality Systemen. Dazu werden verschiedene in der Forschung aktuelle Lösungsansätze vorgestellt und abschließend bewertet. Auf der Entscheidung für einen dieser Lösungsansätze basiert die weitere Durchführung dieser Diplomarbeit.

3.2 Problematik der Objektverdeckung

In der realen Welt weisen Objekte und Gegenstände physikalische Eigenschaften auf. Sie haben ein Volumen, können sich nicht durchdringen oder werfen Schatten. Ebenso können sie sich gegenseitig verdecken, wenn sie sich vom Blickpunkt des Betrachters aus räumlich hintereinander befinden. Um die reale Welt durch eine virtuelle Welt abzubilden, sollten sich die Objekte in gleicher Weise verhalten. Sie können relativ realistisch modelliert werden und entsprechend ihrer realen Vorbilder mit den oben genannten physikalischen Eigenschaften versehen werden. Die Verdeckung von virtuellen Objekten untereinander wird durch den z-Buffer-Algorithmus (Tiefenspeicheralgorithmus) realisiert, der fester Bestandteil einer jeden Grafikkarte ist. Er basiert auf der Ermittlung der Tiefeninformation einer virtuellen Szene. Jedem Pixel wird seine Entfernung zur Kamera entlang der Sichtlinie zugeordnet. Daraus wird berechnet, welche Bildpunkte sich näher am Betrachter befinden beziehungsweise welche Bildpunkte tatsächlich sichtbar sind. Dabei entspricht jeder Tiefenwert einem Grauwert einer vorhandenen Grautonpalette. Aus diesen einzelnen grauen Pixeln ergibt sich eine Tiefenkarte eines virtuellen Objekts. Auf der rechten Seite in Abbildung 3.1 wird diese für einen Würfel auf einer ebenen Fläche dargestellt. Je dunkler die Pixel dabei dargestellt werden, desto näher befindet sich der entsprechende Objektpunkt an der Kamera [Ludw02].

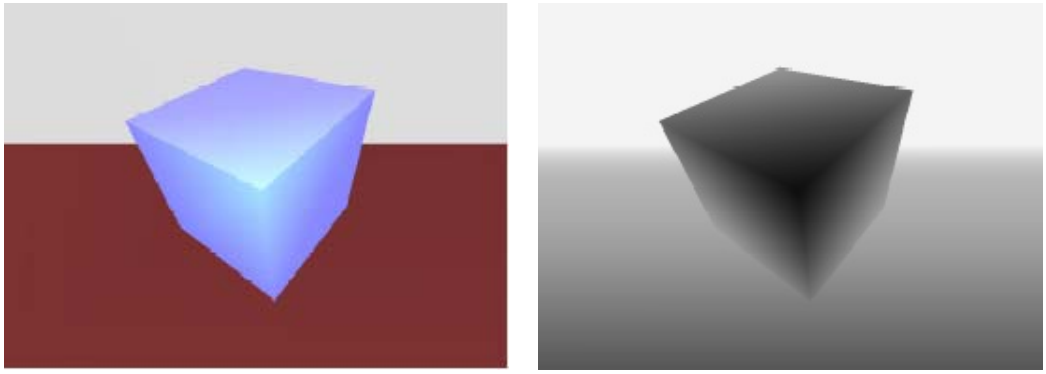


Abbildung 3.1: Das Bild eines virtuellen Objekts (links) und die zugehörige Tiefenkarte [Bree95]

In einem Augmented Reality System werden jedoch beide Arten von Objekten eingesetzt, sowohl reale als auch virtuelle. Damit besteht die zusätzliche Anforderung, dass sich diese Objekte untereinander ebenfalls entsprechend der physikalischen Gesetze der Realität verhalten [Klin98]. Das heißt, virtuelle Objekte können reale Objekte verdecken oder durch sie verdeckt werden. Für eine Augmented Reality Anwendung stellt es dabei kein Problem dar, wenn virtuelle Objekte reale Objekte verdecken. Beim Rendering des virtuellen Anteils eines AR-Bildes werden die virtuellen Modelle auf schwarzem Hintergrund abgebildet. Mit diesem Bild wird die reale Szene überlagert. Dabei sind von der realen Szene nur die Teile zu sehen, die sich hinter den schwarzen Bereichen des Computergenerierten Bildes befinden. Durch diese für Augmented Reality übliche Technik verdecken die virtuellen Objekte immer die realen Objekte. Der Fall, dass reale Objekte virtuelle Objekte verdecken, tritt ungeachtet ihrer realen, räumlichen Verhältnisse nicht auf [Mali02].

Abbildung 3.2 zeigt diese Problematik der Augmented Reality. In beiden Bildern wird eine reale Szene abgebildet. Im rechten Bild soll zusätzlich die virtuelle Information in Form eines Fotos die viereckige reale Fläche überlagern. Damit der richtige Eindruck dieser AR-Szene entsteht, müsste die Hand im rechten Bild jedoch über dem eingeblendeten Foto erscheinen:

Abbildung 3.2: Beispiel für visuelle Verdeckung [Mali02]

Eine falsche Darstellung der räumlichen Verhältnisse kann auch beim Benutzer zu unangenehmen Auswirkungen führen, da das menschliche Gehirn mit den falsch dargestellten visuellen Informationen nicht richtig umgehen kann. Die inkorrekte Darstellung führt nicht nur zu falschen Interpretationen der räumlichen Verhältnisse der Umgebung, sie kann auch zur Überanstrengung der Augen oder Übelkeit, ähnlich der Seekrankheit führen [Fuhr99].

Um in einem AR-System Verdeckungen berechnen und darstellen zu können, müssen die Eigenschaften der realen Welt auf logische, berechenbare Strukturen abgebildet werden. Diese Problematik findet in der Forschung verschiedene Lösungsansätze. Ein Ansatz basiert auf der Verwendung von 3D-Rekonstruktionen, ein weiterer kombiniert diesen mit dem Einsatz von Tiefenkarten. Weitere Ansätze arbeiten gänzlich ohne 3D-Rekonstruktionen und verwenden nur die Tiefeninformationen oder Methoden, die auf Umrisslinien von realen Objekten basieren, um Verdeckungen zu ermitteln. In den folgenden Abschnitten werden diese vier Lösungsansätze kurz vorgestellt. Diese Ansätze beziehen sich immer auf visuelle Verdeckungen, die durch näher am Betrachter und räumlich vor virtuellen Objekten liegende reale Objekte hervorgerufen werden.

3.3 Modellbasierter Ansatz

Um Verdeckungen in einem AR-System zu visualisieren wird nach Klinker und Breen ein sehr präzises Modell der realen Umgebung für die Anwendung in Aug-

mented Reality benötigt. In diesem Abschnitt wird dieser so genannte modellbasierte Lösungsansatz beschrieben [Klin00], [Bree95]. Für diese Methode wird vor allem ein umfangreiches und genaues Modell der realen Welt benötigt. Dabei müssen die einzelnen Modelle nicht so komplex sein wie jene, die in Virtual Reality Anwendungen verwendet werden. In der virtuellen Realität ist die Beschreibung der Oberflächenstruktur der Objekte entscheidend wichtig. In AR-Anwendungen benötigen die Modelle, wenn sie für Verdeckungsverfahren eingesetzt werden, keine genauen Informationen zu Aussehen und Darstellung ihrer Oberfläche. Es ist vielmehr eine detaillierte Wiedergabe der Form ihrer Oberfläche, ihrer Geometrie erforderlich.

Dennoch müssen Modelle, die für Augmented Reality verwendet werden, weitaus präziser in Bezug auf ihre Lage im Raum modelliert sein, als die Modelle für Virtual Reality Anwendungen. Im Gegensatz zu VR haben Benutzer von AR-Systemen den direkten Vergleich der realen und virtuellen Welt vor Augen. Leichte Missverhältnisse und Widersprüche in der Verbindung dieser zwei Welten kann der Benutzer sehr schnell feststellen. Damit wäre eine nahtlose Integration der virtuellen Objekte in die reale Welt nicht mehr gegeben und Objektverdeckungen könnten nicht korrekt dargestellt werden.

Es gibt unterschiedliche Wege, die Modelle der realen Welt zu erstellen. Dabei hängt ihre Erstellung von den Anforderungen und Datenquellen der unterschiedlichen AR-Anwendungen ab. Oft werden die 3D-Informationen einer realen Szene manuell erstellt. Auch können CAD-Modelle verwendet werden. Oder es stehen Karten und Messungen von Landschaften und Stadtbildern aus der Geodäsie zur Verfügung, die ebenfalls als Realitätsmodell verwendet werden können. Obwohl so viele Möglichkeiten zur Modellerstellung vorhanden sind, ist es schwierig, genaue Modelle zu erstellen. Das liegt vor allem daran, dass sich die Realität verändern kann. Bauwerke haben nicht immer das gleiche Erscheinungsbild oder von manchen realen Objekten sind Position und Erscheinungsform im Voraus nicht bekannt. Für den Einsatz in der Verdeckungsbehandlung ist es jedoch immens wichtig, aktuelle Informationen zum Aufbau des Modells der realen Welt zur Verfügung zu haben.

Eine weitere Anforderung an die Modellgeometrien der realen Welt ist ihre korrekte Anordnung in Relation zu ihren entsprechenden Vorbildern aus der realen Welt.

Das bedeutet, die Modelle müssen exakt positioniert werden, damit das Bild der virtuellen Szene exakt das Bild der realen Szene überlagert. Die Modellgeometrie erscheint dann, wenn sie als computergeneriertes Bild dargestellt wird, in der gleichen Lage wie die entsprechenden realen Objekte.

Wenn die Modelle der realen Welt in der virtuellen Welt exakt positioniert sind, können sie Verdeckungen erzeugen indem sie der Hintergrundfarbe der virtuellen Welt entsprechend dargestellt werden. In den meisten AR-Systemen entspricht diese der Farbe schwarz. Wenn sich ein virtuelles Objekt hinter ein Modell eines realen Objekts bewegt, wird die Verdeckung berechnet und das vorne liegende reale Objekt in der Farbe des Hintergrunds dargestellt. Durch die in der Hintergrundfarbe dargestellten Bereiche im Bild ist dann die reale Szene sichtbar. Für den Betrachter erzeugt dies die Illusion, als ob das reale Objekt wirklich das virtuelle Objekt verdeckt.

In Abbildung 3.3 wird die modellbasierte Technik durch zwei virtuelle Stühle hinter einem realen Tisch demonstriert. Das Telefon ist ebenfalls real, die Lampe ein virtuelles Modell. Die Tischplatte und die Tischbeine verdecken auf korrekte Weise die Stühle. In allen drei Dimensionen sind die verschiedenen Objekte in der richtigen räumlichen Anordnung dargestellt. Die virtuelle Lampe überlagert also den realen Tisch, und der reale Tisch verdeckt Teile der zwei virtuellen Stühle. Die Position und Orientierung des geometrischen Modells des Tisches wurden zuvor an die Lage des realen Tisches angepasst.

Abbildung 3.3: Ein realer Tisch verdeckt zwei virtuelle Stühle unter Verwendung der Modellbasierten Methode [Bree95]

3.4 Tiefenbasierter Ansatz mit Realitätsmodell

Dieser Abschnitt beschreibt einen Lösungsansatz, der ebenfalls auf der Zuhilfenahme eines Realitätsmodells beruht. Klinker und Breen ziehen hierbei jedoch zusätzlich die Tiefeninformation des Modells zur Verdeckungsberechnung hinzu [Klin00], [Bree95]. Liegt ein korrektes Modell der Realität vor, können damit visuelle Verdeckungen zwischen realen und virtuellen Objekten auf effiziente Weise durch die geometrische Rendering Hardware der modernen Grafik-Workstations berechnet werden.

Zu Beginn jedes Rendering-Zyklus wird der z-Buffer mit den Tiefenwerten der realen Welt initialisiert. Dadurch kann die Verdeckung von virtuellen Objekten automatisch dargestellt werden. Während das virtuelle Objekt dargestellt wird, werden Bildpunkte dieses Objekts, die sich weiter entfernt von der Kamera befinden als die z-Werte aus der Tiefenkarte, nicht gezeichnet. Das Realitäts-Modell hingegen wird bildpunktweise transparent errechnet. Dadurch kann der z-Buffer der Rendering Maschine entsprechend den Entfernungen der realen Objekte zum Benutzer initialisiert werden. Beim nachfolgenden Rendering der virtuellen Objekte werden nur die

Objekte dargestellt, die sich näher an der Kamera oder dem Benutzer befinden als die realen Objekte [Klin00].

In Abbildung 3.4 kommt diese tiefenbasierte Technik zum Einsatz. Das Bild zeigt ein virtuelles Sofa, das einen Türdurchgang verdeckt und teilweise selbst von ihm verdeckt wird. Für dieses Bild wurde ein geometrisches Modell des Raums mit dem realen Raum registriert. Danach konnte die Tiefenkarte aus dem z-Buffer der Grafik-Hardware ermittelt und mit ihrer Hilfe untenstehendes Bild dargestellt werden [Bree95].

Abbildung 3.4: Ein virtuelles Sofa in einem realen Türdurchgang, unter Verwendung der Tiefenbasierten Methode [Bree95]

3.5 Tiefenbasierter Ansatz ohne Realitätsmodell

Wloka und Anderson verfolgen einen Lösungsansatz, der ohne den Einsatz eines Modells der realen Welt funktioniert [Wlok95]. Sie entwickelten einen Algorithmus,

der jedem Bildpunkt eines Stereo-Videobildpaars Tiefenwerte zuordnet. Mit Hilfe dieser Tiefenwerte werden ähnlich wie in Kapitel 3.4 visuelle Verdeckungen dargestellt. Entwickelt wurde dieser Lösungsansatz für videobasierte HMDs, da diese bereits ein Bildpaar von zweidimensionalen Bitmaps aus den Videobildern liefern.

Auf dem HMD werden zwei Schwarzweißvideokameras nebeneinander möglichst auf einer Höhe montiert. Diese Kameras liefern ununterbrochen ein Bildpaar, bestehend aus einem Bild der linken und einem Bild der rechten Kamera. Das analoge Bildsignal wird zu Bitmaps digitalisiert. Auf die digitalisierten Bilder wird dann der Algorithmus angewendet.

Zuerst werden die z-Werte des linken Bilds in den z-Buffer des Prozessors der Grafikkarte und das zugehörige Bild in den Bildpuffer kopiert. Da jetzt jeder Bildpunkt des linken Videobilds einen zugehörigen Tiefenwert besitzt, können nun die virtuellen Objekte für die linke Seite des HMDs dargestellt werden. Das Problem der visuellen Verdeckung wird durch Verwendung des z-Buffers gelöst (Kapitel 3.4). Nachdem der z-Buffer wieder gelöscht wurde, wird das gesamte Verfahren für das Bild der rechten Kamera wiederholt. Die Darstellung des Videobildpaars im HMD ist damit vollständig. Für das nächste Bild muss der eben beschriebene Zyklus erneut durchlaufen werden.

Der von Wloka und Anderson entwickelte Algorithmus arbeitet im Vergleich zu anderen um ein Vielfaches schneller. Trotzdem arbeitet das tiefenbasierte System nur annähernd in Echtzeit. Pro Bildpaar werden etwa 470 ms benötigt. Den größten Zeit- und Ressourcenverbrauch verzeichnet dabei der verwendete Algorithmus mit 370 ms. Für den Einsatz in Augmented Reality ist das System damit noch nicht leistungsfähig genug. Teilweise weisen die Tiefenkarten der einzelnen Bilder auch Fehler auf, was sich störend auf den Gesamteindruck des AR-Bildes auswirkt.

Ein großer Vorteil ist jedoch, dass zur Darstellung visueller Verdeckungen keine Geometrien der realen Welt benötigt werden. Das System von Wloka und Anderson kann somit auch in realen Umgebungen, die gänzlich unbekannt sind und über die keine Informationen zur Verfügung stehen, eingesetzt werden.

3.6 Konturbasierter Ansatz

Um visuelle Verdeckungen in einem AR-System zu behandeln, verfolgt Berger einen Lösungsansatz, der mit Hilfe der Umrisslinien von realen Objekten Verdeckungen darstellen kann. Der Ansatz basiert auf der Tatsache, dass für die Darstellung einer Verdeckung eines virtuellen Objekts durch ein reales Objekt, eine Art Verdeckungsmaske berechnet werden muss. Diese Maske beschreibt den Bereich eines Bildes, in dem das reale Objekt zu sehen sein soll [Berg97].

Abbildung 3.5 zeigt die einzelnen Arbeitsschritte des konturbasierten Ansatzes für ein Bild in einer AR-Anwendung. Soll ein virtuelles Objekt in eine AR-Szene eingefügt und dort dargestellt werden (hier ein virtuelles Rechteck), werden zuerst die Konturen dieses virtuellen Objekts berechnet (Abbildung 3.5 b). Im Bereich dieser Kontur werden die groben Konturen des realen Bildes ermittelt und durch verschiedene Verfahren verfeinert (Abbildung 3.5 c und d). Jeder einzelne Punkt der Umrisslinie wird nun mit der Zusatzinformation versehen, die besagt, ob er sich vor oder hinter dem virtuellen Objekt befindet (Abbildung 3.5 e und f). Die Berechnung dieser Zusatzinformation erfordert keine 3D-Rekonstruktionen der realen Umgebung. Im letzten Schritt wird in mehreren Zyklen die Verdeckungsmaske aus denjenigen Konturen berechnet, welche die Zusatzinformation enthalten, dass sie sich vor dem virtuellen Objekt befinden (Abbildung 3.5 g bis j). Durch diese Verdeckungsmaske ist der Bereich des Bildes bekannt, in welchem das virtuelle Objekt nicht abgebildet werden darf. Die visuelle Verdeckung kann den räumlichen Verhältnissen entsprechend dargestellt werden [Berg97].

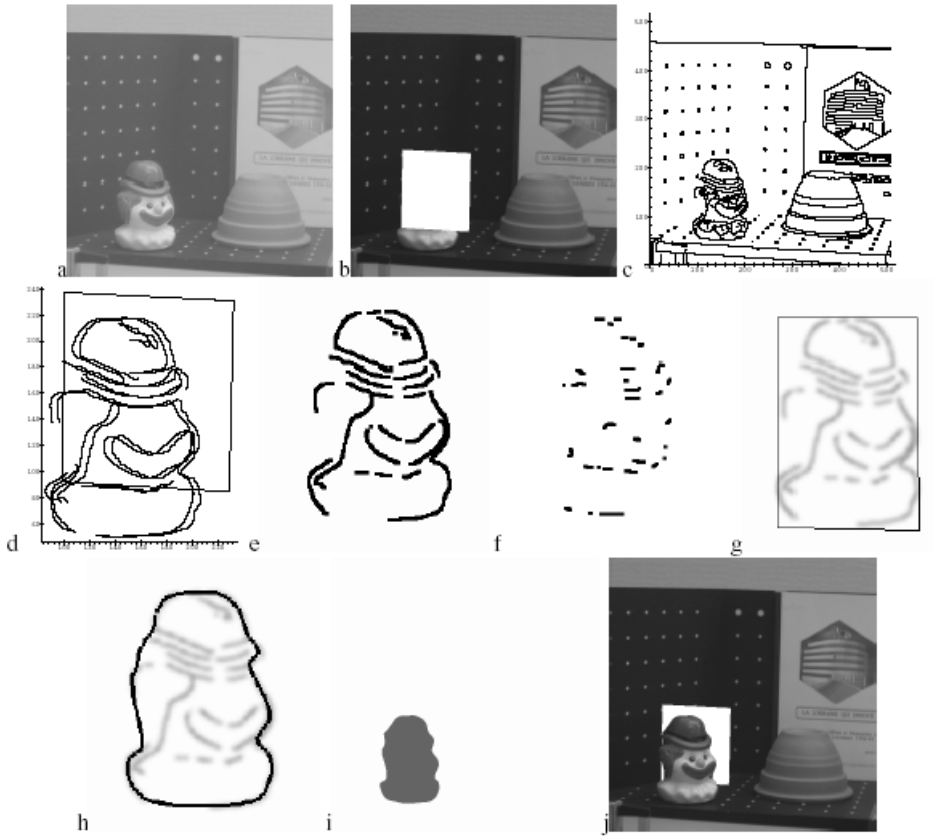


Abbildung 3.5: Einfügen eines virtuellen Rechtecks in eine reale Szene unter der Verwendung der Konturbasierten Methode [Berg97]

Für Bergers Lösungsansatz werden ebenfalls keine 3D-Rekonstruktionen benötigt, was den Einsatz des AR-Systems auch in unbekannten realen Umgebungen möglich macht. Zudem arbeiten die verwendeten Algorithmen schnell und sind leicht zu implementieren. Das Ergebnis des fertigen AR-Bildes ist dabei immer von der Qualität der ermittelten Konturen abhängig. Wird eine Kontur übersehen, kann das dazu führen, dass zwei verdeckende Objekte statt einem errechnet werden. Gerade in realen Umgebungen, die viele verschiedene Muster und Strukturen aufweisen und dadurch keine klaren Konturen erkennen lassen, stößt der verwendete Algorithmus an seine Grenzen. Gut geeignet dagegen ist er für Szenen, die klar erkennbare Konturen aufweisen.

3.7 Bewertung der Lösungsansätze

Die vorgestellten Lösungsansätze weisen jeder für sich unterschiedlich viele Vor- und Nachteile auf. In diesem Abschnitt werden sie noch einmal kurz zusammengefasst.

- Modellbasierter Ansatz:
  Von Vorteil bei diesem Ansatz ist die einfache Implementierung und seine zuverlässige Funktionalität. Die 3D-Geometrien der realen Welt, die verwendet werden, enthalten bereits alle notwendigen räumlichen Daten und damit auch die Verdeckungsinformationen. Diese gelten für jede Ansicht des Benutzers, welche sich in einer Echtzeit-AR-Anwendung sehr häufig und schnell ändern kann. Eingesetzt werden sollte dieser Lösungsansatz, wenn bereits ein umfangreiches, präzises Modell der realen Umgebung vorliegt, in dem das AR-System eingesetzt werden soll. Von Nachteil ist nur, wenn eine große Menge an Objekten berechnet werden muss, was durch moderne Grafikhardware jedoch nicht mehr allzu stark zum Tragen kommt.

- Tiefenbasierter Ansatz mit Realitätsmodell:
  Dieser tiefenbasierte Ansatz arbeitet auch auf komplexen Szenen nahezu fehlerfrei, sogar wenn sie viele Objekte enthalten. Von Nachteil ist, dass die Tiefeninformation der Szene jedes Mal von neuem berechnet werden muss, sobald sich Position und Orientierung des Benutzers ändert, was die benötigte Rechenleistung wieder erhöht. Dieser Lösungsansatz kann ebenfalls nur eingesetzt werden, wenn ein genaues Modell der realen Umgebung vorliegt.

- Tiefenbasierter Ansatz ohne Realitätsmodell:
  Der entscheidende Vorteil dieser Methode ist, dass keinerlei 3D-Rekonstruktionen der realen Umgebung erforderlich sind. Allerdings ist die Berechnung der Tiefenwerte noch nicht vollkommen fehlerfrei. Ein Einsatz in Echtzeitsystemen ist ebenfalls nicht möglich, da die Berechnung eines Bildes etwa 470 ms benötigt. Des Weiteren ist die Implementierung dieses Ansatzes in eine AR-Anwendung nicht trivial.

- Konturbasierter Ansatz:
  Auch hier werden keine Rekonstruktionen der realen Umwelt benötigt. Das heißt, das AR-System, das diesen Lösungsansatz verwendet, kann auch in unbekannten realen Umgebungen eingesetzt werden. Die Algorithmen arbei-

ten schnell und sind leicht zu implementieren. Die Konturen müssen für jedes einzelne darzustellende Bild neu berechnet werden, was dann zum Nachteil wird, wenn sich Position und Orientierung des Benutzers häufig ändern. Geeignet ist der Lösungsansatz vor allem für Umgebungen, die klare, leicht erkennbare Konturen aufweisen. In Umgebungen, die sehr komplex aufgebaut sind und viele Muster und Texturen aufweisen, arbeitet der Algorithmus nicht fehlerfrei genug, um korrekte Ergebnisse liefern zu können.

Neben den oben genannten Ansätzen, beschäftigt sich die moderne Forschung im Bereich der Computer-Visualisierungs-Techniken auch mit dem nicht trivialen Problem der dreidimensionalen Rekonstruktion realer Objekte, zum Beispiel mit Hilfe von Videobildpaaren. Die Darstellung dieser Rekonstruktionen wiederum kann dann für Objektverdeckungszwecke verwendet werden. Zum heutigen Zeitpunkt steht diese Technik noch am Anfang ihrer Entwicklung. Für Echtzeitsysteme kommt dieser Ansatz absolut nicht in Frage, da die notwendigen Berechnungen viel zu umfangreich und komplex sind [Wlok95].

Für die Visualisierung der Objektverdeckung wird in dieser Arbeit der modellbasierte Ansatz verwendet. Da die benötigten 3D-Geometrien zur Verfügung stehen, bietet dieser Ansatz die sinnvollste Lösung der Objektverdeckungsproblematik. Bei anderen Ansätzen ist die Implementierung von wesentlich größerem technischem Aufwand geprägt. Ebenso wichtig ist, dass der verwendete Lösungsansatz auch in einer Echtzeitanwendung sinnvolle Ergebnisse liefert und auch bei häufig wechselnden Positions- und Orientierungsänderungen des Benutzers einsatzfähig bleibt. Die in den folgenden Kapiteln durchgeführte Analyse, Konzeption und Realisierung der Objektverdeckung basiert auf dem modellbasierten Ansatz.

4 Anforderungsanalyse

4.1 Übersicht

Die Implementierung einer Objektverdeckung in ein AR-System stellt verschiedene Bedingungen an die beteiligten Objekte und Komponenten. In diesem Kapitel wird analysiert, unter welchen Bedingungen eine Objektverdeckung auftreten kann, welche Objekte unter welchen Umständen daran beteiligt sind und welche Anforderungen sich daraus für die Darstellung der realen Objekte in der AR-Szene ergeben. Dabei basiert diese Anforderungsanalyse auf dem modellbasierten Ansatz zur Darstellung einer Objektverdeckung (Kapitel 3.3).

Für die Ermittlung einer visuellen Verdeckung von virtuellen Objekten durch reale Gegenstände ist zunächst die Kenntnis der an der Objektverdeckung beteiligten Objekte notwendig. Sie lassen sich in zwei Gruppen einteilen. Die erste Gruppe von Objekten bilden die realen Objekte. Dabei handelt es sich um reale Gegenstände und Objekte, die beim Auftreten einer Objektverdeckung in der virtuellen Welt repräsentiert werden, um auch dort die Verdeckung darzustellen. In Abbildung 1.1 im ersten Kapitel dieser Arbeit entsprechen diese realen Objekte beispielsweise den Bäumen. Die zweite Gruppe umfasst die virtuellen Objekte, die durch andere Objekte verdeckt werden können, was wiederum dem eingeblendeten, virtuellen Gebäude in Abbildung 1.1 entspricht. Für beide Objektgruppen ist jeweils ein eigener Abschnitt vorgesehen, in dem auch ihre Verhaltensweisen und die daraus resultierende Bedeutung für die Berechnung und Darstellung einer Objektverdeckung beschrieben wird.

Weitere Abschnitte befassen sich mit den Komponenten, welche die virtuelle Szene darstellen, und mit den geometrischen und praktischen Bedingungen, unter denen eine Objektverdeckung auftreten kann. Um die Objektverdeckung tatsächlich darzustellen, müssen weitere Anforderungen berücksichtigt werden, die sich aufgrund der Objektverdeckung an die Geometrien der verdeckenden Objekte ergeben. Eine Beschreibung der notwendigen Art der Repräsentation erfolgt in einem weiteren Abschnitt. Abschließend werden die dargestellten Erkenntnisse im letzten Abschnitt noch einmal zusammengefasst.

4.2 Reale Objekte

4.2.1 Begriffsbestimmung

Bei realen Objekten handelt es sich um Objekte, die in der realen Welt physisch existieren, das heißt auch körperlich greifbar sind. Betrachtet werden diejenigen Gegenstände, die räumlich zwischen dem Betrachter und virtuellen Objekten liegen, da sie eine visuelle Verdeckung der weiter im Hintergrund liegenden Objekte verursachen können. Für die Anwendung im Augmented Reality Bereich interessant sind vor allem Bauwerke aller Art wie zum Beispiel Gebäude, Denkmäler und Brunnen. Auch Fahrzeuge, Vegetation und Menschen können eine Objektverdeckung verursachen. Reale Objekte lassen sich in zwei Kategorien einteilen. Die eine Kategorie enthält Objekte, die ein statisches Verhalten aufweisen, die zweite umfasst Objekte, die sich dynamisch verhalten.

4.2.2 Reale Objekte mit statischem Verhalten

Reale Objekte, die im Zeitrahmen der AR-Szene ein statisches Verhalten aufweisen, können Bauwerke aller Art oder sonstige fest installierte Objekte in der realen Welt sein. Zum Beispiel sind Gebäude, Mauern, Denkmäler oder auch Straßenschilder jedes für sich reale Objekte, die sich nicht bewegen können und damit statisch sind.

Ihre Lage in der realen Welt – und damit auch in der virtuellen Repräsentation – ist festgelegt und ändert sich nicht. Voraussetzung für die Integration in eine AR-Szene ist die Kenntnis ihrer genauen Lage im Raum. Nur so können sie bei einer Objektverdeckung berücksichtigt werden. Aufgrund der statischen Verhaltensweise müssen diese Parameter nur einmalig im AR-System registriert werden. Eine weitere ständige Beschreibung des Verhaltens ist nicht notwendig. Zusätzlich muss für das reale Objekt ein geeignetes virtuelles Modell zur Verfügung stehen, um das reale Objekt beim Auftreten einer Verdeckung in der virtuellen Welt repräsentieren zu können.

4.2.3 Reale Objekte mit dynamischem Verhalten

Anders liegt der Fall, wenn reale Objekte ein dynamisches Verhalten aufweisen. Hierbei handelt es sich um Objekte, die ihre Position und Orientierung im Laufe der AR-Szene verändern können. Sie lassen sich in zwei Kategorien von Objekten unterteilen. Die eine Kategorie umfasst reale Gegenstände, die ihre Position selbständig verändern können, die zweite Kategorie Objekte, die verschieden positionierbar sind, dabei aber nicht in der Lage sind, dies selbständig zu tun.

Sich selbständig bewegende Objekte sind zum Beispiel Menschen oder Fahrzeuge. Die Schwierigkeit bei ihrer Repräsentation in einer Objektverdeckung liegt vor allem in der Unvorhersehbarkeit ihrer Bewegung. Es können zwar unter Umständen die grobe Richtung und Geschwindigkeit bestimmt werden, diese Werte sind jedoch nicht aussagekräftig genug, um eine exakte Position und Orientierung vorherzusehen und damit das reale Objekt bei einer Verdeckung zu berücksichtigen.

Ein weiterer Punkt ist, dass zum Beispiel Menschen neben ihrer augenblicklichen Position auch ihre Erscheinungsform verändern können. Während ein Auto immer die gleiche Silhouette besitzt, bewegt ein Mensch Arme, Beine und Kopf und variiert damit spontan und unvorhersehbar seine Erscheinungsform. Das macht es schwierig, repräsentative und vor allem zeitnahe Modelle für die Darstellung einer Objektverdeckung zur Verfügung zu haben. Stehen diese Informationen jedoch zur Verfügung, können dynamische, reale Objekte bei einer Objektverdeckung berücksichtigt und dargestellt werden.

Die Gruppe der dynamischen Objekte, die sich nicht selbständig bewegen können, umfasst alle Gegenstände, die erst durch äußeren Einfluss verschieden positionierbar sind. Das können größere Objekte sein, wie zum Beispiel Fahrzeuge, die stehen und sich nicht bewegen, oder auch kleinere Gegenstände wie ein Werkzeug oder Eimer. Ihre Fortbewegung ist entweder an die Bewegung eines sich selbständig bewegenden Objekts gebunden oder es ist innerhalb der Anwendung nicht vorhersehbar, an welcher Stelle in der realen Welt diese Objekte auftreten können.

In jedem Fall ist es notwendig, die aktuelle Position und Orientierung der dynamischen Objekte zu kennen. Stehen zudem auch die entsprechenden, die realen Objek-

te repräsentierenden Modelle zur Verfügung, können sie auch bei der Darstellung einer Objektverdeckung berücksichtigt werden. Auf welchem Weg die Informationen zur Verfügung gestellt werden, ist für diese Arbeit dabei nicht von Bedeutung. Notwendig ist allein ihr Vorhandensein.

4.3 Virtuelle Objekte

4.3.1 Begriffsbestimmung

Die zweite Gruppe der an einer Objektverdeckung beteiligten Objekte umfasst die virtuellen Objekte. Sie liegen in Form von graphischen, dreidimensionalen Modellen vor, die aus einer Menge von Punkten, Flächen und Kanten bestehen. Damit bilden sie den virtuellen Teil der AR-Szene. Oft sind virtuelle Objekte graphische 3D-Repräsentationen von realen Objekten. Besonders interessant sind dabei Rekonstruktionen und Planungen von Bauwerken, die so angezeigt werden können, dass sie reale Gebäude exakt überlagern. Um eine größere Realitätsnähe zu erreichen, sollten sich virtuelle Objekte ähnlich ihren echten Vorbildern, den realen Objekten, verhalten. Aus diesem Grund lassen sich auch virtuelle Objekte in zwei Verhaltenskategorien einteilen. Die Einteilung erfolgt wieder in virtuelle Objekte mit statischem Verhalten und virtuelle Objekte mit dynamischem Verhalten.

Des Weiteren werden virtuelle Objekte für die Visualisierung einer Objektverdeckung eingesetzt. Im modellbasierten Ansatz, der in dieser Arbeit Anwendung findet, werden die virtuellen Objekte als Repräsentation der realen Objekte, die eine Verdeckung verursachen, verwendet (Kapitel 3.3). Was beachtet werden muss, um virtuelle Objekte für die Darstellung einer Objektverdeckung zu verwenden, wird in Kapitel 4.5 beschrieben.

4.3.2 Virtuelle Objekte mit statischem Verhalten

Virtuelle Objekte mit statischem Verhalten ähneln den realen, statischen Objekten. Das heißt, auch die Beispiele für virtuelle Objekte mit statischem Verhalten gleichen denen der entsprechenden realen Objekte: Sie können Bauwerke aller Art oder sons-

tige fest installierte, nicht bewegliche Objekte aus der realen Welt darstellen. Im AR-Bereich handelt es sich dabei vor allem um Rekonstruktionen von Gebäuden, wie zum Beispiel ein virtuelles, dreidimensionales Modell einer Kirche.

Da sich ein reales Gebäude nicht bewegen kann, muss auch seine virtuelle Repräsentation diese Eigenschaft aufweisen und seine Position und Orientierung während der Laufzeit nicht verändern. Um eine exakte Überlagerung der realen mit der virtuellen Welt zu ermöglichen, müssen Position und Orientierung der Objektrepräsentation der Position und Orientierung des jeweiligen realen Objekts entsprechen. Für die Verdeckungsberechnung und ihre Visualisierung ist es notwendig, diese Objekte in die Berechnungen mit einzubeziehen. Die beteiligten Geometrien müssen dabei identifizierbar und verfügbar sein. Somit können ihre Position und Orientierung ermittelt und zur Verdeckungsberechnung verwendet werden.

4.3.3 Virtuelle Objekte mit dynamischem Verhalten

Bei virtuellen Objekten, die sich dynamisch verhalten, handelt es sich um Objekte, die in einer AR-Szene je nach Bedarf verschiedene Positionen einnehmen können. Diese Positions- und Orientierungsänderung erfolgt nie selbständig, da sie durch die AR-Anwendung gesteuert werden muss. Aus diesem Grund entfällt eine Unterteilung in sich selbständig und sich nicht selbständig bewegende Objekte.

Die Steuerung der virtuellen Objekte stellt technisch keine Schwierigkeit dar. Das bedeutet, generell können virtuelle Objekte beliebig positioniert, orientiert und animiert werden. Wenn ein virtuelles Objekt die Repräsentation eines realen, beweglichen Objekts darstellt, ist jedoch die Kenntnis der momentanen Erscheinungsform des entsprechenden realen Objekts notwendig. Es wird zusätzlich ständig die aktuelle Position und Orientierung des realen Objekts benötigt, um dessen Verhalten durch das virtuelle Objekt möglichst exakt umsetzen zu können.

Die betreffenden Informationen zu Position und Orientierung sind in jedem AR-System verfügbar und können auch je nach Bedarf beeinflusst werden. Weiterhin ist die Verfügbarkeit der virtuellen Modelle Voraussetzung für die Visualisierung der Objektverdeckung, um reale Objekte korrekt zu repräsentieren. Für die konkrete Um-

setzung der Objektverdeckung in dieser Arbeit wird davon ausgegangen, dass die nötigen Informationen zur Verfügung stehen. Somit können auch dynamische, virtuelle Objekte bei der Objektverdeckung berücksichtigt werden.

4.4 Sichtbereich

4.4.1 Darstellung der AR-Szene

Dieser Abschnitt umfasst die drei Komponenten, die für die Darstellung der virtuellen Szene verantwortlich sind und deren Erscheinungsbild maßgeblich beeinflussen. Es handelt sich um den Betrachter, die virtuelle Kamera und die Projektionsfläche. Abhängig von ihrem Verhalten und ihren Beziehungen zueinander wird die Abbildung der AR-Szene gesteuert.

Der Betrachter entspricht dem Anwender eines AR-Systems. Er wird in der AR-Szene durch einen virtuellen Beobachter repräsentiert, der wiederum der virtuellen, abbildenden Kamera entspricht. In einem AR-System steuert der Betrachter diese virtuelle Kamera. Sie bildet die dreidimensionale, virtuelle Szene über ein Projektionsverfahren auf einer Projektionsfläche zweidimensional ab, das heißt die Abbildung der AR-Szene wird abhängig von Position und Orientierung des Betrachters dargestellt. Die virtuelle Kamera kann entweder direkt oder indirekt gesteuert werden. Zumeist wird in der Augmented Reality die direkte Steuerung eingesetzt. In diesem Fall stimmen Position und Orientierung der Kamera mit der Position und Orientierung des Betrachters überein. Damit kann beispielsweise durch den Einsatz von HMDs eine exakte Überlagerung der realen Welt durch eine virtuelle Szene erreicht werden. Hierbei entsprechen der Betrachter der virtuellen Kamera und das HMD der Projektionsfläche, auf der die AR-Szene je nach Position und Blickrichtung des Betrachters dargestellt wird.

Wenn als Projektionsfläche kein HMD, sondern zum Beispiel ein Bildschirm eingesetzt wird, handelt es sich um eine indirekte Steuerung der virtuellen Kamera durch den Benutzer. Seine eigene Position und Orientierung haben keinen unmittelbaren Einfluss auf die Navigation der Kamera und er wird somit nicht direkt in der AR-Szene repräsentiert. Die Kamera wird stattdessen beispielsweise über eine Tastatur

oder ein anderes Eingabemedium geführt, um auf diesem Weg die Abbildung der AR-Szene zu steuern. Das Verhalten der Kamera entspricht dabei nicht direkt dem Verhalten des Betrachters.

Für die Realisierung einer Objektverdeckung ist die Art der Kamerasteuerung irrelevant. Es ist nicht von Bedeutung, ob die virtuelle Kamera direkt der Position und Blickrichtung des Benutzers entspricht oder ob sie durch andere Steuerungen navigiert wird. Ausschlaggebend sind vielmehr Position und Orientierung der virtuellen Kamera sowie ihre Lage bezüglich der Projektionsfläche. Die virtuelle Kamera wird durch verschiedene Parameter wie Position und Orientierung gesteuert. Auf diese und weitere Parameter wird in den nächsten Abschnitten näher eingegangen.

4.4.2 Allgemeines zum Sichtbereich

Der Betrachter, reale und virtuelle Objekte, die Projektionsfläche und die virtuelle Kamera stehen in geometrischen Beziehungen zueinander. Je nach Lage der abbildenden Komponenten können verschiedene Objekte räumlich hintereinander liegen und sich damit verdecken. Welche einzelnen Parameter gegeben sind und dadurch eine Objektverdeckung verursachen beziehungsweise beeinflussen können, lässt sich durch die Eigenschaften des Sichtbereichs definieren. Durch ihn sind veränderliche Größen, wie zum Beispiel Position, Blickrichtung und Blickwinkel, die sich auf die Darstellung der Szene beziehen, festgelegt.

Der Sichtbereich ist in einem AR-System dreidimensional aufgebaut. Alle seine Eigenschaften beziehen sich auf einen Raum mit den drei Dimensionen Breite, Tiefe und Höhe. Das heißt, auch einzelne Parameterwerte werden durch dreidimensionale Koordinaten repräsentiert. Die Position eines Objektpunktes wird beispielsweise mit drei Zahlenwerten angegeben.

Für die Berechnung einer Objektverdeckung sind jedoch nicht alle drei Richtungsachsen von Bedeutung. Normalerweise orientiert sich ein Betrachter vornehmlich in einer horizontalen Ebene. Auch eine Objektverdeckung tritt meist nicht über oder unter ihm auf, sondern eher auf gleicher Höhe mit ihm. Deswegen sind im Rahmen

dieser Arbeit vor allem die Seiten- und die Tiefenachse von Bedeutung. Die Höhenachse und damit die dritte Dimension kann vernachlässigt werden.

Dadurch ergibt sich auch ein Vorteil bei der eigentlichen Berechnung einer Objektverdeckung. Berechnungen, die unter Vernachlässigung der dritten Achse im zweidimensionalen Bereich stattfinden, sind einfacher und dadurch schneller als im dreidimensionalen Raum. Da die Ermittlung einer Objektverdeckung in kürzester Zeit möglich sein soll, um noch genügend Zeit zur Verfügung zu haben, die Verdeckung auch graphisch darzustellen, ist die zweidimensionale Berechnung unbedingt sinnvoll. Demzufolge beziehen sich auch die in diesem Abschnitt dargestellten Abbildungen auf den zweidimensionalen Raum. Während der Sichtbereich im dreidimensionalen Raum als Kegel dargestellt werden müsste, besitzt er hier vielmehr die Form eines Kreissegments beziehungsweise Trapezes. Da es sich um die Höhenachse handelt, die übergangen wird, stellen Abbildungen zum Sichtbereich förmlich die Aufsicht einer dreidimensionalen AR-Szene dar.

In Abbildung 4.1 ist die Aufsicht des Sichtbereichs beispielhaft dargestellt. Auf der hier verwendeten Darstellung der einzelnen Symbole basieren auch die Abbildungen in den folgenden Abschnitten und Kapiteln dieser Arbeit. Das schwarze, mit *VP* gekennzeichnete Symbol stellt dabei das Betrachterauge dar. Von ihm ausgehend erstreckt sich zwischen den zwei Pfeilen hellgrau hinterlegt der Sichtbereich des Betrachters. Die Position des Betrachters ist durch die Werte x_{vp} und z_{vp} auf den Achsen festgelegt. Reale Objekte werden als schwarz umrandete, weiße Flächen dargestellt (*R1, R2*). Virtuelle Objekte sind dunkelgrau hinterlegt (*V*). Konkret ist in dieser Abbildung die Situation dargestellt, in der das reale Objekt *R1* das virtuelle Objekt *V* verdeckt. Weitere Parameter des Sichtbereichs werden in den folgenden Abschnitten ergänzt und näher erläutert.

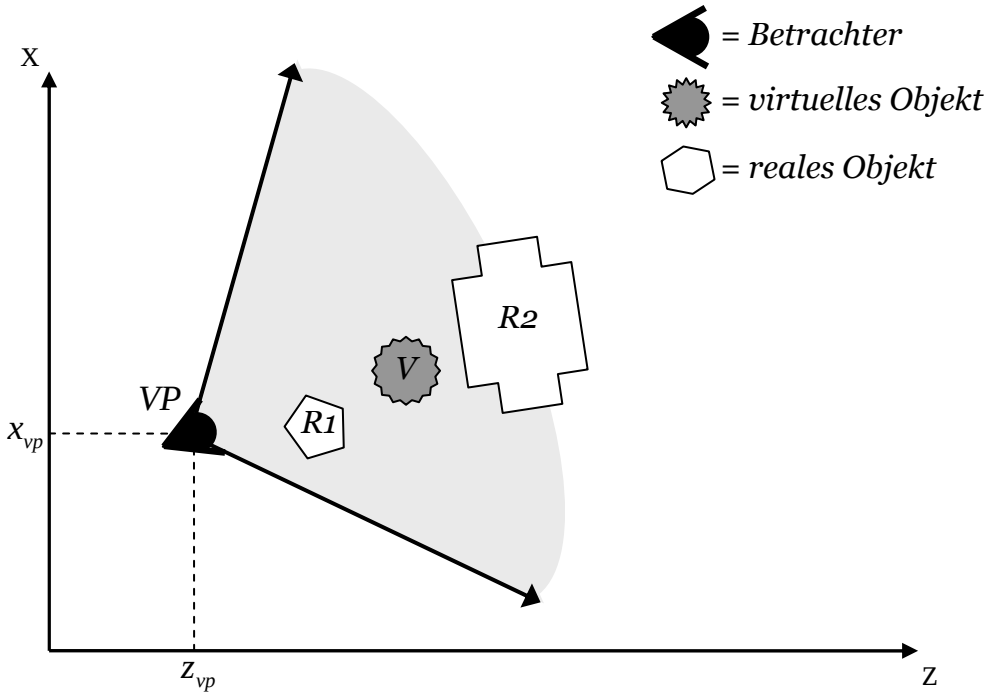


Abbildung 4.1: Schematische Darstellung der Objektverdeckung zwischen realen und virtuellen Objekten

4.4.3 Position der abbildenden Komponenten

Eine Verdeckung von virtuellen Objekten durch reale Objekte tritt auf, wenn sich ein realer Gegenstand räumlich zwischen Benutzer und virtuellem Objekt befindet. Dabei ist von Bedeutung, von welcher Position aus eine AR-Szene betrachtet wird beziehungsweise wo sich die virtuelle Kamera und die Abbildungsfläche des Systems befinden.

Bei der Verwendung von HMDs liegt die Abbildungsfläche für das virtuelle Bild immer zwischen dem Betrachter und der realen Welt und damit optisch zwangsläufig vor den realen Objekten. Dies kann jedoch den Eindruck der räumlichen Tiefe des Gesamtbildes zerstören, wenn ein virtuelles Objekt räumlich hinter einem realen Objekt liegt, optisch jedoch vor diesem abgebildet wird. Da bei der Verwendung von HMDs keine Objekte zwischen Projektionsfläche und Betrachter liegen können, verursachen alle realen Objekte, die räumlich zwischen dem Benutzer beziehungsweise der Abbildungsfläche und dem virtuellen Objekt liegen, eine Objektverdeckung.

Wenn als Abbildungsfläche kein HMD sondern zum Beispiel ein Bildschirm eingesetzt wird, können sich reale Objekte sowohl zwischen dem Betrachter und der Abbildungsfläche als auch zwischen Abbildungsfläche und virtuellem Objekt befinden. Es ist notwendig dies zu unterscheiden, denn nur wenn sich reale Objekte zwischen der Abbildungsfläche und einem virtuellen Objekt befinden, machen sie die Visualisierung der Objektverdeckung im AR-System notwendig.

Unabhängig von der Art der verwendeten Darstellungsmedien ist zu erkennen, dass nicht notwendigerweise die Position des Betrachters ausschlaggebend für das Auftreten einer Objektverdeckung ist. Es ist vielmehr die Position der Projektionsfläche entscheidend, auf der die AR-Szene abgebildet wird.

Im ersten Fall, in dem die Abbildungsfläche mobil als HMD mit dem Betrachter mitgeführt wird, entspricht die Position der Abbildungsfläche nahezu der des Betrachters. Da dieser sich im Raum bewegt, ist vor allem die sich zur Laufzeit ständig ändernde Position des Betrachters zu beachten. Als Anforderung an die Berechnung und Darstellung der Objektverdeckung folgt daraus, dass die aktuelle Position des Betrachters und damit die virtuelle Kamera und Abbildungsfläche ständig zur Laufzeit bekannt sein muss.

Im zweiten Fall, in dem die Abbildungsfläche nicht mobil ist, sondern ein feststehender Bildschirm für die Abbildung der AR-Szene verwendet wird, ändert sich die Position der virtuellen Kamera und damit der Projektionsfläche abhängig von den Navigationsmöglichkeiten des AR-Systems. Es kann zum Beispiel eine Navigation in der AR-Szene unter Einsatz einer Tastatursteuerung der virtuellen Kamera möglich sein. Dabei muss wieder die jeweils aktuelle Position der Kamera zur Verfügung stehen. Die Position des Betrachters hingegen kann vernachlässigt werden. Generell muss die Position der virtuellen Kamera zur Laufzeit ständig aktualisiert werden.

Bei der Verwendung von HMDs ist bei der Bestimmung der Position des Betrachters zu beachten, dass die Messwerte Ungenauigkeiten aufweisen können. Das GPS-Gerät, mit dem die Position des Betrachters beispielsweise ermittelt wird, liefert Werte mit einer Messgenauigkeit von nur etwa ±0,4 m [Sapos]. Die Positionsbestimmung einer durch Tastatur geführten, virtuellen Kamera ist dagegen ausreichend genau.

Eventuell auftretende, minimale Ungenauigkeiten durch Rundungsfehler können vernachlässigt werden. Sie liegen in einem Bereich, der bei der Darstellung der AR-Szene nicht sichtbar wäre. Für die in dieser Arbeit entwickelte Objektverdeckungsbehandlung sollte die Ungenauigkeit der Positionswerte von ±0,4 m bei der Berechnung einer Objektverdeckung berücksichtigt werden, da diese in möglichst vielen AR-Systemen einsetzbar sein soll.

In Abbildung 4.2 ist die Position des Betrachters und damit des Sichtbereichs schematisch dargestellt. Der hellgrau hinterlegte Bereich stellt dabei den Sichtbereich dar. Der kleinere, gestrichelte Kreis um den Betrachter kennzeichnet den Bereich, der sich durch den oben genannten Messfehler *d* ergibt. Dieser Fehler überträgt sich auf den mit durchgezogener Linie gezeichneten Sichtbereich. Daraus resultiert der gesamt mögliche Sichtbereich, der als großer, mit gestrichelter Linie begrenzter Kreis dargestellt wird.

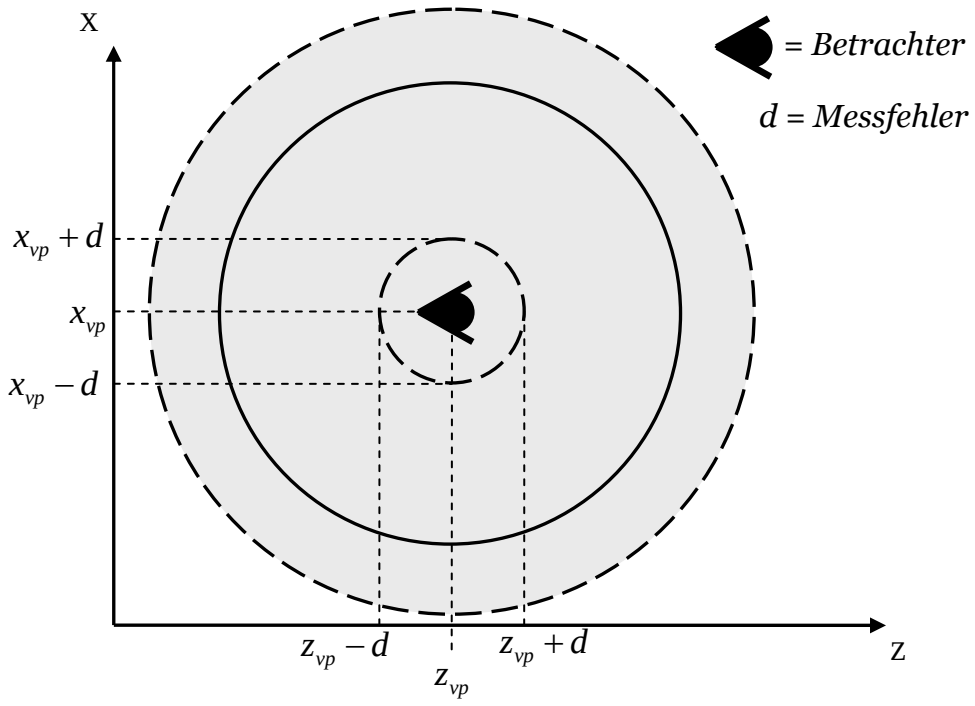


Abbildung 4.2: Schematische Darstellung der Position des Betrachters beziehungsweise der virtuellen Kamera inklusive Messfehler

4.4.4 Orientierung der abbildenden Komponenten

Ein weiterer bei der Objektverdeckungsbehandlung zu berücksichtigender Faktor ist die Blickrichtung des Betrachters beziehungsweise die Orientierung der virtuellen Kamera und damit der Abbildungsfläche. Denn allein durch die Position von Betrachter, Abbildungsfläche, virtueller Kamera und auch der realen und virtuellen Objekte kann nicht ermittelt werden, ob eine Objektverdeckung auftritt. Die Position liefert zwar ein erstes Indiz für das Auftreten einer Verdeckung, diese ist dadurch aber noch nicht eindeutig bestimmt.

Der Betrachter beziehungsweise die abbildende Kamera können jeweils nur in eine Richtung orientiert sein. Das heißt, selbst wenn aus der räumlichen Anordnung von realen und virtuellen Objekten eine Objektverdeckung auftritt, muss diese nicht unbedingt auch berechnet und dargestellt werden. Beispielsweise schaut der Benutzer in eine Richtung, die für die sich verdeckenden Objekte vollkommen irrelevant ist, weil sich diese räumlich hinter ihm befinden. Eine Darstellung der Objektverdeckung ist in diesem Fall nicht notwendig. Es zeigt sich somit die Notwendigkeit die Orientierung bei der Berechnung der Objektverdeckung zu berücksichtigen. Beim Einsatz von HMDs ist zusätzlich zu beachten, dass sich die Blickrichtung des Betrachters im Vergleich zu seiner Position sehr schnell ändern kann. Schon leichte Kopfdrehungen können das Sichtfeld schnell verändern.

Auf jeden Fall lässt sich auf eine Objektverdeckung nur dann schließen, wenn die beiden Parameter Position und Orientierung in ihrer Kombination betrachtet werden. Da sich auch die Orientierung fortwährend ändert, besteht für die Darstellung der Objektverdeckung außerdem die Anforderung, dass die momentane Orientierung des Betrachters oder der virtuellen Kamera zur Laufzeit ständig zur Verfügung steht.

Weiterhin ist auch bei der Bestimmung der Blickrichtung des Betrachters zu beachten, dass die Messwerte Ungenauigkeiten aufweisen können. Wird für die Ermittlung der Blickrichtung des Betrachters ein Orientierungstracker verwendet, dann dieser Werte mit einer Messgenauigkeit von 0,5° liefern [Azum99]. Wie im vorangegangenen Abschnitt zur Position der abbildenden Komponenten ist auch die Orientierungsbestimmung einer durch Tastatur geführten, virtuellen Kamera ausreichend

genau. Auch eventuell auftretende, minimale Ungenauigkeiten durch Rundungsfehler können vernachlässigt werden. Im Fall der Objektverdeckung sollte trotzdem eine Messungenauigkeit berücksichtigt werden, die in etwa der des Orientierungstrackers entspricht. Damit ist das Objektverdeckungssystem umfassender und vor allem im Outdoor-Bereich einsetzbar.

Abbildung 4.3 stellt schematisch dar, wie sich Position und Orientierung auf den Sichtbereich auswirken. Dabei werden die oben genannten Messfehler wieder als gestrichelte Linien dargestellt. Der Messfehler der Orientierung wird als Winkel δ dargestellt. Die Kombination von Position und Orientierung führt zu einer Verkleinerung des Sichtbereichs. Dieser umfasst nun keinen Kreis mehr, sondern vielmehr ein Kreissegment (hellgrau hinterlegt).

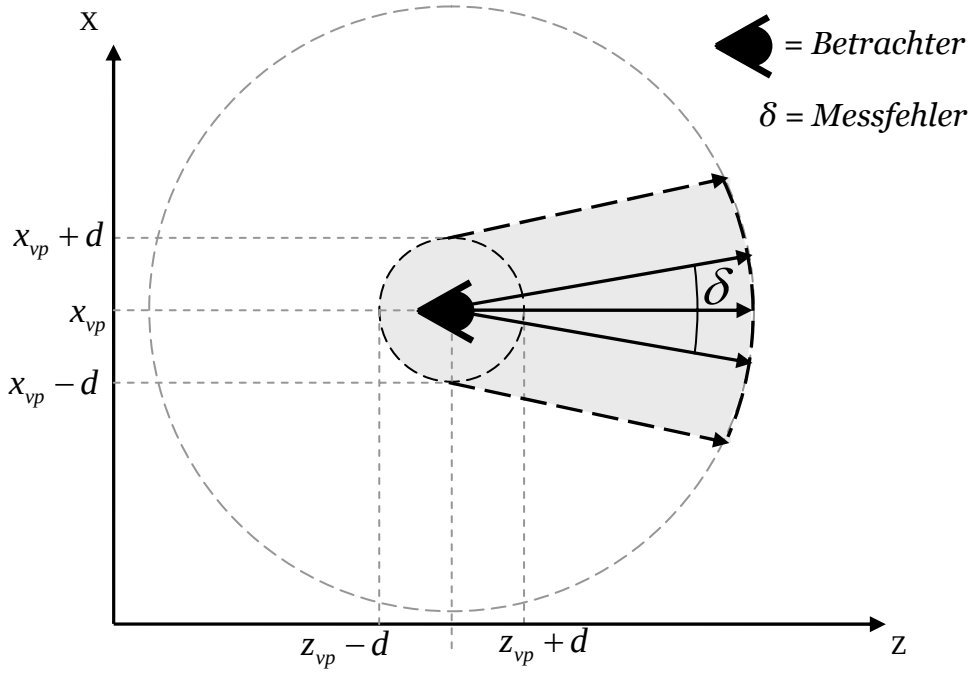


Abbildung 4.3: Schematische Darstellung der Parameter Position und Orientierung inklusive Messfehler

4.4.5 Blickwinkel der abbildenden Komponenten

Der Blickwinkel des Betrachters beziehungsweise Öffnungswinkel der virtuellen Kamera ist eine notwendige Ergänzung zur Orientierung. Während die Orientierung

nur eine Richtung bezeichnet, gibt der Blickwinkel den Winkel an, mit dessen Weite eine Szene abgebildet wird.

Da sich diese Analyse auf den zweidimensionalen Raum beschränkt, wird vom Betrachter nur der horizontale Sehwinkel benötigt. Dieser gibt den Winkel an, „den die von den äußeren Punkten eines Gegenstandes zum Auge ziehenden Linien bilden“ [Univers]. Beim menschlichen Auge beträgt der horizontale Sehwinkel etwa 120° bis 180° (Kapitel 2.2.3). Für die Berechnung einer Objektverdeckung ist dieser Bereich vollkommen ausreichend, da er den möglichen Sichtbereich des menschlichen Auges vollständig abdeckt (Abbildung 4.2).

In Abbildung 4.4 ist der Sichtbereich unter Berücksichtigung des Betrachterblickwinkels φ hellgrau dargestellt. Begrenzt wird er durch die schwarz gestrichelten Linien, die sich aus den Messfehlern der Parameter Position, Orientierung und Blickwinkel ergeben.

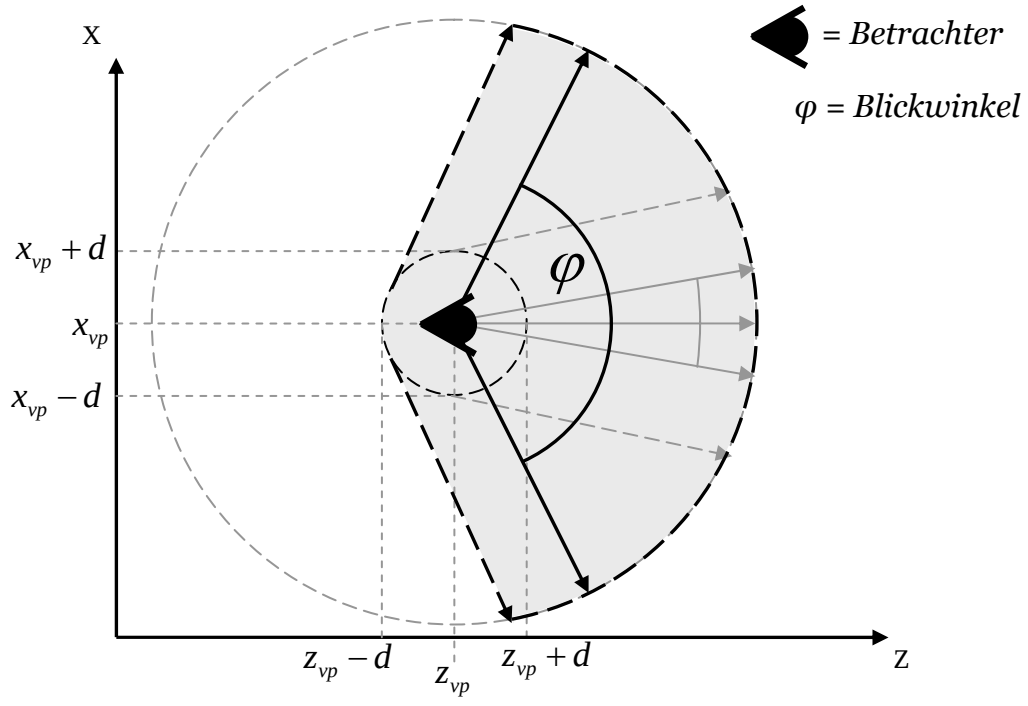


Abbildung 4.4: Schematische Darstellung des Blickwinkels inklusive der Parameter Position und Orientierung

4.4.6 Abstand zwischen virtueller Kamera und Objekten

Weiterhin relevant bei der Objektverdeckung und ihrer Darstellung sind die Abstände zwischen einzelnen Objekten und den abbildenden Komponenten. Grundsätzlich tritt eine Verdeckung dann auf, wenn sich räumlich zwischen der abbildenden Kamera beziehungsweise dem Betrachter und einem virtuellen Objekt ein reales Objekt befindet. Das heißt, der Abstand *b* vom realen Objekt *R1* zur Kamera ist kleiner als der Abstand *a* vom virtuellen Objekt *V* zur Kamera (Abbildung 4.5).

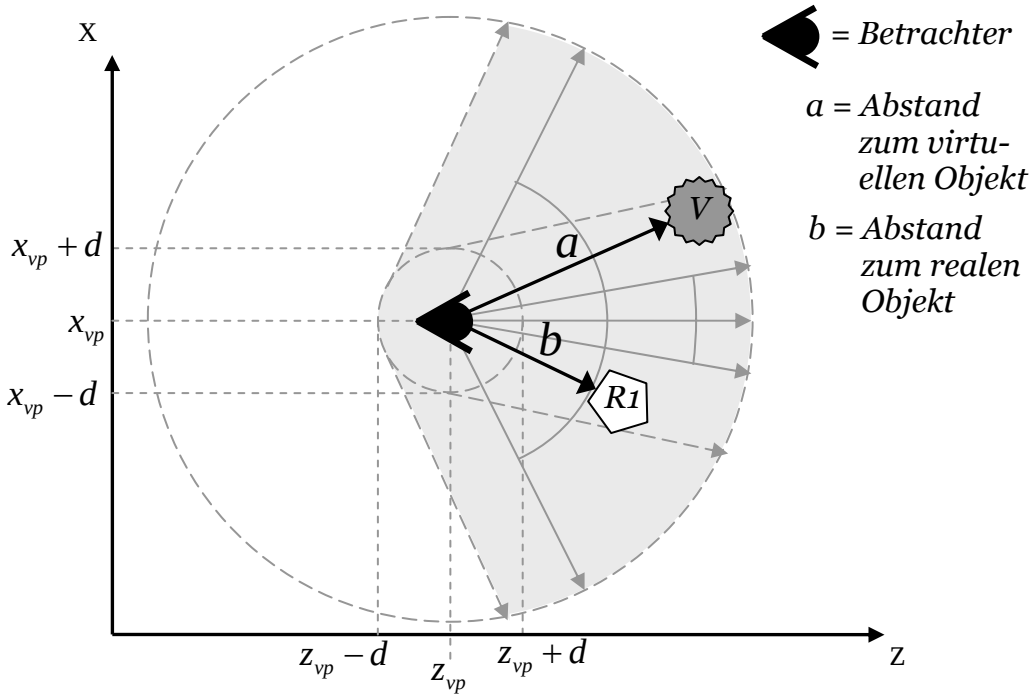


Abbildung 4.5: Schematische Darstellung des Abstands inklusive der Parameter Position, Orientierung und Blickwinkel

Zusätzlich zu den Abständen von Betrachter und Objekten ist zu beachten, dass der Sichtbereich des Betrachters naturgemäß eingeschränkt ist. Ab einer gewissen Distanz können Objekte nicht mehr genau erkannt oder gar nicht mehr gesehen werden. Das ist auf das begrenzte Auflösungsvermögen des menschlichen Auges zurückzuführen. Es beträgt im Durchschnitt 1' bis 1,5' (Winkelminuten), was beispielsweise bedeutet, dass zwei Punkte in 3 m Entfernung einen Mindestabstand von 1 bis 1,5 mm haben müssen, um noch voneinander getrennt wahrgenommen werden zu können. Schon bei einer Entfernung von 100 m ergibt sich daraus eine Mindestgröße noch sichtbarer Details von etwa 3,33 –5 cm. Folglich ist auch die Berechnung einer

Objektverdeckung sowie ihre Darstellung mit zunehmender Entfernung und bei entsprechend kleiner Objektgröße wenig sinnvoll.

4.5 Anforderungen an die Geometrien und ihre Darstellung

Bisher wurde analysiert, unter welchen Bedingungen eine Objektverdeckung auftreten kann und welche Eigenschaften diese weiter beeinflussen können. Um eine Objektverdeckung vollständig in einem AR-System zu simulieren, ist nach dem modellbasierten Ansatz auch ihre Darstellung notwendig. Hierfür wird für jedes reale Objekt, das in der AR-Szene repräsentiert werden soll, eine geometrische Repräsentation in Form eines virtuellen, dreidimensionalen Modells benötigt. Im Folgenden wird beschrieben, welche Anforderungen sich durch eine Objektverdeckung an die Objektgeometrien und ihre Darstellung ergeben.

Bei der Visualisierung der Objektverdeckung müssen reale Objekte dann in der AR-Szene als virtuelles Modell repräsentiert werden, wenn sie aus der Sicht des Betrachters ein virtuelles Objekt verdecken. Dabei besteht nicht die Anforderung, dass die virtuelle Darstellung des realen Objekts der Abbildung des entsprechenden realen Objekts entspricht, denn das reale Objekt selbst soll in der virtuellen Welt nicht sichtbar sein. Vielmehr sollen die Bereiche eines verdeckten Objekts, die durch eine Verdeckung in der realen Welt nicht mehr sichtbar wären, auch in der virtuellen Welt nicht sichtbar sein. Dies wird nach dem modellbasierten Ansatz erreicht, indem die virtuellen Repräsentationen der realen Objekte in der Farbe des Hintergrunds der virtuellen Welt dargestellt werden. Als eine Anforderung an die Darstellungsweise der verdeckenden Objekte folgt hieraus, dass diese ohne Verwendung von Materialien, Texturen oder Farben abzubilden sind. Die einzige farbliche Eigenschaft, die die Geometrien besitzen dürfen beziehungsweise sollen, ist die Farbe des Hintergrunds der virtuellen Szene.

In den meisten AR-Systemen entspricht dies der Farbe schwarz. Damit das Objekt wirklich vollständig nicht sichtbar erscheint, müssen zusätzlich sämtliche Oberflächeneigenschaften wie zum Beispiel Glanz, diffuse Reflektion und Spiegelungen deaktiviert werden. Wenn Bestandteile eines Objekts Bestandteile aufweisen, die teil-

weise oder vollständig transparent sind, muss diese Eigenschaft auch beibehalten werden, wenn sie als verdeckendes Objekt eingesetzt werden.

Damit optisch der Eindruck entsteht, dass ein Teil des Objekts verdeckt wird, ist theoretisch auch die Visualisierung von Objektumrissen in der Farbe des Hintergrunds möglich. Hierbei ist jedoch zu beachten, dass sich Position und Orientierung des Betrachters ständig ändern können, was zu einer Änderung des Umrisses des verdeckenden Objekts führen kann. Für diesen Fall ist für jede erdenkliche Position oder Blickrichtung des Betrachters eine individuelle Silhouettendarstellung des Gebäudes erforderlich. Denkbar wäre diese Arbeitsweise nur, wenn es sich um ein Objekt handelt, welches aus allen Blickrichtungen die gleiche äußere Kontur hat, wie beispielsweise eine Säule oder ein Turm. Dabei kann eine Silhouette für jede auftretende Position und Orientierung des Betrachters verwendet werden. Größenunterschiede, die sich durch verschiedene Abstände ergeben, können durch Skalierung der Objektkontur angezeigt werden. Für das Objektverdeckungsmodul, das in dieser Arbeit entwickelt wird, kommt diese Lösungsmöglichkeit, die auf der alleinigen Objektdarstellung durch eine Silhouette basiert, nicht in Frage. Es gibt nur wenige Objekte, die achsensymmetrisch aufgebaut sind.

Die mögliche häufige Positions- und Orientierungsänderung des Betrachters führt zu der nächsten Anforderung an die Objektgeometrien. Sie ist auch auf den Anforderungen begründet, die sich grundsätzlich in jedem AR-System an die verwendeten Geometrien ergeben. Um ein virtuelles Objekt in eine reale Welt zu integrieren, ist der Einsatz von wirklichkeitsnahen, der realen Welt entsprechend positionierten Modellen notwendig [Mali02], [Azum97]. Dieser Leitsatz gilt auch bei der Objektverdeckung, denn für die möglichst genaue Darstellung einer Objektverdeckung sind ebenfalls präzise Modelle der realen Objekte, sowie ihre exakte Positionierung und Orientierung im Raum notwendig. Abhängig von der Gestaltung und den Eigenschaften der die realen Objekte repräsentierenden Modelle in der virtuellen Welt, kann die Visualisierung der Objektverdeckung für das Auge mehr oder weniger überzeugend ausfallen.

Da in dieser Arbeit mit dem modellbasierten Ansatz gearbeitet wird, können nur solche realen Objekte in der AR-Szene dargestellt werden, von denen bereits ein gutes

Modell zur Verfügung steht. Beim Einsatz des Systems in urbaner Umgebung, ist das Vorhandensein eines kompletten Stadt- oder Landschaftsmodells, bestehend aus vielen einzelnen Geometrien, besonders von Nutzen. Je mehr Modelle von realen Objekten zur Verfügung stehen, umso mehr Verdeckungen können berücksichtigt werden. Für die eigentliche Berechnung der visuellen Verdeckung ist es außerdem von Vorteil, wenn die Position der virtuellen Objekte zur Verfügung steht. Hierdurch kann die Menge der Objekte eingegrenzt werden, die in Frage kommen, eine Verdeckung zu verursachen.

Bis zu welchem Grad an Genauigkeit ein Objekt modelliert sein muss, hängt zusätzlich vom Auflösungsvermögen des menschlichen Auges ab. Diese Begrenzung durch das menschliche Sehvermögen führt zu der Möglichkeit, Objekte abhängig von ihrer Entfernung zum Betrachter zu vereinfachen, das heißt, Objekte mit zunehmendem Abstand zum Betrachter vereinfacht darzustellen. Durch den Einsatz von Geometrien, die so detailliert wie nötig aber vereinfacht wie möglich dargestellt werden, wird zudem der Rechenaufwand des AR-Systems gemindert. Dies führt zu einer besseren Performanz, was nicht zuletzt die Anforderung einer jeden graphischen Anwendung ist.

4.6 Zusammenfassung der Anforderungen

An einer Objektverdeckung sind reale und virtuelle Objekte beteiligt. Reale Objekte sind Bestandteile der realen Welt. Sie spielen bei der Objektverdeckung dann eine Rolle, wenn räumlich gesehen hinter ihnen ein virtuelles Objekt dargestellt werden soll. Um diese Verdeckung nach dem modellbasierten Ansatz zu visualisieren, ist die Repräsentation des verdeckenden, realen Objektes in der virtuellen Welt als virtuelles Objekt erforderlich. Das Verhalten dieses virtuellen Objekts soll dem Verhalten des realen Objekts dabei möglichst ähnlich sein.

Grundsätzlich ist bei der Berechnung und Visualisierung der Objektverdeckung die Kenntnis der aktuellen Position, Lage und Form der realen Objekte notwendig. Auch muss ein virtuelles Modell des betreffenden Objekts existieren. Dabei gibt es sowohl Objekte, die sich weder bewegen noch ihre Form verändern können, als auch wel-

che, die sich dynamisch verhalten. Bei beiden besteht die Anforderung, dass die aktuelle Position zu jeder Zeit bekannt sein muss. Generell kann also jede Art von realen Objekten bei der Berechnung und Darstellung einer Objektverdeckung repräsentiert werden, solange die erforderlichen Informationen verfügbar sind.

Virtuelle Objekte entsprechen der dreidimensionalen, graphischen Repräsentation von realen Objekten. Das Verhalten der virtuellen Objekte und die Art ihrer Darstellung richten sich dabei nach den Eigenschaften, die durch die realen Objekte vorgegeben werden. Soll ein virtuelles Objekt jedoch nach Vorgabe eines realen Objektes positioniert werden, nach dem sich das Verhalten des virtuellen Objektes zu richten hat, werden entsprechende Informationen zur momentanen Position benötigt. Für die Umsetzung der Objektverdeckung in dieser Arbeit wird jedoch davon ausgegangen, dass diese zur Verfügung steht, da sie grundlegend wichtig für die eigentliche Verdeckungsberechnung ist.

Im Abschnitt über die Zusammensetzung des Sichtbereichs wurde beschrieben, von welchen Umständen die Berechnung und Visualisierung einer Objektverdeckung abhängig sind. Dabei wurde genauer auf einzelne Parameter eingegangen und analysiert, welchen Einfluss sie auf die Berechnung eines verdeckenden Objektes haben können. Maßgeblich ist hierbei der Einfluss dieser Parameter auf die virtuelle Kamera, da diese letztlich die AR-Szene abbildet. Stehen diese Parameter, wie Position und Orientierung zur Verfügung, kann die Objektverdeckung berechnet und damit auch dargestellt werden.

Es ist zu beachten, dass sich die Parameter aufgrund des Anspruches der Echtzeit-Anwendung ständig ändern können und dabei ständig im AR-System aktualisiert werden müssen. Zu den wichtigsten Parametern zählen hierbei die Position und Orientierung des Betrachters beziehungsweise der virtuellen Kamera. Des Weiteren sind auch der Blickwinkel und der Abstand des Betrachters beziehungsweise der Kamera zu den abzubildenden Objekten für die Bilddarstellung ausschlaggebend. Bei der Berechnung und Darstellung einer Objektverdeckung in einer AR-Szene sind alle vier Parameter zu berücksichtigen. Denn schon durch die Änderung eines der Parameterwerte, kann die Repräsentation eines realen Objektes in der virtuellen Szene aufgrund einer Objektverdeckung erforderlich oder überflüssig werden.

In einem letzten Abschnitt wurde beschrieben, welche Anforderungen sich an die Darstellung der Geometrien stellen lassen, die aufgrund des verwendeten modellbasierten Ansatzes die verdeckenden Objekte in der AR-Szene repräsentieren. Voraussetzung ist zum einen, dass ausreichend Modelle von realen Objekten zur Verfügung stehen. Des Weiteren müssen verdeckende Objekte ohne die Verwendung von Materialien, Texturen oder Farben abgebildet werden. Stattdessen müssen sie in der Farbe des Hintergrunds der virtuellen Szene dargestellt werden, was in den meisten AR-Systemen der Farbe schwarz entspricht. Zusätzlich ist zu beachten, dass die Oberflächen der Objekte keine Eigenschaften wie Glanz, Reflektion oder Spiegelungen besitzen, damit das Objekt vollständig unsichtbar erscheint.

Aufbauend auf den in diesem Kapitel analysierten Anforderungen, wird im folgenden Kapitel ein Konzept zur Berechnung und Visualisierung der Objektverdeckung vorgestellt.

5 Konzeption der Objektverdeckungskomponente

5.1 Übersicht

Nachdem im vorhergehenden Kapitel analysiert wurde, unter welchen Bedingungen Objektverdeckungen zwischen realen und virtuellen Objekten auftreten können und welche Anforderungen sich daraus an die Implementierung einer Objektverdeckung in ein AR-System ergeben, wird in diesem Kapitel das Lösungskonzept hierfür vorgestellt. Verschiedene Lösungsansätze, mit deren Hilfe die Problematik der visuellen Objektverdeckung in AR-Systemen gelöst werden kann wurden bereits in Kapitel 3 vorgestellt. Das Konzept dieser Arbeit basiert danach auf dem modellbasierten Ansatz nach Klinker und Breen, es wird jedoch an die speziellen Anforderungen der Aufgabenstellung angepasst (Kapitel 3.3).

5.2 Anforderung und Zielsetzung

Das Kernziel dieser Arbeit ist die Berücksichtigung realer Objekte in einem Outdoor Augmented Reality System, um dadurch Objektverdeckungen zwischen virtuellen und realen Objekten in einer AR-Szene zu visualisieren. Hierzu ist das Konzept des Szenengraphen so zu erweitern, dass reale Objekte in ihm repräsentiert und somit potenzielle visuelle Verdeckungen durch diese Objekte dargestellt werden können. Zusätzlich muss hierfür ein geeigneter Algorithmus entwickelt werden, der die Objektverdeckungen berechnen und darstellen kann.

Für die Steuerung und den Einsatz dieser Objektverdeckungskomponente ist eine Schnittstelle zu implementieren, die als alleiniges Kommunikationsinterface zwischen AR-System und Objektverdeckungskomponente dient. Beim Entwurf dieser Schnittstelle ist zu berücksichtigen, dass sie, und damit die Objektverdeckungskomponente, ohne großen Implementierungsaufwand in möglichst vielen auf Java 3D basierten, echtzeitfähigen AR-Systemen eingesetzt werden kann.

Neben der Objektverdeckungskomponente ist eine Testumgebung zur Visualisierung und Demonstration der Objektverdeckung in Form einer Benutzeroberfläche zu entwickeln und die Objektverdeckungskomponente darin zu integrieren. Die Entwicklung und der Aufbau dieser graphischen Benuteroberfläche wird in Kapitel 7 beschrieben. Die Schwierigkeit bei der Erstellung des Gesamtsystems besteht darin, die Objektverdeckung vom Rest des Systems zu kapseln und dabei den vollständigen Funktionsumfang nutzen zu können. Die Benutzeroberfläche basiert dabei auf Swing und verwendet für die graphische Visualisierung der 3D-Welt und der Objektverdeckung Java 3D. Die Grundlage für die 3D-Geometrien bilden Vrml-Modelle, welche durch einen Vrml-Loader in den Java 3D Szenengraphen eingebunden werden. In ihrem Kern besteht die Testumgebung aus drei separaten Komponenten: dem Szenengraph, der Objektverdeckung und ihrer Behandlung sowie der Benutzeroberfläche selbst.

5.3 Konzeptinhalte

5.3.1 Übersicht

Die zu entwickelnde Objektverdeckungskomponente und die Benutzeroberfläche bilden eine komplexe Kombination von Funktionalitäten aus verschiedenen Disziplinen. Um eine saubere Trennung der benötigten Komponenten zu gewährleisten, müssen die einzelnen Aufgaben und Zuständigkeiten klar abgegrenzt und die Kommunikation der Komponenten untereinander möglichst strukturiert abgebildet werden. Zu diesem Zweck wird für die Komponente der Objektverdeckungsbehandlung eine Klasse entworfen, innerhalb welcher jeder Zugriff auf diese Komponente gekapselt wird. Alle Klassen aus anderen Komponenten, die auf die Objektverdeckungskomponente zugreifen wollen, können dies nur über diese Schnittstelle tun. Die Kernkomponenten des Systems sind:

- Verwaltung der 3D-Komponenten
- Berechnung und Visualisierung der Objektverdeckung
- Aufbau der Benutzeroberfläche

Zusätzlich werden für die Implementierung noch Hilfsklassen und –komponenten benötigt, die aber nicht zur eigentlichen Programmlogik zu zählen sind. Die Beziehung zwischen den oben aufgeführten Komponenten stellt Abbildung 5.1 schematisch dar. Hierbei ist jedoch zu beachten, dass es sich lediglich um einen stark vereinfachten Überblick handelt, der zur ersten Orientierung dienen soll.

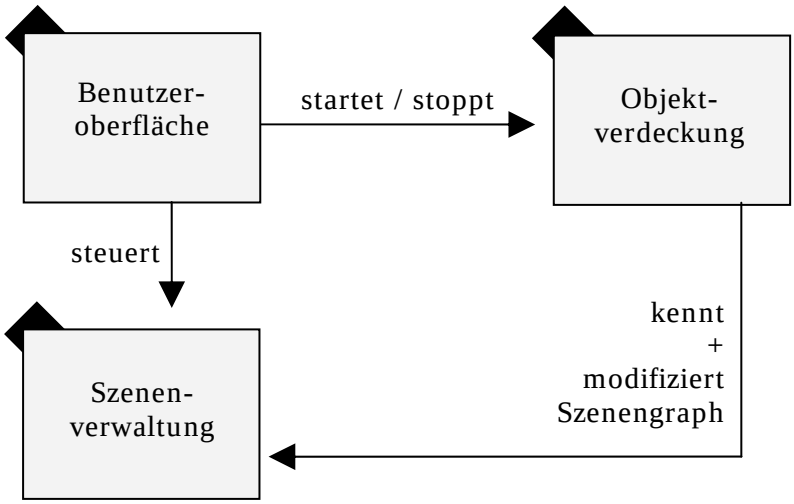


Abbildung 5.1: Schematischer Aufbau der Beziehungen der Komponenten untereinander

5.3.2 Verwaltung der 3D-Komponenten

Eine virtuelle Welt stellt einen künstlichen, dreidimensionalen Raum dar. Dieser Raum hat theoretisch eine unendliche Ausdehnung und besteht aus einer großen Anzahl von Objekten. Das heißt, die virtuelle Welt kann aus einer Vielzahl von unterschiedlichen Daten bestehen, wobei sich diese zur Laufzeit des AR-Systems unvorhersehbar ändern können. Folglich ist eine logische Verwaltung der Daten notwendig, in der alle Änderungen in der virtuellen Welt erfasst werden können.

Alle Daten und Objekte müssen in einem 3D-System darstellbar sein, aber weiter als Einzelteile über eine definierte Referenz oder eine andere Möglichkeit des Zugriffs erreichbar und modifizierbar bleiben, damit auch von außen Änderungen an der Welt vorgenommen werden können. Um dies zu erreichen, muss eine Datenstruktur eingesetzt werden, die diese Zugriffe von außen erlaubt. Es müssen Schnittstellen zu

der virtuellen Welt zur Verfügung stehen, mit der diese modifiziert werden kann. Über diese Schnittstelle sollen zum Beispiel Objekte hinzugefügt, gelöscht oder transformiert werden können.

In Java 3D wird die Szenengraphstruktur zur Verwaltung der dreidimensionalen, virtuellen Welt verwendet. Um aber auch hier die Modifikation von einzelnen Objekten zu gewährleisten und einfach zu halten, sollten weiterhin Referenzen auf diese Objekte in einer anderen Struktur gehalten werden. Es bietet sich eine Art Referenzliste der Objekte einer Szene an, damit bei Bedarf das betreffende Objekt schnell gefunden und sein Status geändert werden kann. Die Aufgabe der Referenzliste ist die Speicherung und Verwaltung der dreidimensionalen Objekte. Über Schnittstellen sollen auf diese zugegriffen werden können. Eine schematische Darstellung dazu bietet Abbildung 5.2.

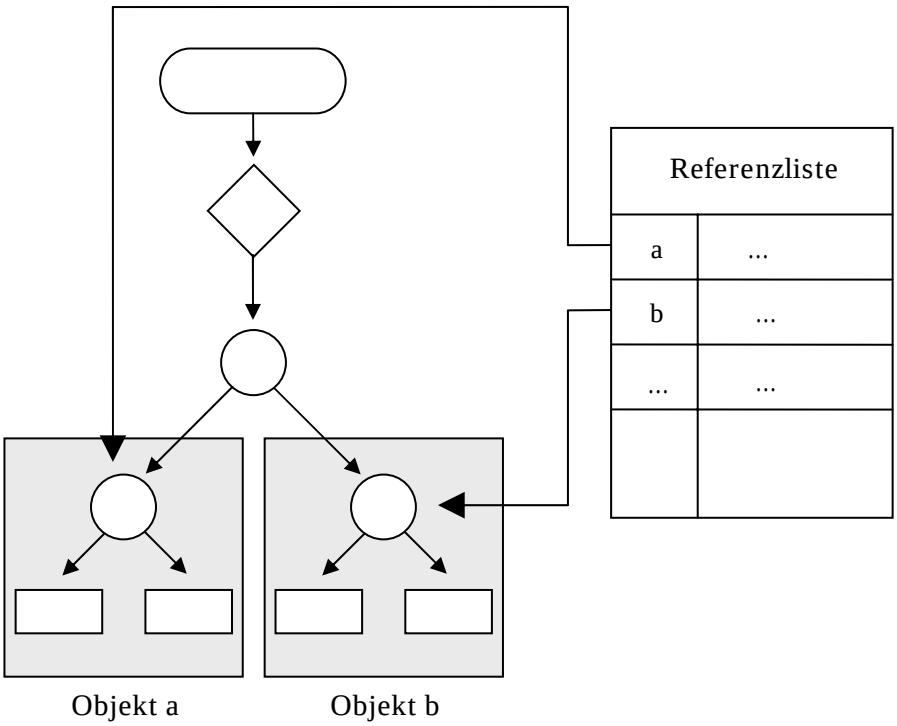


Abbildung 5.2: Schematische Darstellung der Szenengraphstruktur und der Speicherung ihrer Objekte in einer Referenzliste

Weitere wichtige Klassen der Szenenverwaltung sind für die Darstellung der Objekte sowie für den Zugriff auf verschiedene Bestandteile des Szenengraphen vorgesehen. Die Klasse, welche die Objekte repräsentiert, ist für verschiedene Eigenschaften der

Objekte gedacht. Sie bietet gleichermaßen Schnittstellen, über die auf diese Eigenschaften zugegriffen werden kann. Ähnlich ist es mit den Klassen, die den Szenengraphen darstellen. Sie werden benötigt, um beispielsweise die virtuelle Kamera zu steuern. Auch hierfür müssen geeignete Schnittstellen geschaffen werden.

5.3.3 Behandlung der Objektverdeckung

In der realen Welt verdecken Objekte, die vor einem anderen Objekt liegen, immer das dahinter liegende Objekt. In einer allein virtuellen Welt ist das ebenfalls so, doch muss hier bereits mit technischen Mitteln nachgeholfen werden, um eine virtuelle Welt richtig abzubilden. In der Augmented Reality werden virtuelle Objekte jedoch immer vor den Objekten der realen Welt dargestellt, unabhängig von ihrer wahren Anordnung im dreidimensionalen Raum. Um ein an der Natur angelehntes Verdeckungsverhalten der realen und virtuellen Objekte untereinander darzustellen, müssen alle Objekte im Raum, sowohl reale als auch virtuelle, bekannt sein. Dazu gehören die Kenntnis ihrer aktuellen Position und Orientierung sowie die Möglichkeit, auf die Objekte über eine Referenz zugreifen zu können. Zusätzlich sollte die Verwaltung dieser Objekte getrennt nach virtuellen und realen Objekten erfolgen. Da die Verdeckung unterschiedlicher Objekte auch vom Standpunkt des Betrachters abhängt, wird auch hier der Zugriff auf entsprechende Daten wie Position und Orientierung des Benutzers beziehungsweise der virtuellen Kamera benötigt.

Für die Verwaltung all dieser Daten muss eine Schnittstelle innerhalb der Objektverdeckungskomponente geschaffen werden. Diese Klasse verwaltet die gesamten Daten, die für die Simulation der Objektverdeckung benötigt werden. Hierzu gehören unter anderem Strukturen, die verschiedene Objektmengen verwalten. Um berechnen zu können, ob Verdeckungen auftreten, muss eine dieser Strukturen die Objekte enthalten, die theoretisch verdecken können. Das sind alle realen Objekte, die in Frage kommen und mit ihrer Position bekannt sind. Eine andere Struktur muss die Objekte verwalten, die eventuell verdeckt werden können. Das entspricht denjenigen Objekten, die sich im Szenengraphen befinden. Obwohl in der Szenenverwaltungskomponente bereits eine derartige Struktur besteht, muss in der Objektverdeckungskomponente ebenfalls eine solche Struktur gehalten werden. Das ist notwen-

dig, da die Objektverdeckungskomponente auch in anderen Anwendungen einsetzbar sein soll und deshalb unabhängig von anderen Komponenten arbeiten muss. Weitere Strukturen sind zur Speicherung der verdeckenden Objekte erforderlich. Sinnvoll ist hierbei, dass alle Strukturen neben den Objekten zusätzlich die Position dieser Objekte enthalten, da diese für die Verdeckungsberechnung erforderlich ist.

Die interne Schnittstelle, welche die Daten verwaltet und speichert, speichert zusätzlich eine Referenz auf den verwendeten Szenengraphen in der AR-Anwendung. Sie ist erforderlich, um gegebenenfalls Modifikationen am Szenengraphen vornehmen und damit eine visuelle Verdeckung darstellen zu können. Außerdem liefert sie aktuelle Daten zu den Objekten, die sich momentan im Szenengraphen befinden.

Daneben sind die Parameter der virtuellen Kamera der AR-Szene in der Objektverdeckungskomponente erforderlich. Der Transformationsknoten der virtuellen Kamera enthält diese Werte, weswegen die Speicherung einer Referenz auf diesen Knoten in der internen Datenschnittstelle sinnvoll ist. Steht diese Referenz nicht zur Verfügung, ist keine Berechnung einer Objektverdeckung möglich.

Da die Objektverdeckungskomponente das umgebende System aufgrund der Kapselung von außen nicht kennt, kann sie auf diese benötigten Referenzen nicht selbst zugreifen. Um trotzdem die Kommunikation zwischen der Objektverdeckungskomponente und dem umgebenden System zu ermöglichen, wird eine Schnittstelle benötigt. Diese Klasse muss die Möglichkeit bieten, die Verdeckungskomponente einmalig mit den erforderlichen Referenzen zu initialisieren. Daneben ist es erwünscht, die Objektverdeckungsberechnung je nach Bedarf starten und anhalten zu können. Außer dieser Schnittstelle soll für das System, das die Objektverdeckung implementiert, kein Bestandteil der Objektverdeckungskomponente sichtbar sein.

Die Berechnung und Ermittlung einer Objektverdeckung selbst soll dabei ebenfalls in eigenen Komponenten innerhalb der Objektverdeckungskomponente gekapselt werden. In diesen Klassen findet die eigentliche Berechnung und Darstellung der Objektverdeckung statt. Abbildung 5.3 stellt die Beziehungen der internen Arbeitsblöcke in der Objektverdeckungskomponente dar.

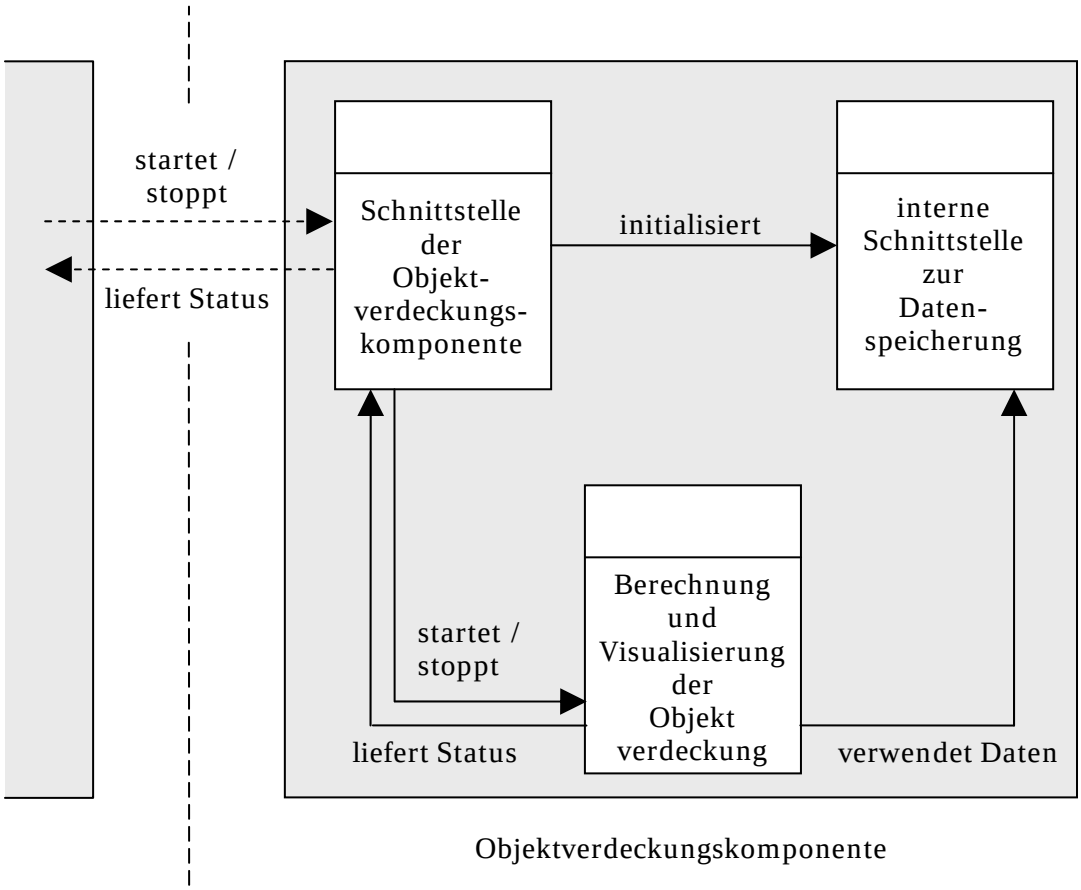


Abbildung 5.3: Schematischer, interner Aufbau der Komponenten innerhalb der Objektverdeckungskomponente

Die Berechnung der Objektverdeckung erfolgt nach einem Algorithmus, der sich in mehrere Teilprozesse und Filter einteilen lässt. Es wird eine Einteilung des Berechnungsablaufs in drei Hauptprozesse gewählt: einen Erhebungsprozess, einen Filterungsprozess und einen Darstellungsprozess. Die drei Hauptprozesse sollten immer in der gleichen Reihenfolge ablaufen, da sie logisch aufeinander aufbauen.

Im Erhebungsprozess sollen die Strukturen, welche die verschiedenen Objekte enthalten, aktualisiert werden, um korrekte Daten für weitere Berechnungen zur Verfügung zu haben. Im Filterungsprozess findet die eigentliche Herausfilterung der Objekte statt, die virtuelle Objekte verdecken. Er soll wiederum in drei Arbeitsschritte beziehungsweise Filterungsschritte gegliedert werden. Dabei ist es sinnvoll, wenn jeder Arbeitsschritt die Ergebnisse des vorherigen Arbeitsschrittes zur weiteren Berechnung verwendet. Ziel dabei ist das allmähliche, schrittweise verfeinern der Menge von Objekten, die für eine Verdeckung in Frage kommen. Wenn mit einer ge-

filterten Menge von Objekten gerechnet werden kann, verringert sich der Rechenaufwand mit jedem Arbeitschritt. Zweckmäßig sind dabei folgende auf den Ergebnissen der Anforderungsanalyse in Kapitel 4 aufbauende Arbeitsschritte:

- Abstandsfilter:
  Der Abstandsfilter berechnet die Menge der verdeckenden Objekte in Abhängigkeit vom Abstand ihres Mittelpunktes zum Benutzer beziehungsweise zur virtuellen Kamera. Damit wird für einen bestimmten Umkreis um die virtuelle Kamera die Menge der verdeckenden Objekte bestimmt. Objekte, die sich weiter als der Radius dieses Umkreises von der Kamera entfernt befinden, kommen für eine Verdeckung nicht in Frage. Erforderlich für diese Berechnung sind die aktuelle Position der virtuellen Kamera sowie die Positionen der eventuell verdeckenden realen Objekte. Des Weiteren muss die Entfernung definiert werden, in welcher eine Berechnung der Objekte sinnvoll ist.

- Sichtkegelfilter:
  Der Sichtkegelfilter berechnet verdeckende Objekte in Abhängigkeit vom Sichtfeld des Benutzers beziehungsweise der virtuellen Kamera. Festgelegt ist das Sichtfeld durch die aktuellen Werte von Position und Orientierung der virtuellen Kamera sowie durch den Blickwinkel. Damit kann berechnet werden, welche der realen Objekte sich im Bereich des Sichtfelds befinden und damit eventuell virtuelle Objekte verdecken.

- Verdeckungsfilter:
  Der Verdeckungsfilter prüft, ob sich reale Objekte tatsächlich näher am Benutzer beziehungsweise der virtuellen Kamera befinden als die virtuellen Objekte, die sie verdecken könnten. Ermittelt wird dies über die Berechnung der Abstände der Kamera zu den realen Objekten und der Abstände der Kamera zu den virtuellen Objekten. Der Abstand ergibt sich jeweils aus der Entfernung des Objektmittelpunktes zur virtuellen Kamera. Nach einem Vergleich dieser Abstände kommen nur die realen Objekte als eventuell verdeckende Objekte in Betracht, die sich näher als andere virtuelle Objekte an der Kamera befinden.

Die Reihenfolge dieser Filter soll dabei beliebig steuerbar sein. Auch soll es möglich sein, einen Arbeitschritt mehrmals auszuführen. Das gibt dem Anwender der Verdeckungsberechnungskomponente die Möglichkeit, eigene Schwerpunkte bei der Berechnung zu setzen und damit an seine AR-Anwendung anzupassen.

Ist der Filterungsprozess vollständig abgelaufen, wird der Darstellungsprozess abgearbeitet. Dieser soll die Aufgabe übernehmen, die Verdeckung an sich darzustellen. Dafür müssen verdeckende Objekte in den Szenengraph eingefügt und andere, nicht mehr verdeckende Objekte aus dem Szenengraph entfernt werden. Abbildung 5.4 zeigt den gesamten schematischen Ablauf der einzelnen Prozesse und Arbeitsschritte.

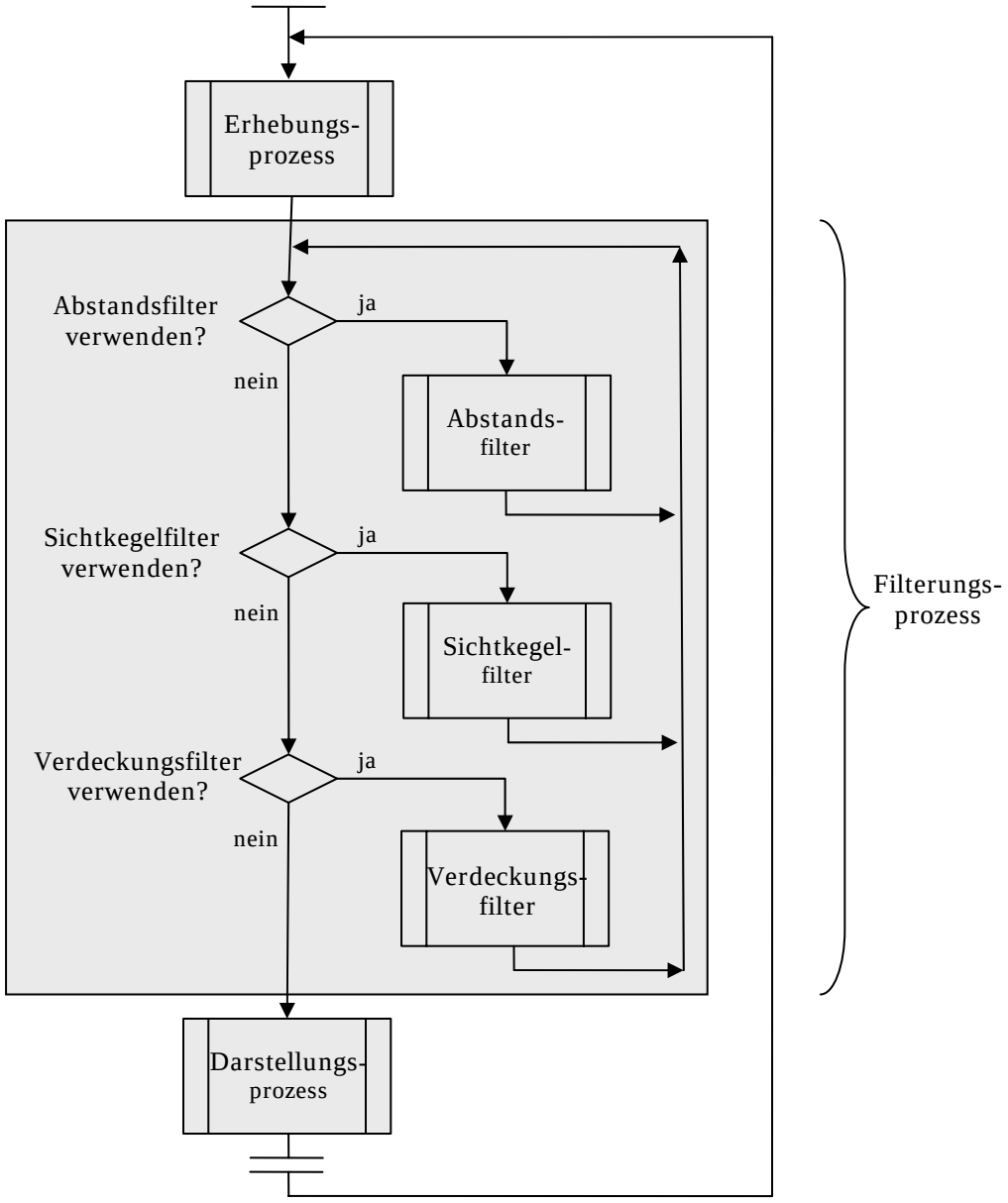


Abbildung 5.4: Schematische Darstellung des Prozessablaufs und der Arbeitsschritte

Da dieses Konzept auf dem modellbasierten Ansatz nach Klinker und Breen aufbaut, müssen verdeckende Objekte im Szenengraph repräsentiert werden, um die Objektverdeckung zu visualisieren. Hierfür wird eine weitere Komponente innerhalb der Objektverdeckungskomponente benötigt, die für die Visualisierung dieser verdeckenden Objekte zuständig ist. Bei der Darstellung der Objekte ist zu beachten, dass sie komplett in der Farbe des Hintergrunds der virtuellen Szene dargestellt werden. In dieser Anwendung wird schwarz als Hintergrundfarbe gewählt, das heißt, auch

die verdeckenden Objekte müssen schwarz dargestellt werden. Konkret müssen den Objekten Materialeigenschaften zugewiesen werden, die diese schwarze Darstellung ermöglichen. Dabei muss berücksichtigt werden, dass die Objektoberflächen keinerlei sonstige Materialeigenschaften wie Glanz, Reflexion oder Spiegelungen aufweisen. Ansonsten sind die Objekte trotz der Schwarzfärbung vor dem Hintergrund sichtbar.

5.4 Zusammenfassung

Das Konzept lässt sich in drei Arbeitspakete einteilen. Dabei entspricht eines der Pakete der Benutzeroberfläche, ein zweites dem Aufbau der dreidimensionalen Szene und ein drittes der eigentlichen Objektverdeckungsberechnung und ihrer Darstellung. Die Benutzeroberfläche dient dabei zur Demonstration und Visualisierung der Objektverdeckung. Sie implementiert die Szenenverwaltung in Form eines Szenengraphen und die Objektverdeckungskomponente. Diese Komponente ist so aufgebaut, dass sie vom restlichen System vollkommen gekapselt wird. Lediglich eine Schnittstelle steht zur Verfügung, über welche diese Komponente auch in weiteren auf Java 3D basierten Systemen implementiert werden kann. Die Objektverdeckungskomponente enthält Komponenten zur Berechnung und Darstellung der Objektverdeckung, sowie zur internen Speicherung der Verdeckungen. Die Implementierung der Objektverdeckungskomponente und die Umsetzung der Algorithmen werden in Kapitel 6 beschrieben. Die Entwicklung der Benutzeroberfläche beziehungsweise der Testumgebung wird in Kapitel 7 erläutert.

6 Entwicklung der Objektverdeckungskomponente

6.1 Übersicht

In diesem Kapitel wird beschrieben, wie die theoretischen Lösungsansätze des vorhergehenden Kapitels umgesetzt werden. Dabei wird auf die bereits erwähnten Betrachtungen jedoch konkreter eingegangen. Die Gliederung dieses Kapitels richtet sich dabei nach der bereits vorgestellten Komponentenstruktur, die im Übrigen auch der Paketstruktur der tatsächlichen prototypischen Realisierung entspricht. Die Entwicklung der als Testumgebung verwendeten, graphischen Benutzeroberfläche wird in Kapitel 7 beschrieben[4].

6.2 Architektur und Paketstruktur

Die Klassen, die in dieser Arbeit entwickelt wurden, gehören jeweils ihrer Funktion entsprechend zu unterschiedlichen Paketen. Die Pakete spiegeln dabei die Struktur und Aufgaben der Komponenten aus der Konzeption wider. Ein Paket enthält die Verdeckungsberechnung und ihre Darstellung, ein weiteres enthält die Klassen, die für die Benutzeroberfläche erforderlich sind. In einem dritten Paket sind die Klassen zur Szenengraphverwaltung enthalten. Weitere benötigte Klassen umfasst ein viertes Paket. Eine solche Einteilung unterstützt die Programmlogik sowie die objektorientierte Arbeitsweise von Java. Zusätzlich wurde bei der Entwicklung darauf geachtet, dass die Klassen wieder verwendbar und flexibel einsetzbar sind. In Abbildung 6.1 wird die Struktur der Pakete untereinander dargestellt. Deutlich wird hieraus, dass jedes Paket für sich einen Aufgabenbereich einschließt und von der Benutzeroberfläche verwendet wird. Abschließend folgt eine genauere Beschreibung der Pakete.

[4] Für die programmiertechnische Umsetzung der Objektverdeckungskomponente und Benutzeroberfläche wurde mehrere Literaturquellen zur Programmiersprache Java und ihrer Erweiterung Java 3D verwendet: [Lutz00], [Goll00], [Krue01], [Selm02].[Goll00]

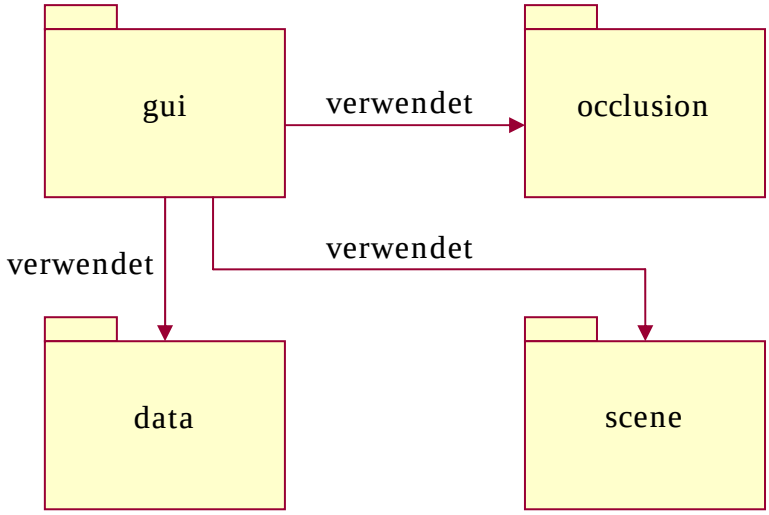


Abbildung 6.1: Schematische Darstellung der implementierten Paketstruktur

- Paket „gui":
  Das Paket gui enthält, wie der Name bereits sagt, alle Klassen, die zur Implementierung und Darstellung der Benutzeroberfläche benötigt werden. Die einzelnen Klassen basieren hierbei auf Swing-Komponenten und sind insgesamt als MDI aufgebaut.[5] Die internen Fenster bieten dabei die Funktionalitäten, die zum Testen der Objektverdeckungskomponente benötigt werden (Kapitel 7.1).

- Paket „occlusion“:
  Das Paket occlusion enthält sämtliche Klassen, die zur Berechnung und Darstellung einer Objektverdeckung zwischen realen und virtuellen Objekten benötigt werden. Bei der Implementierung dieses Pakets wurde darauf geachtet, dass nur eine Klasse den Zugriff auf dieses Paket von außerhalb erlaubt. Die Möglichkeit der Wiederverwendung der Objektverdeckungsbehandlung wird durch einfache Implementierung dieses Pakets in beliebige Java 3D Anwendungen unterstützt.

- Paket „scene“:
  Im Paket scene sind alle Klassen enthalten, die beim Aufbau und bei der Verwaltung der dreidimensionalen Szene eine Rolle spielen. Grundlage für diese

[5] MDI, Multiple Document Interface

Klassen ist das Konzept der Szenengraphstruktur. Das Paket wird in erster Linie vom Paket gui verwendet und dient damit indirekt zur Visualisierung der Objektverdeckung.

- Paket „data“:
  Das Paket data enthält Hilfsklassen, die keinem bestimmten Paket zugeordnet werden können. Sie dienen allein zur Unterstützung verschiedener Funktionalitäten anderer Pakete.

In den folgenden Abschnitten wird beschrieben, wie diese Pakete intern aufgebaut sind und welche Klassen sie enthalten.

6.3 Aufbau der Szenengraphstruktur

6.3.1 Die Klassen des Pakets „scene“

Das Paket scene enthält die Klassen, die zum Aufbau und zur Verwaltung der dreidimensionalen Szene erforderlich sind. Hierfür wird die Szenengraphstruktur von Java 3D verwendet. Um den Szenengraphen, der den Zustand der 3D-Szene speichert, zu erzeugen werden verschiedene Klassen eingesetzt. Des Weiteren sind Schnittstellen in Form von Methoden implementiert, welche die Modifikation des Szenengraphen erlauben. Abbildung 6.2 stellt die in dem Paket enthaltenen Klassen schematisch dar.

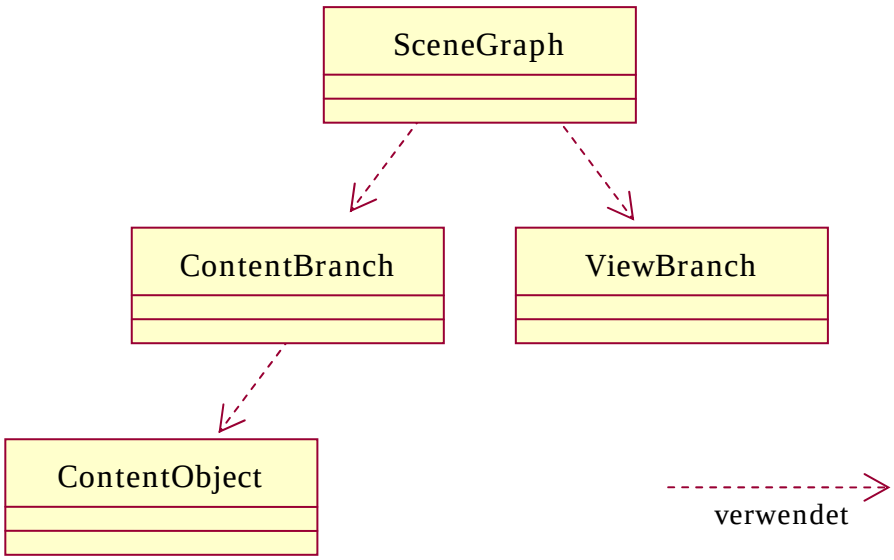


Abbildung 6.2: Schematische Darstellung der Klassen im Paket scene

Mit Hilfe der Klassen `ContentBranch` und `ViewBranch` sowie Objekten der Klassen `VirtualUniverse` und `Locale` erzeugt die Klasse `SceneGraph` einen Standardszenengraphen. Auf diesen Szenengraphen, der in jeder Java 3D Anwendung eingesetzt werden kann, kann nur über eine Zugriffsmethode zugegriffen werden, da die Klasse `SceneGraph` nach dem Singleton-Pattern der objektorientierten Programmierung entwickelt wurde[6]. Dadurch wird erreicht, dass von dieser Klasse nur eine Instanz existiert, indem bei jedem Zugriff auf die Zugriffsmethode `getInstance()` die bereits erzeugte Instanz zurückgegeben wird. Diese Technik gewährleistet, dass alle Klassen innerhalb einer Anwendung auf dem gleichen Szenengraphen arbeiten. Der folgende Programmausschnitt stellt diese Technik noch einmal exemplarisch dar:

```
public class SceneGraph {

        SceneGraph sg = null;

        private SceneGraph(…) {
                // erzeuge ViewBranch
                // erzeuge ContentBranch
        }

        public static SceneGraph getInstance(){
```

[6] Ein Pattern entspricht in der objektorientierten Programmierung einem Entwurfsmuster zum Design einer Klasse.

```
                if ( sg == null ) {
                        sg = new SceneGraph(…);
                }
                return sg;
        }
}
```

Neben der Zugriffsmethode auf die Instanz dieser Klasse bietet `SceneGraph` noch weitere Zugriffsmethoden. Sie liefern bei Bedarf das `VirtualUniverse`-Objekt oder das `Locale`-Objekt des erzeugten Szenengraphen zurück. Zusätzlich bietet die Klasse eine Zugriffsmethode auf das `Canvas3D`-Objekt des implementierten `ViewBranchs`. Das `Canvas3D`-Objekt kann dadurch leicht in eine GUI-Anwendung eingebunden werden.

Die von `SceneGraph` verwendete Klasse `ViewBranch` liefert durch Aufruf ihrer einzigen Zugriffsmethode den `ViewBranch`-Teil eines Java 3D Szenengraphen zurück. Sie ist ebenfalls nach dem Singleton-Pattern aufgebaut. Da dieser `ViewBranch` als `BranchGroup` zurückgegeben wird, kann er in jedem `Locale` eines Szenengraphen verwendet werden. Der `ViewBranch` selbst wird durch eine interne Methode erzeugt und folgt dabei dem Vorbild eines Java 3D Szenengraphen (Kapitel 2.3.4). Hierfür wird eine `BranchGroup` erzeugt, an die eine `TransformGroup` gehängt wird. An diese `TransformGroup` wird ein `ViewPlatform`-Objekt gehängt, das wiederum ein `View`-Objekt enthält. Das `View`-Objekt enthält seinerseits ein `Canvas3D`-Objekt. Zusätzlich wird die Bildwiederholrate auf ein Maß von 25 Bildern pro Sekunde gestellt. Damit in der Szene auch mit der Maus navigiert werden kann, werden Objekte der Java 3D Klassen `MouseZoom`, `MouseRotate` und `MouseTranslate` an den Transformationsknoten der `ViewPlatform` gehängt[7]. Den strukturellen Aufbau des erzeugten ViewBranch-Graphen zeigt Abbildung 6.3:

[7] MouseZoom, MouseRotate und MouseTranslate werden von der Java 3D API bereitgestellt und ermöglichen die bereits fertig implementierte Steuerung einer TransformGroup durch die Maus.

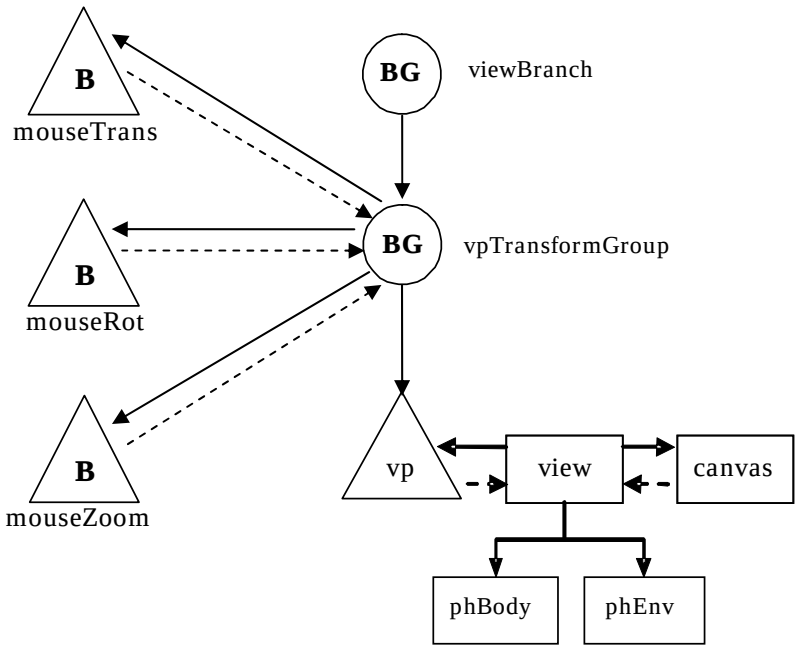


Abbildung 6.3: Schematischer Aufbau des durch die Klasse `ViewBranch` erzeugten ViewBranchs

Damit der `ViewBranch` in eine GUI-Anwendung eingebunden werden kann, existiert eine Zugriffsmethode auf das `Canvas3D`-Objekt des ViewBranches. Um auch auf den Transformationsknoten, der die `ViewPlatform` beziehungsweise die virtuelle Kamera steuert, zugreifen zu können, enthält die Klasse `ViewBranch` eine statische Referenz auf diesen Knoten. Auf diese Referenz kann ebenfalls mittels einer Zugriffsmethode zugegriffen werden. Ferner wurden Methoden implementiert, die einen vereinfachten Zugriff auf die Navigation der `ViewPlatform` ermöglichen. Über die Methoden `setPosition()` und `setOrientation()` kann ihre Position und Orientierung durch Übergabe eines Parameters gesetzt werden. Für die Abfrage dieser Parameter stehen die Zugriffsmethoden `getPosition()` und `getOrientation()` zur Verfügung.

Die Klasse `ContentBranch` erzeugt, wie ihr Name schon sagt, einen ContentBranch für einen Java 3D Szenengraphen. Zu Beginn besteht er lediglich aus einer `BranchGroup`, die jedoch mit beliebigen Java-3D-Objekten gefüllt werden kann. `ContentBranch` stellt hierfür Methoden zur Verfügung, mit denen Objekte in diesen Branch eingefügt oder aus ihm gelöscht werden können. Des Weiteren existiert die Möglichkeit, über eine weitere Methode die Position, an der ein Objekt ein-

gefügt werden soll, mit anzugeben. Konkret wird als Objektformat das Vrml-Format implementiert, das heißt in den `ContentBranch` können vornehmlich Vrml-Objekte eingefügt werden. Die Klasse speichert zusätzlich alle Objekte, die sich momentan im Szenengraphen befinden in einer Struktur. Damit die Objekte im `ContentBranch` auch sichtbar sind, werden mit dem Erzeugen des ContentBranches auch Lichter initialisiert und dem Graphen hinzugefügt.

Die vierte Klasse im Paket scene, die Klasse `ContentObject` repräsentiert Objektgeometrien, die im Java 3D Szenengraphen verwendet werden können. Hierfür verwaltet die Klasse verschiedene Datenfelder und Methoden, um die Objekte und ihre Eigenschaften speichern und modifizieren zu können. Das wichtigste Datenfeld dabei ist die `BranchGroup`, welche die Geometrien enthält. Um auf diese `BranchGroup` zugreifen zu können, existiert die Methode `getBranchGroup()`, welche die `BranchGroup` zurückgibt. Die `ContentObject`-Geometrie kann zusätzlich transliert und skaliert werden. Hierfür stehen die Methoden `setTranslation()` und `setScale()` zur Verfügung. Damit die interne `BranchGroup` im Szenengraphen später leichter wieder auffindbar ist, kann sie über die Methode `setUserData()` mit einem beliebigen Objekt versehen werden. In dieser Arbeit wird hierfür beispielsweise ein String verwendet, der dem eindeutigen Namen des Objekts entspricht.

6.3.2 Aufbau des Szenengraphen

Um die Objektverdeckung mit der später beschriebenen Benutzeroberfläche zu testen, wird in der Benutzeroberfläche ein Szenengraph implementiert. Dieser Szenengraph basiert dabei auf oben vorgestellter Struktur. Da die Objektverdeckung jedoch immer auf bestimmten Ebenen des Szenengraphen nach Objekten sucht, die für eine Objektverdeckung in Frage kommen, ist der `ContentBranch` des in dieser Arbeit verwendeten Szenengraphen feiner strukturiert aufgebaut. Das heißt, die Objekte befinden sich nicht direkt unter dem obersten Knoten des `ContentBranchs`. Stattdessen entsprechen erst die Knoten auf der dritten Ebene den Objekten.

In Abbildung 6.4 wird der konkrete Szenengraph, wie er in dieser Arbeit aufgebaut und verwendet wird, schematisch dargestellt. Er wird durch die oben beschriebenen Klassen erzeugt und enthält dadurch einen Knoten mit Lichtobjekten wie `PointLight` und `AmbientLight`. Auf der dritten Ebene befinden sich die Knoten mit den Objektgeometrien. Der ViewBranch ist aus Gründen der Übersichtlichkeit stark vereinfacht dargestellt. Sein Aufbau entspricht dem der Abbildung 6.4.

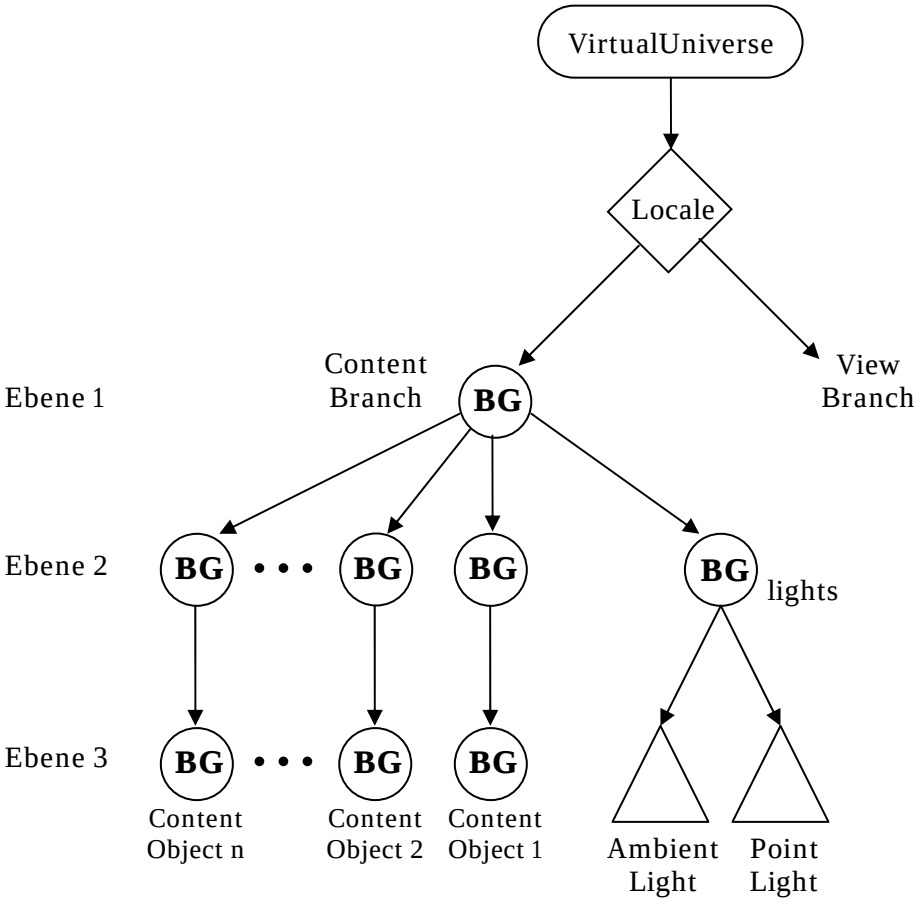


Abbildung 6.4: Schematische Darstellung des in dieser Arbeit verwendeten Szenengraphen.

6.4 Behandlung der Objektverdeckung

6.4.1 Die Klassen des Pakets „occlusion“

Im Paket occlusion sind alle Klassen enthalten, die für eine Objektverdeckungsbehandlung in einer auf Java 3D basierten AR-Anwendung benötigt werden. Dabei ist

das Paket so aufgebaut, dass nur eine einzige Klasse den Zugriff von außen auf dieses Paket ermöglicht. Über diese Schnittstelle wird die Objektverdeckungsbehandlung implementiert und gesteuert. Weitere Klassen dienen zur Datenspeicherung innerhalb des Pakets sowie zur eigentlichen Berechnung und Visualisierung der Objektverdeckung.

Einen Überblick über die Klassenstruktur der im Paket occlusion enthaltenen Klassen gibt Abbildung 6.5. Dabei dient die Klasse OcclusionManager als eben genannte Schnittstelle zu dieser Objektverdeckungskomponente. Über sie wird die Objektverdeckungsbehandlung initialisiert, gestartet und angehalten. Mit ihrer Initialisierung bekommt die Klasse Daten zur Verfügung gestellt, die in den Klassen AbstractOcclusionData und OcclusionData für die Verdeckungsberechnung gespeichert werden. Die Verdeckungsbehandlung selbst wird durch starten und anhalten des Threads OcclusionThread gesteuert. Er verwendet zur eigentlichen Berechnung der verdeckenden Objekte die Klasse OcclusionFilter. In ihr sind die verschiedenen Filter, wie sie auch schon in Kapitel 5.3.3 vorgesehen wurden, implementiert. MaskingObject repräsentiert die Klasse, mit deren Hilfe verdeckende Objekte in der AR-Szene dargestellt werden.

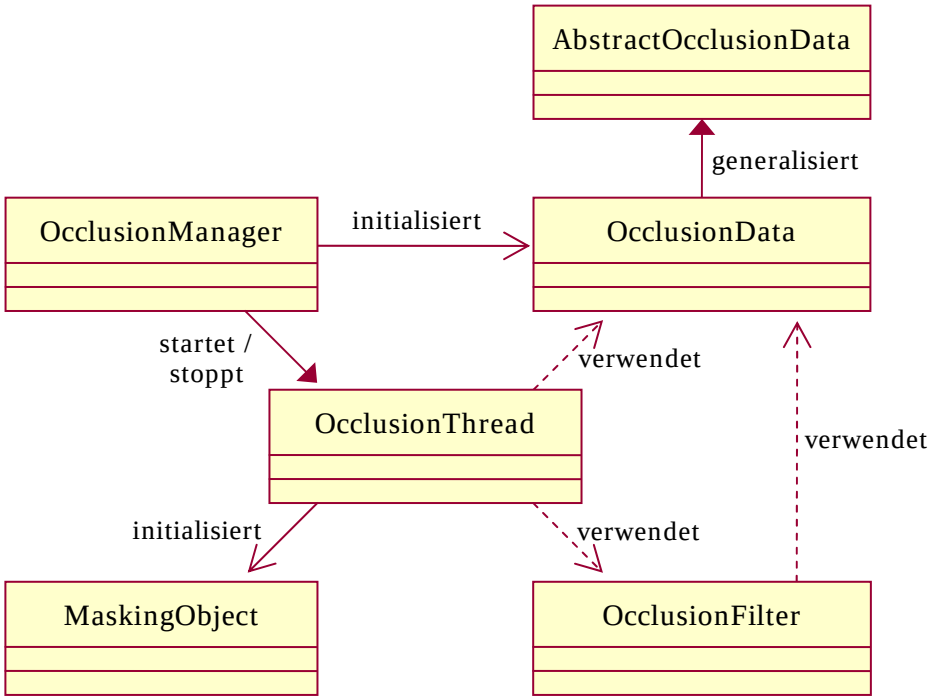


Abbildung 6.5: Schematische Darstellung der Klassen im Paket occlusion

Eine detaillierte Beschreibung der einzelnen Klassen findet in den folgenden Abschnitten statt, wobei dabei einzelne Klassen ihrer Funktion im Paket entsprechend zusammengefasst werden.

6.4.2 Datenhaltung

In diesem Abschnitt werden die Klassen `AbstractOcclusionData` und `OcclusionData` beschrieben. `OcclusionData` ist eine Spezialisierung der Klasse `AbstractOcclusionData`. Diese definiert bereits einen Großteil der Daten und Zugriffsmethoden, die für die Objektverdeckungsbehandlung erforderlich sind. In `OcclusionData` wird der Umfang dieser Funktionalität um weitere benötigte Daten und Methoden ergänzt. Da sie durch Vererbung auch die Attribute und Methoden der Vaterklasse erbt, wird innerhalb der Verdeckungsbehandlung nur mit `OcclusionData` gearbeitet. `AbstractOcclusionData` wird nicht als konkrete Instanz benötigt.

Die Klasse `OcclusionData` ist als Singleton implementiert, um die Bildung mehrer Instanzen dieser Klasse zu vermeiden[8]. Dadurch und durch die statische Deklaration der Datenfelder wird erreicht, dass alle anderen Klassen, die auf diese Daten zugreifen müssen, auf denselben Daten arbeiten. Zusätzlich sind die Datenfelder `private` oder `protected` deklariert, wodurch der lesende und schreibende Zugriff auf diese Daten von außerhalb des occlusion-Pakets verhindert wird. Für die Klassen innerhalb des occlusion-Pakets stehen jeweils lesende und schreibende Zugriffsmethoden auf die Daten zur Verfügung.

Der Übersichtlichkeit wegen werden im Folgenden nur die wichtigsten Daten, die in `OcclusionData` verwaltet werden, aufgelistet und ihre grundlegende Funktion beschrieben. Die meisten der Datenfelder ergeben sich dabei aus der Anforderungsanalyse und der Konzeption (Kapitel 4 und 5). Initialisiert werden die Daten durch die Klasse `OcclusionManager` (Kapitel 6.4.3):

- `protected static TransformGroup viewTransform;`
  Die Objektverdeckungsberechnung erhält durch diese Referenz auf die `TransformGroup` der `ViewPlatform` des Szenengraphen Zugriff auf die aktuellen Parameter wie deren Position und Orientierung.
- `protected static BranchGroup contentBranch;`
  In diesem Attribut wird die Referenz auf den `ContentBranch` des verwendeten Szenengraphen gespeichert. Diese ist erforderlich, um den Szenengraphen nach Objekten zu durchsuchen und eine Objektverdeckung zu visualisieren.
- `protected static int bgLevel;`
  Über dieses Attribut wird die Ebene im `ContentBranch` des Szenengraphen definiert, auf welcher sich Objekte befinden, die bei der Objektverdeckungsberechnung als virtuelle, eventuell verdeckte Objekte berücksichtigt werden sollen.

[8] Für die Funktionsweise des Singleton-Patterns siehe Kapitel 6.3.1

Die nächsten Datenfelder sind `Hashtables`, über die alle Objekte mittels eines Schlüssel-Werte-Paares in einer Struktur gespeichert werden. Über den Schlüssen kann auf die Eigenschaften des entsprechenden Objekts zugegriffen werden. Der Schlüssel entspricht dem eindeutigen Namen des Objekts in Form eines Strings. Verwaltet werden auf diese Weise die Eckpunkte der Boundingbox für jedes sich momentan im Szenengraph befindliche Objekte[9]. Dabei können drei- oder zweidimensionale Koordinaten abgelegt werden. Bei der Verwendung von dreidimensionalen Koordinaten müssen zwei sich gegenüberliegende Ecken angegeben werden. Die zweidimensionalen Koordinaten sollen alle vier Ecken definieren. Für die Berechnung der Objektverdeckung werden nur zweidimensionale Wertepaare benötigt. Die `Hashtables` stellen damit die in Kapitel 5.3.2 beschriebene Referenzstruktur dar. Ihr Aufbau wird von den folgenden `Hashtables` verwendet.

- `protected static Hashtable sgObjects;`
  `sgObjects` speichert alle Objekte, die sich momentan im Szenengraph befinden in oben beschriebener `Hashtable`.

- `protected static Hashtable wrlObjects;`
  Im Gegensatz zu `sgObjects` enthält `wrlObjects` alle Objekte, die für eine Objektverdeckung in Frage kommen. Dies entspricht der Menge aller realen Objekte.

- `private static Hashtable possOcclObjects;`
  `possOcclObjects` enthält die Menge der Objekte, die virtuelle Objekte verdecken können. Das heißt, `possOcclObjects` entspricht der Differenz aller realen Objekte und der Objekte, die sich momentan im Szenengraphen befinden.

- `private static Hashtable occlObjects;`
  Diese Struktur dient zur Zwischenspeicherung der tatsächlich verdeckenden Objekte. Alle Objekte, die in dieser Struktur enthalten sind, müssen im Szenengraph als verdeckendes Objekt repräsentiert sein.

[9] Eine Boundingbox ist ein Quader, der ein Objekt vollständig einhüllt und dessen Kanten dabei parallel zu den Koordinatenachsen liegen.

Neben diesen Strukturen, existieren noch weitere Datenfelder in `OcclusionData`. Über das Bool'sche Attribut `isLive` wird der aktuelle Status der Verdeckungsbehandlung definiert Ein String `methodOrder` definiert, in welcher Reihenfolge welche Filter bei der Verdeckungsberechnung in der Klasse `OcclusionFilter` verwendet werden[10]. Über einen String `pathname` kann angegeben werden, in welchem Verzeichnis sich Informationen befinden, die beschreiben, welche realen Objekt vorhanden sind und in die Objektverdeckungsbehandlung mit einbezogen werden müssen. Diese Informationen können in Form von Textdateien vorliegen, welche die Namen und Boundingbox-Eckpunkte der Objekte enthält. Zusätzlich gibt `pathname` an, wo sich die Objektgeometrien zu den darzustellenden, verdeckenden Objekten befinden. Dabei definiert ein weiterer String `filename` den Namen dieser Textdatei. Weitere Datenfelder definieren den Sichtwinkel oder den Abstand zur `ViewPlatform`, der bei der Berechnung der Objektverdeckung verwendet werden soll.

Für alle Datenfelder stehen lesende und schreibende Zugriffsmethoden zur Verfügung. Sie werden hier nicht einzeln aufgezählt, da sie keine andere Funktion besitzen, als oben genannte Datenfelder mit neuen Werten zu versehen oder eben die Daten selbst zurückzugeben. Weitere Methoden mit größerem Funktionsumfang werden im Folgenden beschrieben:

- `protected void setSGComponents(Object val, int l);`
  Mit Hilfe dieser Methode kann der gesamte Subgraph unterhalb einer `BranchGroup` nach den Objekten durchsucht werden, die sich gerade im Szenengraph befinden und damit verdeckt werden können. Bis zu welcher Ebene diese Suche stattfinden soll, wird über das oben genannte Attribut bgLevel definiert. Da der Szenengraph, der in dieser Arbeit verwendet wird, seine 3D-Geometrien auf der dritten Ebene enthält, durchläuft diese Methode rekursiv alle Knoten, die sich auf erster, zweiter oder dritter Ebene befinden und speichert die Objekte, die sich auf der dritten Ebene befinden, in der Hashtable `sgObjects` ab. Aktiviert wird `setSGComponents()` durch eine weitere Methode `getSGObjects()` (Abbildung 6.6).

[10] Die Verdeckungsberechnung wird inklusive der implementierten Filter in Kapitel 6.4.4 beschrieben.

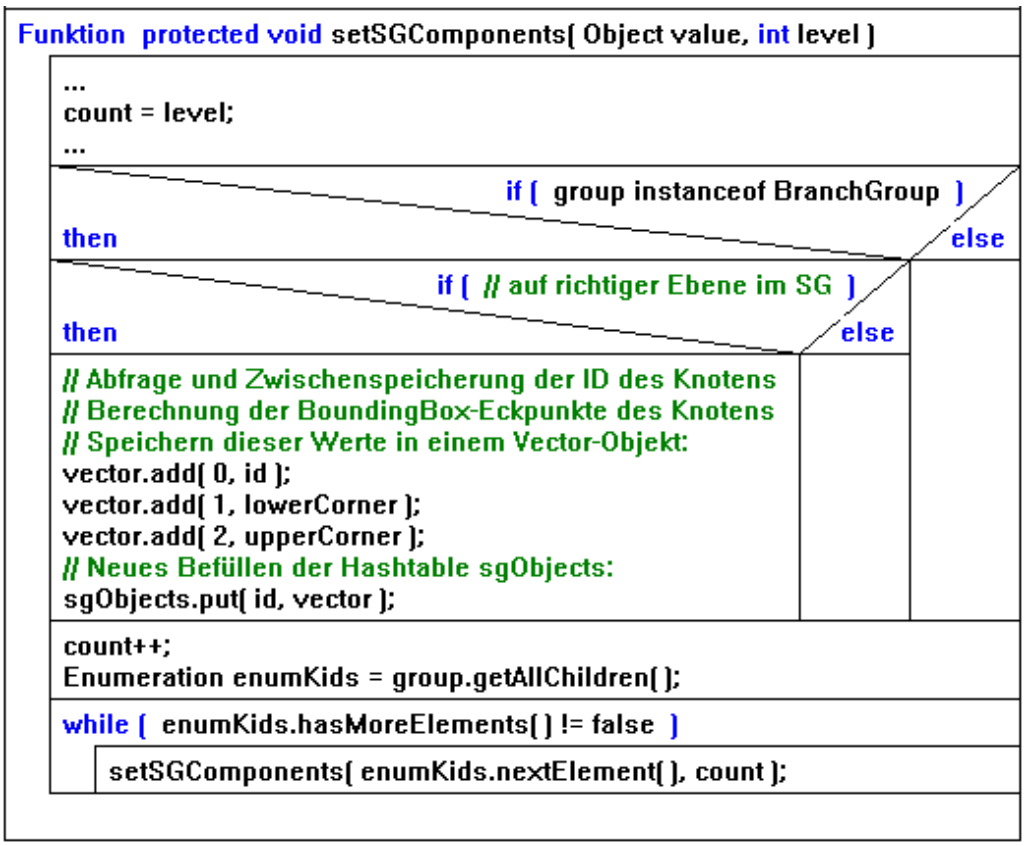


Abbildung 6.6: Struktureller Ablauf der Methode setSGComponents()

- `private void setPossOcclObjects();`

  Über diese Methode wird die Differenz der zwei Strukturen `wrlObjects` und `sgObjects` berechnet und in `possOcclObjects` als Menge der eventuell verdeckenden Objekte abgelegt.

- `private void recursiveSetCapabilities(Object o);`

  Diese Methode durchläuft rekursiv den übergebenen Branch und setzt die Capability der Knoten auf `ALLOW_CHILDREN_READ`[11]. Ist dieses Capability-Bit nicht gesetzt, ist es nicht möglich, den Szenengraphen zu durchsuchen.

- `protected Hashtable setBBoxCorners(Hashtable h);`

  Für die Berechnung der Objektverdeckung werden nur zweidimensionale Werte der Boundingbox-Ecken benötigt. Liegen die Objekte jedoch mit drei-

[11] Java 3D verwendet bei jedem Knoten Capabilty-Bits, durch die ein Zugriff auf bestimmte Eigenschaften des Knotens auch im live-Zustand des Szenengraphen möglich ist.

dimensionalen Werten in einer `Hashtable` vor, kann diese Methode aufgerufen werden. Sie errechnet die zweidimensionalen Wertepaare und legt sie statt der dreidimensionalen Koordinaten in der `Hashtable` ab.

Durch die Implementierung von `OcclusionData` wird eine zentrale Verwaltung der Daten gebildet, die für die Berechnung einer Objektverdeckung benötigt werden. Dabei wird gewährleistet, dass alle die Daten verwendenden Klassen auf denselben Daten arbeiten. Initialisiert werden die Datenfelder durch die Schnittstellenklasse `OcclusionManager`, die im folgenden Abschnitt näher dargestellt wird.

6.4.3 Schnittstelle des Objektverdeckungspakets

Die Klasse `OcclusionManager` stellt die Schnittstelle zum occlusion-Paket dar. Das heißt die Objektverdeckungsbehandlung kann nur über diese Schnittstelle initialisiert und verwendet werden. Der Implementierungsaufwand dabei ist gering. Es muss lediglich eine Instanz von `OcclusionManager` erzeugt werden. Daten, die zur Berechnung der verdeckenden Objekte benötigt werden, müssen bereits bei der Instanzbildung übergeben werden.

Die Daten, die von der Objektverdeckungsbehandlung benötigt werden und damit schon der Klasse `OcclusionManager` übergeben werden müssen, resultieren größtenteils auf den Ergebnissen der Anforderungsanalyse. Um die Anzahl der Parameter flexibel zu halten, sollen alle Daten, die erforderlich sind, in einem Vector-Objekt übergeben werden. Unbedingt erforderlich sind hierbei eine Referenz auf den ContentBranch des Szenengraphen und auf das `Locale`-Objekt sowie eine Referenz auf den Transformationsknoten der `ViewPlatform`. Zusätzlich müssen das Verzeichnis und der Name der Textdatei, die eine Liste der realen Objekte inklusive ihrer Boundingbox-Eckpunkte enthält, in Form von Strings angeben werden. Bei Bedarf kann auch die Ebene des Szenengraphen angegeben werden, auf der sich die Objekte im `ContentBranch` befinden. Intern werden aus diesen im `Vector` enthaltenen Angaben die Daten generiert, die letztendlich für die Objektverdeckungsberechnung notwendig sind. Dazu wird das `Vector`-Objekt nach benötigten Daten-

typen durchsucht. Sind die entsprechenden Referenzen im Vector enthalten, werden sie in den zugehörigen Datenfeldern der Klasse `OcclusionData` gespeichert.

Stehen alle Daten zur Verfügung, erzeugt `OcclusionManager` einmalig eine `BranchGroup`, die allein für die verdeckenden Objekte vorgesehen ist, und hängt sie in den `ContentBranch` des Szenengraphen. Dadurch sind diese Objekte mühelos im Szenengraph auffindbar und können bearbeitet oder gelöscht werden. Zuletzt wird die Objektverdeckungsbehandlung durch Instanzbildung der Klasse `OcclusionThread` initialisiert. Diese Klasse der Verdeckungsbehandlung wartet nun darauf, gestartet zu werden. Hierfür steht die öffentliche Methode `setOcclusionStatus()` in `OcclusionManager` zur Verfügung, die durch Übergabe eines Bool'schen Wertes den Status der Verdeckungsberechnung setzt. Der aktuelle Status kann wiederum über die komplementäre Methode `getOcclusionStatus()` abgerufen werden.

Zusätzlich bietet die Klasse `OcclusionManager` auch Methoden, mit denen sich die Berechnung der verdeckenden Objekte teilweise an die Ansprüche des Anwenders anpassen lässt. Diese Methoden sind zu diesem Zweck als öffentlich deklariert und werden nachfolgend beschrieben.

- `public void changeMethodOrder(String order);`
  Durch den String, der hier übergeben wird, kann beeinflusst werden, welche Filter in welcher Reihenfolge bei der Objektverdeckungsberechnung eingesetzt werden. Der String muss dafür die entsprechenden Filternamen enthalten. Hierfür können je nach erwünschter Filterart die Worte ABSTAND, SICHTKEGEL oder VERDECKUNG verwendet werden.[12]

- `public void changeRadius(int r);`
  Mit dieser Methode kann der Anwender der Objektverdeckungsbehandlung festlegen, in welchem Abstand zur `ViewPlatform` Objektverdeckungen berechnet werden sollen. Voreingestellt ist ein Abstand von 100 Metern.

- `public void changeViewAngle(double rad);`
  Über diese Methode kann der Anwender definieren, in welchem Sichtwinkel

[12] Näheres zur Struktur und Funktionsweise der einzelnen Filter siehe Kapitel 6.4.4

der `ViewPlatform` Objektverdeckungen berechnet werden sollen. Wird diese Methode nicht aufgerufen, wird ein Blickwinkel von 180° verwendet.

Ergänzend stehen zu diesen Methoden stehen auch Zugriffsmethoden zur Abfrage der genannten Parameter zur Verfügung. Wofür die einzelnen Parameter genau verwendet werden, wird im folgenden Abschnitt über die konkrete Berechnung und Visualisierung der Objektverdeckung beschrieben.

6.4.4 Berechnung und Visualisierung

Den Kern der Objektverdeckungsbehandlung bilden die drei Klassen `OcclusionThread`, `OcclusionFilter` und `MaskingObject`. Bei der Berechnung sind sie auf die Daten, welche durch die Datenklasse `OcclusionData` bereitgestellt werden, angewiesen. Im Kapitel 5.3.3 wurde bereits die konzeptionelle Funktionalität der Objektverdeckungsberechnung beschrieben. Die drei dort vorgestellten Hauptprozesse (Erhebungsprozess, Filterungsprozess und Darstellungsprozess) werden nun durch diese Klassen implementiert. Dabei steuert die Klasse `OcclusionThread` den gesamten Prozessablauf und verwendet die Klasse `OcclusionFilter` für den Filterungsprozess und die Klasse `MaskingObject` für den Darstellungsprozess.

`OcclusionThread` ist von der Klasse `Thread` abgeleitet, welche die Java-API standardmäßig zur Verfügung stellt. Normalerweise wird immer nur ein Programmstrang ausgeführt, `Threads` ermöglichen es jedoch, dass mehrere Programmstränge parallel ausgeführt werden. Da die Objektverdeckungsberechnung parallel zum Hauptprogramm der AR-Anwendung ausgeführt werden soll, ohne diese zu beeinträchtigen oder zu unterbrechen, ist die Implementierung von `OcclusionThread` als `Thread` sinnvoll. Dabei erzeugt die Schnittstelle `OcclusionManager` mit seiner Erzeugung eine Instanz der Klasse `OcclusionThread` und ruft an ihr die `start()`-Methode auf. Diese von der Klasse `Thread` vorgegebene Methode ruft

`run()` auf, welche alle Abläufe, die parallel zum Hauptprogramm ausgeführt werden sollen, implementiert.[13]

Die Methode `run()` ist so aufgebaut, dass sie mit der Erzeugung der Klasse `OcclusionThread` und dem Aufruf der `start()`-Methode darauf wartet, dass die Verdeckungsberechnung aktiviert wird. Dazu überwacht sie das Attribut `isLive` der Klasse `OcclusionData`. Wird dieses Attribut vom Anwender durch die Methode `setOcclusionStatus()` im `OcclusionManager` auf den Status `true` gesetzt, startet die Verdeckungsbehandlung. Dieser Gesamtprozess wiederholt sich solange bis die Berechnung durch `false`-Setzen des Attributs `isLive` wieder unterbrochen wird. Um eine gerade laufende Berechnung nicht zu unterbrechen, wird der Prozessablauf zuvor vollständig beendet. In Abbildung 6.7 wird anhand eines Struktogramms der interne Ablauf der Methode `run()` dargestellt:

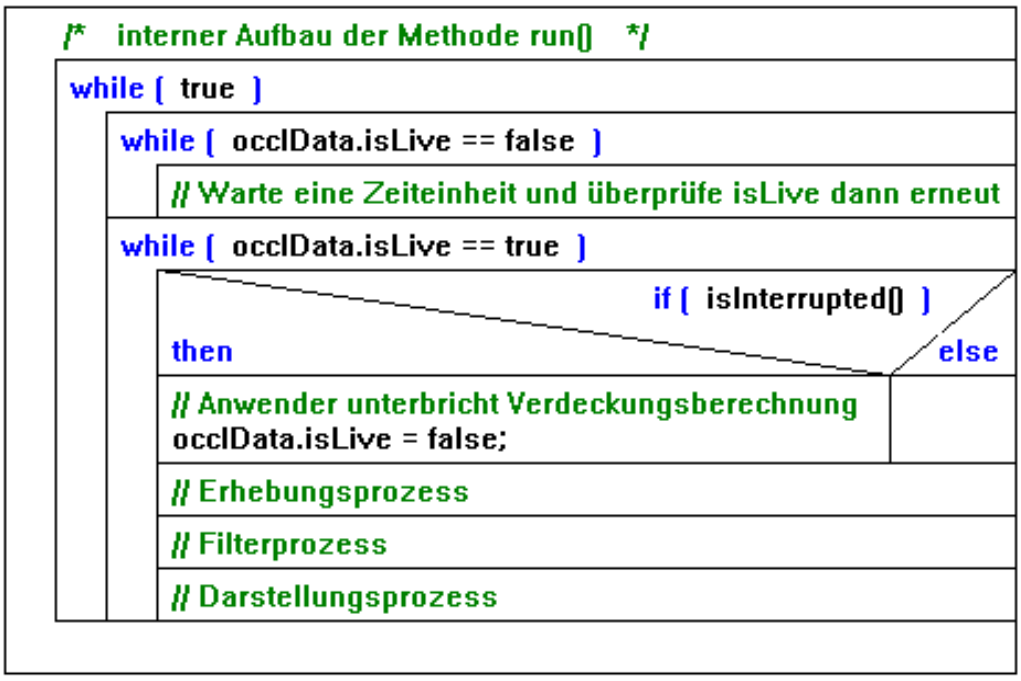


Abbildung 6.7: Struktureller Ablauf der Methode run() in der Klasse OcclusionThread

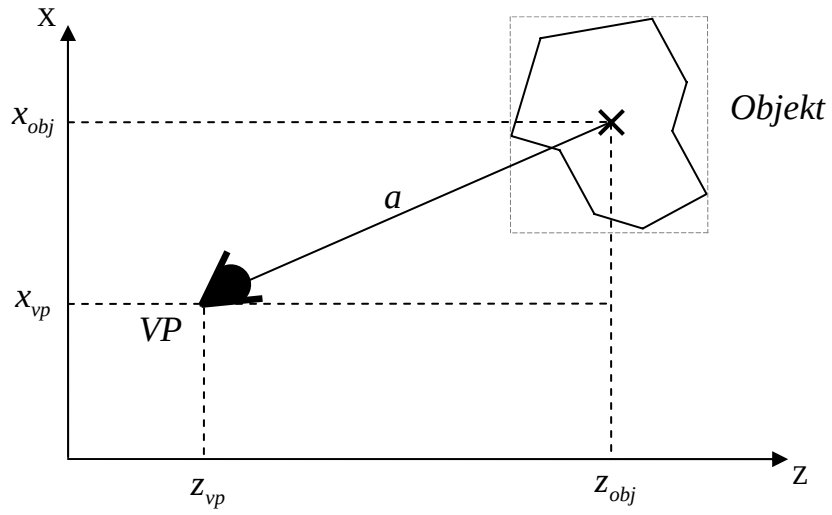
[13] Die Methode run() wird von der Klasse Thread vererbt und muss in jedem Fall überschrieben werden.

Ist die Berechnung gestartet, wird zuerst der Erhebungsprozess durchlaufen. Hier wird die Menge der verdeckenden Objekte zwischengespeichert, die aus dem vorhergehenden Rechendurchlauf bekannt ist. Handelt es sich um den ersten Durchlauf der Verdeckungsberechnung, ist diese Menge leer. Die Zwischenspeicherung dient im Darstellungsprozess zum Vergleich der ursprünglichen Menge mit der im Filterungsprozess neu berechneten Menge der verdeckenden Objekte.

Nach dem Erhebungsprozess folgt der Filterungsprozess. Hierfür wird die Klasse `OcclusionFilter` verwendet. Sie berechnet die Menge der verdeckenden Objekte durch die drei verschiedenen Filter Abstandsfilter, Sichtkegelfilter und Verdeckungsfilter. Diese Objektmenge wird der `Hashtable` `occlObjects` in `OcclusionData`. `OcclusionFilter` ist ebenfalls von der Klasse `Thread` abgeleitet, weswegen auch hier die `run()`-Methode sämtliche Funktionalitäten der Klasse steuert. Um die Klasse `OcclusionFilter` als Filterungsprozess nutzen zu können, bildet `OcclusionThread` eine Instanz von ihr und ruft zum Starten die Methode `start()` auf. In `run()` werden zu Beginn jedes Durchlaufs die Mengen der Objekte, die sich momentan im Szenengraphen befinden, und der Objekte, die sie eventuell verdecken könnten und sich nicht im Szenengraphen befinden, zwischengespeichert. Die Daten hierfür verwaltet `OcclusionData`. Auf der Grundlage dieser Objektmengen werden die verschiedenen Filter durchlaufen, dabei wird für jeden Filtertyp eine eigene Methode verwendet. Jede dieser Methoden liefert als Rückgabewert eine `Hashtable` mit den herausgefilterten, verdeckenden Objekten.

- Abstandsfilter:
  Der Abstandsfilter berechnet die verdeckenden Objekte, die sich innerhalb eines bestimmten Abstands zur `ViewPlatform` befinden. Hierfür wird die private Methode `distanceFilter()` verwendet. Sie berechnete in Abhängigkeit von einem bestimmten Abstand um die `ViewPlatform` herum, die Menge der verdeckenden Objekte. Hierfür wird je Objekt, das für eine Verdeckung in Frage kommt, der Abstand seines Mittelpunkts der Boundingbox zur `ViewPlatform` bestimmt. Erforderlich für diese Berechnung ist die Position der `ViewPlatform`, wobei diese und auch der Abstand durch die Klasse `OcclusionData` verfügbar sind. Abbildung 6.8 stellt die Formel für die Berechnung des Abstands a graphisch dar. Dabei entsprechen

x_{vp} und z_{vp} den Positionskoordinaten der `ViewPlatform` und x_{obj} und z_{obj} den Koordinaten des Objektmittelpunkts.



$$a = \sqrt{\left(x_{obj} - x_{vp}\right)^2 + \left(z_{obj} - z_{vp}\right)^2}$$

Abbildung 6.8: Schematische Darstellung der Abstandsberechnung zwischen einem Objekt und der ViewPlatform

- Sichtkegelfilter:

  Der Sichtkegelfilter berechnet die Objekte mit Hilfe der Methode `viewAngleFilter()`. Sie filtert aus der Menge der ihr übergebenen Objekte die Objekte heraus, die sich im Sichtfeld der `ViewPlatform` befinden. Für die entsprechende Berechnung werden Position und Orientierung der `ViewPlatform` benötigt. Diese Parameter sowie der Sichtwinkel werden wieder durch `OcclusionData` bereitgestellt.

  Um berechnen zu können, ob sich ein Objekt in einem bestimmten Sichtwinkel befindet, werden zuerst zwei Richtungsvektoren ermittelt, wobei einer vom Mittelpunkt des Objekts zur `ViewPlatform` gerichtet ist und der zweite der Orientierung der `ViewPlatform` entspricht. Ist der Winkel α zwischen diesen beiden Vektoren kleiner als die Hälfte des gesamten Sichtwinkels β, befindet sich das Objekt innerhalb des Sichtwinkels, da die Orientierung den Sichtwinkel in der Hälfte teilt (Abbildung 6.9).

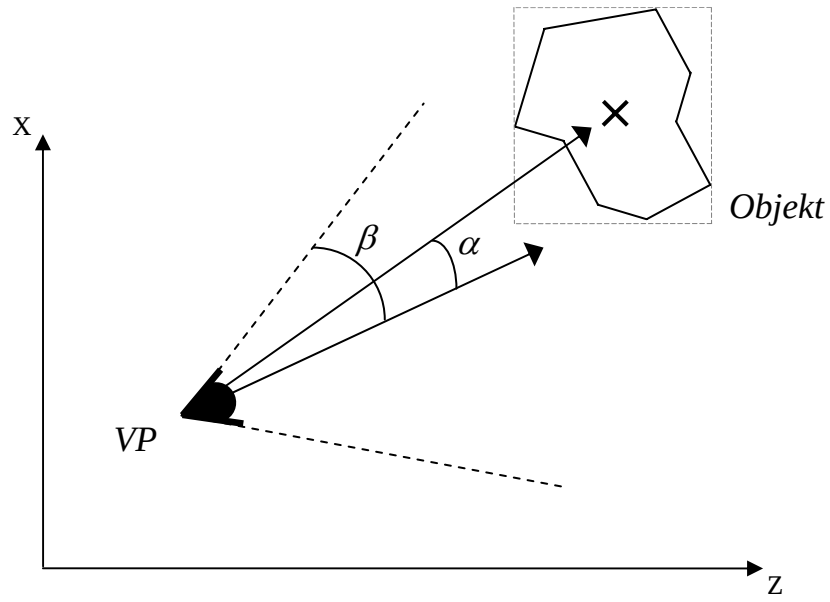


Abbildung 6.9: Berechnung des Sichtwinkelfilters.

- Verdeckungsfilter:
  Der Verdeckungsfilter prüft über die Methode `behindFilter()` welche der realen Objekte sich tatsächlich vor einem virtuellen Objekt befinden. Es wird berechnet, welche Objekte der ersten übergebenen `Hashtable` Objekte aus der zweiten übergebenen `Hashtable` verdecken können. Dafür wird zuerst die maximale Entfernung von eventuell verdeckten Objekten zur `ViewPlatform` berechnet. Reale Objekte die sich näher an der `ViewPlatform` befinden, können dann andere virtuelle Objekte verdecken.

Die Reihenfolge und Häufigkeit der oben beschriebenen Filter wird durch das Attribut `methodOrder` in `OcclusionData` bestimmt. Je nachdem, in welcher Reihenfolge in diesem String die Worte „ABSTAND“, „SICHTKEGEL“ oder „VERDECKUNG“ enthalten sind, werden die entsprechenden Filter-Methoden ausgeführt. Dabei sind die drei Filter am wirkungsvollsten, wenn sie alle drei verwendet werden, da sie sich dann gegenseitig kombinieren und eine genauere Eingrenzung der verdeckenden Objekte ergeben. Die Filter bauen aufeinander auf, indem jeder Filter die Ergebnismenge des vorhergehenden Filters zur Berechnung verwendet. Das heißt, zum Schluss enthält die Menge der verdeckenden Objekte nur die Objekte, die von allen drei Filtern als verdeckende Objekte erkannt wurden. `OcclusionThread`

wartet auf dieses Ergebnis und setzt dann die Objektverdeckungsbehandlung auf der Basis der errechneten Objektmenge mit dem Darstellungsprozess fort sobald der Filterungsprozess abgeschlossen ist.

Im Darstellungsprozess wird die Menge der im Erhebungsprozess zwischengespeicherten Objekte mit der neu errechneten Menge von verdeckenden Objekten verglichen. Verdeckende Objekte, die noch nicht im Szenengraphen dargestellt werden, werden dem Szenengraphen hinzugefügt, andere Objekte, die nicht mehr verdecken, werden aus ihm gelöscht. Alle verdeckenden Objekte befinden sich im Szenengraphen in der vom `OcclusionManager` festgelegten `BranchGroup`. Um verdeckende Objekte in den Szenengraph einzufügen, verwendet `OcclusionThread` die Klasse `MaskingObject`.

Für jedes verdeckende Objekt wird ein Objekt der Klasse `MaskingObject` gebildet und in den Szenengraph eingefügt. Die Klasse `MaskingObject` bewirkt, dass die Objektgeometrien dem modellbasierten Ansatz entsprechend in der Farbe des Hintergrundes dargestellt werden und die Objekte keine Oberflächeneigenschaften wie Glanz, Reflexion und Spiegelung aufweisen. Durch eine entsprechende Methode werden den Objektgeometrien automatisch die eben genannten Farbeigenschaften zugewiesen.

Nachdem durch die Klasse `OcclusionThread` für jedes verdeckende reale Objekt ein dieses Objekt repräsentierendes `MaskingObject` in den Szenengraphen eingefügt wurde und Objekte, die nicht mehr verdecken, aus diesem gelöscht wurden, ist ein Durchlauf der Verdeckungsbehandlung abgeschlossen. Der `Thread` beginnt seine Arbeit im nächsten Zyklus von neuem beim Erhebungsprozess. Die drei Prozesse werden solange abgearbeitet, bis eine Unterbrechung durch den Anwender erfolgt.

Durch diesen Ablauf wird gewährleistet, dass sich stets alle verdeckenden Objekte in möglichst kleiner Anzahl im Szenengraphen befinden und damit die Objektverdeckung visualisieren. Die der Hintergrundfarbe entsprechend eingefärbten Objekte sind nur als Silhouettenausschnitt vor den virtuellen Objekten sichtbar. Durch die-

sen Ausschnitt hindurch werden im AR-System die realen Objekte der natürlichen Umgebung sichtbar gemacht.

Zum Testen und Darstellen der Objektverdeckung wurde eine graphische Benutzeroberfläche entwickelt. Ihr Aufbau und die Funktion ihrer Klassen werden im nächsten Kapitel beschrieben.

7 Entwicklung der Testumgebung

7.1 Übersicht

Um die im vorhergehenden Kapitel beschriebene Objektverdeckungskomponente testen zu können, wird eine Benutzeroberfläche zur Visualisierung und Demonstration der Objektverdeckung benötigt. In diese Oberfläche soll die Objektverdeckungskomponente integriert werden. Damit diese sinnvoll getestet werden kann, sind einige Funktionalitäten innerhalb der Benutzeroberfläche erforderlich. Welche Funktionalitäten benötigt werden und wie diese in der Testumgebung umgesetzt werden, wird in diesem Kapitel erläutert.

7.2 Entwurf der Benutzeroberfläche

Als Testumgebung ist eine GUI-Anwendung vorgesehen, die aus einem einzigen Hauptfenster besteht und mehrere Kindfenster in diesem Hauptfenster enthält. Jede Art von Funktionalität, welche die Benutzeroberfläche bieten soll, soll dabei in einem separaten Fenster angeboten werden. Diese Technik wird unter Windows als MDI (Multiple Document Interface) bezeichnet. Da im Java AWT keine Möglichkeit besteht, diese Oberflächenstruktur zu verwenden, wird für die Entwicklung der Benutzeroberfläche Swing verwendet (Kapitel 2.3.3).

Das Hauptfenster, das den Rahmen der GUI-Anwendung darstellt, muss dabei keine besonderen Fähigkeiten bieten. Sämtliche Funktionalitäten der Benutzeroberfläche werden in separaten Fenstern zur Verfügung gestellt. In den folgenden Absätzen wird beschrieben, welche Funktionalitäten im Einzelnen in der Testumgebung benötigt werden, und in welcher Form sie in der Anwendung bereitgestellt werden sollen.

Zunächst muss die dreidimensionale Szene beziehungsweise der Szenengraph visualisiert werden. Dabei soll als Ausgabegerät für die hier entwickelte Testumgebung der Bildschirm eines Computers dienen. Diese Technik wird „Window-on-World“-Verfahren genannt und stellt eine einfache und weit verbreitete Technik zur Visualisierung von dreidimensionalen Welten dar. Generell basiert die Darstellung von

dreidimensionalen Objekten oder Szenen dabei auf Projektionen der dreidimensionalen Szene auf eine zweidimensionale Fläche. Diesen Abbildungsvorgang bezeichnet man als Rendering. Das Rendering wird bereits von der Java 3D API bereitgestellt und liefert die erzeugten Bilder in Form von GUI-Komponenten. Diese GUI-Komponenten können dann in die Benutzeroberfläche eingebunden werden. Für die Darstellung der Szene ist ein separates Fenster vorgesehen.

Des Weiteren muss die Möglichkeit bestehen auswählen zu können, welche Objekte in der virtuellen Szene dargestellt beziehungsweise in den Szenengraph eingefügt oder auch wieder gelöscht werden sollen. Zusätzlich ist es bei manchen Objekten sinnvoll, ihre Position in der virtuellen Szene definieren zu können. Für diese Funktionalitäten werden Felder in der Benutzeroberfläche benötigt, über die diese Informationen an den Szenengraph weiter geleitet werden können. Integriert werden diese Felder in einem eigenen Fenster, das gleichzeitig zeigt, welche Objekte für eine Darstellung im Visualisierungsfenster verfügbar sind und damit in den Szenengraphen eingefügt oder aus ihm entfernt werden können.

Damit der Anwender in der dreidimensionalen Szene navigieren kann, muss er die Position und Orientierung der Kamera beliebig steuern können. Hierfür wird eine Schnittstelle zur virtuellen Kamera benötigt, durch welche der Anwender eben diese Parameter setzen kann. Ermöglicht wird dies über Eingabefelder, durch welche sich Position und Orientierung der Kamera setzen lassen. Zusätzlich hilfreich ist die Möglichkeit zur Abfrage der aktuellen Parameter der Kamera, um sich in der Szene besser orientieren zu können. Auch dies soll über Felder realisiert werden, die in diesem Fall jedoch zur Ausgabe dienen. Alle Felder zur Ein- und Ausgabe der Kameraparameter sollen innerhalb eines separaten Fensters bereitgestellt werden.

Die Objektverdeckungskomponente soll ebenfalls in die Gesamtanwendung der Benutzeroberfläche integriert werden. Um die Komponente auch nutzen und nach Bedarf einsetzen zu können, ist es notwendig, mit ihr kommunizieren zu können. Dafür stellt die Objektverdeckungskomponente eine, bereits in Kapitel 5.3.3 erwähnte, Schnittstelle bereit, über welche die Objektverdeckungsberechnung gestartet und angehalten werden kann. Um diese Funktionalität graphisch in die Benutzeroberfläche einzubinden, soll ein Fenster mit entsprechenden Schaltflächen implementiert

werden. In Abbildung 7.1 werden die Komponenten, aus denen die Benutzeroberfläche besteht und ihre Beziehungen untereinander sowie zur Objektverdeckungsschnittstelle schematisch dargestellt:

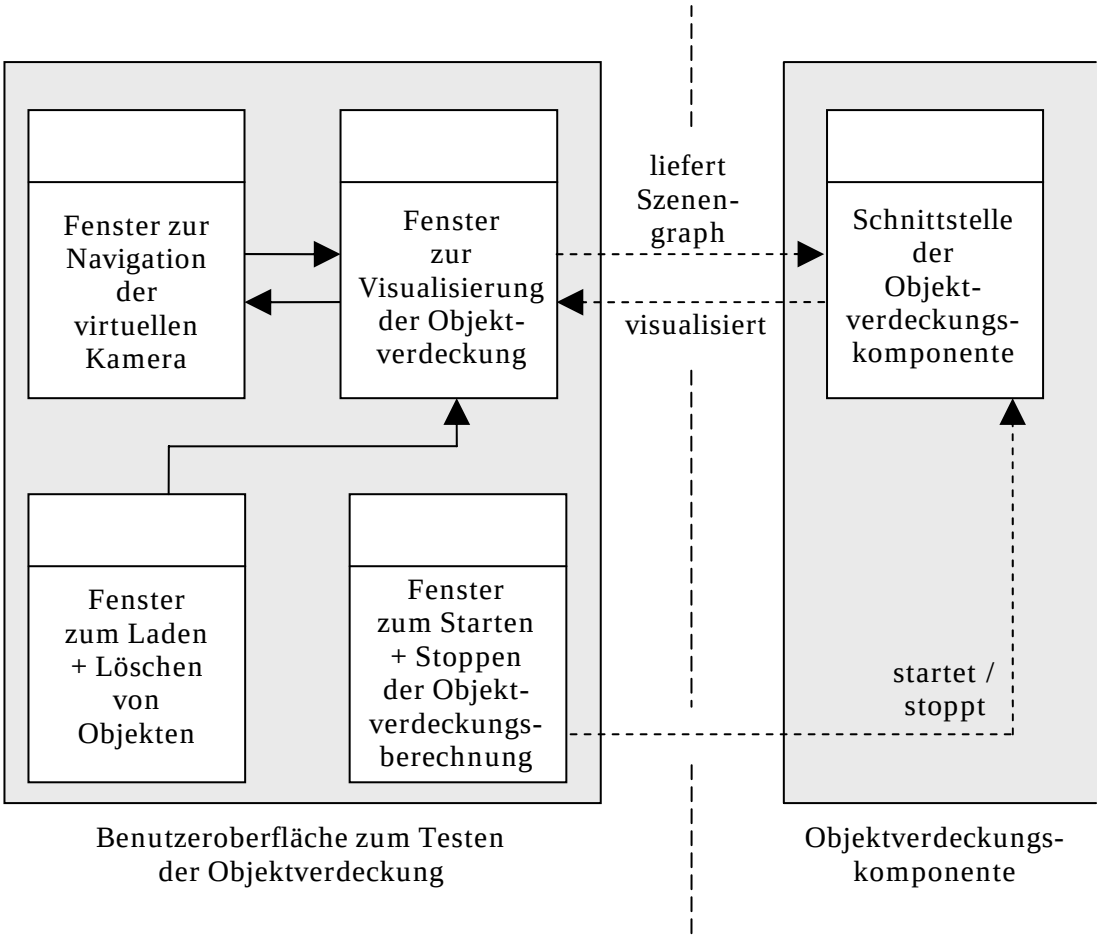


Abbildung 7.1: Schematische Darstellung der Benutzeroberfläche und der Kommunikation zur Objektverdeckungskomponente

7.3 Benutzeroberfläche als Testumgebung

7.3.1 Die Klassen des Pakets „gui“

Das Paket gui enthält alle Klassen, die für den Aufbau der Benutzeroberfläche erforderlich sind. Diese besteht aus einem Hauptfenster mit Menüleiste enthält mehrere Kindfenster (MDI, Multiple Document Interface). In diesen internen Fenstern sind

die verschiedenen Funktionalitäten der Benutzeroberfläche implementiert. Sie haben den Vorteil, dass sie einzeln verschoben und skaliert werden können, wodurch der Anwender die Möglichkeit hat, die Benutzeroberfläche nach seinem Bedarf zu gestalten. Die einzelnen Klassen, aus denen die Benutzeroberfläche besteht, basieren auf Swing (Kapitel 2.3.3).

Jedes Fenster wird durch eine Klasse erzeugt. Hierbei stellt die Klasse `MainFrame` das Hauptfenster dar. Sie ist von der Klasse `JFrame` aus dem Swing-Paket abgeleitet und bindet die von `JMenuBar` abgeleitete Klasse `AppMenuBar` als Menüleiste ein. Alle Kindfenster sind von der Klasse `AppInternalFrame` abgeleitet, welche wiederum von der Swing-Klasse `JInternalFrame` abgeleitet ist. In `AppInternalFrame` werden Funktionen implementiert, die von allen internen Fenstern benötigt werden. Abbildung 7.2 stellt die Beziehungen der verwendeten Klassen noch einmal schematisch dar. Eine genauere Beschreibung der einzelnen Klassen findet in den folgenden Abschnitten statt.

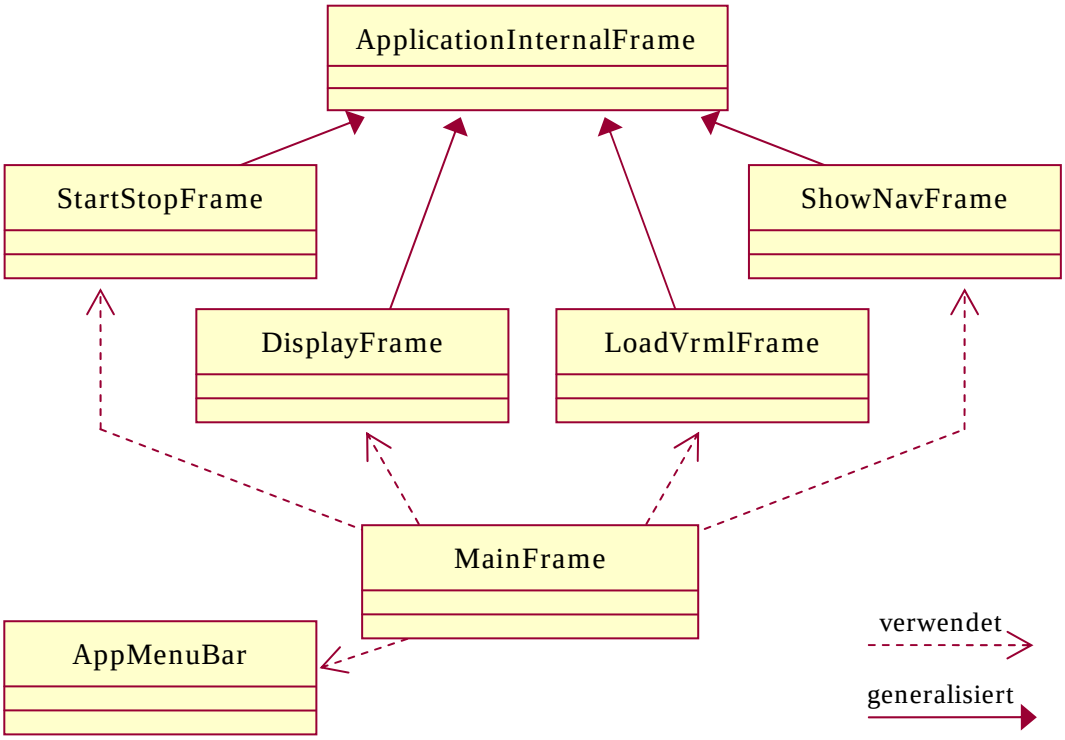


Abbildung 7.2: Schematische Darstellung der Klassen im Paket gui

7.3.2 Aufbau des Hauptfensters und der Menüleiste

Das Hauptfenster wird durch die Klasse `MainFrame` dargestellt, die wiederum die Klasse `AppMenuBar` verwendet, um eine Menüleiste zu erzeugen und einzubinden. `MainFrame` ist wie ein Standardfenster aufgebaut und besitzt beim Erzeugen noch keinen Inhalt außer der Menüleiste. Diese Menüleiste ist nicht sehr umfangreich, da die meiste notwendige Funktionalität durch die implementierten Fenster erlangt wird. Es ist ein Menü „File“ vorgesehen, das bisher jedoch nur den Menüunterpunkt „Exit“ enthält. Durch Klicken auf diesen Menüpunkt wird die GUI-Anwendung beendet.

Ein weiteres Menü heißt „Window“. Es enthält ein Menü, in welchem für jedes Kindfenster ein `JCheckBoxMenuItem` aufgelistet ist. Diese Swing-Klasse stellt einen Menüeintrag mit einem Kontrollkästchen dar. Das Kontrollkästchen ändert durch Anfügen oder Entfernen eines Häkchens den Status des Menüeintrags. Dieser Status kann überwacht werden und wird im Fall des Window-Menüs dafür eingesetzt, die internen Fenster zu verwalten. Je nachdem, in welchem Kästchen das Häkchen gesetzt ist, wird das entsprechende interne Fenster angezeigt. Damit hat der Anwender die Möglichkeit, die verschiedenen Kindfenster seinen Ansprüchen entsprechend anzeigen zu lassen. Ein drittes Menü stellt unter dem Namen „Help“ Menüeinträge zur Verfügung, die Hilfe bei der GUI-Anwendung versprechen. Mangels Relevanz für diese Arbeit wurde jedoch nur ein Eintrag implementiert, unter dem man generelle Informationen zu der Anwendung erhält. Abbildung 7.3 stellt einen Ausschnitt aus `MainFrame` mit aufgeklapptem Window-Menü dar.

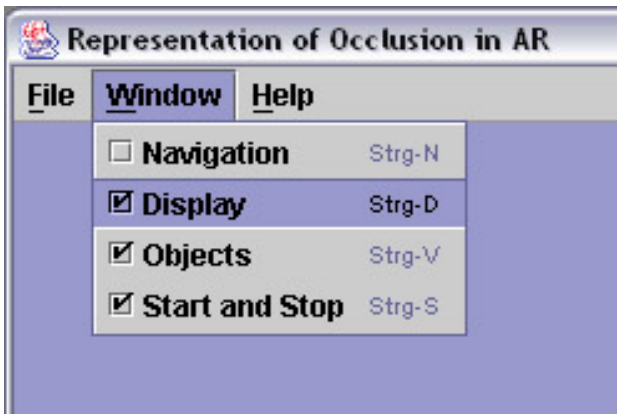


Abbildung 7.3: Die Menüleiste im Hauptfenster

Die Klasse `MainFrame` implementiert zwei Methoden, mit denen in dieses Hauptfenster interne Fenster eingebunden werden können. Beide Methoden heißen `addChild()` und unterscheiden sich neben ihren Übergabeparameter nur geringfügig. Die Methoden erwarten als Parameter unter anderem ein Objekt eines internen Fensters und fügen es dem Hauptfenster hinzu. Weitere Übergabeparameter bestimmen Größe und Position des hinzugefügten Fensters innerhalb des Hauptfensters. Der Unterschied der Methoden liegt dabei nur darin, dass bei der einen `addChild()`-Methode die Skalierung des Fensters nicht mit angegeben wird. Fenster, die durch diese `addChild()`-Methode hinzugefügt werden, nehmen also automatisch die Größe an, die von den GUI-Komponenten innerhalb des Fensters benötigt wird.

In den folgenden Abschnitten werden die internen Fenster, die vom Hauptfenster implementiert werden, näher erläutert. Dabei basieren alle internen Fenster auf der zunächst beschriebenen Basisklasse `AppInternalFrame`. Abbildung 7.4 zeigt das Hauptfenster, wenn alle internen Fenster hinzugefügt sind.

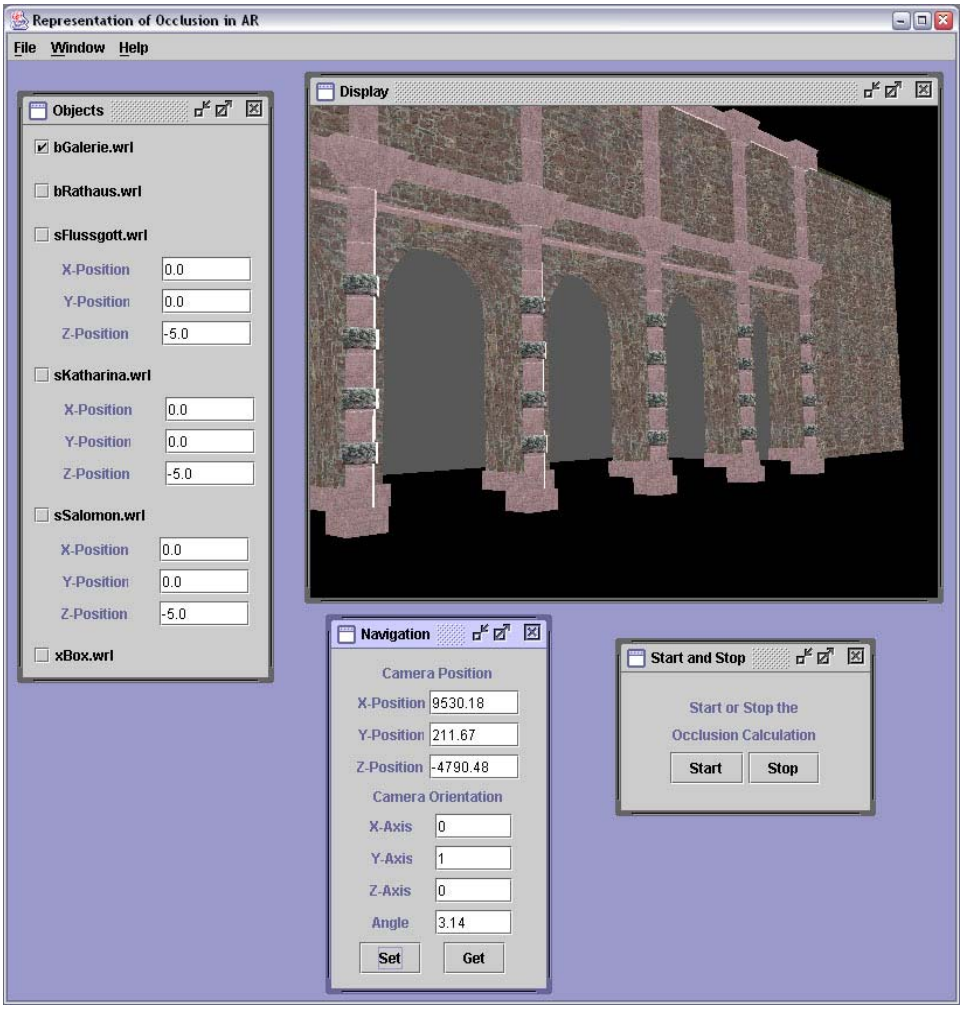


Abbildung 7.4: Das Hauptfenster mit den internen Fenstern

7.3.3 Genereller Aufbau der internen Fenster

Die Klasse `AppInternalFrame` bildet die Basisklasse aller in der GUI-Anwendung verwendeten internen Fenster. Durch sie werden verschiedene Methoden zur Verfügung gestellt, mit deren Hilfe sich das Layout der Komponenten in den internen Fenstern konfigurieren und an ein einheitliches Aussehen anpassen lässt. Außerdem implementiert die Klasse bereits die verschiedenen Schnittstellen und Methoden, die notwendig sind, um Aktionen wie Mausklicks auf GUI-Komponenten innerhalb der Fenster zu behandeln. Die Vererbung dieser Methoden hat den Vorteil, dass Funktionalitäten, die in allen Fenstern benötigt werden, automatisch ent-

halten sind. Eine Auswahl der Methoden und Datenfelder wird im Folgenden kurz vorgestellt.

In auf Java basierten graphischen Benutzeroberflächen können für die Anordnung der einzelnen GUI-Komponenten im Fenster verschiedene Layoutmanager verwendet werden. Diese sind unterschiedlich aufwendig und nicht immer flexibel zu handhaben. Für die internen Fenster in dieser Arbeit wird der Layoutmanager `GridBagLayout` verwendet, der am genauesten zu steuern ist. Es ist teilweise umständlich einzurichten, bietet jedoch die exakteste Möglichkeit, Komponenten in einem Fenster wie gewünscht darzustellen. Das `GridBagLayout` kann von allen abgeleiteten Klassen verwendet werden, da `AppInternalFrame` erforderliche Datenfelder wie ein `GridBagLayout`-Objekt und ein `GridBagConstraints`-Objekt sowie Methoden zur Konfiguration des Layouts vorsieht. Die wichtigste Methode hierbei ist `makegbCon()`. Diese Methode wird von den abgeleiteten Klassen für jede GUI-Komponente aufgerufen und erzeugt damit ein den übergebenen Parametern entsprechendes Layout. Die Parameter legen dabei fest, wo und in welcher Größe die betreffende Komponente im Fenster dargestellt wird.

Eine weitere Methode speichert die Menge aller Fenster, die `AppInternalFrame` als Basisklasse verwenden, welche diese Klasse als Basisklasse verwenden, in einem `Vector`-Objekt. Durch diese Verwaltung kann der Zustand der jeweiligen Fenster überwacht werden. Erforderlich ist dies vor allem für das Window-Menü in der Menüleiste des Hauptfensters, da es mit Hilfe dieser Struktur überwacht, welche internen Fenster geöffnet oder geschlossen sind.

In den folgenden Abschnitten werden nun die Klassen beschrieben, die von `AppInternalFrame` abgeleitet sind und damit die oben erwähnten Funktionen einerben und verwenden.

7.3.4 Aufbau des Visualisierungsfensters

Das Visualisierungsfenster ist von allen internen Fenstern am einfachsten aufgebaut und wird durch die Klasse `DisplayFrame` erzeugt. Es dient der Visualisierung der dreidimensionalen Java 3D Szene und damit auch der Darstellung der Objektverde-

ckung. Hierfür enthält das Fenster das `Canvas3D`-Objekt des verwendeten Szenengraphen und bildet damit die Szene aus der Sicht der `ViewPlatform` ab. Um eine Referenz auf den Szenengraphen und damit auf das `Canvas3D`-Objekt zu erhalten, verwendet `DisplayFrame` die Klasse `SceneGraph` aus dem Paket scene. Von allen internen Fenstern verwendet `DisplayFrame` als einzige Klasse, keinen Layoutmanager, da nur die Abbildungsfläche in dem Fenster angeordnet werden muss. Da der verwendete Szenengraph verschiedene `Behavior`-Objekte zur Navigation der `ViewPlatform` enthält, ist es möglich, mit der Maus durch die 3D-Szene zu navigieren.

7.3.5 Aufbau des Navigationsfensters

Zur präziseren Navigation im Visualisierungsfenster dient das Navigationsfenster, das durch `ShowNavFrame` erzeugt wird. Es stellt Funktionalitäten zur Verfügung, mit denen die `ViewPlatform` des im `DisplayFrame` abgebildeten Szenengraphen gesteuert werden kann. Zusätzlich lassen sich Position und Orientierung der `ViewPlatform` anzeigen. Hierfür greift das Fenster über die durch die Klasse `ViewBranch` verfügbaren Methoden auf den Transformationsknoten der `ViewPlatform` zu.

Diese Steuerung der `ViewPlatform` wird im Fenster durch Textfeld-Komponenten ermöglicht. Dabei werden in drei Textfeldern jeweils die x-, die y- und die z-Koordinate der Position angegeben und in weiteren vier Feldern die vier Werte, welche die Orientierung angeben. Die Textfelder sind beschriftet, damit deutlich wird, für welche Koordinate ein Textfeld vorgesehen ist. Um die Zahleneingaben in den Textfeldern an den Transformationsknoten der `ViewPlatform` zu übertragen steht eine Schaltfläche zur Verfügung. Zum Abruf der aktuellen Positions- und Orientierungswerte genügt ein Mausklick auf einen weiteren Button (Abbildung 7.5).

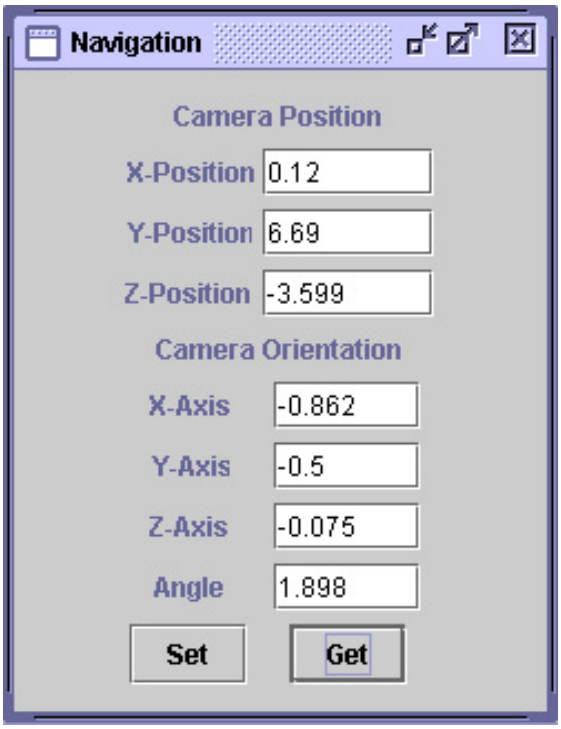


Abbildung 7.5: Das Navigationsfenster

Obwohl der `ViewBranch` bereits die Navigation mit der Maus ermöglicht, ist oben beschriebene Art der Steuerung doch sinnvoll. Indem konkrete Zahlenwerte eingeben werden können, kann die `ViewPlatform` wesentlich präziser gesteuert werden. Gerade zum Testen der Objektverdeckungen ist diese Funktionalität sinnvoll, da genau überprüft werden kann, ob die Berechnung der verdeckenden Objekte und ihre Darstellung auf korrekte Weise arbeiten.

7.3.6 Aufbau des Objektfensters

Das Objektfenster erlaubt im Gegensatz zum Navigationsfenster die Modifikation des `ContentBranchs`. Dieses interne Fenster wird durch die Klasse `LoadVrmlFrame` erzeugt und ermöglicht es, Objekte in den Szenengraphen zu laden oder aus diesem zu entfernen. Hierfür wird in dem Fenster eine Liste von Checkboxen generiert. Dabei entspricht jede Checkbox einer Objektgeometrie aus der Menge der für die Java 3D Anwendung zur Verfügung stehenden Objekte. Wird nun ein Modell durch das Setzen eines Häkchens in der betreffenden Checkbox aktiviert, werden die

zugehörigen Geometrien in den Szenengraphen geladen. In dieser Arbeit werden dafür Vrml-Modelle verwendet, die über einen Parser in Java 3D Geometrien konvertiert und dann in den Szenengraphen gehängt werden. Wird das Häkchen einer Checkbox wieder entfernt, wird das entsprechende Objekt aus dem Szenengraphen gelöscht. Hierfür werden entsprechende Methoden der Klasse `ContentBranch` verwendet (Kapitel 6.3.1).

Bei manchen Objekten kann es durchaus erwünscht sein, sie an einer beliebigen Stelle positionieren zu können. Das ist vor allem bei dynamischen Objekten sinnvoll. Deswegen wird je dynamischen Objekt nicht nur eine Checkbox in dem internen Fenster erzeugt, sondern auch drei Textfelder. Über diese Textfelder kann dann die Position festgelegt werden, an der das Objekt erscheinen soll. Intern wird hierfür eine `TransformGroup` erzeugt, welche das Objekt an die gewünschte Stelle transliert.

Die Liste der Checkboxen wird durch das Auslesen einer Textdatei generiert. Es wird dabei die gleiche Textdatei verwendet, die auch die Informationen über die zur Verfügung stehenden Objekte in der Objektverdeckungsbehandlung enthält. Eine derartige Objektliste kann auch auf andere Weise zur Verfügung gestellt werden. Dafür steht ein zweiter Konstruktor der Klasse zur Verfügung, in welchem per Verzeichnis- und Dateiname angegeben werden kann, wo sich eine entsprechende Textdatei befindet (Abbildung 7.6).

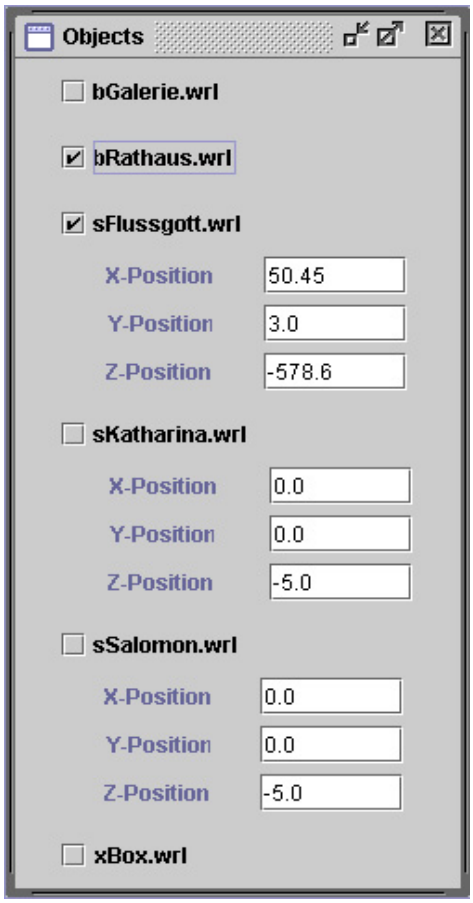


Abbildung 7.6: Das Objektfenster

Das Objektfenster ermöglicht es, gezielt Objektverdeckungssituationen einzustellen und damit die Objektverdeckungskomponente auf ihre Funktionsweise zu überprüfen.

7.3.7 Aufbau des Fensters zur Steuerung der Objektverdeckung

Das vierte Fenster ist wiederum sehr einfach aufgebaut. Es wird durch die Klasse `StartStopFrame` generiert und dient dazu, die Objektverdeckungsbehandlung zu starten oder anzuhalten. Hierfür werden zwei Buttons im Fenster angezeigt, wobei einer zum Starten und einer zum Unterbrechen der Verdeckungsbehandlung vorgesehen ist (Abbildung 7.7).

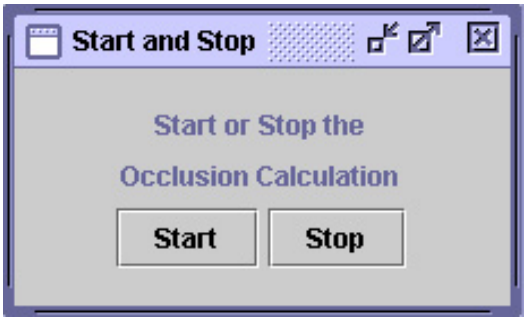


Abbildung 7.7: Das Fenster zur Steuerung der Objektverdeckung

Wird auf den Start-Button geklickt, wird im `OcclusionManager` die Methode `setOcclusionStatus()` mit dem Parameter `true` aufgerufen, das heißt `OcclusionThread` startet die Objektverdeckungsbehandlung. Wird auf den Stopp-Button geklickt, wird `setOcclusionStatus()` mit dem Parameter `false` ausgeführt und dadurch die Arbeit des `OcclusionThread` unterbrochen, sobald er am Ende seiner `run()`-Methode angekommen ist. Es wird also verhindert, dass ein weiterer Durchlauf der Verdeckungsbehandlung gestartet wird (Kapitel 6.4.4).

Damit die Verdeckungsbehandlung überhaupt verwendet werden kann, muss zuvor eine Instanz der Klasse `OcclusionManager` gebildet und damit das Objektverdeckungspaket occlusion initialisiert werden. Wie in Kapitel 6.4.3 bereits beschrieben, muss `OcclusionManager` dabei ein Vektor mit Daten und Referenzen übergeben werden. Diese Daten stehen `StartStopFrame` durch die Zugriffsmethoden, die in den Klassen des Pakets scene vorhanden sind, zur Verfügung. Folgendes Quellcodebeispiel zeigt, wie der Konstruktor von `StartStopFrame` den `OcclusionManager` initialisiert und den Datenvektor übergibt, der vorher gefüllt worden ist:

```
public class StartStopFrame {

        OcclusionManger om = null;

        public StartStopFrame(…) {
                …
                om = OcclusionManager.getInstance(createVector());
                …
        }

        private Vector createVector(){
                …
                occlVector = new Vector();
                occlVector.add(ViewBranch.getVPTransform());
```

```
                occlVector.add(ContentBranch.getInstance());
                occlVector.add(level);
                occlVector.add(pathname);
                occlVector.add(filename);
                occlVector.add(SceneGraph.getLocale());
                return occlVector;
        }
        …
}
```

So einfach wie in diesem Programmausschnitt kann die Objektverdeckungsbehandlung in jeder Java 3D Anwendung integriert werden. Es müssen nur die benötigten Daten durch den übergebenen Vektor verfügbar gemacht werden.

7.4 Die Klassen des Pakets „data“

Das vierte Paket in dieser Arbeit enthält nur die zwei Klassen `TheVrmlLoader` und `BBoxAsXML`, die weitestgehend als Hilfsklassen dienen (Abbildung 7.8). Dabei wird die Klasse `TheVrmlLoader` verwendet, um ein Vrml-Modell in eine Java 3D Geometrie zu konvertieren. Die Klasse importiert die Klasse `VrmlLoader` des Java-Pakets com.sun.j3d.loaders.vrml97.

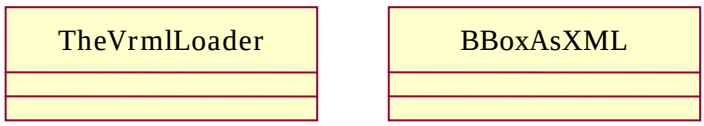


Abbildung 7.8: Schematische Darstellung der Klassen im Paket data

`TheVrmlLoader` stellt anstatt der Instanzbildung eine statische Methode zur Verfügung, die aufgerufen werden kann, ohne dass ein Objekt der Klasse existiert. Dieser Methode wird der Dateiname des Vrml-Modells übergeben. `TheVrmlLoader` sucht diese Datei und generiert aus ihr eine `BranchGroup`, welche dann die gesamten in Java 3D Format konvertierten Objektgeometrien enthält. Diese `BranchGroup` gibt die statische Methode wieder zurück. Die aufrufende Klasse kann den erhaltenen Knoten nun in jedem Szenengraphen verwenden. `TheVrmlLoader` wird vor allem vom `LoadVrmlFrame` eingesetzt, um Objekte in den Szenengraphen zu laden.

Die Klasse `BBoxAsXML` wird eingesetzt, um die Textdatei zu generieren, die in Kapitel 6.4.3 von der Klasse `OcclusionManager` benötigt wird und welche eine Aufzählung aller realen Objekte inklusive ihrer Boundingbox-Eckpunkte enthält. Hierfür implementiert `BBoxAsXML` die Methode `getFiles()`. Dieser Methode wird als Parameter der Name des Verzeichnisses übergeben, das die realen Objektmodelle enthält. Aus diesem Verzeichnis werden dann alle Dateinamen ausgelesen und es wird überprüft, ob es sich um Vrml-Modell handelt. Trifft dies zu, werden von diesem Objekt die Boundingbox-Eckpunkte berechnet und mit dem Namen der Datei in eine Textdatei geschrieben. Die Textdatei heißt „GeometrieData.txt“ und wird in das gleiche Verzeichnis geschrieben, in dem sich die Modelle befinden.

7.5 Testergebnisse

In diesem Kapitel wird die praktische Umsetzung einer Objektverdeckungsbehandlung zwischen realen und virtuellen Objekten beschrieben. Für die Realisierung wurde eine graphische Benutzeroberfläche entwickelt, mit welcher die Objektverdeckung anschaulich getestet und visualisiert werden kann. Die Oberfläche bietet die Möglichkeit, beliebige zur Verfügung stehende Objekte in einem Fenster darzustellen, in der dreidimensionalen Szene zu navigieren und bei Bedarf die Objektverdeckungsbehandlung einzusetzen. Letztere dient vor allem zur Veranschaulichung der Unterschiede zwischen der Verwendung und der Nichtverwendung einer Objektverdeckungsbehandlung.

In Abbildung 7.9 wird ein und dieselbe virtuelle Szene dargestellt. Dabei entstand das obere Bild ohne und das untere Bild mit Verwendung der entwickelten Objektverdeckungskomponente. Es handelt sich bei dem Gebäude um ein Haus auf dem Darmstädter Marktplatz. Auf dem Platz selbst steht ein Brunnen. Dieser ist rund und in seiner Mitte befindet sich eine Säule. Im unteren Bild ist gut zu erkennen, dass dieser Brunnen als reales, verdeckendes Objekt erkannt wurde und als virtuelles, verdeckendes und schwarz eingefärbtes Modell in den Szenengraphen eingefügt wurde. Hierbei ist zu beachten, dass bei Verwendung eines HMDs die reale Umgebung durch die schwarzen Flächen im Bild hindurch sichtbar wird. In dem Brunnenausschnitt erscheint dann der reale Brunnen. Damit sind die wahren räumlichen

Verhältnisse auch bei der Visualisierung der AR-Szene wieder hergestellt und der Brunnen erscheint vor dem virtuellen Objekt.

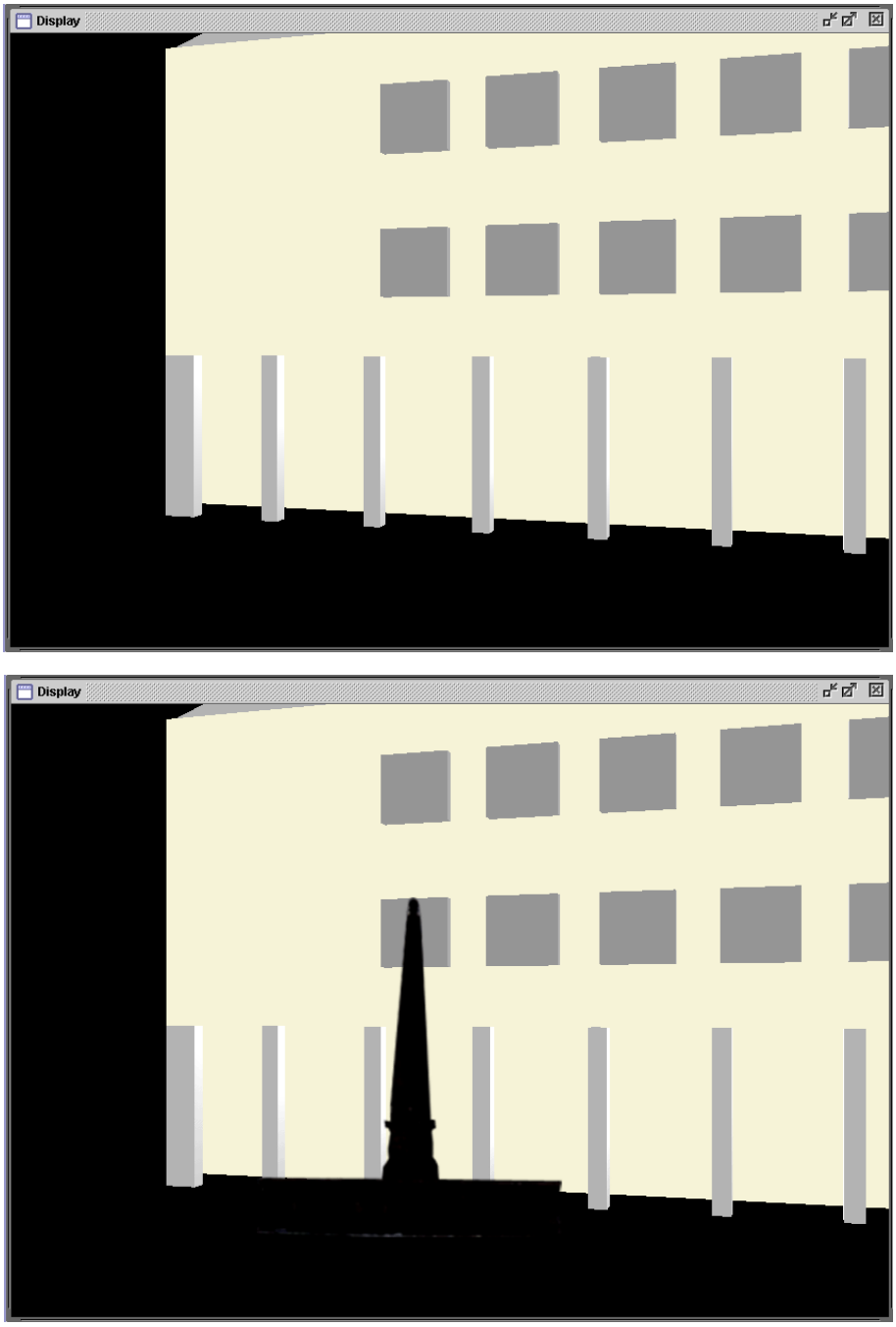


Abbildung 7.9: Vergleich einer virtuellen Szene ohne und mit Verwendung der Objektverdeckungskomponente

Die Abbildung 7.10 wurde auf gleiche Weise erstellt, allerdings aus einem anderen Sichtwinkel. Wieder wurde im oberen Bild keine Objektverdeckungskomponente

verwendet, im Gegensatz zur unteren Abbildung, in der der schwarze Brunnenausschnitt vor dem Darmstädter Rathaus gut zu erkennen ist.

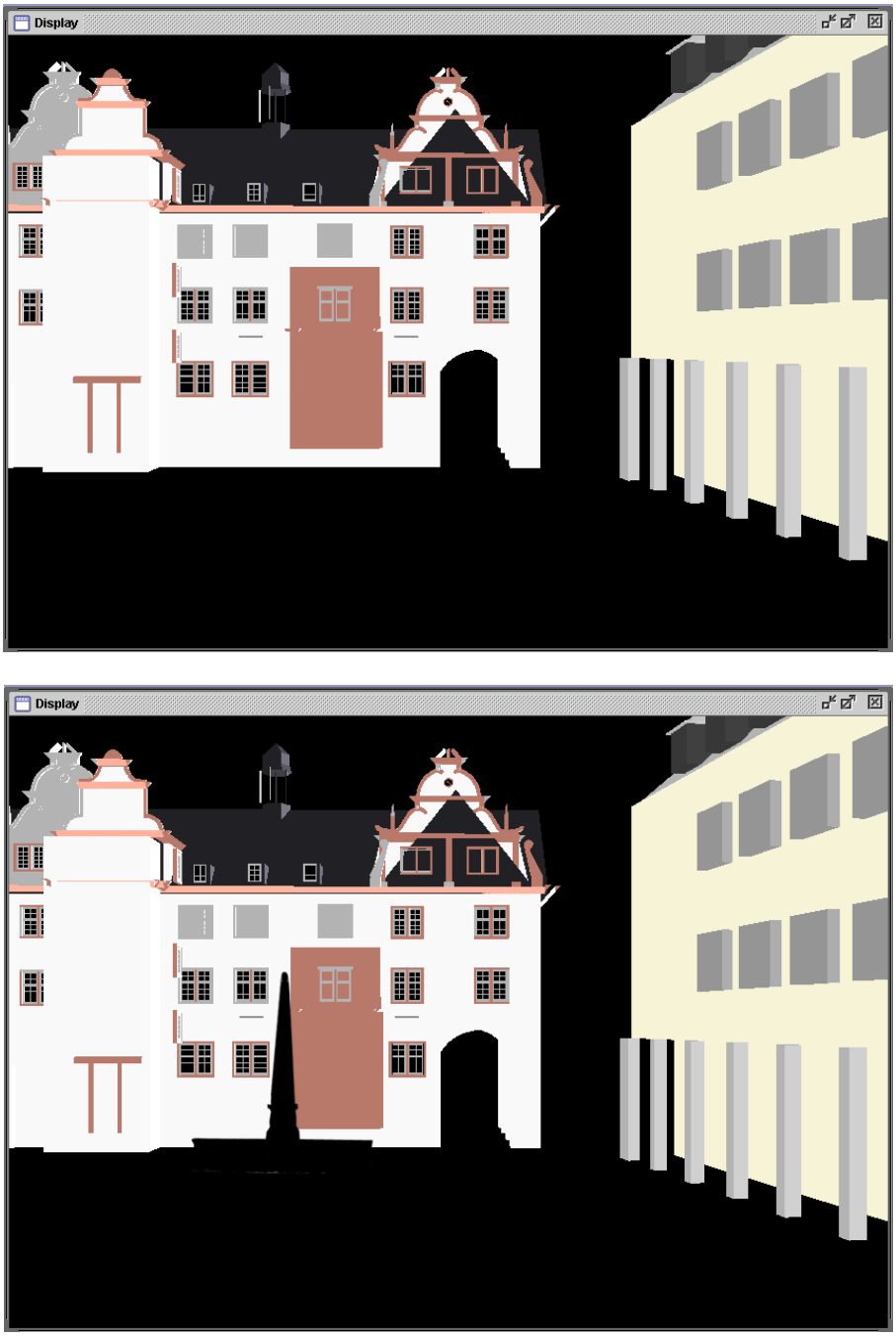


Abbildung 7.10: Vergleich einer virtuellen Szene ohne und mit Verwendung der Objektverdeckungskomponente

Bei der Entwicklung der Objektverdeckungskomponente wurde darauf geachtet, dass die in Kapitel 4 analysierten Anforderungen bei der Berechnung und Visualisie-

rung der Objektverdeckung berücksichtigt werden. Dies schließt auch ein, dass die entwickelte Objektverdeckungskomponente eine für sich gekapselte Einheit darstellt und für ihre Ausführung nicht an die in dieser Arbeit entwickelte Benutzeroberfläche gebunden ist. Sie verfügt über die in der Konzeption bereits beschriebene Schnittstelle, über welche sie in nahezu jeder Java 3D Anwendung eingesetzt werden kann, in der eine Objektverdeckungsbehandlung erwünscht sind.

Diese Schnittstelle, die Klasse OcclusionManager, benötigt zur Initialisierung der Objektverdeckungskomponente verschiedene Parameter, wie zum Beispiel Referenzen auf den ContentBranch des Szenengraphen, der verwendet werden soll, sowie auf seinen Locale-Knoten und den Transformationsknoten der ViewPlatform. Des Weiteren müssen Informationen über die Objekte der realen Umgebung in Form von Namen und Positionsdaten verfügbar sein. Über eine Methode hat der Anwender, der diese Schnittstelle in sein AR-System implementiert, die Möglichkeit den Status der Objektverdeckungsbehandlung zu beeinflussen. Weitere Methoden ermöglichen die Anpassung der in der Verdeckungsberechnung verwendeten Parameter an die Bedürfnisse des Anwenders. Hier kann zum Beispiel bestimmt werden, in welchem Abstand zur ViewPlatform Objektverdeckungen berechnet werden sollen. Auch auf die bei der Berechnung verwendeten Filter kann direkt Einfluss genommen werden. [14]

Da die Objektverdeckungsbehandlung auf dem in Kapitel 3.3 vorgestellten modellbasierten Ansatz basiert, werden bei der entwickelten Objektverdeckungskomponente reale Objekte, die virtuelle Objekte verdecken, im Szenengraph als schwarz eingefärbte und dadurch nicht sichtbare Objekte repräsentiert. Der Eindruck der visuellen Verdeckung eines virtuellen Objekts durch ein reales Objekt wird dadurch erreicht, indem der verdeckte Bereich des räumlich weiter hinten liegenden virtuellen Objekts schwarz gezeichnet wird. Dies entspricht der Farbe des Hintergrunds, wodurch in dem Bereich, in dem das 3D-Modell des realen Objekts schwarz dargestellt wird, die Objekte der realen Umgebung sichtbar werden. Für den Einsatz des entwickelten Systems ist jedoch erforderlich, dass möglichst viele der Objekte aus

[14] Für weitere Informationen siehe Kapitel 6.4.4 sowie die auf der CD-Rom im Anhang enthaltene Klassendokumentation.

der realen Umgebung, in dem das System eingesetzt werden soll, als virtuelle 3D-Geometrien vorliegen.

Zur Berechnung der visuellen Verdeckung zwischen realen und virtuellen Objekten, die innerhalb der Objektverdeckungskomponente erforderlich ist, wird nach dem in Kapitel 4 vorgestellten Prozessablauf verfahren. In diesem Prozessablauf werden zuerst optische Verdeckungen zwischen realen und virtuellen Objekten und daraus die Menge der verdeckenden Objekte berechnet. Die Ergebnismenge der verdeckenden Objekte wird dann in oben beschriebener Weise im Szenengraph visualisiert.

8 Zusammenfassung und Ausblick

Um den realistischen Eindruck der Darstellung von AR-Szenen zu verbessern, beschäftigt sich diese Arbeit mit der Berücksichtigung und Visualisierung von optischen Verdeckungen, die in AR-Systemen zwischen virtuellen und realen Objekten auftreten können. Hierfür wurde eine auf Java 3D basierte Softwarekomponente entwickelt, die reale Objekte im Szenengraph repräsentiert und dadurch die Objektverdeckung visualisiert. Um die Funktion dieser Komponente zu veranschaulichen und zu testen, wurde zudem eine graphische Benutzeroberfläche entwickelt.

Hierfür wurden die Grundlagen vorgestellt, auf welchen die Umsetzung dieser Aufgabenstellung aufbaut. Dies dient zum besseren Verständnis der Arbeit und verdeutlicht die Problematik, die sich aus der Visualisierung der Objektverdeckung ergibt. Für diese Problemstellung wurden verschiedene Lösungsansätze vorgestellt und verglichen, darunter auch der modellbasierte Ansatz, auf welchem diese Arbeit aufbaut. Ergänzend wurde ein Einblick in die Techniken gegeben, die an der Realisierung der Objektverdeckungskomponente und der graphischen Oberfläche beteiligt sind.

Die Ausarbeitung der Anforderungen an die Objektverdeckungskomponente und ihre Darstellung ist ein weiterer Bestandteil. Dabei wurde auf die verschiedenen Objekte, die maßgeblich an einer Objektverdeckung beteiligt sind, und den Aufbau des Sichtbereichs eingegangen. Aus der Analyse ihrer Eigenschaften wurde deutlich, welchen Einfluss Parameter wie zum Beispiel Verhalten, Position und Orientierung auf das Auftreten einer Objektverdeckung haben können. Die bei der Visualisierung der Objektverdeckung zu beachtenden Faktoren, wurden in einem weiteren Abschnitt über die Anforderungen, die sich an die Geometrien der verdeckenden Objekte ergeben, aufgezeigt.

Aufbauend auf diesen Anforderungen wurde ein Konzept entworfen und vorgestellt, auf dem die Umsetzung der Objektverdeckungsberechnung und ihrer Darstellung aufbaut. Diese Konzeption konnte erfolgreich durch die Implementierung in Java

und Java 3D umgesetzt werden. Eine Beschreibung dieser konkreten Umsetzung und eine Erläuterung relevanter Funktionalitäten, Klassen und Methoden fanden im weiteren Verlauf der Arbeit statt.

Hierfür wurde ein Algorithmus entwickelt, der in mehreren Filterstufen die Menge der realen Objekte berechnet, die virtuelle Objekte verdecken, wobei ein Filter jeweils auf dem Ergebnis des vorhergehenden Filters aufbaut. Die einzelnen Stufen berechnen verdeckende Objekte nach den in der Anforderungsanalyse vorgestellten Parametern wie zum Beispiel Position, Blickrichtung oder Sichtwinkel der virtuellen Kamera. Die Objekte, die der Algorithmus als verdeckende Objekte erkennt, werden im Szenengraph der AR-Anwendung indirekt visualisiert, indem sie der Hintergrundfarbe der Szene entsprechend eingefärbt und dargestellt werden. Dadurch erscheint es so, als ob aus den dargestellten virtuellen Objekten Bereiche ausgeschnitten werden, durch welche der Anwender wieder die reale Umgebung sehen kann.

Um zu demonstrieren, dass der erarbeitete und implementierte Algorithmus sowie die gesamte Objektverdeckungskomponente nutzbringend und ohne großen Implementierungsaufwand in AR-Systemen eingesetzt werden können, wurde zusätzlich für Testzwecke und zur Demonstration eine auf Swing basierte, graphische Benutzeroberfläche entwickelt. Die Performanzfähigkeit der Objektverdeckungskomponente und der Benutzeroberfläche ist aufgrund der Verwendung von Swing etwas eingeschränkt und größtenteils abhängig von der Leistungsfähigkeit der verwendeten Hardware.

Hervorzuheben ist insbesondere die einfache Implementierung der entwickelten Objektverdeckungskomponente. Da sie als eine Einheit konzipiert und entwickelt wurde, die unabhängig vom umgebenden System arbeitet, sind ihre Einsatzmöglichkeiten sehr vielfältig und nicht von einem einzelnen System abhängig. Die Komponente greift dabei auf Daten zu, die ihr einmalig bei ihrer Initialisierung über eine Schnittstelle zur Verfügung gestellt werden. Dabei handelt es sich um Datenfelder, welche in den meisten 3D-Anwendungen ohnehin vorhanden sind, weswegen auch dies keinen weiteren Aufwand für den Einsatz dieser Komponente bedeutet.

Ein wesentliches Ergebnis dieser Diplomarbeit ist jedoch, dass in vielen auf Echtzeit basierten AR-Systemen der realistische Eindruck der mit virtuellen Informationen überlagerten realen Umgebung durch den Einsatz der Objektverdeckungskomponente verbessert werden konnte. Das Räumlichkeitsempfinden des AR-Benutzers folgt wieder den aus der Realität bekannten geometrischen Beziehungen zwischen unterschiedlichen Objekten. Die überlagernden virtuellen Objekte und Informationen werden wesentlich besser, als es vor Durchführung dieser Arbeit möglich war, an die reale Umgebung angepasst und in deren Beschaffenheit integriert.

Ein möglicher Nachteil bei der Funktionsweise der Objektverdeckungsbehandlung ist lediglich, dass sie nur in Systemen eingesetzt werden kann, die über ein flächendeckendes, präzises Modell der realen Umgebung verfügen. Ist das nicht der Fall, können Verdeckungen weder berechnet noch dargestellt werden, da keine Berechnungsgrundlage vorliegt. Ein zweiter Nachteil kann sich möglicherweise durch mangelnde Rechenleistung der im AR-System verwendeten Hardware ergeben. Da dies jedoch mit zunehmender technischer Weiterentwicklung der Hardware-Komponenten immer weniger zum Tragen kommt, ist durch die hier entwickelte Objektverdeckungskomponente ein entscheidender Beitrag für den zunehmenden Einsatz von Outdoor-AR-Systemen geleistet worden.

Die Einsatzmöglichkeiten für das entwickelte System sind breit gefächert: Zum Beispiel kann die Komponente von Architekten bei städtebaulichen Maßnahmen eingesetzt werden, um zu visualisieren wie sich ein in der Planung befindliches Gebäude in die Stadtumgebung integrieren lässt. Auch die Unterhaltungs- und Tourismusbranche bietet ein weites Feld an Einsatzmöglichkeiten. Hier können tragbare AR-Systeme eingesetzt werden, um den Benutzern zusätzliche Informationen wie zum Beispiel Rekonstruktionen von Gebäuden anzeigen zu können. Gerade bei solchen Anwendungen sollte sich das überlagernde, virtuelle Objekt nahtlos mit der Realität verbinden. Dies alles ist durch die Entwicklung der Objektverdeckungskomponente erst wirklich möglich geworden.

9 Literaturverzeichnis

[Adam03] Christian Adam:
Entwicklung einer Autorenumgebung für komponentenbasierte Mixed Reality Anwendungen.
Bachelorarbeit. Fachbereich Informatik, Fachhochschule Darmstadt, Darmstadt, 2003.

[Azum97] Ronald T. Azuma:
A Survey of Augmented Reality.
In: Presence: Teleoperators and Virtual Environments 6, August 1997, S. 355-385.

[Azum99] Ronald T. Azuma:
The Challenge of Making Augmented Reality Work Outdoors.
In: Yuichi Ohta and Hiedyuki Tamura (Ed.): Mixed Reality: Merging Real and Virtual Worlds. Springer-Verlag, 1999, S. 379-390.

[Berg97] Marie-Odile Berger:
Resolving Occlusion in Augmented Reality: A Contour Based Approach without 3D Reconstruction.
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1997, S. 91-96.

[Bree95] David E. Breen, Eric Rose, Ross T. Whitaker:
Interactive Occlusion and Collision of Real and Virtual Objects in Augmented Reality.
Technischer Bericht ECRC-95-02.
European Computer-Industry Research Centre GmbH, München, 1995.

[Frei03] Thorsten Frei:
Analyse und Konzeption einer Autorenumgebung für MR-Anwendungen.
Diplomarbeit. Fachbereich Biologie und Informatik, Professur für Graphische Datenverarbeitung, Johann Wolfgang Goethe-Universität Frankfurt am Main, Frankfurt am Main, 2003.

[Fuhr99] Anton Fuhrmann, Gerd Hesina, François Faure, Michael Gervautz:
Occlusion in Collaborative Augmented Environments.
Proceedings 5th Eurographics Workshop on Virtual Environments, Springer-Verlag, Wien, 1999, S. 179-190.
ftp.cg.tuwien.ac.at/pub/TR/98/TR-186-2-98-29Paper.pdf

[Goll00] Joachim Goll, Cornelia Weiß, Peter Rothländer:
Java als erste Programmiersprache. Java 2 Plattform.
2. Auflage, Verlag B. G. Teubner, Stuttgart/Leipzig/Wiesbaden, 2000.

[Grau99] Oliver Grau, Hellward Broszio:
Vorstudie ueber Potentiale und Chancen zur Entwicklung Hannovers als VR-Standort.
Studie im Auftrag der Technologie-Centrum Hannover GmbH.
Hannover, April 1999.

[Klin98] Gudrun Klinker, Didier Stricker, Dirk Reiners :
The Use of Reality Models in Augmented Reality Applications.
In: Reinhard Koch, Luc Van Gool (Ed.): European Workshop on 3D Structure from Multiple Images of Large-scale Environments (SMILE ‘98), Springer-Verlag Berlin, Heidelberg, 1998.

[Klin00] Gudrun Klinker:
Augmented Reality: A Problem in Need of Many Computer Vision-Based Solutions.
Confluence of Computer Vision and Computer Graphics.
Kluwer Academic Publishers, 2000, S. 267-284.

[Krue01] Guido Krüger:
GoTo Java 2. Handbuch der Java-Programmierung.
2. Auflage, Addison Wesley Verlag, München, 2001.

[Lepe00] Vincent Lepetit, Marie-Odile Berger:
Handling Occlusion in Augmented Reality Systems: A Semi-Automatic Method.
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2000, S. 225-230.

[Ludw02] Manfred Ludwig:
Mathematische Grundlagen der Visualisierung.
Skript. Fachrichtung Mathematik, Technische Universität Dresden. Dresden, 2002.

[Lutz00] Monika Lutz, Sonja Emmel:
Clixx Java 3D. Der Grundkurs.
Verlag Harri Deutsch, Frankfurt am Main, 2000.

[Mali02] Shahzad Malik:
Robust Registration of Virtual Objects for Real-Time Augmented Reality.
Masterarbeit. Ottawa-Carleton Institute for Computer Science, Carleton University, Ottawa, Ontario, Canada, 2002.

[Rueg99] Ingrid Rügge:
Alternativen zum Desktop: Virtual, Augmented und Real Reality.
Frauenarbeit und Informatik, 19/1999, S. 17-22.

[Sapos] Satellitenpositionierungsdienst der deutschen Landesvermessung.
http://www.sapos.de

[Selm02] Daniel Selman:
Java 3D Programming.
Manning Publications, Greenwich, Connecticut, USA, 2002.

[SUN-1] SUN Microsystems – Java Technology
http://java.sun.com

[SUN-2] Java Foundation Classes (JFC/Swing)
http://java.sun.com/products/jfc/

[SUN-3] Java 3D
http://java.sun.com/products/java-media/3D/collateral/

[Thie03] Frank Thiemann:
3D-Gebäude-Generalisierung.
Deutsche Gesellschaft für Kartographie, Kartographische Schriften,
Band 7: Visualisierung und Erschließung von Geodaten.
Beiträge des Seminars GEOVIS 2003, 27.-28. Februar 2003,
Hannover, S. 185-192.

[Ulbr02] Christiane Ulbricht:
Tangible Augmented Reality für Computerspiele.
Diplomarbeit. Institut für Softwaretechnik und Interaktive Systeme
der Technischen Universität Wien, Wien, 2002.

[UniT03] Jörg Ihringer:
Skript zur Vorlesung Experimental Physik 1. Das Auge.
Institut für Angewandte Physik, Universität Tübingen,
http://www.uni-tuebingen.de/uni/pki/skripten/V8_2_4Auge.doc,
Tübingen, 2003.

[Univers] Universal-Lexikon.
Mohndruck GmbH, Gütersloh, 2003.

[Wlok95] Matthias M. Wloka, Brian G. Anderson:
Resolving Occlusion in Augmented Reality.
ACM Symposium on Interactive 3D Graphics Proceedings.
New York, 1995, S. 5-12.

Anhang – Die CD zur Diplomarbeit

Die CD zu dieser Diplomarbeit enthält in einem Ordner den gesamten Quellcode der Software, die in dieser Arbeit entwickelt wurde. Dabei entspricht der Unterordner „occlusion“ dem Klassenpaket, das die Objektverdeckungskomponente enthält. Weitere Unterordner enthalten die Klassen der Benutzeroberfläche und der Szenengraphstruktur.

Zudem enthält die CD die gesamte Klassendokumentation. Sie wurden mit dem javadoc-Werkzeug, das vom Java Development Kit zur Verfügung gestellt wird, erstellt. Das Programm generiert aus Java-Quelltexten Dokumentationen im HTML-Format und verwendet hierfür Klassen-, Attribut- und Methodendeklarationen. Zusätzlich fügt es Informationen zu dem Quelltext aus Quelltextkommentaren hinzu.

Ferner enthält die CD die Kurzfassung sowie die gesamte Diplomarbeit im PDF- und im Microsoft-Word-Format.