Virtuelle Realität

Prof. Dr.-Ing. Reinhard Schmidt
Fachhochschule Esslingen
Hochschule für Technik
Fachbereich Informationstechnik
Flandernstraße 101
D-73732 Esslingen

Wichtiger Hinweis:
Das Skript ersetzt nicht den Besuch der Vorlesung. Es gibt auch nicht den vollständigen Inhalt der Vorlesung wider. Das Skript dient dazu, den Hörer während der Vorlesung von der Arbeit der Mitschrift weitgehend zu entlasten.

Titelbild:
Die Titelseite zeigt einen virtuellen Akteur in Gestalt von Marilyn Monroe. Die Simulation wurde am MIRLab der University of Geneva, unter der Leitung von Prof. Dr. Nadja Thalmann, durchgeführt.

Das Bild links, zeigt das Original.

Esslingen, im September 2004.

Vorwort

Diese Vorlesung gibt eine Einführung in die neue Schlüsseltechnologie “Virtuelle Realität”. Die Vorlesung ist nach dem Bottom-Up-Prinzip aufgebaut.

Zunächst werden wichtige Definitionen und Begriffe der Virtuelle Realität erläutert und industrielle Anwendungen vorgestellt. Anschließend werden Systeme zur Visualisierung von dreidimensionalen Szenen und Systeme zur Navigation innerhalb von dreidimensionalen Szenen sowie zur Interaktion mit dreidimensionalen Szenen besprochen.

Virtuelle Realität basiert zum großen Teil auf Computergraphik. Die Vorlesung zeigt im nächsten Teil einige Basisalgorithmen der Computergraphik auf. Es wird erläutert, wie dreidimensionale Körper modelliert, wie daraus dreidimensionale Szenen aufgebaut werden und wie schließlich die Farbgebung für eine dreidimensionale Szene erfolgt.

Eine Einführung in die standardisierte Beschreibungssprache “Virtual Reality Modelling Language” (VRML) ergänzt die vorangegangenen Vorlesungsteile und zeigt Anwendungen für das Internet auf. Im letzten Teil werden weiterführende Probleme diskutiert, wie beispielsweise Aufbau und Animation von virtuellen Akteuren, sowie die Konfiguration von virtuelle Welten für Mehrbenutzeranwendungen.

Lernziele in Stichworten:
Aufbau von VR-Systemen, geometrische Modellierung von dreidimensionalen Körpern, Algorithmen der Computergraphik, Verfahren zur Visualisierung von virtuellen Umgebungen, Programmierung virtueller Umgebungen, Techniken der Animation, virtuelle Realität für Mehrbenutzer-Anwendungen.

Inhaltsverzeichnis

Teil I

Grundlagen

Kapitel 1

Einführung in die virtuelle Realität

Der Begriff “Virtual Reality” wurde 1989 von Jaron Lanier geprägt. Dieser Begriff ist zwar sehr vielen bekannt, die wenigsten wissen aber, was es sich hinter diesem Begriff verbirgt. Virtual Reality entsprang zunächst der Phantasie von Science-Fiction-Autoren, inzwischen hat sich daraus eine neue Schlüsseltechnologie entwickelt. Sucht man nach einer Erklärung für den Begriff Virtual Reality (deutsch: Virtuelle Realität, VR), so findet man beispielsweise in: “The Silicon Mirage, The Art and Science of Virtual Reality”, [4], [5], der beiden VR-Experten Steve Aukstakalnis und David Blatner die folgende Antwort.

Virtual Reality

Virtuelle Realität

```
"Virtuelle Realität ermöglicht es Menschen, äußerst komplexe
 Datenmengen sichtbar zu machen, sie zu manipulieren und mit
 dem dazu gehörigen Rechnern zu interagieren. So einfach ist
 das. Das Handeln von Menschen und Maschine berührt sich an
 den so genannten Schnittstellen, und virtuelle Realität ist
 nur die vorläufig letzte und modernste einer langen Reihe
 von Schnittstellen. Also was ist daran so besonders? Wieso
 machen Presse und Fernsehen so ein Aufhebens um eine simple
 Schnittstelle?"
```

Beide Autoren geben eine sehr anschauliche, technische Definition für den Begriff virtuelle Realität an. Vom Standpunkt der Computertechnik aus betrachtet, ist virtuelle Realität eine logische und evolutionäre Weiterentwicklung der Mensch-Maschine-Schnittstelle. Die folgenden Tabelle zeigt die Evolutionsstufe.

Mensch-Maschine-Schnittstelle

Tabelle 1.1: Evolutionsstufen der Mensch-Maschine-Schnittstelle

Stufe 1	Stufe 2	Stufe 3
Mainframe	Desktop-PC	Client/Server
zentral	lokal	dezentral
Text	graphische Oberfläche	3D-Szenerie, Datenraum
Tastatur	Maus	Datenhandschuh, Space-Mouse
ASCII-Terminal	XWindow-Terminal	Datenhelm, CAVE
tippen	klicken	navigieren, interagieren

Die kreative Idee für den Entwicklung der virtuellen Realität scheint auf den ureigenen Drang der Menschheit zurückzugehen, in neue Welten aufzubrechen und diese zu erforschen. Betrachtet man den Verlauf der Geschichte, so spiegelt sich der Drang diese Neugierde zu stillen ständig wider.

Unter diesem Aspekt kann das Ziel der virtuellen Realität folgendermaßen verstanden werden. Virtuelle Realität verfolgt das Ziel, reale oder künstliche Welten, mit allen ihren Gesetzen, in den Computer abzubilden und diese Welten möglichst wirklichkeitsnah darzustellen und erlebbar zu machen.

Die Möglichkeit, eine vom Rechner erzeugte, fremde Welt erleben und in diese Welt eingreifen zu können, beflügelt die Phantasie der Menschen und verleiht dieser neuen Technologie eine besondere Faszination.

Aufgrund der neuartigen Möglichkeiten zur Visualisierung von Informationen und der einfachen Interaktion mit dem steuernden Rechnersystem leistet die virtuelle Realität einen riesigen Innovationssprung in der Simulationstechnik. Viele Experten zählen deshalb die virtuelle Realität zu den modernen Schlüsseltechnologien.

Cyberspace

Ein weiterer Begriff, der im Zusammenhang mit virtueller Realität häufig gebraucht wird, ist das Wort “Cyberspace”. Dieses Wort geht auf den Romanautor Wiliam Gibson zurück. In seinem Roman “Neuromancer” verwendet er erstmalig den Begriff Cyberspace für eine im Computer existierende Welt. Der Begriff Cyberspace wurde von den VR-Entwicklern für diese neue Technologie übernommen.

1.1 Industrielle Anwendungen von virtueller Realität

Die folgende Zusammenstellung zeigt einige Beispiele für industriellen Einsatz der virtuellen Realität. Zwar haben einige der genannten Anwendungen aufgrund noch immer einen projektartigen Charakter, inzwischen gibt es aber ein sehr breites industrielles Anwendungsspektrum.

- Militärtechnik
  Militärtechnik zählt zwar nicht zu den üblichen industriellen Anwendungen, jedoch war eine der ersten großen Anwendungen der VR-Technologie - wie auch so viele andere Technologien vorher - militärischer Art und Weise. Die US-Army verfügt derzeit über die beste VR-Ausstattung. Für die Fahrsimulation mit Panzerfahrzeugen steht ein über 15 Quadratkilometer großes virtuelles Kampfgebiet zur Verfügung. Diese Anwendung wird als (Close Combat Tactical Trainer, CCTT) bezeichnet. Das CCTT-Projekt ist bis zum heutigen Tag die größte und komplexeste virtuelle Umgebung. Sie wurde vom amerikanischen Verteidigungsministerium gefördert. Der gigantische Aufwand wird mit mehreren hundert vernetzten Hochleistungsrechnern bewerkstelligt. Das Training von Kampfjet-Piloten ist eine weitere Anwendung.

  Close Combat Tactical Trainer

- Architektur und Stadtplanung
  In der Architektur und Stadtplanung werden seit einiger Zeit VR-Systeme eingesetzt. Architekten und Bauherren können das Bauvorhaben bereits vorab begutachten. Beispiele hierfür sind die Planungen des Regierungsbezirk für Berlin und die Verlegung des Stuttgarter Hauptbahnhofs unter die Erdoberfläche.

- Interieur-Design
  Für die Planung von Inneneinrichtungen oder Arbeitsumgebungen bieten sich VR-Systeme an. Die Anwendungen reichen hier von der Auswahl der Einbauküche bis zum ergonomischen Design und umfassen auch bauphysikalische Simulationen.

- Medizintechnik
  In der Medizintechnik ergeben sich viele neue Möglichkeiten bei der Diagnose, der Operationsplanung und der Ärzteausbildung. Die Kombination von realen Daten, z.B. Computer-Tomographie-Aufnahmen, mit berechneten, synthetischen Daten bietet neue Möglichkeiten zur Diagnose und zur Operationsplanung. Eine weitere Anwendung ist beispielsweise das Training von endoskopische Untersuchungen. Ärzten üben endoskopische Eingriffe zunächst an virtuellen Patienten.

- Automatisierungstechnik
  Die Planung von Industrieanlagen kann mit Hilfe von virtueller Realität erfolgen. Nicht nur die bauliche Planung kann mit Hilfe des Computers durchgeführt werden, sondern eine virtuelle Anlage kann vorab nach funktionellen, logistischen und wirtschaftlichen Kriterien überprüft werden. Sogar Schulungen, wie die Anlage später zu bedienen ist oder wie im Falle eines Fehler zu reagieren ist, sind für das Fachpersonal vorab möglich.

- Robotik
  Die Programmierung von Industrierobotern kann in virtueller Umgebung erfolgen. Mit Hilfe eines Datenhandschuhs oder anderen Eingabemedium werden die Bahnbewegungen des Roboterarmes simuliert. Die vom Rechnern gespeicherten Bahn- und Bewegungsdaten können an den Roboter übertragen werden.

- Rapid Prototyping
  Virtuelle Realität unterstützt eine schnelle Produktentwicklung. Die Gestaltgebung und die Funktionsüberprüfung lassen sich in einer virtuellen Umgebung sehr einfach und schnell durchführen. In der heutigen Zeit sind Produkte schnelllebiger denn je und

die Produktzyklen werden immer kürzer. Ein rasches Prototyping wird für den Erfolg einer Firma immer wichtiger.

- Wissensvermittlung
  Zur Informations- und Wissensvermittlung ist virtuelle Realität hervorragend geeignet. Durch die Möglichkeit des direkten Erlebens des Lernstoffes wird das Interesse und die Motivation am Lernstoff sehr stark gesteigert. Es gibt inzwischen zahlreiche VR-Anwendungen für Unterrichtszwecke. Projekte, die in diesem Zusammenhang Schlagzeilen gemacht haben, sind beispielsweise ein virtueller Rundgang durch das antike Pompeij (Firma SIMLAB, USA) oder der Besuch eines virtuellen Wikinger-Dorfes (Telecom Research Norwegen).

- Entertainment
  Die Spieleindustrie war neben der Militärtechnik eine der ersten Industriezweige, die die virtuelle Realität als Marktpotential für sich entdeckte. In Spielhallen findet man häufig Fahr- und Flugsimulatoren, Box- und Schießspiele. Daneben gibt es inzwischen zahlreiche Computerspiele mit Interaktionsmöglichkeiten.

- Telepräsenz
  Mit Hilfe ferngesteuerter Roboter können gefährliche oder für den Menschen schwer zugängliche Orte erkundet oder anfallende Arbeiten ausgeführt werden. Anwendungsbeispiele sind hier Atomkraftwerke, auch das Abwasserkanalsystem einer Stadt oder Experimente im Weltraum .

- Virtuelle Fernsehstudios
  Einige Fernsehsender zeichnen inzwischen Sendungen in einem virtuellen Studio auf. Ein virtuelles Studio besteht aus einem leeren Raum, dessen Wände mit einem kräftigen Blau ausgekleidet sind. Virtuelle Studios werden als “Blue Box” bezeichnet. Bei der Aufzeichnung befindet sich lediglich der Moderator in der Blue Box. Die verschiedenen Bühnenbilder, so genannte “Studio-Sets”, und Interviewpartner werden von einem Rechner hinzugefügt. Der Vorteil besteht darin, dass Bühnenbilder sehr schnell geändert werden können. Außerdem werden Spezialeffekten möglich, die mit herkömmlicher Studiotechnik bisher nicht machbar waren.

Blue Box

- Cooperatives Arbeiten in einer virtuellen Umgebung
  Beim kooperativen Arbeiten in einer virtuellen Umgebung (engl.: Computer Supported Collaborative Working, CSCW) treffen sich mehrere Experten im Cyberspace. Obwohl sie sich an verschiedenen Orten befinden, können sie gemeinsam an einem Projekt arbeiten. Derartige Anwendungen befinden sich zwar noch im Forschungsstadium, die technischen Möglichkeiten sind bereits vorhanden. Bei derartigen Anwendungen findet eine Verknüpfung von Videokonferenz und virtueller Realität statt. Beispiele sind hier, mehrere Konstrukteure entwickeln ein CAD-Modell oder ein Ärzteteam diagnostiziert einen virtuellen Patienten.

Beispiele für aktuelle Forschungsschwerpunkte sind:

- Virtuelle Akteure, Leben im Cyberspace
  Virtuelle Akteure verleihen VR-Anwendungen eine neue Dimension - der Cyberspace wird bevölkert. Virtuelle Akteure können als Lehrer, Diskussions- oder Spielpartner auftreten. Bei einem Besuch in einem virtuellen Wikinger-Dorf könnte man einen virtuellen Wikinger durch sein Dorf geführt werden und ihn nach seinem Alltag und seinen Lebensumständen befragen.

- Systeme zur Vermittlung des Tastgefühls
  Nach wie vor ist es ein großes Manko, dass in virtuellen Welten das Spüren von wirkenden Kräften fehlt, wie beispielsweise das Tastgefühl beim Greifen eines virtuellen

Objektes. Inzwischen gibt es prototypische Lösungsansätze zur Vermittlung des Tastgefühls, diese sind jedoch mehr oder weniger rudimentär.

- Alternative Systeme zur Bildwiedergabe
  Eine neue Technologien zur stereoskopen Visualisierung befasst sich mit der direkten Laserprojektion auf die Netzhaut des menschlichen Auges. Weitere Forschungsthemen sind die stereoskope Bildwiedergabe mittels Laser auf großflächigen Projektionswänden oder die holographische Projektion, ohne dass ein Hilfsmittel zur stereoskopen Betrachtung notwendig ist, z.B. Shutterglasses.

- Systeme zur Vermittlung von Gerüchen
  Die Übermittlung von Gerüchen ist ein sehr schwieriges und kompliziertes Problem. Ein Durchbruch auf diesem Gebiet ist momentan nicht zu erwarten, es würde aber eine neue, weitere Dimension bei VR-Anwendungen eröffnen.

1.2 Typisierung von VR-Anwendungen

virtuelle Welt

virtuelle Umgebung

Virtuelle Realität bietet die Möglichkeit, Abläufe, wie sie in der realen Welt geschehen, mit Hilfe des Computers zu simulieren. Die Gesetze, die diese Abläufe kontrollieren, müssen in Form von Regeln im Computer vorliegen und die Welt in Form eines geeigneten Modells gespeichert werden. Im Verlauf der Simulation wird ständig der aktuelle Zustand der virtuellen Welt berechnet und das Abbild realitätsnah visualisiert. Der Begriff "Welt" ist dabei sehr weit gefasst. Es kann stets nur ein kleiner Ausschnitt einer realen Welt modelliert werden. Aus diesem Grunde ist die Bezeichnung "virtuelle Umgebung" ebenfalls gebräuchlich. In den meisten Fällen handelt es sich um hoch komplexe technische Systeme, deren Verhalten simuliert wird.

Der Anwender hat die Möglichkeit, sich mit einem Navigationsmittel in der virtuellen Welt zu bewegen und damit den Betrachtungsstandpunkt zu verändern. Er kann weiterhin Eingriffe, so genannte Interaktionen, in der virtuellen Welt vornehmen. Dazu muss der Anwender dem zugehörigen Rechnersystem Eingriff mittel eines Eingabegerätes mitteilen können. Der Anwender kann damit eine Änderung des Zustandes der virtuellen Welt veranlassen und die Folgen dieser Zustandsänderung unmittelbar von allen Seiten betrachten. Diese Eigenschaften bewirken die Faszination der VR-Technologie.

Die wesentlichen Eigenschaften einer VR-Anwendung sind:

Interaktion

- Interaktion
  Der Anwender kann sich innerhalb der virtuellen Welt bewegen, mit ihr kommunizieren und Interaktionen veranlassen.

Simulation

- Simulation
  Das Verhalten der virtuellen Welt wird zu jedem Zeitpunkt berechnet.

Animation

- Animation
  Das Erscheinungsbild der virtuellen Welt wird visualisiert und gegebenenfalls wird durch akustische Eindrücke das Erscheinungsbild verstärkt.

1.2.1 Interaktion

Aus der Sicht des Anwenders kann man für VR-Anwendungen drei Ebenen der Interaktion [4] angeben. Je größer der Grad der Interaktion ist, desto größer ist auch der Realisierungsaufwand.

passive Ebene

- Passive Ebene
  Die passive Ebene ist die unterste Ebene. Der Anwender kann die virtuelle Welt nur passiv erleben. Er bekommt die virtuelle Welt vorgeführt, etwa mittels eines als Video aufgezeichneten Fly-Through oder Walk-Through.

aktive Ebene

Navigation

- Aktive Ebene
  Die nächste höhere Ebene ist die aktive Ebene. Der Anwender kann sich in der virtuellen Welt bewegen. Er kann sich mit Hilfe eines geeigneten Eingabegerätes durch die virtuelle Welt navigieren, der Anwender kann aber nicht oder nur geringfügig verändernd in die Welt eingreifen.

interaktive Ebene

- Interaktive Ebene
  Die interaktive Ebene ist die höchste Ebene. Auf dieser Ebene kann der Anwender die virtuelle Welt nicht nur erleben sondern sie auch interaktiv verändern.

1.2.2 Simulation

Der zeitliche Ablauf der Simulation kann je nach Aufgabenstellung mit unterschiedlichen Geschwindigkeitensebenen erfolgen.

- Echtzeitverarbeitung
  Die Geschwindigkeit der Simulation erfolgt in Echtzeit.
  Echtzeit-verarbeitung

- Zeitraffer
  Die Geschwindigkeit der Simulation erfolgt schneller als in Echtzeit.
  Zeitraffer

- Zeitlupe
  Die Geschwindigkeit der Simulation erfolgt langsamer als in Echtzeit.
  Zeitlupe

1.2.3 Animation

Die Animation kann bei VR-Anwendungen in folgende Gruppen unterteilt werden.

- Immersive Reality
  Immersive Reality
  Meistens steht der immersive[1] Charakter einer Anwendung im Vordergrund, wenn von virtueller Realität die Rede ist. Der Anwender trägt dabei einen Datenhelm, der ein Stereobild vor den Augen des Anwenders darstellt. Gegebenenfalls wird durch eine 3D-Audioausgabe der Erlebniseffekt verstärkt. Die Navigation und die Interaktion kann über einen Datenhandschuh erfolgen. Durch Handgesten, die über den Datenhandschuh an den Steuerrechner gemeldet werden, wird die Navigation gesteuert bzw. die Interaktion veranlasst. Die Position des Anwenders wird durch ein Tracking-System erfasst und ebenfalls dem Steuerrechner mitgeteilt. Der Steuerrechner berechnet daraufhin die resultierende Animation.

  Eine aufwendigere Art der Visualisierung ist mit Fish-Tank- oder CAVE-Systemen[2] möglich. Der Anwender befindet sich in diesem Fall in einem Projektionsraum, in dem er sich frei bewegen kann. Bei einem CAVE- System hat der Raum die Form eines Würfels. Mit Hilfe mehrerer Projektoren wird die virtuelle Welt auf die Wandflächen projiziert. Die Position des Anwenders wird durch ein Tracking-System erfasst, er kann mit Hilfe eines free-flying Joysticks navigieren.

- Augmented Reality
  Augmented Reality
  Die kombinierte Visualisierung einer virtuellen Welt mit einer realen Welt bezeichnet man als Augmented Reality[3]. Dabei werden gespeicherte reale Daten mit berechneten virtuellen Daten gemischt. Es erfolgt quasi eine Überlagerung des Bildes der realen Welt mit dem Bild der virtuellen Welt. Der Anwender nutzen beide Datensätze als Information. Anwendungen findet man beispielsweise in der Medizintechnik.

- Telepresence
  Telepresence
  Umweltdaten, die mit Hilfe von Remote-Sensoren in einer entfernten, aber realen Umgebung aufgezeichnet werden, werden so aufbereitet und visualisiert, dass der Anwender den Eindruck erhält, er befinde sich in dieser entfernten Umgebung. Bei vielen Anwendungen handelt es sich häufig nur um die Videoübertragung einer mobilen, steuerbaren Kamera (Kameraroboter), aber auch die Übertragung und Auswertung von Audiosignalen, Temperaturwerten oder Drücken ist vorstellbar.

[1] Der Begriff “Immersive Reality” bedeutet, dass der Anwender in die in virtuelle Welt eintaucht (englisch: immerse = eintauchen).
[2] Die Abkürzung CAVE steht für Cave Automatic Virtual Environment
[3] Der Begriff “Augmented Reality” bedeutet, dass der Anwender eine gesteigerte Realität erlebt (englisch: augmented = gesteigert).

1.3 Aufbau eines VR-Systems

VR-Systeme sind Computersysteme mit einem sehr leistungsfähige Graphiksubsystem, häufig handelt es sich um Mehrprozessorsysteme. Das folgende Bild 1.1 zeigt den prinzipiellen Aufbau eines VR-Systems.

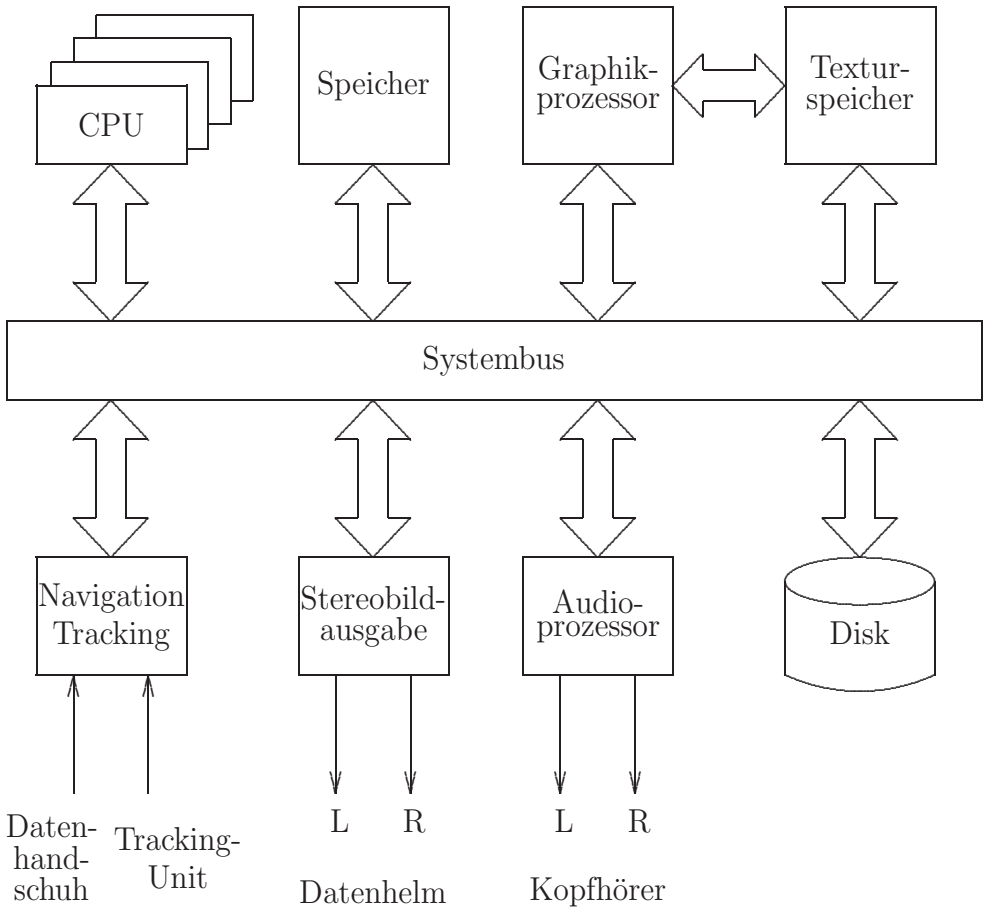

Abbildung 1.1: Aufbau eines VR-Systems

An ein VR-System werden folgende Anforderungen gestellt:

- 3D-Hochleistungsgraphik
  An das Graphiksubsystem werden sehr hohe Anforderungen gestellt. Die Hardware des Graphiksubsystmes sollte möglichst viele verschiedene Algorithmen der Computergraphik eigenständig, d.h. unabhängig vom Prozessor, abarbeiten können. Eine parallel Verarbeitungsweise ist ebenfalls wünschenswert. Einige Beispiele für Algorithmen, die ein Graphiksubsystem eigenständig durchführen können sollte, sind:
  - Rasterung und Antialising,
  - geometrische Transformationen und Projektionen,
  - Entfernung nicht sichtbarer Objektteile aus der Verarbeitung,
  - Entfernung nicht sichtbarer Polygonstücke aus der Verarbeitung,
  - Shattierung von Dreiecken und Polygonen,
  - Projektion von Texturen auf Polygone.

Texturspeicher

- Texturspeicher
  Im Texturspeicher werden Texturen gespeichert, die vorab generiert oder eingescannt wurden. Dabei handelt es sich meistens um reale Texturen. Diese Texturen werden auf Dreiecke oder Polygone projiziert werden. Dieser Vorgang wird als Texture Mapping bezeichnet. Durch das Texture Mapping kann einer virtuellen Umgebung sehr einfach ein realistisches Aussehen verliehen werden. Werden mehr Texturen benötigt, als im Texturspeicher Platz finden, so werden diese bei Bedarf aus dem Arbeitsspeicher oder von der Festplatte nachgeladen. Das Graphiksubsystem sollte über einen schnellen Zugriff auf den Texturspeicher verfügt, der unabhängig vom Bussystem arbeitet.

- Multiprozessorsystem
  Bei komplexen Echtzeitsimulationen oder aufwendigen Bildsyntheseverfahren ist es empfehlenswert, ein Multiprozessorsystem zu verwenden. Verschiedene Aufgaben, die bei der Simulation anfallen, können dann auf verschiedene Prozessoren verteilt werden.

- Sterobildausgabe
  Der VR-Rechner sollte in der Lage sein, mindestens 30 Bilder pro Sekunde für jeden Stereokanal (linkes und rechtes Auge) zu berechnen., ansonsten erscheinen die Bildsequenzen zu ruckhaft. An die Schnittstelle der Stereobildausgabe kann ein Datenhelm, stereofähiger Monitor oder Beamer angeschossen werden.

Sterobild

- Tracking
  Über eine so genannte “Tracking Unit” wird die Position des Anwenders erfasst und an den VR-Rechner übermittelt. Das VR-System berechnet dann den Bildausschnitt für die jeweilige Anwenderposition neu. Eine Tracking Unit wird üblicherweise an eine serielle Schnittstelle angeschlossen.

Tracking

- Navigation
  Für die Navigation durch eine virtuelle Umgebung werden die Eingabedaten eines Datenhandschuhs, free-flying Joysticks oder einer Spacemouse über eine serielle Schnittstelle dem VR-Rechner mitgeteilt. Die Navigation erlaubt das Betrachten der virtuellen Umgebung von verschiedenen Punkten

Navigation

- Audioprozessor
  Damit ein sehr realistischer Eindruck einer virtuellen Welt entsteht, ist neben einer Stereobildausgabe auch eine 3D-Audioausgabe notwendig. Da die Algorithmen für 3D-Audio enorm rechenintensiv sind, gehört eine 3D-Audioausgabe bei VR-Anwendungen noch nicht zur Regel. Mehrere parallel arbeitende Signalprozessoren sind für die Berechnung notwendig.

- Diskvolumen
  Ein großes Diskvolumen ist heutzutage selbstverständlich.

Die Bildrate der Stereobildausgabe sollte möglichst konstant sein, damit ein ruckfreier Ablauf gewährleistet ist. Außerdem sollte die Stereobildausgabe eine geringe Latenz aufweisen. Die Zeitdauer zwischen einer Initiierung einer Aktion und der daraufhin wahrnehmbaren resultierenden Reaktion des Systems wird als Latenzzeitbezeichnet. Kritisch ist die Latenzzeit zwischen Systemreaktion und Drehbewegung des Kopfes. Ist die Latenzzeit zu groß, so tritt häufig bei längerer Anwendung die so genannte Simulatorkrankheit auf. Die Symptome sind Gleichgewichtsstörungen und Übelkeit.

Latenzzeit

Simulator-krankheit

1.4 Professionelle Hard- und Software

Bei industriellen Anwendungen finden man häufig folgende VR-Rechner:

- Silicon Graphics UNIX-Workstations: Indigo
- Evans & Sutherland: UNIX-Workstations

Seit einiger Zeit setzen sich Low-Cost-Lösungen auf PC-Basis (Windows oder LINUX) immer mehr durch. PC-Lösungen finden seit längerem auch im CAD-Bereich Anwendung. Für professionelle VR-Anwendungen sind allerdings nach wie vor UNIX-Workstations notwendig, insbesondere dann, wenn es sich um Visualisierungen mit einer CAVE handelt.

Der entscheidender Faktor ist dabei weniger die Prozessorleistung, wichtiger ist vielmehr die Leistung des Graphiksubsystems. Ein leistungsfähiges Graphiksubsystem zeichnet sich durch einen großen Umfang an Hardware-Unterstützung für Graphikalgorithmen auf hohem Niveau aus sowie einen hohen Grad an paralleler Verarbeitung aus. Basis für die Hardware-Unterstützung ist entweder OpenGL oder DirectX. OpenGL wird in der Regel bei CAD-Anwendung, während DirectX meist bei Computerspielen eingesetzt wird.

Professionelle VR-Entwicklungssoftware wird von folgenden Firmen angeboten:

- World Tool Kit von Sense8
- Open Inventor von Silicon Graphics
- Performer von Silicon Graphics
- Vision Works von Paradigm
- Total Virtual Reality Authoring Solution VRT von SUPERSCAPE
- dVISE und dVS von Division

Die oben genannte Software ist sowohl für Windows als auch für UNIX verfügbar.

Softwarepakete zur Modellierung dreidimensionaler Körper finden zusätzlich Einsatz. Softwarepakete zur Modellierung stammen meist aus dem CAD-Bereich oder aus dem Trickfilm-Bereich. Professionelle Software wird von folgenden Firmen angeboten:

- CATIA
- AutoCAD
- TrueSpace
- RayDream
- 3DStudioMAX
- Maya
- Softimage

Seit längerer Zeit sind VR-Anwendung auch im Internet möglich. Bei diesen Anwendungen wird eine virtuelle Welt durch eine Beschreibungssprache modelliert und vom Server zum Client übertragen. Der Client interpretiert die Beschreibung und führt die Visualisierung der virtuellen Welt mit Hilfe eines Browsers durch. Die inzwischen international standardisierte Beschreibungssprache “Virtual Reality Modelling Language”, VRML, wird im Abschnitt 7 vorgestellt.
Für die Entwicklung von VRML-Anwendungen gibt es weiterhin eine Reihe von speziellen Editoren, die VRML-Syntax direkt unterstützen. Einige Beispiele sind:

- VRMLPad
- Spazz3D
- CosmoWorlds, wird von SGI leider nicht mehr weiterentwickelt

In der Regel bieten 3D-Modeller-Softwarepakete auch eine Exportmöglichkeit nach VRML.

1.5 Systeme zur Ein- und Ausgabe

In diesem Abschnitt werden Systeme beschrieben, die eine Visualisierung, Navigation, Interaktion oder eine Positionsbestimmung bei VR-Systemen ermöglichen.

1.5.1 Systeme zur Visualisierung

Systeme zur Visualisierung dienen dazu, dem Anwender eine Betrachtung der virtuellen Welt zu ermöglichen. Dabei muss ein Stereobild dargestellt oder projiziert werden.

Für die stereoskope Projektion gibt es prinzipiell folgende Möglichkeiten:

- Aufprojektion
  Bei dieser Variante wird das Bild auf die Vorderseite einer Projektionswand projiziert. Der Anwender steht vor der Projektionswand, der Beamer befindet sich hinter dem Anwender. Nachteilig ist dabei, dass der Anwender zwischen den Strahlenverlauf geraden kann und dadurch Schatten verursachen kann.

  Aufprojektion

- Rückprojektion
  Bei der Variante der Rückprojektion projiziert ein Beamer das Stereobild auf die transparente Rückseite der Projektswand, der Anwender befindet sich vor der Projektionswand. Eine Möglichkeit zur Schattenbildung durch den Anwender ist damit nicht gegeben. Dieser Vorteil wird durch einen größeren Platzbedarf erkauft.

  Rückprojektion

Damit ein steroskopischer Bildeffekt entsteht, muss je ein Bild für das linke bzw. rechte Auge berechnet werden. Ein stereoskoper Bildeindruck kann durch durch folgende Möglichkeiten erreicht werden:

- Direkte Projektion
  Jedes der beiden Bilder wird direkt vor dem betreffenden Auge visualisiert, wie beispielsweise bei einem Datenhelm.

  direkte Projektion

- Aktive Projektion
  Ein Bildquelle, z.B. ein Beamer, liefert im zeitlichen Wechsel ein Bild für linke bzw. rechte Auge. Mit Hilfe einer Shutterbrille erfolgt die Abdeckung jeweils eines der Augen. Damit wird erreicht, dass jeweils das Bild für das entsprechende Auge dargestellt wird. Bildwechsel und Augenabdeckung werden über einen Infrarot-Sender und -Empfänger synchronisiert.

  aktive Projektion

- Passive Projektion
  Jede von zwei Bildquellen, z.B. Beamer, liefern kontinuierlich ein Bild für ein Auge. Beide Bildquellen sind mit senkrecht zueinander orientierten Polfiltern versehen. Mit Hilfe einer Brille, die mit den korrespondierenden Polfilter ausgestattet ist, gelingt die Trennung der beiden Bilder, so dass das jeweilig richtige Bild zum richtigen Auge gelangt.

  passive Projektion

Für Display-Systeme gibt es verschiedene Varianten:

- Helmet-mounted Display
  Bei einem Helmet-mounted Display (HMD) ist die Optik an einem Helm angebracht, den der Betrachter trägt. Das Stereobild kann damit direkt vor den Augen des Betrachters dargestellt.

  Helmet-mounted Display

  Man unterscheidet folgende Realisierungen von Helmet-mounted Displays:

  - LCD Helmet-mounted Display
    Beim LCD Helmet-mounted Display wird auf zwei LCD Displays (Liquid Crystal

    LCD Helmet-mounted Display

Display , LCD) das Stereobild vor den Augen des Betrachters dargestellt. Der Vorteil von LCD Displays ist das kleine Gewicht. Nachteilig ist jedoch die geringe Auflösung im Vergleich zu den anderen Realisierungen.

CRT Helmet-mounted Display

- ▷ CRT Helmet-mounted Display
  Bei einem CRT Helmet-mounted Display werden zwei kleine Farbmonitore (Kathodenstrahlröhren, Cathode Ray Tube, CRT) verwendet, um das Stereobild vor den Augen des Anwenders darzustellen. Dabei handelt es sich um die gleiche Technik, wie bei TV-Geräten. Der Vorteil von CRT-Systemen ist die hohe Auflösung. Nachteilig ist jedoch das große Gewicht der Kathodenstrahlröhren. Ein Problem sind die hohen elektromagnetischen Felder, die in unmittelbarer am Kopf des Anwenders wirken. Die Auswirkungen auf die Gesundheit sind bislang noch nicht hinreichend untersucht. Aus diesem Grunde sollte auf ein häufiges und längeres Tragen eines CRT Helmet-mounted Displays verzichtet werden.

Fiber-Optic Helmet-mounted Display

- ▷ Fiber-Optic Helmet-mounted Display
  Bei einem Fiber-Optic Helmet-mounted Display werden zwei Glasfaserbündel mit etwa vier Millionen Fasern zur Bildübertragung verwendet. Das auf zwei Kathodenstrahlröhren dargestellte Stereobild wird in das Glasfaserbündelpaar eingekoppelt und zum Helm des Anwenders übertragen. Vor den Augen des Anwenders wird das Stereobild auf zwei kleine Projektionsflächen gelenkt und dargestellt. Fiber-Optic Helmet-mounted Displays genügen höchsten Ansprüchen. Sie ermöglichen eine sehr hohe Bildqualität. Anwendung finden Fiber-Optic Helmet-mounted Displays bei militärischen Flugsimulationen.

Shutterbrille und Stereomonitor

- ▶ Shutterbrille und Stereomonitor
  Monitore für Workstations verfügen heutzutage über eine entsprechend hohe Bildwechselfrequenz und sind damit in der Lage, schnelle Bildwechsel zu vollführen. Im Stereomodus wird am Monitor das vom Rechner ermittelte Bild für das linke bzw. rechte Auge für eine kurze Zeitdauer dargestellt. Zur Betrachtung des Stereobildes muss der Betrachter eine Shutterbrille tragen. Die Shutterbrille besitzt LCD-Gläser, die lediglich zwischen transparent bzw. undurchsichtig schalten. Das Schalten der Shutterbrille ist mit der Frequenz der Stereodarstellung synchronisiert. Der Betrachter nimmt deshalb jeweils nur mit einem Auge das zugehörige Bild wahr, während das andere Auge abgedeckt ist. Die Shutterfrequenz liegt üblicherweise bei 30Hz bzw. bei 60Hz.

Responsive Workbench

- ▶ Responsive Workbench
  Die Responsive Workbench wurde von der Fraunhofergesellschaft in Darmstadt entwickelt. Dabei handelt es sich um eine große Projektionsfläche, die horizontal wie ein Tisch aufgestellt wird. Mit Hilfe von Stereobeamer wird ein Stereobild auf diese Fläche projiziert. Zur Betrachtung muss der Anwender eine Shutterbrille tragen.

BOOM

- ▶ BOOM, Binocular Omni-Orientation Monitor
  Ein BOOM besteht aus zwei Monitore die an einem “galgenähnlichen” Schwenkarm angebracht sind. Jeder der Monitore liefert das Bild für ein Auge. Der Schwenkarm nimmt den größten Anteil des Gewichtes auf. Deshalb können vergleichsweise schwere Monitore eingesetzt werden, die ein sehr gutes Bild liefern. Am Schwenkarm sind sechs mechanische Sensoren angebracht, die die Bewegung des Schwenkarms erfassen und die aktuelle Position an den Rechner übermitteln. Einsatzgebiet sind beispielsweise CAD-Anwendung oder Architektur.

CAVE-System
Fish-Tank-System

- ▶ CAVE-System oder Fish-Tank-System
  Bei einem Fish-Tank-System (Aquarium), das auch als CAVE-System (Visual Experience Automatic Virtual Environment) bezeichnet wird, befindet sich der Anwender in

einem würfelförmigen "Visualisierungsraum". Jede Seitenwand sowie Decke und Boden des Visualisierungsraumes werden als Projektionsfläche verwendet. Mit Hilfe von Workstations werden die Stereobilder für die einzelnen Projektionsflächen berechnet und mit Beamern von der Wandrückseite her projiziert (Rückprojektion). Der Anwender trägt eine Shutterbrille und einen Positionssensor.

- Autosstereoskopes Display
  autostereoskopes Display
  Bei einem autostereoskopes Display benötigt der Anwender keinerlei Hilfsmittel mehr, wie beispielsweise eine Shutter- oder Polfilterbrille. Die Technik für ein autostereoskopes Display wurde vom Heinrich-Hertz-Institut in Berlin entwickelt. Dabei handelt es sich um einen Monitor, dessen Bildschirmoberfläche vertikal mit optischen Linsen in halbzylindrischer Form überzogen ist. Die halbzylindrischer Form der Linsen lenkt die beiden stereoskopen Bilder so ab, dass die beiden verschiedenen Bilder zum jeweils richtigen Auge gelangen. Die Kopfposition des Anwenders wird mit einer Kamera erfasst und der Abstand zum Monitor ermittelt. Aufgrund dieser Daten wird die Ablenkungsrichtung der stereoskopen Bilder nachgeregelt.

- Virtual Showcases
  virtual Showcase
  Virtual Showcases sind mehrnutzerfähige Displays und haben die Form einer virtuellen Vitrine. Durch trickreichen Einsatz von Spiegelsystemen und Projektoren wird der visuelle Eindruck einer virtuellen Vitrine erreicht. Virtual Showcases werden unter anderem vom Fraunhoferinstitut für graphische Datenverarbeitung in Darmstadt entwickelt.

1.5.2 Systeme zur Navigation und Interaktion

Eingabesysteme erfüllen zwei Aufgaben: Navigation und Interaktion. Im Fall der Navigation verändert der Anwender mit Hilfe des Eingabegerätes seine Position in der virtuellen Welt und im Fall der Interaktion veranlasst der Anwender eine Aktion, d.h. eine Veränderung in der virtuellen Welt.

- Datenhandschuh
  Datenhandschuh
  Ein Datenhandschuh wird wie ein normaler Handschuh angezogen. An den Fingergliedern und am Handrücken sind Dehnmessstreifen angebracht. Diese Dehnmessstreifen erzeugen eine Spannung, die mit Hilfe von Phototransitoren in Licht umgewandelt wird. Über Glasfasern wird das erzeugte Licht zum Rechner übertragen. Die räumliche Position der Hand wird mit Hilfe eines Tracking-Systems erfasst.
  Jede Fingerstellung (Geste) muss vor der Anwendung programmiert werden. Ein ausgestreckter Zeigefinger kann beispielsweise eine Vorwärtsbewegung bedeuten.

- Spacemouse
  Spacemouse
  Die Spacemouse hat einen federnd gelagerten Steuerknopf. der die Eingabe einer Bewegung mit sechs Freiheitsgraden $(x, y, z, \alpha_x, \alpha_y, \alpha_z)$ ermöglicht. Weiterhin gibt es eine Reihe von programmierbaren Funktionstasten, die mit einer Interaktion belegt werden können.

- Spaceball
  Spaceball
  Ein Spaceball ist genauso aufgebaut, wie eine Spacemouse, lediglich der federnd gelagerte Steuerknopf ist zu einer Kugel ausgebildet.

- Freeflying Joystick
  Freeflying Joystick
  Flystick
  Bei einem freeflying Joystick ist der Steuerknüppel nicht feststehend, sondern kann frei beweglich in der Hand gehalten werden. In Anlehnung an das Wort "Joystick" bezeichnet man den freeflying Joystick häufig auch verkürzt als "Flystick". Ein freeflying Joystick liefert die sechs Bewegungsparameter $(x, y, z, \alpha_x, \alpha_y, \alpha_z)$. Inzwischen gibt es

auch kabellose, funkbetriebene Ausführungen.

1.5.3 Systeme zur Positionsbestimmung

Tracking-Systeme

Systeme zur Positionsbestimmung werden als Tracking-Systeme bezeichnet. Tracking-Systeme haben die Aufgabe, die räumliche Position des Anwenders während der Visualisierung einer virtuellen Welt zu erfassen. Die Position wird dem Rechner zu übermittelt. Nach einer Bewegung des Anwenders berechnet VR-Rechner das der neuen Position entsprechende Stereobild.

Man unterscheidet folgende Tracking-Technologien:

Elektromagnetische Tracking-Systeme

- Elektromagnetische Tracking-Systeme
  Bei elektromagnetischen Tracking-Systemen wird die Information über die räumliche Position des Anwenders mit Hilfe von drei orthogonal zueinander angeordneten Spulen bestimmt. Der in einer Spule induzierte Strom, ist dann am größten, wenn das elektromagnetische Feld parallel zur Spulenachse liegt. Eine Senderspule strahlt eine elektromagnetische Welle ab. Ein Sensor, der vom Anwender getragen wird, aktiviert nacheinander die drei Empfängerspulen, damit sie sich nicht gegenseitig beeinflussen. Die registrierten induzierten Ströme dienen zur Berechnung der Bewegung des Anwenders.

Akustische Tracking-Systeme

- Akustische Tracking-Systeme
  Bei akustischen Tracking-Systemen wird eine Ultraschallwelle von einem Sender ausgestrahlt und von Ultraschallmikrophonen registriert. Über eine Laufzeit- oder eine Phasendifferenzmessung kann die Entfernung des Anwenders festgestellt werden.

Optische Tracking-Systeme

- Optische Tracking-Systeme
  Bei einem optischen Tracking-System trägt der Anwender einen Infrarotsensor mit sich. Mehrere Infrarotleuchtdioden, die an der Decke angebracht sind, werden vom Sensor erfasst. Aus der registrierten Lichtintensität der einzelnen Infrarotquellen kann auf die Position des Anwenders geschlossen werden.

Mechanische Tracking-Systeme

- Mechanische Tracking-Systeme
  Bei mechanischen Tracking-Systemen kann es sich bei den Sensoren um Akzelerometer, Gyroskope oder Drehwinkelmeter handeln.

Die spezifischen Eigenschaften, nach denen Tracking-Systeme bewertet werden, sind:

- Auflösung
  Die Auflösung gibt die kleinste wahr genommene Positionsänderung an.

- Genauigkeit
  Die Genauigkeit gibt die räumliche Abweichung zwischen gemessener und tatsächlicher Position an.

- Aktualisierungsrate
  Die Aktualisierungsrate ist die Zeit zwischen den einzelnen Positionsmeldungen.

- Latenzzeit
  Die Latenzzeit gibt die Zeit zwischen Positionswechsel und der Visualisierung der neuen Position an.

- Wiederholbarkeit
  Die Wiederholbarkeit gibt die Abweichung zwischen dem Start- und dem Endpunkt nach der Durchführung einer Bewegungsschleife an.

Kapitel 2

Grundlagen der Computergraphik

In der Computergraphik werden dreidimensionale Körper innerhalb eines Koordinatensystems dargestellt. Eine der dabei häufig angewandten Darstellungsarten ist die Randdarstellung von dreidimensionalen Körpern. Im Abschnitt 2.4 wird auf weitere Darstellungsarten von Körpern näher eingegangen. Bei der Randdarstellung wird die Oberfläche des Körpers angenähert und das umschlossene Volumen als starrer Körper betrachtet. Im einfachsten Fall nimmt man zur Randdarstellung ein Polygonnetz (Drahtgittermodell). Jedes Polygon wird dabei durch seine Eckpunkte festgelegt. Die Eckpunkte selbst sind Ortsvektoren innerhalb des Koordinatensystems.

2.1 Geometrische Transformationen

In der Computergraphik werden Objekte im dreidimensionalen Raum verschoben, gedreht, aus verschiedenen Entfernungen und unter unterschiedlichen Blickwinkeln betrachtet etc. Jede dieser Objektmanipulation kann durch eine mathematische Transformation beschrieben werden. Es ist vorteilhaft, für alle Transformationen dieselbe Beschreibung zu verwenden. Komplexe Transformationen können dann aus Grundtransformationen zusammengesetzt werden.

Da ein Objekt im dreidimensionalen Raum durch eine Menge von Punkten beschrieben wird, muss die gleiche Transformation auf die gesamte Punktemenge angewandt werden. Die folgenden Betrachtungen können deshalb auf einen Punkt beschränkt bleiben. Jeder Punkt wird durch einen Ortsvektor repräsentiert.

Damit alle geometrischen Transformationen einheitlich durch eine Matrixoperation beschreibbar sind, muss von den gewöhnlichen Koordinaten des dreidimensionalen Raumes $\mathcal{R}^3$ auf so genannten "homogenen Koordinaten" gewechselt werden. Die homogenen Koordinaten bilden die Basis des vierdimensionalen Raumes $\mathcal{R}^4$.

homogene Koordinaten

Ein Ortsvektor des dreidimensionalen Raumes $\mathcal{R}^3$ wird durch folgende Vorschrift in den vierdimensionalen Raum $\mathcal{R}^4$ übergeführt:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \longleftrightarrow \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \qquad x' = \frac{x}{w}\ , \quad y' = \frac{y}{w}\ , \quad z' = \frac{z}{w}\ , \quad w \neq 0\ . \tag{2.1}$$

Bei geometrischen Transformationen setzt man $w = 1$.

Die Einführung von homogenen Koordinaten bringt folgende Vorteile mit sich:

- Einheitliche Darstellung
  Es wird eine einheitliche Darstellung für alle Transformationen erreicht. Jede Transformation wird durch eine 4×4 Matrix beschrieben. Mit der gleichen Hardware können alle Transformationen durchgeführt werden.
- Aufbau komplexer Transformationen aus Basistransformationen
  Komplexe Transformationen können in Basistransformationen zerlegt werden.
- Ausführung mehrerer Transformationen in einem Schritt
  Die sequentielle Ausführung verschiedener Transformationen erfordert nur die einmalige Multiplikation aller Transformationsmatrizen. Dadurch wird die Rechenzeit reduziert.

Zur Unterscheidung von homogenen Koordinaten und dreidimensionalen Koordinaten werden Vektoren und Matrizen, die sich auf homogene Koordinaten beziehen, mit eckigen Klammern geschrieben. Vektoren und Matrizen, die sich dagegen auf dreidimensionale Koordinaten beziehen, werden mit runden Klammern angegeben.

Die Basistransformationen sind:

- Translation
- Skalierung
- Scherung
- Rotation
- Spiegelung

Diese Basistransformationen werden im folgenden näher betrachtet.

2.1.1 Translation

Verschiebt man einen Punkt mit den Koordinaten $P(x, y, z)$ um den Vektor $\vec{v} = (v_x,\ v_y,\ v_z)^T$, so lauten bekanntlich die neuen Koordinaten des Punktes $P(x', y', z') = P(x+v_x, y+v_y, z+v_z)$. In der Darstellung mit homogenen Koordinaten erhält man:

$$\begin{bmatrix} x+v_x \\ y+v_y \\ z+v_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.2}$$

Die Transformationsmatrix lautet damit:

$$\underline{\underline{T_v}}(v_x, v_y, v_z) = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.3}$$

Die inverse Transformation lautet:

$$\underline{\underline{T_v}}^{-1}(v_x, v_y, v_z) = \underline{\underline{T_v}}(-v_x, -v_y, -v_z) = \begin{bmatrix} 1 & 0 & 0 & -v_x \\ 0 & 1 & 0 & -v_y \\ 0 & 0 & 1 & -v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.4}$$

2.1.2 Skalierung

Ein Objekt kann für jede der drei Koordinatenachsen unterschiedlich skaliert werden. In homogenen Koordinaten erhält man:

$$\begin{bmatrix} x \cdot s_x \\ y \cdot s_y \\ z \cdot s_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.5}$$

Die Skalierungsmatrix lautet damit:

$$\underline{\underline{S_K}}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.6}$$

Die inverse Transformation lautet hierzu:

$$\underline{\underline{S_K}}^{-1}(s_x, s_y, s_z) = \underline{\underline{S_K}}(\tfrac{1}{s_x}, \tfrac{1}{s_y}, \tfrac{1}{s_z}) = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.7}$$

2.1.3 Scherung

Bei einer Scherung wird die Form des Objektes verzerrt. Es gelten folgende Beziehungen für die kartesischen Koordinaten:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x + y \cdot s_1 + z \cdot s_2 \\ y + x \cdot s_3 + z \cdot s_4 \\ z + x \cdot s_5 + y \cdot s_6 \end{pmatrix} . \tag{2.8}$$

In homogenen Koordinaten folgt:

$$\begin{bmatrix} x + y \cdot s_1 + z \cdot s_2 \\ y + x \cdot s_3 + z \cdot s_4 \\ z + x \cdot s_5 + y \cdot s_6 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_1 & s_2 & 0 \\ s_3 & 1 & s_4 & 0 \\ s_5 & s_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} . \tag{2.9}$$

Die Transformationsmatrix für eine Scherung lautet:

$$\underline{\underline{S_H}}(s_1, s_2, s_3, s_4, s_5, s_6) = \begin{bmatrix} 1 & s_1 & s_2 & 0 \\ s_3 & 1 & s_4 & 0 \\ s_5 & s_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.10}$$

Für die inverse Transformation muss die Matrix $\underline{\underline{S_H}}(s_1, s_2, s_3, s_4, s_5, s_6)$ invertiert werden. Sie ist abhängig von den Werten $s_1, s_2, s_3, s_4, s_5, s_6$.

2.1.4 Rotation

Bei einer Rotation wird unterschieden, um welche der drei Koordinatenachsen die Drehung erfolgen soll. Rotationen um eine der drei Koordinatenachsen lassen sich sehr einfach angeben. Dagegen werden Rotationen um eine beliebige Raumachse aus Basistransformationen zusammengesetzt. Die Drehung wird dabei stets in mathematisch positiver Richtung durchgeführt.

2.1.4.1 Rotation um die x-Achse

Eine Rotation um den Winkel α in mathematisch postiver Drehungrichtung um die x-Achse wird in kartesischen Koordinaten durch die folgenden Gleichung beschrieben:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \cdot \cos(\alpha) - z \cdot \sin(\alpha) \\ y \cdot \sin(\alpha) + z \cdot \cos(\alpha) \end{pmatrix} . \tag{2.11}$$

In homogenen Koordinaten folgt die Darstellung:

$$\begin{bmatrix} x \\ y \cdot \cos(\alpha) - z \cdot \sin(\alpha) \\ y \cdot \sin(\alpha) + z \cdot \cos(\alpha) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} . \tag{2.12}$$

Die Transformationsmatrix lautet:

$$\underline{\underline{R_x}}(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.13}$$

Die inverse Transformation lautet:

$$\underline{\underline{R_x}}^{-1}(\alpha) = \underline{\underline{R_x}}(-\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.14}$$

2.1.4.2 Rotation um die y-Achse

Eine Rotation um den Winkel α um die y-Achse wird in kartesischen Koordinaten durch die folgende Gleichung beschrieben:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \cdot \cos(\alpha) + z \cdot \sin(\alpha) \\ y \\ -x \cdot \sin(\alpha) + z \cdot \cos(\alpha) \end{pmatrix} . \tag{2.15}$$

In homogenen Koordinaten folgt die Darstellung:

$$\begin{bmatrix} x \cdot \cos(\alpha) + z \cdot \sin(\alpha) \\ y \\ -x \cdot \sin(\alpha) + z \cdot \cos(\alpha) \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} . \tag{2.16}$$

Die Transformationsmatrix lautet:

$$\underline{\underline{R_y}}(\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.17}$$

Die inverse Transformation lautet:

$$\underline{\underline{R_y}}^{-1}(\alpha) = \underline{\underline{R_y}}(-\alpha) = \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.18}$$

2.1.4.3 Rotation um die z-Achse

Eine Rotation um den Winkel α um die z-Achse wird in kartesischen Koordinaten durch die folgende Gleichung beschrieben:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \cdot \cos(\alpha) - y \cdot \sin(\alpha) \\ x \cdot \sin(\alpha) + y \cdot \cos(\alpha) \\ z \end{pmatrix} . \tag{2.19}$$

In homogenen Koordinaten folgt die Darstellung:

$$\begin{bmatrix} x \cdot \cos(\alpha) - y \cdot \sin(\alpha) \\ x \cdot \sin(\alpha) + y \cdot \cos(\alpha) \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} . \tag{2.20}$$

Die Transformationsmatrix lautet:

$$\underline{\underline{R_z}}(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.21}$$

Die inverse Transformation lautet:

$$\underline{\underline{R_z}}^{-1}(\alpha) = \underline{\underline{R_z}}(-\alpha) = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.22}$$

2.1.4.4 Rotation um eine beliebige Achse

Die Rotation um den Winkel α um eine beliebige Raumachse stellt eine komplizierte Transformation dar. Man führt sie geschickterweise auf Basistransformationen zurück. Die Raumachse muss dazu durch eine Geradengleichung beschrieben werden. Die Geradengleichung lautet:

$$\vec{g}(\mu) = \vec{a} + \mu \cdot \vec{r} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} + \mu \cdot \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} . \tag{2.23}$$

Dabei ist $\vec{a}$ ein Ortsvektor, der auf einen Punkt der Gerade $\vec{g}(\mu)$ zeigt und $\vec{r}$ der Richtungsvektor der Geraden. Das Bild 2.1 zeigt die Gerade $\vec{g}(\mu)$.

Zur Berechnung der Transformationsmatrix geht man folgendermaßen vor. Die Gerade wird derart verschoben und gedreht wird, bis sie parallel zu einer der Koordinatenachsen liegt. Anschließend wird der Punkt P(x,y,z) um den Winkel α um die transformierte Rotationsachse gedreht. Schließlich wird die Gerade wieder in die ursprüngliche Position zurück transformiert. Damit liegt schließlich das gewünschte Ergebnis vor.

Im einzelnen sind folgende Schritte notwendig:

1. Verschiebung der Geraden in den Ursprung.
   Die Gerade wird so verschoben, dass sie durch den Koordinatenursprung verläuft. Der Translationsvektor ist $-\vec{a}$. Die zugehörige Transformationsmatrix lautet:

$$\underline{\underline{T_v}}(-a_x, -a_y, -a_z) = \begin{bmatrix} 1 & 0 & 0 & -a_x \\ 0 & 1 & 0 & -a_y \\ 0 & 0 & 1 & -a_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.24}$$

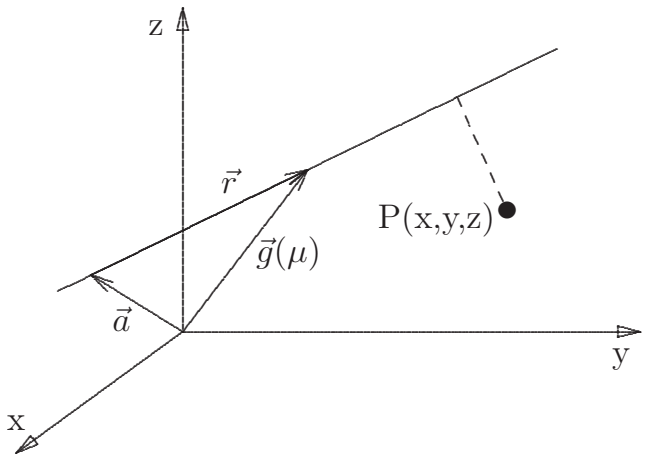

Abbildung 2.1: Gerade im dreidimensionalen Raum

Das Bild 2.2 zeigt die Gerade, die jetzt durch den Koordinatenursprung verläuft.

1

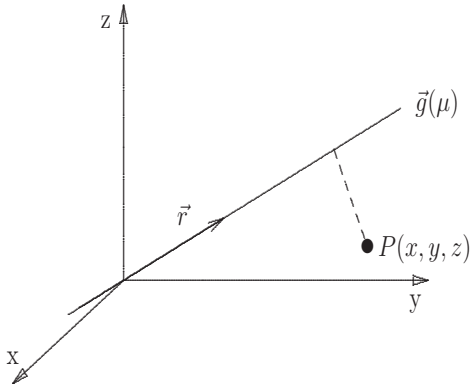

Abbildung 2.2: Gerade durch Ursprung

2. Drehung um die z-Achse.
   Die Gerade wird um die z-Achse gedreht, bis sie in der x-z-Ebene liegt. Dazu muss sie um den Winkel $-\alpha_z$ gedreht werden. Der Winkel α_z berechnet sich aus:

$$\cos(\alpha_z) = \frac{r_x}{\sqrt{r_x^2 + r_y^2}} , \quad \sin(\alpha_z) = \frac{r_y}{\sqrt{r_x^2 + r_y^2}} . \tag{2.25}$$

   Die Transformationsmatrix lautet:

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} \cos(\alpha_z) & \sin(\alpha_z) & 0 & 0 \\ -\sin(\alpha_z) & \cos(\alpha_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

bzw.

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} \frac{r_x}{\sqrt{r_x^2+r_y^2}} & \frac{r_y}{\sqrt{r_x^2+r_y^2}} & 0 & 0 \\ \frac{-r_y}{\sqrt{r_x^2+r_y^2}} & \frac{r_y}{\sqrt{r_x^2+r_y^2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.27}$$

Das Bild 2.3 veranschaulicht die Berechnung des Drehwinkels α_z.

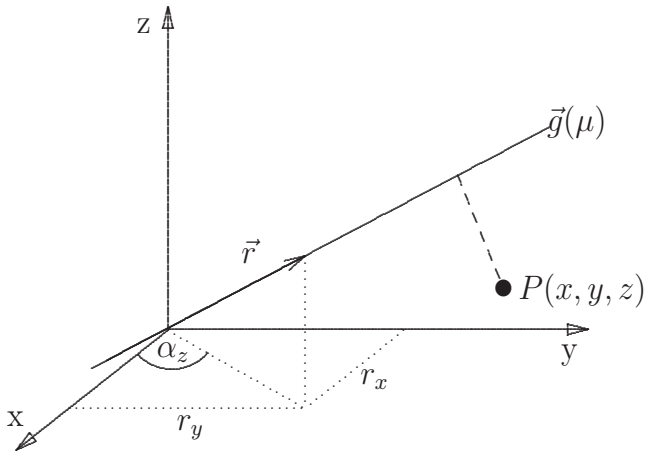

Abbildung 2.3: Drehwinkel α_z

3. Drehung um die y-Achse.
Schließlich muss die Gerade noch um die y-Achse um den Winkel $-\alpha_y$ gedreht werden, damit sie parallel zur z-Achse verläuft.

Das Bild 2.4 zeigt die Gerade, die jetzt in der x-z-Ebene verläuft, mit dem eingezeichneten Winkel α_y.

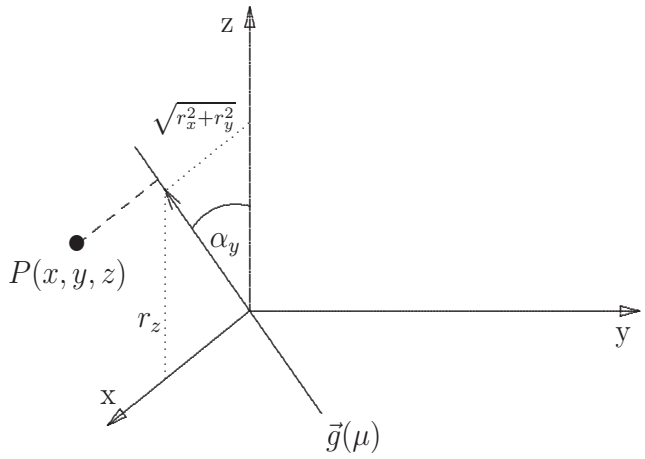

Abbildung 2.4: Gerade in xz-Ebene

Der Winkel α_y berechnet sich aus:

$$\cos(\alpha_y) = \frac{r_z}{\sqrt{r_x^2 + r_y^2 + r_z^2}}\ , \quad \sin(\alpha_y) = \frac{\sqrt{r_x^2 + r_y^2}}{\sqrt{r_x^2 + r_y^2 + r_z^2}}\ . \tag{2.28}$$

Die Transformationsmatrix lautet:

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} \cos(\alpha_y) & 0 & -\sin(\alpha_y) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha_y) & 0 & \cos(\alpha_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.29}$$

bzw.

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} \frac{r_z}{\sqrt{r_x^2+r_y^2+r_z^2}} & 0 & \frac{-\sqrt{r_x^2+r_y^2}}{\sqrt{r_x^2+r_y^2+r_z^2}} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{r_x^2+r_y^2}}{\sqrt{r_x^2+r_y^2+r_z^2}} & 0 & \frac{r_z}{\sqrt{r_x^2+r_y^2+r_z^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\ . \tag{2.30}$$

4. Drehung um die z-Achse.
   Jetzt erst kann die eigentliche Drehung um den Winkel α durchgeführt werden. Die Drehung muss um die z-Achse erfolgen. Mit Gl.(2.21) folgt für die Transformationsmatrix:

$$\underline{\underline{R_z}}(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\ . \tag{2.31}$$

5. Gesamte Transformtion.
   Die durchgeführten Transformationen müssen wieder rückgängig gemacht werden. Wie bereits erwähnt, kann eine Transformation rückgängig gemacht werden, wenn man die inverse Transformationsmatrix anwendet. Man erhält folgenden Zusammenhang.

$$\begin{aligned} \underline{\underline{R_r}}(\alpha) &= \left(\underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \cdot \underline{\underline{T_v}}(-a_x, -a_y, -a_z)\right)^{-1} \cdot \\ &\quad \underline{\underline{R_z}}(\alpha) \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \cdot \underline{\underline{T_v}}(-a_x, -a_y, -a_z) \\ \underline{\underline{R_r}}(\alpha) &= \underline{\underline{T_v}}^{-1}(-a_x, -a_y, -a_z)\underline{\underline{R_z}}^{-1}(-\alpha_z) \cdot \underline{\underline{R_y}}^{-1}(-\alpha_y) \cdot \\ &\quad \underline{\underline{R_z}}(\alpha) \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \cdot \underline{\underline{T_v}}(-a_x, -a_y, -a_z) \\ \underline{\underline{R_r}}(\alpha) &= \underline{\underline{T_v}}(a_x, a_y, a_z)\underline{\underline{R_z}}(\alpha_z) \cdot \underline{\underline{R_y}}(\alpha_y) \cdot \\ &\quad \underline{\underline{R_z}}(\alpha) \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \cdot \underline{\underline{T_v}}(-a_x, -a_y, -a_z) \end{aligned} \tag{2.32}$$

2.1.5 Spiegelung

Bei einer Spiegelung wird ebenfalls unterschieden werden, ob es sich um eine Spiegelung an einer der Hauptebenen oder um eine Spiegelung an einer beliebigen Ebene handelt. Beide Arten sollen im folgenden betrachtet werden.

2.1.5.1 Spiegelung an einer Hauptebene

Die Spiegelung an einer Hauptebene kann als Sonderfall der Skalierung betrachtet werden. Ein Skalierungsfaktor nimmt dabei den Wert -1 an. Im einzelnen lauten die Transformationsmatrizen:

- Spiegelung an der y-z-Ebene

$$\underline{\underline{S_{yz}}} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.33}$$

- Spiegelung an der x-z-Ebene

$$\underline{\underline{S_{xz}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.34}$$

- Spiegelung an der x-y-Ebene

$$\underline{\underline{S_{xy}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.35}$$

- Spiegelung am Koordinatenursprung

$$\underline{\underline{S_{P_0}}} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.36}$$

2.1.5.2 Spiegelung an einer beliebigen Ebene

Die Spiegelung an einer beliebigen Ebene kann auf eine Reihe einfacher Basistransformationen zurückgeführt werden. Die Ebene wird dazu in der Hesse Normalform angegeben.

$$\vec{x}^T \cdot \vec{n} = d \tag{2.37}$$

Dabei ist $\vec{n}$ der Normalenvektor der Ebene und d der Abstand der Ebene vom Koordinatenursprung. Liegt der Vektor $\vec{x}$ in der Ebene, so erfüllt er die Gl.(2.37).

Für eine Spiegelung an dieser Ebene sind folgende Schritte notwendig:

1. Verschiebung der Ebene in den Koordinatenursprung.
   Die Ebene muss dazu um die Translation $\vec{n} \cdot d$ verschoben werden. Die Translationsmatrix lautet:

$$\underline{\underline{T_v}}(-d \cdot n_x, -d \cdot n_y, -d \cdot n_z) = \begin{bmatrix} 1 & 0 & 0 & -d \cdot n_x \\ 0 & 1 & 0 & -d \cdot n_y \\ 0 & 0 & 1 & -d \cdot n_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.38}$$

2. Drehung um die z-Achse bis der Normalenvektor in der x-z-Ebene liegt.
Dazu muss der Normalenvektor um den Winkel $-\alpha_z$ gedreht werden. Der Winkel α_z berechnet sich aus:

$$\cos(\alpha_z) = \frac{n_x}{\sqrt{n_x^2 + n_y^2}} = \frac{n_x}{\sqrt{1 - n_z^2}} \tag{2.39}$$

$$\sin(\alpha_z) = \frac{n_y}{\sqrt{n_x^2 + n_y^2}} = \frac{n_y}{\sqrt{1 - n_z^2}} \tag{2.40}$$

Die Transformationsmatrix lautet:

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} \cos(\alpha_z) & \sin(\alpha_z) & 0 & 0 \\ -\sin(\alpha_z) & \cos(\alpha_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.41}$$

bzw.

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} \frac{n_x}{\sqrt{1-n_z^2}} & \frac{n_y}{\sqrt{1-n_z^2}} & 0 & 0 \\ \frac{-n_y}{\sqrt{1-n_z^2}} & \frac{n_y}{\sqrt{1-n_z^2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.42}$$

3. Drehung um die y-Achse bis der Normalenvektor parallel zur z-Achse liegt.
Der Winkel α_y berechnet sich aus:

$$\cos(\alpha_y) = \frac{n_z}{\sqrt{n_x^2 + n_y^2 + n_z^2}} = n_z \tag{2.43}$$

$$\sin(\alpha_y) = \frac{\sqrt{n_x^2 + n_y^2}}{\sqrt{n_x^2 + n_y^2 + n_z^2}} = \sqrt{n_x^2 + n_y^2} = \sqrt{1 - n_z^2}. \tag{2.44}$$

Die Transformationsmatrix lautet:

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} \cos(\alpha_y) & 0 & -\sin(\alpha_y) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha_y) & 0 & \cos(\alpha_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.45}$$

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} n_z & 0 & -\sqrt{1 - n_z^2} & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{1 - n_z^2} & 0 & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.46}$$

4. Gesamte Transformation.
Die gesamte Transformation berechnet sich aus den Produkten der Transformationsmatrizen.

$$
\begin{aligned}
\underline{\underline{S_n}}(\alpha) &= \Big(\underline{\underline{R_y}}(-\alpha_y)\cdot\underline{\underline{R_z}}(-\alpha_z)\cdot\underline{\underline{T_v}}(-d\cdot n_x,-d\cdot n_y,-d\cdot n_z)\Big)^{-1}\cdot \\
&\quad \underline{\underline{S_{xy}}}\cdot\underline{\underline{R_y}}(-\alpha_y)\cdot\underline{\underline{R_z}}(-\alpha_z)\cdot\underline{\underline{T_v}}(-d\cdot n_x,-d\cdot n_y,-d\cdot n_z) \\
\underline{\underline{S_n}}(\alpha) &= \underline{\underline{T_v}}^{-1}(-d\cdot n_x,-d\cdot n_y,-d\cdot n_z)\cdot\underline{\underline{R_z}}^{-1}(-\alpha_z)\cdot\underline{\underline{R_y}}^{-1}(-\alpha_y)\cdot \\
&\quad \underline{\underline{S_{xy}}}\cdot\underline{\underline{R_y}}(-\alpha_y)\cdot\underline{\underline{R_z}}(-\alpha_z)\cdot\underline{\underline{T_v}}(-d\cdot n_x,-d\cdot n_y,-d\cdot n_z) \\
\underline{\underline{S_n}}(\alpha) &= \underline{\underline{T_v}}(d\cdot n_x,d\cdot n_y,d\cdot n_z)\cdot\underline{\underline{R_z}}(\alpha_z)\cdot\underline{\underline{R_y}}(\alpha_y)\cdot \\
&\quad \underline{\underline{S_{xy}}}\cdot\underline{\underline{R_y}}(-\alpha_y)\cdot\underline{\underline{R_z}}(-\alpha_z)\cdot\underline{\underline{T_v}}(-d\cdot n_x,-d\cdot n_y,-d\cdot n_z)
\end{aligned}
\tag{2.47}
$$

2.1.6 Transformation von Normalenvektoren

Komplexe Objekte werden sehr häufig durch ihre Eckpunkte beschrieben. Diese Darstellung eines Objektes bezeichnet man als Polygonrepräsentation. Die Polygondarstellung eines Objektes wird im Abschnitt 2.4.4.1 näher betrachtet. Neben den Eckpunkten eines Objektes werden die zugehörigen Normalenvektoren mit abgespeichert, da diese für viele Berechnungen benötigt werden. Wird das Objekt transformiert, so müssen auch Normalenvektoren transformiert werden. Die Transformation eines Normalenvektors verläuft nach folgendem Schema.

Es seinen $P_{A1}(x_{A1},y_{A1},z_{A1},w)$ und $P_{B1}(x_{B1},y_{B1},z_{B1},w)$ zwei Punkte der durch den Normalenvektor $\vec{n}_1 = (n_{1_x},n_{1_y},n_{1_z},n_{1_w})^T$ festgelegten Tangentialebene. Der Differenzvektor $\vec{r}_1 = \vec{x}_{A1} - \vec{x}_{B1}$ liegt dann in der Tangentialebene und steht auf dem Normalenvektor $\vec{n}_1$ senkrecht. Durch eine Transformation werden die Punkte $P_{A1}(x_{A1},y_{A1},z_{A1},w)$ und $P_{B1}(x_{B1},y_{B1},z_{B1},w)$ in die Punkte $P_{A2}(x_{A2},y_{A2},z_{A2},w)$ und $P_{B2}(x_{B2},y_{B2},z_{B2},w)$ übergeführt. Der Differenzvektor $\vec{r}_2 = \vec{x}_{A2} - \vec{x}_{B2}$ muss dann wieder in der vom transformierten Normalenvektor $\vec{n}_2$ festgelegten Tangentialebene liegen und senkrecht auf dem Normalenvektor $\vec{n}_2$ stehen. Es gilt:

$$
\begin{aligned}
\vec{n}_1^T\cdot\big(\vec{x}_{A1}-\vec{x}_{B1}\big) &= \vec{n}_1^T\cdot\underline{\underline{E}}\cdot\big(\vec{x}_{A1}-\vec{x}_{B1}\big) \\
&= \vec{n}_1^T\cdot\underline{\underline{A}}^{-1}\cdot\underline{\underline{A}}\cdot\big(\vec{x}_{A1}-\vec{x}_{B1}\big) \\
\vec{n}_1^T\cdot\big(\vec{x}_{A1}-\vec{x}_{B1}\big) &= \underbrace{((\underline{\underline{A}}^{-1})^T\cdot\vec{n}_1)^T}_{\vec{n}_2^T}\cdot\underbrace{\underline{\underline{A}}\cdot\big(\vec{x}_{A1}-\vec{x}_{B1}\big)}_{\vec{x}_2}
\end{aligned}
\tag{2.48}
$$

Der Normalenvektor muss mit der Transponierten der inversen Transformationsmatrix multipliziert werden.

$$
\vec{n_2} = (\underline{\underline{A}}^{-1})^T\cdot\vec{n_1} \tag{2.49}
$$

2.2 Geometrische Projektionen

Ebene geometrische Projektionen sind in der Computergraphik von besonderer Bedeutung. Sie werden dann angewandt, wenn ein dreidimensionales Objekt auf einer zweidimensionalen Fläche visualisiert werden soll. Das folgende Bild 2.5 zeigt eine Übersicht über verschiedene Projektionsvarianten.

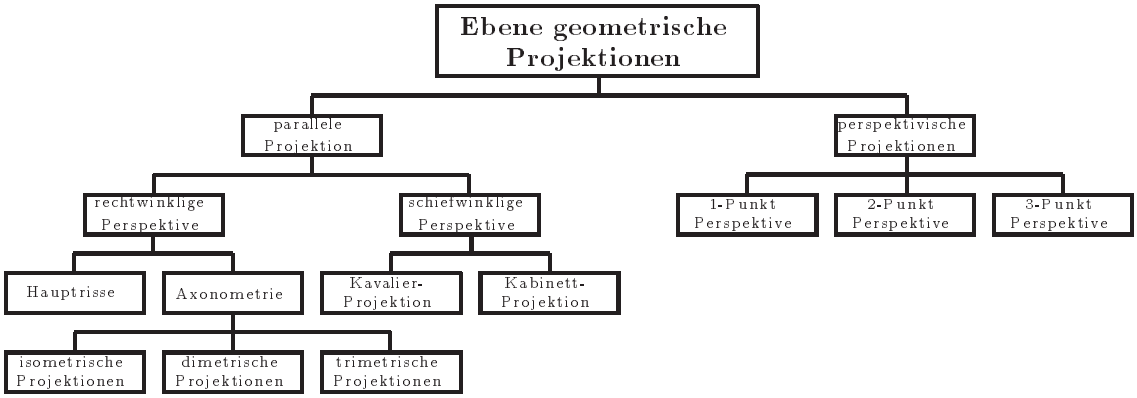

Abbildung 2.5: Einteilung der geometrischen Projektionen

Man kann ebene geometrische Projektionen in zwei Hauptgruppen einteilen: parallele und perspektivische Projektionen. Beide unterscheiden sich darin, dass sich der Betrachter bei einer perspektivischen Projektion in einem endlichen Abstand vom betrachteten Objekt befindet, bei einer parallelen Projektion dagegen im Unendlichen. Bei einer perspektivischen Projektion konvergieren alle Projektionsstrahlen, bei einer parallelen Projektion verlaufen diese dagegen parallel.

- Parallele Projektion
  Man unterteilt parallele Projektionen weiter in:
  - Rechtwinklige bzw. orthographische Projektionen
    Bei einer rechtwinkligen parallelen Projektion treffen die Projektionsgeraden senkrecht auf die Projektionsfläche. Rechtwinklige Projektionen unterteilt man weiter in:
    - Hauptrisse
      Unter dem Begriff "Hauptriß" versteht man eine der sechs möglichen Seitenansichten eines Körpers. Hauptrisse werden bei technischen Zeichnungen häufig verwendet.
    - Axonometrische Projektionen
      Axonometrische Projektionen erlauben eine beliebige Blickrichtung auf einen Körper.
      Diese unterteilt man weiter in:
      - Isometrische Darstellung
        Die Projektionsgeraden verlaufen in Richtung einer Hauptraumdiagonalen. Dabei bildet die Normale der Projektionsebene denselben Winkel mit allen drei Koordinatenachsen.
      - Dimetrische Darstellung
        Zwei Koordinatenachsen bilden mit der Normalen der Projektionsebene denselben Winkel.

- ◇ Trimetrische Darstellung
  Alle Winkel zwischen Koordinatenachsen und der Normalen der Projektionsebene sind verschieden.
- Schiefwinklige parallele Projektion
  Die schiefwinklige parallele Projektion unterteilt man weiter in:
  - ○ Kavalierperspektive
    Es wird im Winkel von 30^o oder 60^o projiziert.
  - ○ Kabinettperspektive
    Es wird im Winkel von 30^o oder 60^o projiziert und nach hinten verlaufende Kanten werden um den Faktor 1/2 Verkürzt.

▶ Perspektivische Projektion
Perspektivische Projektionen sind keine affine Abbildungen mehr, da die Längenverhältnisse nicht invariant sind. Je nachdem, wieviel Fluchtpunkte für die Perspektive verwendet werden, unterscheidet man in:

- 1-Punkt-Perspektive
  Die Projektion schneidet eine Koordinatenachse (Zentralprojektion)
- 2-Punkt-Perspektive
  Die Projektionsebene schneidet zwei Koordiantenachsen.
- 3-Punkt-Perspektive
  Die Projektionsebene schneidet alle drei Koordinatenachsen.

In der Computergraphik verwendet man entweder eine rechtwinklige Parallelprojektion oder eine Zentralprojektion. Diese beiden Beispiele sollen im folgenden betrachtet werden.

2.2.1 Orthogonalprojektion

Das Kennzeichen einer Orthogonalprojektion ist, dass der Normalenvektor der Projektionsebene und die Projektionsrichtung parallel zueinander sind. Man unterscheidet Projektionen auf eine der Hauptebenen und Projektionen auf eine beliebige Ebene.

2.2.2 Projektion auf eine der Hauptebenen

▶ Projektion auf die x-y-Ebene (z = 0)
Die Projektionsmatrix $\underline{\underline{P_{xy}}}$ lautet:

$$\underline{\underline{P_{xy}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.50}$$

▶ Projektion auf die x-z-Ebene (y = 0).
Die Projektionsmatrix $\underline{\underline{P_{xy}}}$ lautet:

$$\underline{\underline{P_{xy}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.51}$$

▶ Projektion auf die y-z-Ebene (x = 0).
Die Projektionsmatrix $\underline{\underline{P_{xy}}}$ lautet:

$$\underline{\underline{P_{xy}}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.52}$$

2.2.3 Projektion auf eine beliebige Ebenen

Die Orthogonalprojektion auf eine beliebige Ebene kann wieder mit Hilfe von elementaren Transformationen in einfache Schritte zerlegt werden. Die Projektionsebene wird in der Hesse Normalform beschrieben:

$$\vec{x}^T \cdot \vec{n} = d \ . \tag{2.53}$$

Die Vorgehensweise bei den Teilschritten kann beispielsweise so erfolgen, dass die Projektionsebene parallel zur x-y-Ebene ausgerichtet wird. Anschießend erfolgt die Projektion auf die x-y-Ebene. Schließlich wird die Ausrichtung wieder rückgängig gemacht. Da die Projektion parallel erfolgt, muss die Projektionsebene nicht in den Koordinatenursprung verschoben werden.

1. Drehung des Normalenvektors um die z-Achse in die x-z-Ebene.
   Hier können die Ergebnisse von Gl.(2.25) bis Gl.(2.27) übernommen werden.

$$\cos(\alpha_z) = \frac{n_x}{\sqrt{1-n_z^2}} \ , \quad \sin(\alpha_z) = \frac{n_y}{\sqrt{1-n_z^2}} \ . \tag{2.54}$$

   Die Transformationsmatrix lautet:

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} cos(\alpha_z) & \sin(\alpha_z) & 0 & 0 \\ -sin(\alpha_z) & cos(\alpha_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.55}$$

   bzw.

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} \frac{n_x}{\sqrt{1-n_z^2}} & \frac{n_y}{\sqrt{1-n_z^2}} & 0 & 0 \\ \frac{-n_y}{\sqrt{1-n_z^2}} & \frac{n_y}{\sqrt{1-n_z^2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ . \tag{2.56}$$

2. Drehung des Normalenvektors um die y-Achse parallel zur z-Achse.
   Auch hierfür können wieder die Ergebnisse von Gl.(2.28) bis Gl.(2.30) übernommen werden.

   Der Winkel α_y berechnet sich aus:

$$\cos(\alpha_y) = n_z \ , \quad \sin(\alpha_y) = \sqrt{1-n_z^2} \ . \tag{2.57}$$

   Die Transformationsmatrix lautet:

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} \cos(\alpha_y) & 0 & -sin(\alpha_y) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha_y) & 0 & \cos(\alpha_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.58}$$

   bzw.

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} n_z & 0 & -\sqrt{1-n_z^2} & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{1-n_z^2} & 0 & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ . \tag{2.59}$$

3. Projektion auf die x-y-Ebene.
   Die Projektionsmatrix lautet gemäß Gl.(2.50):

$$\underline{\underline{P_{xy}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.60}$$

4. Gesamtmatrix der Projektion.
   Die gesamte Projektionsmatrix lautet damit:

$$\begin{aligned} \underline{\underline{P_n}} &= \left(\underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z)\right)^{-1} \cdot \underline{\underline{P_{xy}}} \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \\ \underline{\underline{P_n}} &= \underline{\underline{R_z}}^{-1}(-\alpha_y) \cdot \underline{\underline{R_y}}^{-1}(-\alpha_z) \cdot \underline{\underline{P_{xy}}} \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \\ \underline{\underline{P_n}} &= \underline{\underline{R_z}}(\alpha_y) \cdot \underline{\underline{R_y}}(\alpha_z) \cdot \underline{\underline{P_{xy}}} \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \end{aligned} \tag{2.61}$$

2.2.4 Zentralprojektion

Die Zentralprojektion entspricht der 1-Punkt-Perspektive. Alle Projektionsgeraden schneiden sich in einem Punkt. Dieser Punkt wird in den Koordinatenursprung gelegt. Im Abstand d vom Ursprung befindet sich die Projektionsebene. Das folgende Bild 2.6 veranschaulicht die Zentralprojektion.

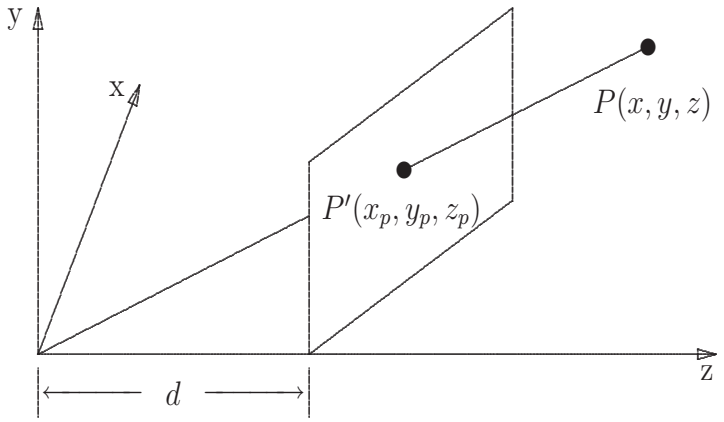

Abbildung 2.6: Zentralprojektion

Der Punkt $P(x,y,z)$ eines Objektes wird in den Punkt $P'(x,y,z)$ der Projektionsebene abgebildet werden. Diese befindet sich im Abstand d vom Koordinatenursprung. Aus dem Strahlensatz gelten die Beziehungen:

$$x_p = \frac{x \cdot d}{z} , \quad y_p = \frac{y \cdot d}{z} , \quad z_p = d \tag{2.62}$$

In der Darstellung der homogenen Koordinaten erhält man die Projektionsmatrix:

$$\underline{\underline{P_Z}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \tag{2.63}$$

2.3 Koordinatensysteme

Zur Darstellung einer virtuellen Welt sind verschiedene Koordinatenräume zweckmäßig. Je nach Bedarf wechselt man von einem Koordinatensystem in ein anderes. Die Umrechnung der Koordinaten erfolgt mit Hilfe von Transformationen und Projektionen. Diese wurden in den Abschnitten 2.1 und 2.2 näher betrachtet.

Folgende Koordinatensysteme werden verwendet:

- Objektkoordinatensystem
  Objekt-koordinaten-system
  Das Objektkoordinatensystem (Object System) wird auch lokales Koordinatensystem genannt. Im Objektkoordinatensystem sind die Objekte definiert. Das folgende Bild 2.7 zeigt einige Objekte mit ihrem lokalen Koordinatensystem.

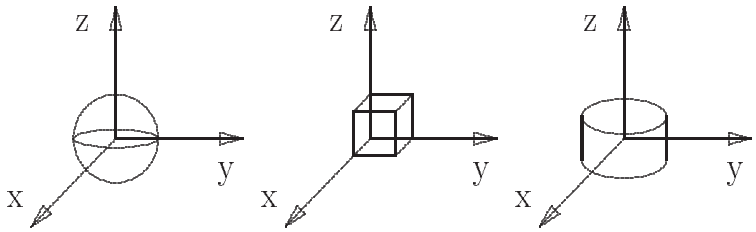

Abbildung 2.7: Objektkoordinatensystem

  Im lokalen Koordinatensystem liegen die Objekte beispielsweise als Primitive, Polygone, Bézier- bzw. Spline-Flächen vor.

- Weltkoordinatensystem
  Welt-koordinaten-system
  Die virtuelle Welt wird in diesem Koordinatensystem konstruiert. Die einzelnen Objekte werden in das Weltkoordinatensystem (World System) transformiert. Dort nehmen die Objekte ihre eigentliche Größe an und ihre Position ein. Das Weltkoordinatensystem schafft einen räumlichen Bezug zwischen den einzelnen Objekten. Das folgende Bild 2.8 zeigt die im Weltkoordinatensystem platzierten Objekte. Als Objekte sind primitive geometrische Körper dargestellt.

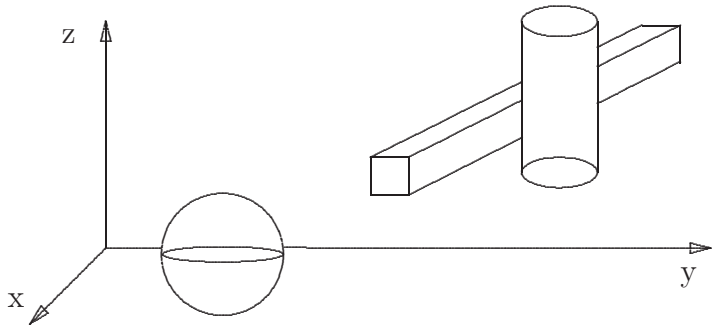

Abbildung 2.8: Weltkoordinatensystem

- Betrachterkoordinatensystem
  Betrachter-koordinaten-system
  Das Weltkoordinatensystem möchte man von allen Richtungen aus betrachten. Dieses erfolgt mit Hilfe des so genannten Betrachterkoordinatensystems (View Reference System). Dazu wird eine Betrachtungsrichtung und der Abstand vom Ursprung des

Betrachterkoordinatensystems, dem Fluchtpunkt, vorgegeben. Betrachterrichtung und Fluchtpunkt legen die Betrachterebene fest und damit auch das Betrachterkoordinatensystem.

Durch die Projektion der Welt auf die Betrachterebene entsteht ein sichtbares Volumen (View Volume) das einem Pyramidenstumpf gleicht. Dieser Pyramidenstumpf wird als Sichtbarkeitspyramide (View Frustum) bezeichnet. Das folgende Bild 2.9 zeigt die Sichtbarkeitspyramide mit dem enthaltenen Weltkoordinatensystem.

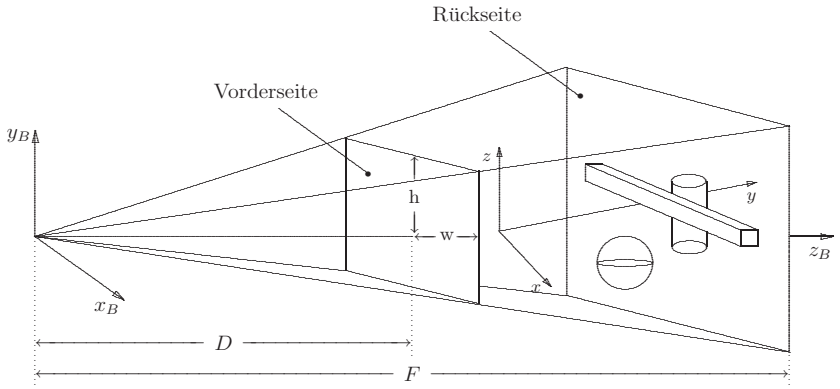

Abbildung 2.9: Betrachterkoordinatensystem

Das Betrachtersystem wird sehr häufig als linksdrehendes Koordinatensystem definiert. Dem zugrunde liegt die Betrachtungsweise, dass mit größer werdendem z-Wert die Entfernung zunimmt. Gleichzeitig möchte man in der Betrachterebene die y-Achse nach oben und die x-Achse nach rechts orientieren. Im Bild 2.9 ist ein linksdrehendes Koordinatensystem für die Sichtbarkeitspyramide dargestellt.

Die Festlegung des Betrachterkoordinatensystem erfolgt durch die Lage der Betrachterebene. Die Betrachterebene wird durch die zwei Vektoren $\vec{u}$ und $\vec{v}$ festgelegt wird. Die Vektoren stehen nicht notwendigerweise senkrecht zueinander. Der Vektor $\vec{v}$ zeigt dabei stets nach oben, er wird als View Up Vektor bezeichnet. Der Vektor $\vec{u}$ gibt die Betrachterrichtung an. Der Endpunkt des Vektor $\vec{u}$ wird als Normal Reference Point bezeichnet. Der Ortsvektor $\vec{r}$ gibt die Lage des Betrachterkoordinatensystems in Bezug auf den Weltkoordinatenursprung an. Der Endpunkt des Ortsvektors $\vec{r}$ wird als View Reference Point bezeichnet. Der Abstand D markiert den Fluchtpunkt. Er liegt auf der vom Vektor $\vec{u}$ festgelegten Geraden. Im Bild 2.10 ist der Zusammenhang zwischen Welt- und Betrachterkoordinatensystem dargestellt.

Für die Transformation des Weltkoordinatensystems in das Betrachterkoordinatensystem sind mehrere Schritte notwendig.

1. Berechnung der Achsen des Betrachterkoordinatensystems
   Der Vektor $\vec{u}$ gibt die Betrachterrichtung an. Er dient zur Festlegung der z-Achse des Betrachterkoordinatensystems. Der Vektor $\vec{u}$ wird damit zum Normalenvektor der Betrachterebene. Der View Up Vektor $\vec{v}$ liegt nicht notwendigerweise in der Betrachterebene liegen und ist damit auch nicht notwendigerweise senkrecht zum Vektor $\vec{u}$. Der Vektor $\vec{v}$ muss deshalb erst in die Betrachterebene projiziert werden. Die dritte Achse des Betrachterkoordinatensystems erhält man schließlich über das Kreuzprodukt. Im einzelnen folgt für die Einheitsvektoren $(\vec{e}_{x_B}, \vec{e}_{y_B}, \vec{e}_{z_B})$ des Betrachterkoordinatensystems:

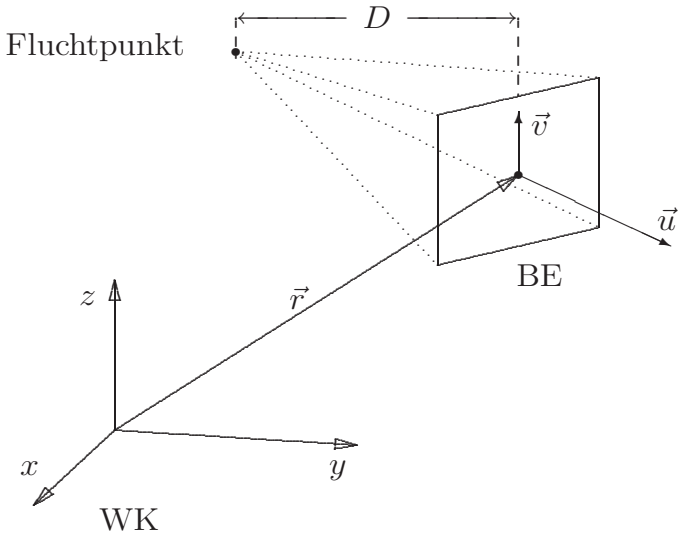

Abbildung 2.10: Zusammenhang Welt- und Betrachterkoordinatensystem

$$\begin{aligned} \vec{e}_{z_B} &= \frac{\vec{u}}{|\vec{u}|} = \vec{n} \\ \vec{e}_{y_B} &= \frac{\vec{v} - (\vec{u} \circ \vec{v}) \cdot \vec{u}}{|\vec{v} - (\vec{u} \circ \vec{v}) \cdot \vec{u}|} \\ \vec{e}_{x_B} &= \vec{e}_{y_B} \times \vec{e}_{z_B} \end{aligned} \tag{2.64}$$

Damit liegt das Betrachterkoordinatensystem fest.

2. Verschiebung des Betrachterkoordinatensystems
Die Verschiebung des in den Ursprung des Weltkoordinatensystem erfolgt um den Ortsvektor $\vec{r}$ gemäß der Translationsmatrix von Gl.(2.3).

$$\underline{\underline{T_v}}(-r_x, -r_y, -r_z) = \begin{bmatrix} 1 & 0 & 0 & -r_x \\ 0 & 1 & 0 & -r_y \\ 0 & 0 & 1 & -r_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.65}$$

3. Drehung um die z-Achse
Der Normalenvektor muss um den Winkel $-\alpha_z$ um die z-Achse gedreht werden, bis der Normalenvektor in x-z-Ebene liegt. Der Winkel α_z berechnet sich aus:

$$\begin{aligned} \cos(\alpha_z) &= \frac{n_x}{\sqrt{n_x^2 + n_y^2}} = \frac{n_x}{\sqrt{1 - n_z^2}} \\ \sin(\alpha_z) &= \frac{n_y}{\sqrt{n_x^2 + n_y^2}} = \frac{n_y}{\sqrt{1 - n_z^2}} \end{aligned} \tag{2.66}$$

Die Transformationsmatrix lautet gemäß Gl.(2.42):

$$\underline{\underline{R_z}}(-\alpha_z) = \begin{bmatrix} \frac{n_x}{\sqrt{1-n_z^2}} & \frac{n_y}{\sqrt{1-n_z^2}} & 0 & 0 \\ \frac{-n_y}{\sqrt{1-n_z^2}} & \frac{n_y}{\sqrt{1-n_z^2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.67}$$

4. Drehung um die y-Achse
   Der Normalenvektor wird so gedreht, dass er parallel zur z-Achse liegt. Der Winkel α_y berechnet sich aus:

$$\cos(\alpha_y) = \frac{n_z}{\sqrt{n_x^2 + n_y^2 + n_z^2}} = n_z \tag{2.68}$$

$$\tag{2.69}$$

$$\sin(\alpha_y) = \frac{\sqrt{n_x^2 + n_y^2}}{\sqrt{n_x^2 + n_y^2 + n_z^2}} = \sqrt{n_x^2 + n_y^2} = \sqrt{1 - n_z^2}. \tag{2.70}$$

$$\tag{2.71}$$

   Die Transformationsmatrix lautet gemäß Gl.(2.46):

$$\underline{\underline{R_y}}(-\alpha_y) = \begin{bmatrix} n_z & 0 & -\sqrt{1-n_z^2} & 0 \\ 0 & 1 & 0 & 0 \\ \sqrt{1-n_z^2} & 0 & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.72}$$

5. Angleichung der Koordinatenachsen
   Zwar sind die z-Achsen der beiden Koordinatensysteme parallel in einer Line, jedoch sind die x- und y- Achsen noch nicht ausgerichtet. Sie sind noch um den Winkel β_z zueinander versetzt. Der Winkel β_z berechnet sich aus dem Skalarprodukt z.B. der beiden x-Achsen oder der beiden y-Achsen. Es muss um die z-Achse um den Winkel $-\beta_z$ noch gedreht werden. Es gilt:

$$\begin{aligned} \cos(-\beta_z) &= (e_x \circ e_{x_B}) \\ \beta_z &= -\arccos\left((e_x \circ e_{x_B})\right) \\ \sin(-\beta_z) &= \sqrt{1 - cos^2(\beta_z)} \end{aligned}$$

   Die zugehörige Transformationsmatrix lautet:

$$\underline{\underline{R_z}}(-\beta_z) = \begin{bmatrix} \cos(\beta_z) & \sin(\beta_z) & 0 & 0 \\ -\sin(\beta_z) & \cos(\beta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{2.73}$$

6. Verschiebung der Betrachterebene entlang der z-Achse
   Die Betrachterebene muss schließlich um den Abstand D entlang der z-Achse verschoben werden. Die Verschiebung um $(0, 0, D)$ erfolgt gemäß der Translationsmatrix von Gl.(2.3).

$$\underline{\underline{T_v}}(0,0,D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.74}$$

7. Gesamte Transformation
   Die gesamte Transformationsmatrix $\underline{\underline{T}}_{BW}$ lautet damit:

$$\underline{\underline{T}}_{BW} = \Big(\underline{\underline{T_v}}(0,0,D) \cdot \underline{\underline{R_z}}(-\beta_z) \cdot \underline{\underline{R_y}}(-\alpha_y) \cdot \underline{\underline{R_z}}(-\alpha_z) \cdot \underline{\underline{T_v}}(-r_x,-r_y,-r_z)\Big) \tag{2.75}$$

   Die Transformationsmatrix $\underline{\underline{T}}_{BW}$ transformiert das Betrachterkoordinatensystem in das Weltkoordinatensystem und zwar so, dass die z-Achse des Weltkoordinatensystems senkrecht zur Betrachterebene ist. Die Betrachterebene befindet sich im Abstand D vom Ursprung und liegt parallel zur x-y-Ebene. Damit liegt der Fluchtpunkt im Koordinatenursprung. Das Bild 2.11 zeigt die Lage der transformierten Betrachterebene im Weltkoordinatensystem.

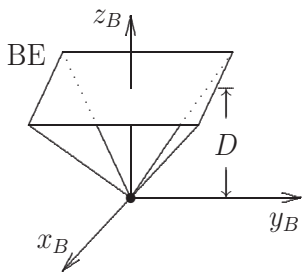

Abbildung 2.11: Betrachterebene im Weltkoordinatensystem

   Durch die Inversion der Transformationsmatrix $\underline{\underline{T}}_{BW}^{-1} = \underline{\underline{T}}_{WB}$ wird das Weltkoordinatensystem in das Betrachterkoordinatensystem transformiert.

8. Zentralprojektion
   Abschließend muss noch die Zentralprojektion erfolgen. Die Projektionsmatrix lautet entsprechend Gl.(2.63):

$$\underline{\underline{P_Z}}(D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{D} & 0 \end{bmatrix} \tag{2.76}$$

   Die resultierende Transformationsmatrix lautet schließlich:

$$\begin{aligned} \underline{\underline{T}}_{BE} &= \underline{\underline{P_Z}}(D) \cdot \underline{\underline{T}}_{WB} \\ \underline{\underline{T}}_{BE} &= \underline{\underline{P_Z}}(D) \cdot \underline{\underline{T_v}}(r_x,r_y,r_z) \cdot \underline{\underline{R_z}}(\alpha_z) \cdot \underline{\underline{R_y}}(\alpha_y) \cdot \underline{\underline{R_z}}(\beta_z) \cdot \underline{\underline{T_v}}(0,0,-D) \end{aligned} \tag{2.77}$$

► Einheitswürfel
Einheitswürfel

Aus Gründen der einfacheren Handhabung für weitere Algorithmen wird die Sichtbarkeitspyramide in einen Einheitswürfel abgebildet. Es sei $z_B = D$ die Entfernung der Vorderseite und $z_B = F$ die Entfernung der Rückseite vom Koordinatenursprung. Die Ebenengleichungen der Seitenflächen der Sichtbarkeitspyramide lauten damit:

$$\begin{aligned} z_B &= D \quad , \quad z_B = F \\ x_B &= \pm w \cdot \frac{z_B}{D} \\ y_B &= \pm h \cdot \frac{z_B}{D} \ . \end{aligned} \tag{2.78}$$

Das folgende Bild 2.12 zeigt den Einheitwürfel.

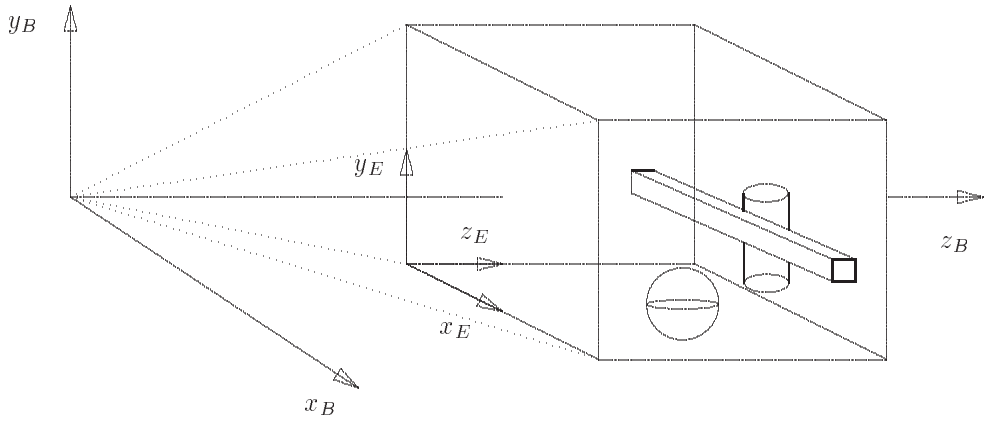

Abbildung 2.12: Betrachterkoordinatensystem als Einheitswürfel

Für die Transformationsvorschrift des Pyramidenstumpfes auf den Einheitswürfel gelten folgende Vorschriften.

$$\begin{aligned} x_E &= x_B \cdot \frac{D}{z_B} \cdot \frac{1}{w} \ , \quad x_E \in [-1,1] \\ y_E &= y_B \cdot \frac{D}{z_B} \cdot \frac{1}{h} \ , \quad y_E \in [-1,1] \\ z_E &= \frac{z_B - D}{F - D} \cdot \frac{F}{z_B} \ , \quad z_E \in [\,0,1] \end{aligned} \tag{2.79}$$

In homogenen Koordinaten lautet die Transformation:

$$\begin{aligned} x_E &= x_B \cdot \frac{1}{w} \\ y_E &= y_B \cdot \frac{1}{h} \\ z_E &= \frac{F \cdot z_B}{D \cdot (F - D)} - \frac{F}{F - D} \\ w_E &= \frac{z_B}{D} \end{aligned} \tag{2.80}$$

$$\begin{bmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{1}{h} & 0 & 0 \\ 0 & 0 & \frac{F \cdot z_B}{D \cdot (F-D)} & -\frac{1}{D} \\ 0 & 0 & -\frac{F}{F-D} & 0 \end{bmatrix} \cdot \begin{bmatrix} x_B \\ y_B \\ z_B \\ 1 \end{bmatrix} = \begin{bmatrix} x_E \\ y_E \\ z_E \\ w_E \end{bmatrix} . \tag{2.81}$$

- Koordinatensystem der Betrachterebene

Koordinatensystem der Betrachterebene

Die Objekte, die sich innerhalb der Sichtbarkeitspyramide befinden, werden auf die Betrachterebene projiziert. Dieses ist gleichbedeutend mit der Projektion der Objekte auf die x_E-y_E-Ebene innerhalb des Einheitswürfels. Die Projektionsmatrix lautet hierfür:

$$\underline{\underline{P_{xy}}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.82}$$

Die Vorderfront des Einheitswürfels umfasst die Fläche $[-1,1] \times [-1,1]$. Eventuell wird das Fenster der Betrachterebene noch auf das Einheitsquadrat $[0,1] \times [0,1]$ normiert. Dazu ist eine Translation und eine Skalierung notwendig. Man erhält dann geräteunabhängige Koordinaten. Die beiden Transformationsmatrizen lauten:

$$\underline{\underline{S_K}}(1,1) \cdot \underline{\underline{T_v}}(1,1) = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} . \tag{2.83}$$

- Koordinatensystem des Ausgabegerätes

Koordinatensystem des Ausgabegerätes

Das Koordinatensystem des Ausgabegerätes (View Port) spiegelt die realen Koordinaten des Ausgabesystems wider. Dazu wird die Betrachterebene auf das Fenster der Ausgabeeinheit transformiert. Diese Transformation wird als Window-View-Port-Transformation bezeichnet. Das folgende Bild 2.13 veranschaulicht die Window-View-Port-Transformation.

Hat der Viewport die Abmessungen $[v_{x_{min}}, v_{y_{min}}] \times [v_{x_{max}}, v_{y_{max}}]$ und das Fenster (Window) der Betrachterebene die Abmessungen $[-w, w] \times [-h, h]$ (siehe Bild 2.9), so sind für die Window-View-Port-Transformation drei Transformationsschritte notwendig:

1. Verschiebung des Koordinatenursprungs in die linke untere Ecke des Windows um den Vektor $\vec{v} = (-w, -h)^T$.
2. Skalierung des Koordinatensystems, so dass das Window die Maße des Viewports annimmt.
3. Verschiebung des Koordinatensystems, so dass das Fenster an die Postion des Viewports gelangt, um den Vektor $\vec{v} = (v_{x_{min}}, v_{y_{min}})^T$.

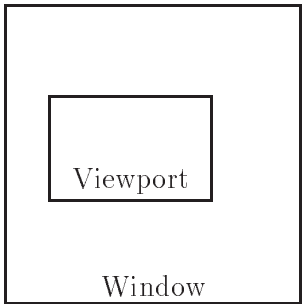

Abbildung 2.13: Window-Viewport-Transformation

Die drei Matrizen lauten:

$$\begin{bmatrix} 1 & 0 & v_{x_{min}} \\ 0 & 1 & v_{y_{min}} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -w \\ 0 & 1 & -h \\ 0 & 0 & 1 \end{bmatrix} , \tag{2.84}$$

$$s_x = \frac{v_{x_{max}} - v_{x_{min}}}{2 \cdot w} , \quad s_y = \frac{v_{y_{max}} - v_{y_{min}}}{2 \cdot h} .$$

2.4 Modellierung von Körpern

In der Computergraphik gibt es verschiedene Methoden dreidimensionale Objekte zu modellieren. Diese Methoden lassen sich in die folgenden Gruppen unterteilen.

- Modellierung von Körpern mit Hilfe von primitiven geometrischen Grundkörpern
- Modellierung von Körpern als Translationskörper
- Modellierung von Körpern mit Hilfe der räumlichen Zerlegung
- Modellierung von Körpern mit Hilfe der Randdarstellung

Welche Methode verwendet wird, hängt von der jeweiligen Anwendung ab. Die genannten Methoden werden kurz vorgestellt.

2.4.1 Modellierung von Körpern mit Hilfe von primitiven geometrischen Grundkörpern

Primitive

Constructive Solid Geometry

Bei der Modellierung von Körpern mit Hilfe von Primitiven versucht man komplexere Körper durch einfache geometrische Körper aufzubauen. Einfache geometrische Körper sind z.B. Kugel, Kegel, Würfel, Zylinder etc. Die Gestalt eines primitiven Objekt wird durch Parameter festgelegt, wie beispielsweise der Radius einer Kugel oder die Kantenlängen eines Würfels. Diese Vorgehensweise wird als “Constructive Solid Geometry” bezeichnet. Der Aufbau komplexer Körper aus primitiven Objekten kann durch lineare Transformationen, arithmetische und Boolesche Operationen beschrieben werden. Das folgende Bild 2.14 zeigt, wie aus einem Zylinder und einer Platte ein neuer Körper konstruiert wird.

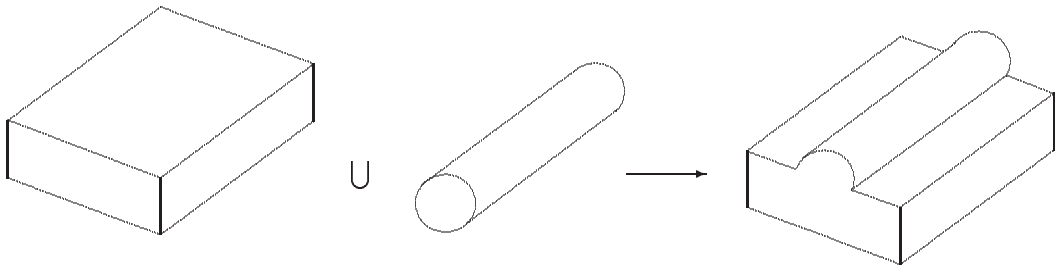

Abbildung 2.14: Modellierung mit Primitiven

Für die Konstruktion kommen folgende Operationen in Betracht:

- Boolesche Mengenoperationen, wie z.B. $\bigcup$, $\bigcap$,
- Arithmetische Operationen, wie z.B. $+$, $-$.

Der Aufbau komplexer Körper kann in Form einer binären Baumstruktur abgelegt werden. Die Abarbeitung entspricht derselben Vorgehensweise wie bei der umgekehrten polnischen Notation.

2.4.2 Modellierung als Translationskörper

Bewegt man ein Objekt entlang einer vorgegebenen Trajektorie, so entsteht ein neuer dreidimensionaler Körper, wenn man das überstrichene Volumen markiert. Das Objekt kann während der Bewegung die Form ändern. Dieser Vorgang wird als Extrusion oder auch als Sweeping bezeichnet. Die resultierenden Körper nennt man verallgemeinerte Translationskörper oder auch Sweep.

Sweeping

Translationskörper

Sweep

Ein Spezialfall sind die einfachen Rotationskörper. Rotationskörper entstehen, wenn eine erzeugende Fläche um eine Raumachse rotiert wird. Im Bild 2.15 ist ein Beispiel für einen

Rotationskörper dargestellt.

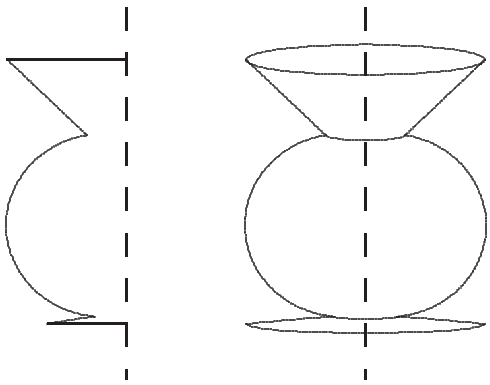

Abbildung 2.15: Modellierung als Rotationskörpern

2.4.3 Modellierung von Körpern mit Hilfe der räumlichen Zerlegung

Zellenzerlegung
Cell Decomposition

Bei der Methode der räumlichen Zerlegung wird ein dreidimensionaler Körper in einfachere Körper aufgeteilt. Dieser Vorgang wird als Zellenzerlegung oder auch Cell Decomposition bezeichnet. Diese Methode stellt die Umkehrung zur Konstruktion mit Primitiven dar. Einen Sonderfall stellt die so genannte räumliche Enumeration dar, dabei wird der Körper in identische Zellen erlegt, die auf einem festen, regelmäßigen dreidimensionalen Gitter angeordnet sind. Als Zellen verwendet man kleine Würfel. In Anlehnung an den Aufbau eines zweidimensionalen Bildes, wo jeder Bildpunkt durch ein Pixel beschrieben wird, nennt man hier ein Volumenelement Voxel. Diese Methode ist extrem speicherintensiv. Sie wird vorwiegend in der Medizintechnik bei der dreidimensionalen Visualisierung von Computer-Tomographie-Daten angewandt. Für Anwendung auf dem Gebiet der virtuelle Realität ist diese Methode zu aufwendig. Das folgende Bild 2.16 zeigt ein Beispiel für die räumliche Zellenzerlegung für einen Torus.

Voxel

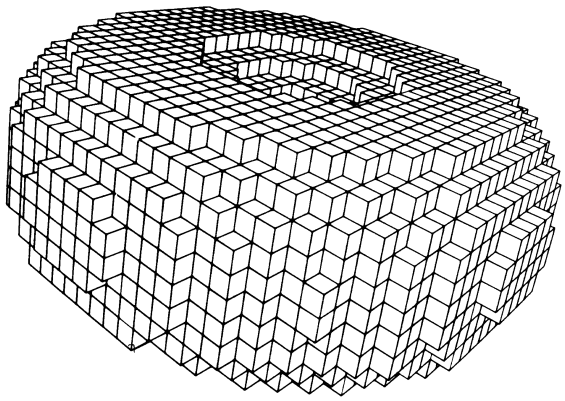

Abbildung 2.16: Torus zerlegt in Würfelzellen

2.4.4 Modellierung von Körpern mit Hilfe der Randdarstellung

Die Repräsentation von Körpern durch ihre umgebenden Flächen bezeichnet man als Randdarstellung oder auch als Boundary Representation. Für die Flächenbeschreibung der Berandungen kommen folgende Flächenformen in Betracht.

Boundary Representation

- Ebene Flächenstücke, aufgebaut aus Polygonen oder Dreiecken.
- Freiformflächen, bestehend z.B. aus Bézier-Flächen oder Non-Uniform Rational B-Spline-Flächen (NURBS).

Bei VR-Anwendungen ist die Darstellung mit ebenen Flächenstücken am gebräuchlichsten. Freiformflächen sind dagegen für VR-Anwendungen noch zu aufwendig, sie werden überwiegend bei CAD-Anwendungen eingesetzt.

2.4.4.1 Randdarstellung mit Hilfe von Polygonen

Ein Polygon ist ein ebenes Vieleck, dass durch seine Eckpunkte festgelegt wird. Das Bild 2.17 zeigt ein Polygon. Zusätzlich ist in jedem Eckpunkt der zugehörige Normalenvektor mit dargestellt.

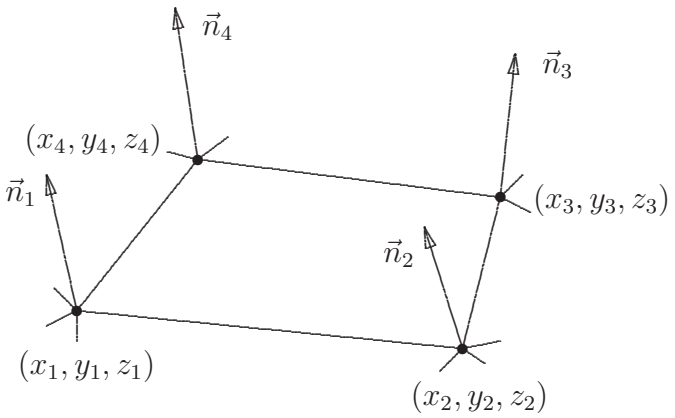

Abbildung 2.17: Polygon mit Normalenvektoren

Die polygonale Repräsentation ist die gebräuchlichste Darstellungsform bei VR-Anwendungen.

Eine polygonale Repräsentation eines Körpers besitzt folgende Vorteile:

- Einfache Generierung von Ploygonen
  Polygone sind sehr einfach zu erzeugen. Die dreidimensionalen Abtastwerte, die bei der Vermessung eines Körpers entstehen, werden als Polygonpunkte verwendet.

- Beliebige Approximation
  Stark gekrümmte Objektbereiche lassen sich durch kleine Polygone annähern. Bei wenig gekrümmten Objektbereichen genügen größere Polygonfläche.

- Einfache Triangulation
  Polygone lassen sich sehr einfach in Dreiecke zerlegen. Die Zerlegung von Polygonen in Dreiecke bezeichnet man als Triangulation. Dreiecke sind die kleinste unabhängige Einheit, die ein Graphikprozessor verarbeiten kann.

  Triangulation

- Einfache Schattierung
  Schattierungsalgorithmen sind auf Polygone bzw. Dreiecke sehr einfach anwendbar.

Das folgende Bild 2.18 zeigt den Aufbau eines Zylinders aus Polygonen.

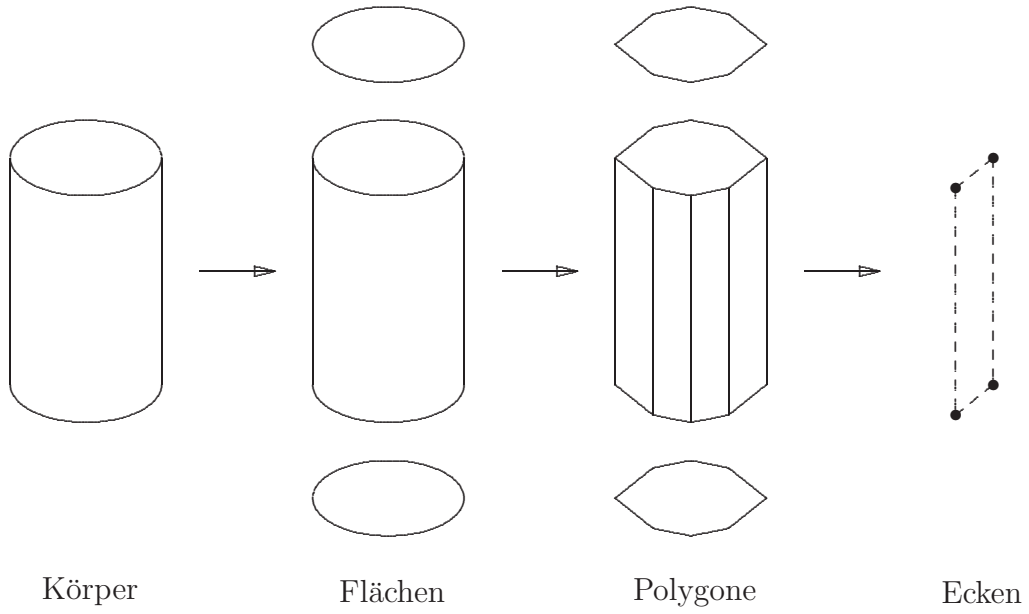

Abbildung 2.18: Darstellung eines Zylinders durch Polygone

Da Normalenvektoren in vielen Algorithmen benötigt werden, die Bestimmung der Normalenvektoren aber sehr viel Zeit in Anspruch nimmt, berechnet man sie häufig vorab und speichert die Normalenvektoren, die in den Eckpunkten vorliegen, zusammen mit den Koordinaten der Eckpunkte ab. Eventuell werden weitere topologische Informationen über die Körperberandung mit abgespeichert.

Wird das dreidimensionale Objekt durch eine geometrische Transformation verändert, so müssen selbstverständlich auch die Normalenvektoren angepasst werden. Die Transformation von Normalenvektoren ist im Abschnitt 2.1.6 in Gl.(2.49) diskutiert worden.

Die verwendete Datenstruktur für die Speicherung der Polygoneckpunkte und der zugehörigen Normalenvektoren ist ein Baum. Im Bild 2.19 ist die Datenstruktur dargestellt.

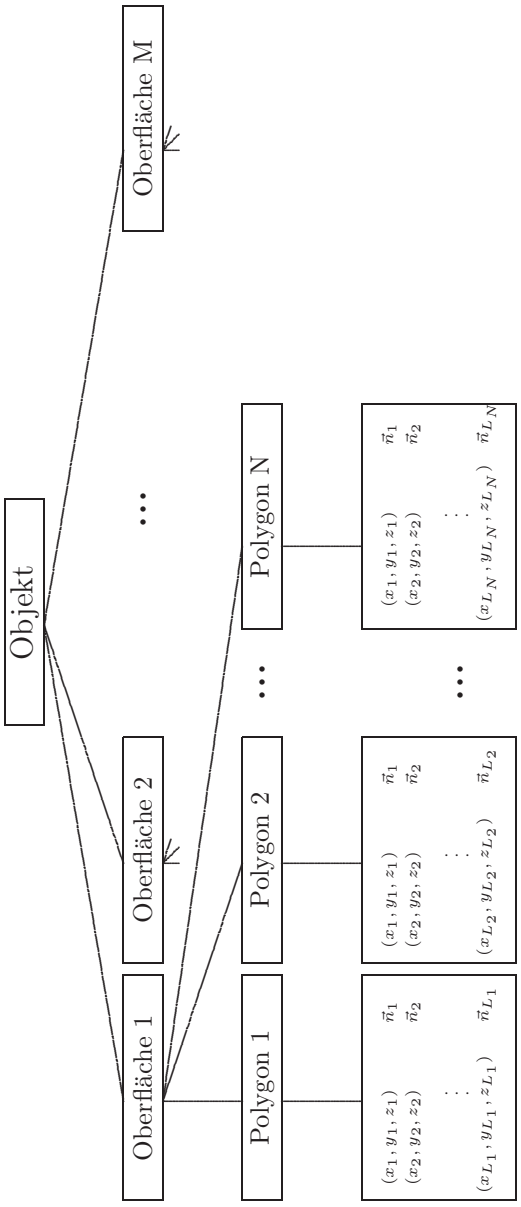

Abbildung 2.19: Datenstruktur für ein polygonales Objekt

2.4.4.2 Randdarstellung mit Hilfe von Bézier-Flächen

In den Sechziger Jahren entwickelte der französische Mathematiker Pierre Bézier parametrische Polynome zur Beschreibung von Freiformflächen. Er setzte seine Ergebnisse in ein CAD-System um, das bis Mitte der siebziger Jahre bei der Firma Renault zur Entwicklung von Fahrzeugkarrosserien eingesetzt wurde. Inzwischen sind Bézier-Polynome zu einem wichtigem Bestandteil in der Computergraphik geworden. Bézier-Polynome gestatten eine sehr einfache Möglichkeit zur Approximation und zur Interpolation von Freiformflächen.

Bézier-Polynomen besitzen folgende Vorteile:

- Es muss kein lineares Gleichungssystem wie bei der Verwendung von Spline-Funktionen gelöst werden. Es werden keine Ableitungen der Freiformfläche benötigt. Die Interpolationsvorschrift resultiert direkt aus den Stützpunkten.
- Bézier-Polynome erfüllen von sich aus die Tangentialbedingung. Funktionswert und Ableitung sind am Rand gleich und damit ist eine stetige und glatte Fortsetzung garantiert.
- Bézier-Polynome verlaufen stets innerhalb der Stützpunkte, Die Verbindungsgeraden bilden damit die Einhüllende der Interpolation.

Ein eindimensionales Bézier-Basispolynome vom Grad N hat die Form:

$$B_{i,N}(t) = \binom{N}{i} \cdot t^i \cdot (1-t)^{N-i} \ , \quad i = 0(1)N \tag{2.85}$$

Für den Grad N gibt es $N+1$ verschiedene Bézier-Basispolynome. Die Bézier-Basispolynomene werden bei einer Interpolation mit den Stützpunkten gewichtet.
Für eine Kurve in der Ebene, die durch die Stützpunkte $P_i \ , i = 0(1)N$ gegeben ist, ergibt sich die Interpolationsvorschrift:

$$\vec{P}(t) = \sum_{i=0}^{N} \binom{N}{i} \cdot t^i \cdot (1-t)^{N-i} \cdot \vec{P}_i \tag{2.86}$$

Setzt man die Grenzen $t = 0$ und $t = 1$ in Gl.(2.86) ein, so erhält man:

$$\begin{array}{lcl} \vec{P}(0) & = & \vec{P}_0 \\ \vec{P}(1) & = & \vec{P}_N \\ \vec{P}'(0) & = & N \cdot (\vec{P}_1 - \vec{P}_0) \\ \vec{P}'(1) & = & N \cdot (\vec{P}_{N-1} - \vec{P}_N) \end{array} \tag{2.87}$$

Häufig verwendet man Polynome vom Grad 3 auf. Die vier Bézier-Basispolynome lauten gemäß Gl.(2.86):

$$\begin{array}{lcl} B_{0,3}(t) & = & (1-t)^3 \\ B_{1,3}(t) & = & 3 \cdot t \cdot (1-t)^2 \\ B_{2,3}(t) & = & 3 \cdot t^2 \cdot (1-t) \\ B_{3,3}(t) & = & t^3 \end{array} \tag{2.88}$$

Das Bild 2.20 zeigt diese vier Basispolynome. Die Variable t nimmt nur Werte zwischen $0 \leq t \leq 1$ an.

Bézier-Flächen entstehen durch direkte Erweiterung der eindimensionalen Interpolation von Gl.(2.87) auf den zweidimensionalen Fall. Für ein Gitter mit $N+1 \times M+1$ Stützpunkten $\vec{P}_{i,j}, \ i = 0(1)N, \ j = 0(1)M$ entsteht durch die folgende Interpolationsvorschrift die Bézier-Fläche.

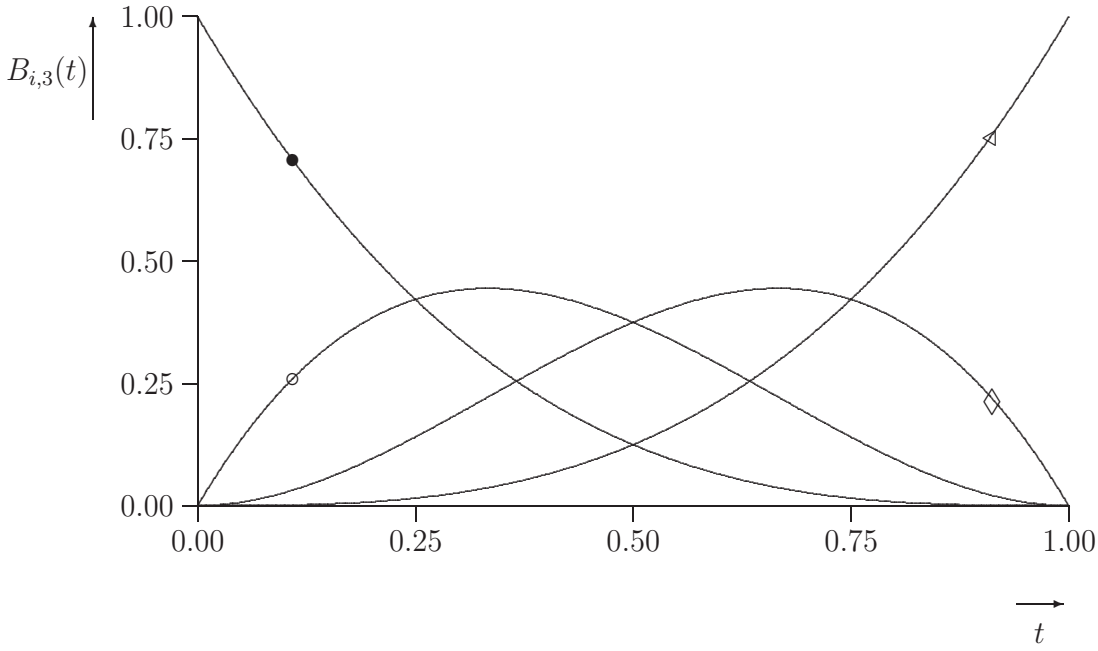

Abbildung 2.20: Bézier-Basispolynome vom Grad 3
• $B_{0,3}(t)$ ○ $B_{1,3}(t)$ ◇ $B_{2,3}(t)$ ◁ $B_{3,3}(t)$

$$\vec{F}(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} B_{i,N}(u) \cdot B_{j,M}(v) \cdot \vec{P}_{i,j} \tag{2.89}$$

Die vorteilhaften Eigenschaften bleiben erhalten.

2.4.4.3 Randdarstellung mit Hilfe von NURBS

NURBS

Die Abkürzung NURBS steht für Non-Uniform Rational B-Splines. NURBS sind inzwischen zu einem Industriestandard zur Darstellung von Freiformflächen im CAD/CAM/CAE-Bereich geworden. NURBS sind rationale kubische Kurvensegmente deren Kontrollpunkte nicht äquidistant vorliegen müssen. Rationale kubische Kurvensegmente entstehen allgemein aus Quotienten von Polynomen. Denkt man sich eine Kurve, deren Kontrollpunkte in homogenen Koordinaten gegeben sind, durch kubische Polynome $(X(t),\ Y(t),\ Z(t),\ W(t))$ angenähert, so erhält man die rationalen Kurvensegmente aus der Vorschrift:

$$x(t) = \frac{X(t)}{W(t)}\ , \quad y(t) = \frac{Y(t)}{W(t)}\ , \quad z(t) = \frac{Z(t)}{W(t)}\ . \tag{2.90}$$

NURBS weisen folgende auf:

- Der Entwurf von Freiformflächen mit Hilfe von NURBS erfolgt intuitiv. Die Kontrollpunkte bestimmen die Freiformfläche.
- NURBS-Kurven und NURBS-Flächen sind invariant bei geometrischen Transformation und Projektionen.
- NURBS gestatten eine einheitliche und geschlossene Darstellung für Kegelschnitte.
- NURBS-Algorithmen sind numerisch stabil.

Bei VR-Anwendungen finden NURBS erst langsam Anwendung. Der Grund liegt darin, dass alle Rendering-Algorithmen bisher auf Polygone bzw. Dreiecke ausgelegt sind. Bei der

Verwendung von NURBS-Flächen müssen diese momentan noch in Dreiecke zerlegt werden. Über kurz oder lang werden aber auch NURBS fester Bestandteil bei VR-Anwendungen sein.

Kapitel 3

Algorithmen der Computergraphik

3.1 Basisalgorithmen

Bei der Abbildung einer dreidimensionalen Szene auf das Ausgabefenster sind verschiedene Verarbeitungsschritte notwendig.

- Entfernung nicht sichtbarer Objekte.ñewline Nachdem alle Objekte im Weltkoordinatensystem plaziert sind, müssen sie in die Sichtbarkeitspyramide abgebildet werden. Dabei können Objekte eventuell außerhalb des Volumens der Sichtbarkeitspyramide liegen. Damit keine unnötigen Berechnungen durchgeführt werden, werden diese Objekte als nicht sichtbar markiert.

- Entfernung nicht sichtbarer Objektflächen.
  Alle Objektflächen, die vom Betrachter abgewandt sind, werden ebenfalls entfernt.

- Ermittlung der verdeckten Objektteile.
  Es wird bestimmt, welche Objektteile anderen Objektteile verdecken.

- Abschneiden nicht sichtbarer Objektteile.
  Bei der Projektion der Objekte auf das Ausgabefenster werden nicht sichtbare Objektteile abgeschnitten.

Für die angegeben Schritte ist jeweils ein Basisalgorithmus notwendig, der im folgenden kurz vorgestellt wird.

3.1.1 Sichtbarkeit eines Objektes

Sichtbarkeitspyramide
Object Clipping
Bounding Box

Zur Prüfung, ob ein Objekt nicht in der Sichtbarkeitspyramide liegt, und damit auch nicht dargestellt werden muss, verwendet man das so genannte Object Clipping. Im allgemeinen wird für jedes Objekt eine Bounding Box mit abgespeichert. Eine Bounding Box stellt aus praktischen Gründen einen einfachen geometrischen Körper dar, der das betreffende Objekt vollständig umschließt. Anhand dieses einfachen geometrischen Körpers, z.B. einer Kugel oder einem Quader, wird geprüft, ob dieser vollständig außerhalb der Sichtbarkeitspyramide liegt. Man spricht dabei auch von einer Einhüllenden oder einer konvexen Hülle.

Konvexe Hülle

Die folgende Überlegung zeigt, welche Bedingung ein Punkt erfüllen muss, damit er in der Sichtbarkeitspyramide liegt.

Aus dem Bild 3.1 wird deutlich, dass jeder Objektpunkt $P(x, y, z)$ die Bedingung

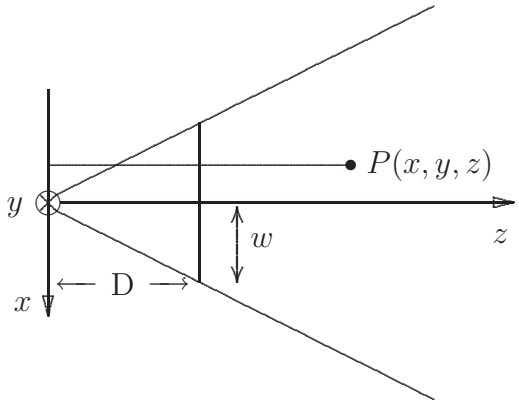

Abbildung 3.1: Sichtbarkeitstest für einen Objektpunkt

$$\begin{aligned} \frac{|x|}{z} &\leq \frac{w}{D}\,, \quad z \geq 0 \\ \frac{|y|}{z} &\leq \frac{h}{D}\,, \quad z \geq 0 \end{aligned} \tag{3.1}$$

erfüllen muss, damit er sichtbar ist. Diese Bedingung folgt unmittelbar aus der Anwendung des Strahlensatzes.

Zur schnelleren Durchführung des Sichtbarkeitstestes, skaliert man den Öffnungswinkel der Sichtbarkeitspyramide auf 45^o. In diesem Fall wird $D = w = h$. Damit wird aus Gl.(3.1) die einfachere Form:

$$\begin{aligned} \frac{|x|}{z} &\leq 1\,, \quad z \geq 0 \\ \frac{|y|}{z} &\leq 1\,, \quad z \geq 0 \end{aligned} \quad \text{bzw.} \quad \begin{aligned} |x| &\leq z\,, \quad z \geq 0 \\ |y| &\leq z\,, \quad z \geq 0 \end{aligned} \tag{3.2}$$

Die zugehörige Transformationsmatrix für die Skalierung lautet mit Gl.(2.6):

$$\underline{\underline{S_K}}(s_x, s_y, s_z) = \begin{bmatrix} \frac{D}{w} & 0 & 0 & 0 \\ 0 & \frac{D}{h} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\,. \tag{3.3}$$

Dabei sind w und h die Breite bzw. die Höhe der Vorderfläche der Sichtbarkeitspyramide und D ist der Abstand der Vorderfläche vom Ursprung.

Ist die Bounding Box ein einfacher geometrischer Körper, so sind nur wenige Koordinatenpunkte zu prüfen.

3.1.2 Back-Face-Culling-Algorithmus

Back Face Culling

Der Back-Face-Culling-Algorithmus dient zum Entfernen von nicht sichtbaren Objektflächen. Der Algorithmus überprüft dazu, ob die Richtung der Flächennormale antiparallel zur Betrachterrichtung liegt. Ist das skalare Produkt von Flächennormale und Betrachterrichtung positiv, so ist die Objektfläche nicht sichtbar. Üblicherweise wird für jedes Polygon der Normalenvektor mit abgespeichert, damit kann die Prüfung schnell durchgeführt werden. Das

folgende Bild 3.2 veranschaulicht diesen Zusammenhang.

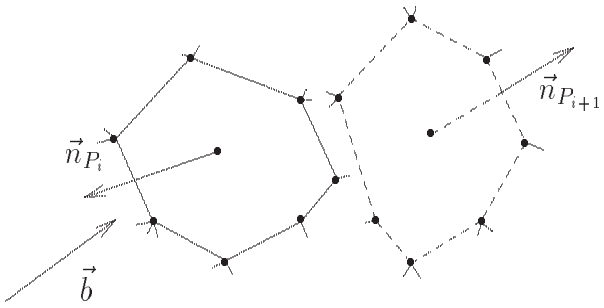

Abbildung 3.2: Sichtbarkeitstest für einen Objektpunkt

Es muss gelten:

$$\vec{b}^{\,T} \cdot \vec{n}_{P_i} < 0 \tag{3.4}$$

Wird die Prüfung von Gl.(3.4) im Koordinatensystem des Betrachters durchgeführt, so ist die Betrachterrichtung entlang der z-Achse orientiert. Die Prüfung von Gl.(3.4) vereinfacht sich dann zu:

$$\begin{array}{rcl} \vec{e}_z \cdot \vec{n}_{P_i} & < & 0 \\ \vec{n}_{z_{P_i}} & < & 0 \end{array} \tag{3.5}$$

Damit muss nur die z-Komponente des transformierten Normalenvektors geprüft werden, ob diese kleiner als Null ist.

Bei nicht konvexen Körpern kann der Back-Face-Algorithmus möglicherweise versagen, da die Relation zwischen Normalenvektor und Betrachterrichtung nicht mehr stimmt. Eventuell auftretende Fehler werden aber durch den anschließenden Z-Buffer-Algorithmus wieder ausgeglichen.

3.1.3 Z-Buffer-Algorithmus

Z-Buffer-Algorithmus

Der Z-Buffer-Algorithmus dient zum Entfernen von verdeckten und nicht sichtbaren Objektteilen. Für den Z-Buffer-Algorithmus wird ein zusätzlicher Speicher, der so genannte "Z-Buffer" benötigt. Dieser Speicher nimmt die z-Koordinate der Objektpunkte auf. Der Z-Buffer muss so viele z-Werte speichern können, wie der Frame-Buffer Bildpunkte umfasst.

Zur Lösung des Verdeckungsproblemes muss die Sichtbarkeit jedes Punktes geprüft werden. Legt man für das Betrachterkoordinatensystem ein linksorientiertes Koordinatensystem zugrunde, so ist ein Objektpunkt $P_1(x_1, y_1, z_1)$ dann sichtbar, wenn es keinen anderen Punkt $P_2(x_1, y_1, z_2)$ gibt, mit $z_2 \leq z_1$. Sichtbar ist also derjenige Objektpunkt mit der kleinsten z-Koordinate. Diese Vorschrift bezieht sich auf das Betrachterkoordinatensystem bzw. auf das Koordinatensystem des Einheitswürfels.
In der Literatur gibt es mehrere Varianten des Z-Buffer-Algorithmus. Allen Varianten arbeiten nach dem folgenden Prinzip.

1. Der Z-Buffer-Speicher wird mit dem größten auftretenden z-Wert initialisiert. Dieses ist üblicherweise die Koordinate $z = 1$, da das sichtbare Volumen normiert vorliegt $0 \leq z \leq 1$.

2. Gleichzeitig wird der Frame-Buffer mit der Farbe des Hintergrundes belegt.

3. Die einzelnen Objekte werden nacheinander mit Hilfe eines Rasterisierungsverfahren in den Frame-Buffer und in den Z-Buffer übertragen. Die Werte des Z-Buffers dienen dabei zur Kontrolle der Sichtbarkeit. Wird ein Objektpunkt, der bereits im Frame-Buffer dargestellt ist, durch einen anderen verdeckt, so wird der verdeckte Objektpunkt durch den sichtbaren überschrieben. Im Z-Buffer wird die kleinere z-Koordinate eingetragen und im Frame-Buffer die Farbe des sichtbaren Objektpunktes.

Zur Veranschaulichung des Z-Buffer-Algorithmus ist im Bild 3.3 die Arbeitsweise dargestellt.

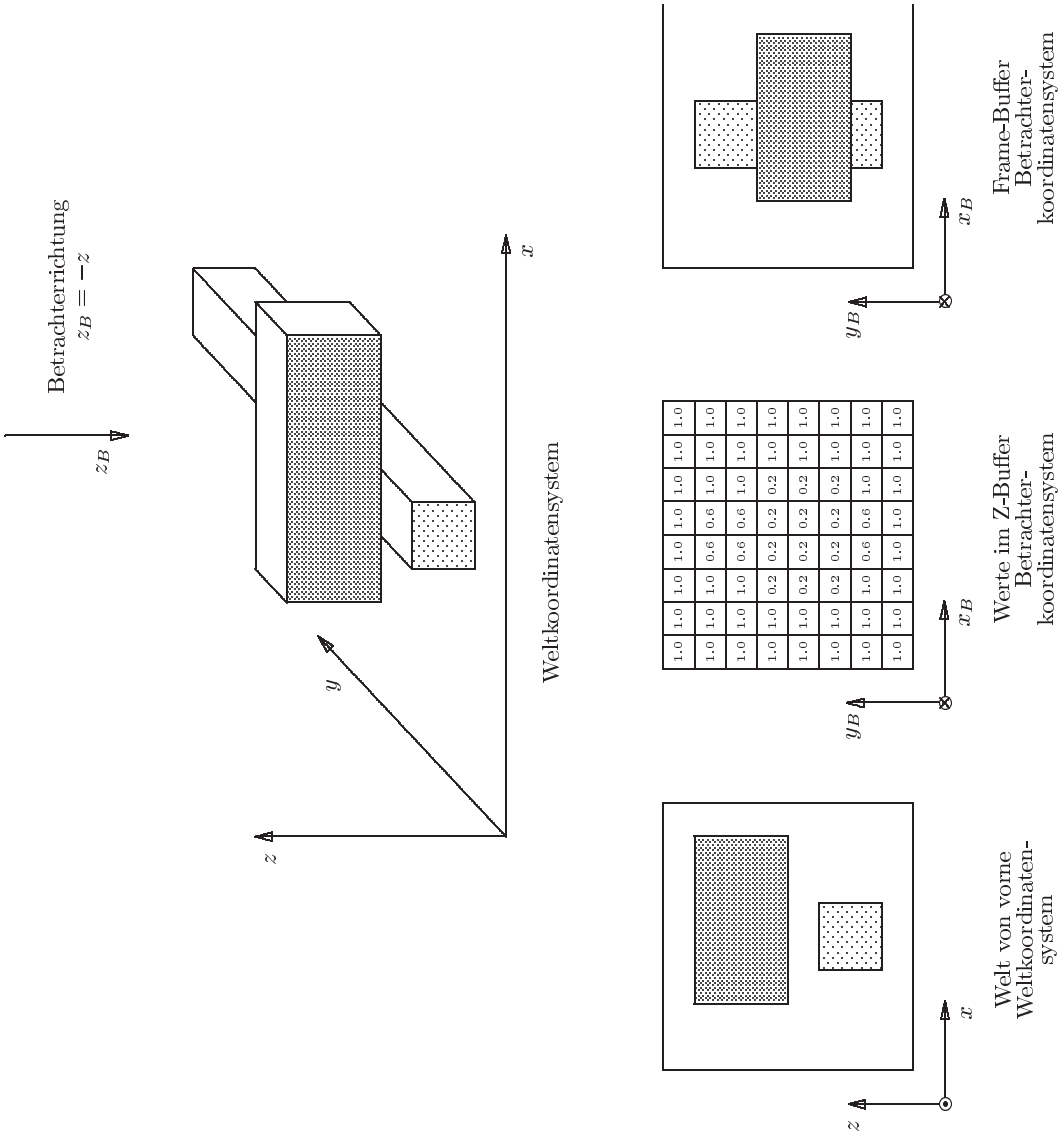

Abbildung 3.3: Z-Buffer-Algorithmus

3.1.4 Clipping

Bei der Projektion der virtuellen Welt auf das Fenster der Betrachterebene müssen sehr häufig Objekte geschnitten werden. Da die Objekte durch Polygone definiert sind, müssen Schnittpunkte von Geradenstücken berechnet werden. Die Berechnung erfolgt im zweidimensionalen Fenster der Betrachterebene. Das folgende Bild 3.4 zeigt mögliche Schnittpunkte des Fensters mit den Geradenstücken der Polygone.

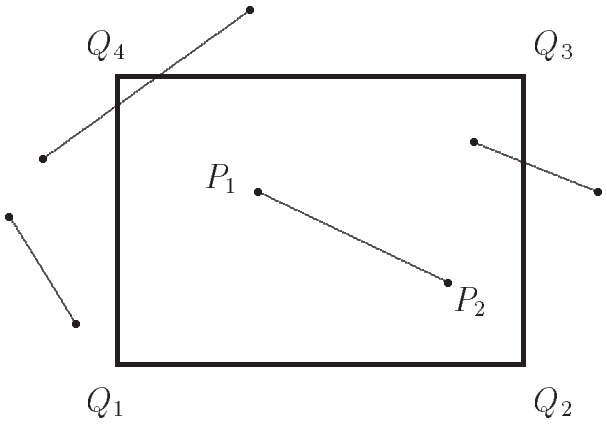

Abbildung 3.4: Mögliche Schnittpunkte des Fenster mit einem Geradenstück

Aus Bild 3.4 ist erkennbar, dass entweder ein, zwei oder kein Schnittpunkt möglich sind. Die Eckpunkte des Fensters haben die Koordinaten:

$$\vec{Q}_1 = \begin{pmatrix} x_{min} \\ y_{min} \end{pmatrix} , \quad \vec{Q}_2 = \begin{pmatrix} x_{max} \\ y_{min} \end{pmatrix} , \quad \vec{Q}_3 = \begin{pmatrix} x_{max} \\ y_{max} \end{pmatrix} , \quad \vec{Q}_4 = \begin{pmatrix} x_{min} \\ y_{max} \end{pmatrix} .$$

Ein Geradenstück sei durch die Eckpunkte gegeben:

$$\vec{P}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} , \quad \vec{P}_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} .$$

Die Bestimmungsgleichungen für die Schnittpunkte der Geradenstücke mit einer der Begrenzungslinien des Fensters lauten:

$$\vec{P}_1 + \mu \cdot (\vec{P}_2 - \vec{P}_1) = \vec{Q}_i + \nu \cdot (\vec{Q}_j - \vec{Q}_i) , \quad 1 \leq i \leq 4 , \quad j \neq i. \tag{3.6}$$

Die Gl.(3.6) liefert zwei Bestimmungsgleichungen für die beiden unbekannten Parameter μ und ν. Damit ein Schnittpunkt vorliegt, müssen beide Parameter im Intervall [0, 1] liegen. Die Gl.(3.6) muss für alle vier Begrenzungslinien geprüft werden. Diese Vorgehensweise ist sehr zeitaufwendig. Mit Hilfe des einfachen Algorithmus von Cohen und Sutherland wird zunächst geprüft, ob überhaupt ein Schnittpunkt vorliegt, bevor die Gl.(3.6) ausgewertet wird. Der Algorithmus von Cohen und Sutherland ist im Bild 3.5 dargestellt.

Die dualen Größen T_1 und T_2 geben die Lage des Geradenstückes an. Die dualen Zahlen haben folgende Bedeutung.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Dabei bedeuten die einzelnen Bits:

$$\begin{array}{lcll} x < x_{max} & : & Bit1 = 1 & \text{für} \quad x - x_{max} < 0 \\ x > x_{max} & : & Bit2 = 1 & \text{für} \quad x_{max} - x < 0 \\ y < y_{max} & : & Bit3 = 1 & \text{für} \quad y - y_{max} < 0 \\ y > y_{max} & : & Bit4 = 1 & \text{für} \quad y_{max} - y < 0 \end{array}$$

Diesen 4-Bit-Code bezeichnet man auch als Outcode.

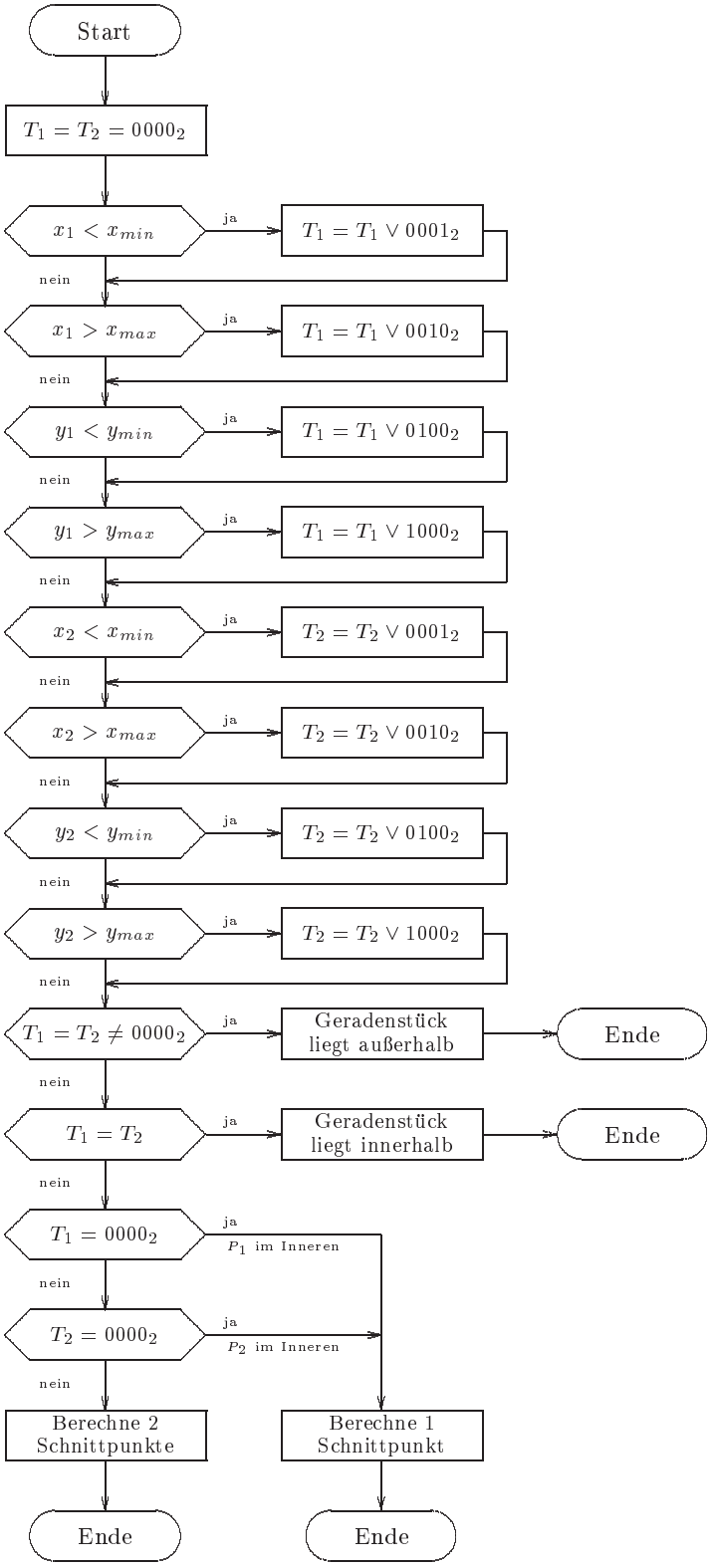

Abbildung 3.5: Clipping-Algorithmus nach Cohen-Sutherland

Teil II

Bildsynthese

Kapitel 4

Beleuchtung und Schattierung

4.1 Photometrische Betrachtung

Zum besseren Verständnis der in den folgenden Abschnitten beschriebenen Bildsyntheseverfahren, ist es hilfreich, die wesentlichen Grundbegriffe der Photometrie zu kennen. Die Photometrie ermöglicht die physikalische Beschreibung eines Strahlungsfeldes durch geeignete Größen.

Photometrie

Eine Lichtquelle gibt in den ganzen Raum den Strahlungsfluss ϕ ab. Die Einheit des Strahlungsflusses ist Watt (W). Im allgemeinen gibt eine Lichtquelle ihren Strahlungsfluss nicht nach allen Raumrichtungen gleichmäßig ab. Der Strahlungsfluss ϕ ist vom Raumwinkel $d\omega$ abhängig. Der Raumwinkel $d\omega$ ist definiert als das Verhältnis der Kugelfläche dA zum Quadrat des Kugelradius r.

Raumwinkel

$$d\omega = \frac{dA}{r^2} \tag{4.1}$$

Die Größe Raumwinkel selbst ist dimensionslos, sie wird jedoch in Steradiant(sr) angegeben. Ein Steradiant (1 sr) ist der Raumwinkel eines Kegels, der auf einer Kugel eine Fläche mit der Größe des Quadrats des Radius r schneidet. Das folgende Bild 4.1 veranschaulicht die Definition.

Steradiant

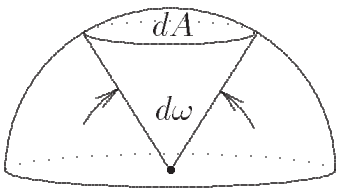

Abbildung 4.1: Raumwinkel

Nimmt man beispielsweise für den Raumwinkel die gesamte Halbkugel, so erhält man als Maß den Zahlenwert $4\pi \cdot r^2/(2 \cdot r^2) = 2 \cdot \pi$.

Die auf den Raumwinkel bezogen abgegebene Strahlungsleistung wird als Strahlungsstärke J bezeichnet. Sie ist im allgemeinen eine Funktion des Raumwinkels und wird in Watt pro Steradiant ($W\ sr^{-1}$) angegeben.

$$J = \frac{d\phi}{d\omega} \tag{4.2}$$

Das Flächenstück dA_L einer Lichtquelle wird im allgemeinen die Strahlung nicht nach allen Richtungen gleichmäßig abstrahlen. Die Strahlungsdichte I gibt die Strahlungsstärke, bezogen auf die effektive Fläche, an. Das Flächenstück dA_L erscheint aus der Richtung δ betrachtet, um den Faktor $\cos(\delta)$ verkürzt. Der Winkel δ wird auf die Flächennormale des Flächenstück dA_L bezogen. Die Strahlungsdichte hat die Einheit $W\ sr^{-1}\ m^{-2}$.

$$B = \frac{dJ}{dA_L \cdot \cos(\delta)} = \frac{d^2\phi}{dA_L \cdot \cos(\delta) \cdot d\omega} \tag{4.3}$$

Die bisher eingeführten Größen beziehen sich auf die Lichtquelle selbst. Wird ein Körper von einer Lichtquelle bestrahlt so trifft auf das Flächenstück dA der Strahlungsfluss ϕ. Die empfangene Strahlung ist dann am größten, wenn die Flächennormale des Flächenstück dA senkrecht zur Strahlungsrichtung orientiert ist. In jeder anderen Lage nimmt die empfangene Strahlungsleistung um den Faktor $\cos(\delta)$ ab, wenn δ der Winkel zwischen Flächennormale und Strahlungsrichtung ist. Am Ort des Flächenelementes dA liegt die Strahlungsflussdichte D vor.

$$D = \frac{d\phi}{dA \cdot \cos(\delta)} \tag{4.4}$$

Die Einheit der Strahlungsflussdichte ist $W\ m^{-2}$. Die Strahlungsflussdichte wird häufig auch als Intensität bezeichnet.

4.1.1 Reflexionsfunktion einer Objektoberfläche

Reflexionsfunktion

Die Reflexionsfunktion ermöglicht eine Charakterisierung von Objektoberflächen entsprechend ihres Reflexionsverhaltens. Die Reflexionsfunktion soll im folgenden hergeleitet werden.

Die Reflexionsfunktion gibt an, welcher Anteil des einfallenden Strahlungsflusses $d\phi_i$ vom Flächenelement dA der Objektoberfläche in eine beliebige, vorgegebene Raumrichtung reflektiert wird. Das folgende Bild 4.2 zeigt das Oberflächenelement dA mit einem lokalen Koordinatensystem.

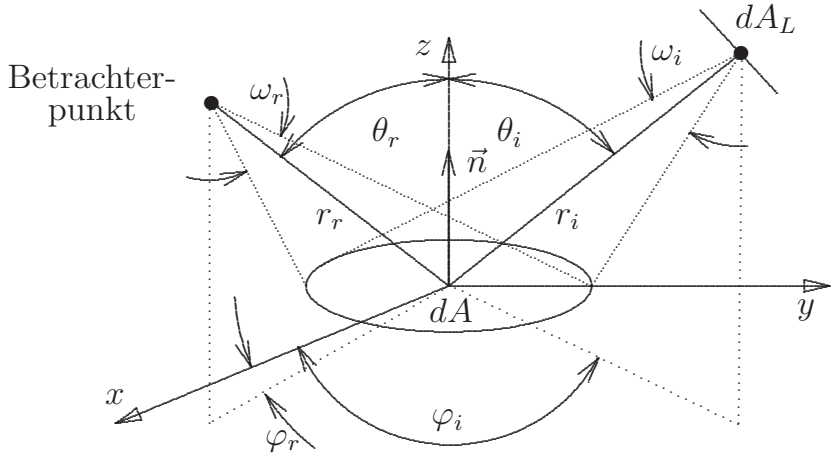

Abbildung 4.2: Reflexionsfunktion einer Objektoberfläche

Das Flächenelement dA_L einer Lichtquelle sendet in den Halbraum der Vorderseite die Strahlungsleistung ϕ_L aus. Diese Strahlungsleistung verteilt sich im allgemeinen nicht gleichmäßig

über alle Raumwinkel. Es ist $B_L(\theta_i, \varphi_i)$ die Strahlungsdichte, die das Flächenelement dA_L der Lichtquelle in der Richtung (θ_i, φ_i) erzeugt, so empfängt das Flächenelement dA der Objektoberfläche aus dem Raumwinkel $d\omega_i$ entsprechend Gl.(4.3) die einfallende Strahlungsleistung

$$d^2\phi_i = B_L(\theta_i, \varphi_i) \cdot dA_L \cdot d\omega_i \ . \tag{4.5}$$

Vom Flächenelement dA_L aus betrachtet, erscheint das Flächenelement dA der Objektoberfläche um den Faktor $\cos(\theta_i)$ verkürzt, wobei θ_i der Winkel zwischen der Flächennormale $\vec{n}$ und der einfallenden Strahlungsrichtung ist. Für den Raumwinkel $d\omega_i$ gilt:

$$d\omega_i = \frac{dA \cdot \cos(\theta_i)}{r_i^2} \ . \tag{4.6}$$

Hierbei ist r_i der Abstand zwischen den beiden Flächenelementen dA_L und dA. Aus Gl.(4.4) berechnet sich die vom Flächenelement dA_L auf dem Oberflächenelement dA erzeugte Strahlungsflussdichte:

$$dD_i(\theta_i, \varphi_i) = \frac{d^2\phi_i}{dA \cdot \cos(\theta_i)} = B_L(\theta_i, \varphi_i) \cdot \frac{dA_L}{r_i^2} \tag{4.7}$$

Das Oberflächenelement dA wird selbst zur Senderfläche und reflektiert die empfangene Strahlungsleistung. Für die reflektierte Strahlungsleistung gilt:

$$d^2\phi_r = \rho \cdot d^2\phi_i \tag{4.8}$$

Der Proportionalitätsfaktor ρ gibt das Reflexionsvermögen des Oberflächenelementes dA an.

Von Interesse ist die in Richtung (θ_r, φ_r) des Betrachters abgestrahlte Leistung. Vom Betrachter aus gesehen, erscheint das Flächenelement dA um den Faktor $\cos(\theta_r)$ verkürzt. Für die in den Raumwinkel $d\omega_r$ vom Oberflächenelement dA erzeugte Strahlungsdichte $dB_r(\theta_i, \varphi_i, \theta_r, \varphi_r)$ ergibt sich aus Gl.(4.3):

$$dB_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{d^3\phi_r}{dA \cdot \cos(\theta_r) \cdot d\omega_r} \ . \tag{4.9}$$

Der Differentialquotient

$$f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{dB_r(\theta_i, \varphi_i, \theta_r, \varphi_r)}{dD_i(\theta_i, \varphi_i)} \tag{4.10}$$

wird in der Literatur als Reflexionsfunktion (reflectance distribution function) bezeichnet. Die Dimension der Reflexionsfunktion ist sr^{-1}. Sie gibt damit an, wie die auf ein Oberflächenelement einfallende Strahlung pro Raumwinkel abgestrahlt wird.

Ist die Reflexionsfunktion von einer Objektoberfläche bekannt, so kann die Strahlungsdichte $B_r(\theta_i, \varphi_i, \theta_r, \varphi_r)$ berechnet werden. Nach Gl.(4.10) gilt der Zusammenhang:

$$\begin{aligned} B_r(\theta_i, \varphi_i, \theta_r, \varphi_r) &= \int f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) \cdot dD_i(\theta_i, \varphi_i) \\ &= \int_{\omega_i} f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) \cdot B_L(\theta_i, \varphi_i) \cdot d\omega_i \ . \end{aligned} \tag{4.11}$$

Verwendet man für die Darstellung Kugelkoordinaten, so folgt für den Raumwinkel $d\omega_i$

$$d\omega_i = \frac{dA_L}{r_i^2} = \cos(\theta_i) \cdot \sin(\theta_i) \cdot d\theta_i \ d\varphi_i \ . \tag{4.12}$$

Damit wird aus Gl.(4.11) die Form:

$$B_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) \cdot B_L(\theta_i, \varphi_i) \cdot \cos(\theta_i) \cdot \sin(\theta_i) \cdot d\theta_i \; d\varphi_i \; . \tag{4.13}$$

4.1.1.1 Reflexionsfunktion einer diffus reflektierenden Oberfläche

Eine ideal diffus reflektierend Oberfläche reflektiert die einfallende Strahlung gleichförmig nach allen Richtungen. Derartige Oberflächen werden in der Physik als Lambertsche Flächen bezeichnet. Die Reflexionsfunktion hat die einfache Form:

$$f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \rho \cdot \frac{1}{\pi} \; . \tag{4.14}$$

4.1.1.2 Reflexionsfunktion einer spiegelnd reflektierenden Oberfläche

Eine ideal spiegelnde Oberfläche reflektiert die einfallende Strahlung genau in der an der Flächennormalen gespiegelten Richtung. Die Reflexionsfunktion einer ideal spiegelnden Oberfläche kann durch Deltafunktionen formuliert werden. Die Reflexionsfunktion lautet:

$$f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \rho \cdot \frac{\delta_0(\theta_i - \theta_r) \cdot \delta_0(\varphi_i - \varphi_r + \pi)}{\sin(\theta_i) \cdot \cos(\theta_i)} \; . \tag{4.15}$$

4.1.2 Berechnung der reflektierten Strahlungsdichte

Mit Hilfe der in den Abschnitten 4.1.1.2 und 4.1.1.1 hergeleiteten Reflexionsfunktionen, Gl.(4.14) und Gl.(4.15), kann die reflektierte Strahlungsdichte für jede Raumrichtung berechnet werden. Man kann hierbei vier Fälle unterscheiden:

1. Eine diffus reflektierende Oberfläche wird von einem ungerichteten Strahlungsfeld bestrahlt.
2. Eine diffus reflektierende Oberfläche wird von einem gerichteten Strahlungsfeld bestrahlt.
3. Eine spiegelnd reflektierende Oberfläche wird von einem ungerichteten Strahlungsfeld bestrahlt.
4. Eine spiegelnd reflektierende Oberfläche wird von einem gerichteten Strahlungsfeld bestrahlt.

Das ungerichtete Strahlungsfeld wird durch die Strahlungsdichte

$$B_L^{(u)} = B_0^{(u)} \; . \tag{4.16}$$

beschrieben. Die Strahlen kommen von jeder Raumrichtung.

Bei einem gerichteten Strahlungsfeld kann die Strahlungsdichte durch

$$B_L^{(g)} = B_0^{(g)} \cdot \delta_0\Big(\cos(\theta) - \cos(\theta_L)\Big) \cdot \delta_0(\varphi - \varphi_L) \tag{4.17}$$

beschrieben werden. Die Strahlen kommen aus der Richtung (θ_L, φ_L)

Die oben genannten vier Fälle werden im folgenden näher untersucht:

1. Diffus reflektierende Oberfläche in einem ungerichteten Strahlungsfeld
   Mit Hilfe von Gl.(4.13) folgt für eine diffus reflektierende Oberfläche in einem ungerichteten Strahlungsfeld.

$$B_r^{(d,u)}(\theta_i, \varphi_i, \theta_r, \varphi_r) \;=$$

$$\int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) \cdot B_L^{(u)}(\theta_i, \varphi_i) \cdot \cos(\theta_i) \cdot \sin(\theta_i) \cdot d\theta_i \; d\varphi_i \;=$$

$$\frac{1}{\pi} \cdot \rho \cdot B_0^{(u)} \cdot \int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} \cos(\theta_i) \cdot \sin(\theta_i) \cdot d\theta_i \; d\varphi_i \;,$$

$$B_r^{(d,u)}(\theta_i, \varphi_i, \theta_r, \varphi_r) = \rho \cdot B_0^{(u)} \;. \tag{4.18}$$

2. Diffus reflektierende Oberfläche mit einem gerichteten Strahlenfeld
   Wird eine diffus reflektierend Oberfläche mit einem gerichteten Strahlungsfeld bestrahlt, so erhält man für die reflektierte Strahlungsdichte:

$$B_r^{(d,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) \;=$$

$$\int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) \cdot B_L^{(g)} \cdot \cos(\theta_i) \cdot \sin(\theta_i) \cdot d\theta_i \; d\varphi_i \;=$$

$$\frac{1}{\pi} \cdot \rho \cdot B_0^{(g)} \cdot \int_{-\pi}^{\pi} \int_{0}^{\frac{\pi}{2}} \delta_0\Big(\cos(\theta_i) - \cos(\theta_L)\Big) \cdot \delta_0(\varphi_i - \varphi_L) \cdot \cos(\theta_i) \cdot \sin(\theta_i) \cdot d\theta_i \; d\varphi_i \;,$$

$$B_r^{(d,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{1}{\pi} \cdot \rho \cdot B_0^{(g)} \cdot \cos(\theta_L) \;. \tag{4.19}$$

   Der Cosinus des Winkels θ_L berechnet sich aus dem skalaren Produkt der Flächennormalen des Oberflächenelementes und dem normierten Richtungsvektor $\vec{n}_L$, der die Strahlungsrichtung (θ_L, φ_L) angibt.

3. Spiegelnd reflektierende Oberfläche in einem ungerichteten Strahlenfeld
   Eine spiegelnd reflektierende Oberfläche in einem ungerichteten Strahlungsfeld hat die

folgende reflektierte Strahlungsdichte zu Folge:

$$B_r^{(s,u)}(\theta_i,\varphi_i,\theta_r,\varphi_r) \;=$$

$$\int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}} f_r(\theta_i,\varphi_i,\theta_r,\varphi_r)\cdot B_L^{(u)}\cdot\cos(\theta_i)\cdot\sin(\theta_i)\cdot d\theta_i\; d\varphi_i \;=$$

$$\rho\cdot B_0^{(u)}\cdot \int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}} \frac{\delta_0\Big(\theta_i-\theta_r\Big)\cdot\delta_0(\varphi_i-\varphi_r+\pi)}{\sin(\theta_i)\cdot\cos(\theta_i)}\cdot\cos(\theta_i)\cdot\sin(\theta_i)\cdot d\theta_i\; d\varphi_i \;,$$

$$B_r^{(s,u)}(\theta_i,\varphi_i,\theta_r,\varphi_r) = \rho\cdot B_0^{(u)} \;. \tag{4.20}$$

4. Spiegelnd reflektierende Oberfläche in einem gerichteten Strahlenfeld
   Eine spiegelnd reflektierende Oberfläche in einem gerichteten Strahlungsfeld liefert die reflektierte Strahlungsdichte:

$$B_r^{(s,g)}(\theta_i,\varphi_i,\theta_r,\varphi_r) \;=$$

$$\int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}} f_r(\theta_i,\varphi_i,\theta_r,\varphi_r)\cdot B_L^{(g)}\cdot\cos(\theta_i)\cdot\sin(\theta_i)\cdot d\theta_i\; d\varphi_i \;=$$

$$B_0^{(g)}\cdot\rho\cdot \int_{-\pi}^{\pi}\int_{0}^{\frac{\pi}{2}} \delta_0\Big(\cos(\theta_i)-\cos(\theta_L)\Big)\cdot\delta_0(\varphi_i-\varphi_L)\cdot \frac{\delta_0(\theta_L-\theta_r)\cdot\delta_0(\varphi_L-\varphi_r+\pi)}{\sin(\theta_L)}\cdot\cos(\theta_i)\cdot\sin(\theta_i)\cdot d\theta_i\; d\varphi_i \;,$$

$$B_r^{(s,g)}(\theta_i,\varphi_i,\theta_r,\varphi_r) = B_0^{(g)}\cdot\rho\cdot\frac{\delta_0(\theta_r-\theta_L)\cdot\delta_0(\varphi_r-\varphi_L+\pi)}{\sin(\theta_L)} \;. \tag{4.21}$$

Aus den Ergebnissen von Gl.(4.18) bis Gl.(4.21) lassen sich folgende Aussagen ableiten.

- Beleuchtungsfälle

  Es gibt drei vom Ergebnis her verschiedene Beleuchtungsfälle:

  1. Wird ein Oberflächenelement, diffus oder spiegelnd reflektierend, mit einem ungerichteten Strahlungsfeld bestrahlt, so reflektiert es die empfangene Strahlungsdichte bis auf den absorbierten Anteil nach allen Richtungen.
  2. Wird ein diffus reflektierendes Oberflächenelement mit einem gerichteten Strahlungsfeld bestrahlt, so erscheint es von allen Seiten gleich hell. Die reflektierte Strahlungsdichte hängt ab vom Cosinus des Winkels zwischen der Einfallsrichtung und der Flächennormale des Oberflächenelement.
  3. Wird ein spiegelnd reflektierendes Oberflächenelement einem gerichteten Strahlungsfeld ausgesetzt, so wird die reflektierte Strahlungsdichte unter dem Reflexionswinkel abgestrahlt.

- Realistischer Eindruck
  Die oben genannten Beleuchtungfälle zeigen, dass Lichtquellen mit einem gerichteten Strahlungsfeld wichtig sind für die Erzeugung eines realistischen Eindrucks. Lichtquellen mit ungerichtetem Strahlungsfeld erzeugen dagegen nur eine Grundfarbe bzw. eine Grundhelligkeit.

- Reale Oberflächen
  Eine reale Oberfläche wird im allgemeinen weder rein diffus noch spiegelnd reflektieren. Die Realität wird zwischen diesen beiden idealisierten Modellen liegen. Es ist deshalb nahe liegend, für die Beschreibung des Reflexionsverhaltens von realen Oberflächen eine Kombination von beiden Modellen zu nehmen.

- Farbe
  Die photometrischen Größen sind außerdem abhängig von der Wellenlänge. Die Farbinformation ist über die Wellenlänge und die am Ort des Oberflächenelementes herrschende Strahlungsflussdichte gegeben. Die Wellenlänge legt Aussehen der Farbe fest und die Strahlungsflussdichte bestimmt die Farbintensität. In der Computergraphik ist es einfacher, die Farbinformation mit Hilfe eines Farbmodells anzugeben.

4.2 Reflexionsmodell

Die auf der Basis der Strahlungsphysik entwickelten Ergebnisse des Abschnittes 4.1.2 können nicht direkt für die Computergraphik umgesetzt werden. Insbesondere bereiten die im Reflexionsmodell enthaltenen Deltafunktionen Schwierigkeiten bei einer direkten Umsetzung in einen Algorithmus. In diesem Abschnitt sollen die Ergebnisse der photometrischen Betrachtung in ein für die Computergraphik handhabbares Reflexionsmodell übergeführt werden.

Im Abschnitt 4.1.2 wurde bereits vorgeschlagen, eine Linearkombination aller Beleuchtungsfälle für das reale Reflexionsverhalten eines Oberflächenelementes anzunehmen. Diese Anteile sollen für nochmals betrachtet werden und anschließend linear überlagert werden.

1. Beleuchtung mit ungerichteter Strahlung
   Im Fall einer Beleuchtung mit ungerichtetem Licht zeigt sich sowohl bei einer diffus als auch bei einer spiegelnd reflektierenden Oberfläche lediglich ein konstanter Anteil. Mit den Ergebnissen von Gl.(4.18) und Gl.(4.20) folgt:

$$\begin{aligned} B_r^{(d,u)}(\theta_i, \varphi_i, \theta_r, \varphi_r) &= \rho_d \cdot B_0^{(u)} \\ B_r^{(s,u)}(\theta_i, \varphi_i, \theta_r, \varphi_r) &= \rho_s \cdot B_0^{(u)} \end{aligned} \tag{4.22}$$

   Die Gl.(4.22) liefert nur einen konstanten Anteil.

2. Diffus reflektierende Oberfläche bei gerichteter Strahlung
   Wird eine diffus reflektierenden Oberfläche mit einer gerichteten Lichtstrahlung beleuchtet, so folgt mit dem Ergebnis von Gl.(4.19):

$$B_r^{(d,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{1}{\pi} \cdot \rho_d \cdot B_0^{(g)} \cdot \cos(\theta_L) \ . \tag{4.23}$$

   Das folgende Bild 4.3 veranschaulicht den Strahlenverlauf bei der diffusen Reflexion.

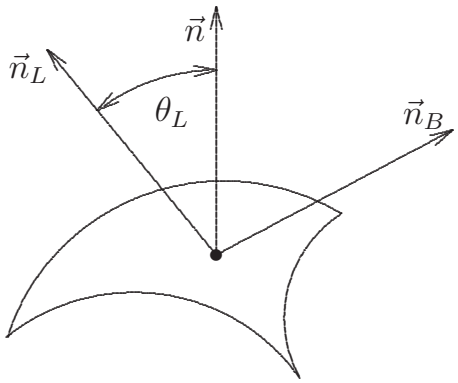

Abbildung 4.3: Diffuse Reflexion

   Es sei $\vec{n}$ der Normalenvektor des Oberflächenelementes und $-\vec{n}_L$ der normierte Richtungsvektor des einfallenden Lichtstrahls. Dann berechnet sich der Cosinus des Winkels θ_L aus dem Skalarprodukt der Vektoren. Die Gl.(4.23) kann dann in der Form

$$B_r^{(d,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{1}{\pi} \cdot \rho_d \cdot B_0^{(g)} \cdot (\vec{n} \circ \vec{n}_L) \tag{4.24}$$

   angegeben werden.

3. Spiegelnd reflektierende Oberfläche bei gerichteter Strahlung
   Wird eine spiegelnd reflektierende Oberfläche einer gerichteten Strahlung ausgesetzt, so wird die Strahlung entsprechend des Reflexionsgesetzes reflektiert. Mit dem Ergebnis von Gl.(4.21) folgt:

$$B_r^{(s,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) = B_0^{(g)} \cdot \rho_s \cdot \frac{\delta_0(\theta_r - \theta_L) \cdot \delta_0(\varphi_r - \varphi_L + \pi)}{\sin(\theta_L)} \ . \tag{4.25}$$

   Das folgende Bild 4.4 veranschaulicht den Strahlenverlauf bei einer spiegelnde Reflexion.

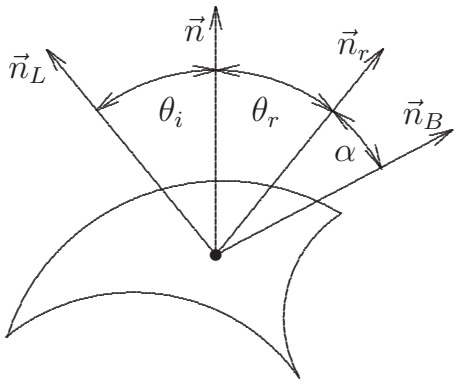

Abbildung 4.4: Spiegelnde Reflexion

Es sei $\vec{n}$ der Normalenvektor des Oberflächenelementes und $-\vec{n}_L$ der normierte Richtungsvektor des einfallenden Lichtstrahls. Gemäß dem Reflexionsgesetz, das auch in Gl.(4.25) enthalten ist, wird der Lichtstrahl entlang des Vektors $\vec{n}_r$ reflektiert. Ein Betrachter, der sich in der Richtung des Vektors $\vec{n}_B$ befindet, würde aufgrund der Deltafunktionen in Gl.(4.25) das Oberflächenelement nicht sehen können, da das Licht nur in Richtung $\vec{n}_r$ reflektiert wird, nicht aber in die Betrachterrichtung. Der Betrachter würde das Oberflächenelement nur dann sehen, wenn er sich direkt im Strahlengang des reflektierten Strahls befinden würde, d.h. es muss $\alpha = 0$ gelten. Diese Formulierung ist für eine praktische Umsetzung nicht geeignet. Die Deltafunktionen in Gl.(4.25) müssen durch eine geeignete Annäherung ersetzt werden. Bei einer Abweichung um den Winkel α von der Reflexionsrichtung, muss das Oberflächenelement immer noch sichtbar sein und die Intensität des reflektierten Lichtes muss mit größer werdendem α abnehmen. Ein Vorschlag, für eine nicht perfekte Reflexion nähert die Deltafunktionen durch eine $\cos^m$-Funktion an. Der Exponent m bestimmt, wie schmal die Cosinus-Funktion wird.

In Bild 4.5 ist die Cosinus-Funktion für verschiedene Exponenten m dargestellt.

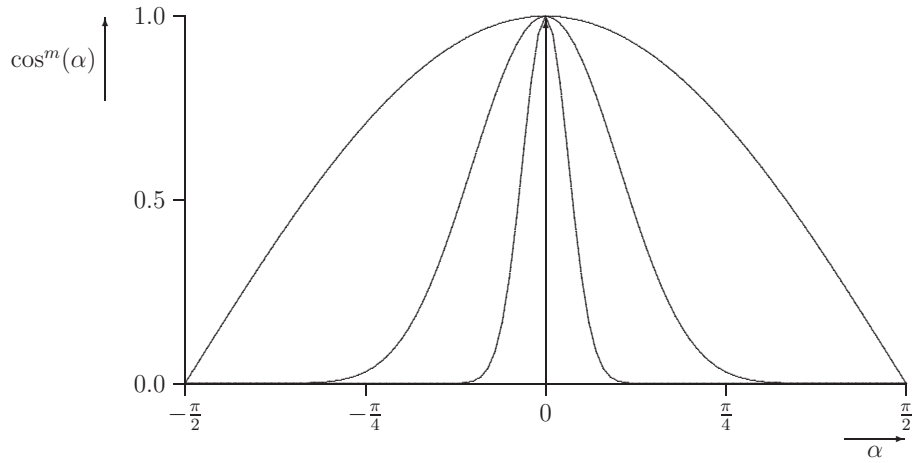

Abbildung 4.5: Annäherung der Deltafunktion durch Cosinus-Funktion für Exponenten $m = 1$, $m = 10$, $m = 100$

Zur weiteren Veranschaulichung zeigt das Bild 4.6 das Abstrahlungsprofil.

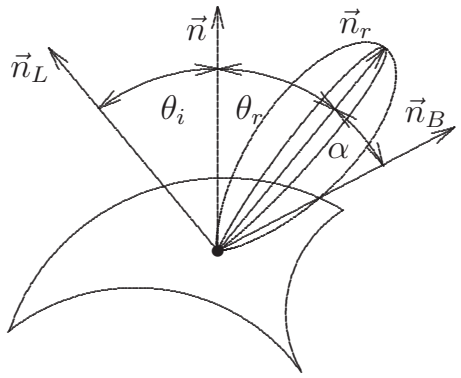

Abbildung 4.6: Abstrahlungscharakteristik der Cosinus-Funktion für Exponenten $m = 10$, $m = 100$

Phong-Reflexionsmodell

Der in Bild 4.6 darstellte Vorschlag zur Approximation der Deltafunktionen wird in der Literatur als Phong-Reflexionsmodellbezeichnet. Aus Gl.(4.24) wird damit die Form:

$$B_r^{(s,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) = B_0^{(g)} \cdot \rho_s \cdot \cos^m(\alpha) \ . \tag{4.26}$$

Der Winkel α kann mit Hilfe der Vektorrechnung durch die Vektoren $\vec{n}_L$ und $\vec{n}_B$ ausgedrückt werden. Es folgt:

$$\begin{array}{rcl} \cos(\alpha) & = & \vec{n}_r \circ \vec{n}_B \\ \cos(\alpha) & = & 2 \cdot (\vec{n} \circ \vec{n}_L) \cdot (\vec{n} \circ \vec{n}_B) - (\vec{n} \circ \vec{n}_B) \ . \end{array} \tag{4.27}$$

Damit wird aus Gl.(4.26) die Form:

$$\begin{array}{rcl} B_r^{(s,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) & = & B_0^{(g)} \cdot \rho_s \cdot (\vec{n}_r \circ \vec{n}_B)^m \\ B_r^{(s,g)}(\theta_i, \varphi_i, \theta_r, \varphi_r) & = & B_0^{(g)} \cdot \rho_s \cdot \Big((2 \cdot (\vec{n} \circ \vec{n}_L) \cdot (\vec{n} \circ \vec{n}_B) - (\vec{n} \circ \vec{n}_B) \Big)^m \end{array} \tag{4.28}$$

4. Lineare Überlagerung der Teilmodelle
   Die lineare Überlagerung der Teilmodelle liefert die folgende Gleichung:

$$B_r = \rho_d \cdot B_0^{(u)} + \rho_s \cdot B_0^{(u)} + \frac{1}{\pi}\rho_d \cdot B_0^{(g)} \cdot (\vec{n} \circ \vec{n}_L) + \rho_s \cdot B_0^{(g)} \cdot \cos^n(\alpha) \tag{4.29}$$

   Alle multiplikativen Konstanten werden zu neuen Konstanten zusammengefasst. Durch die Festlegung der Gewichtungsfaktoren kann das gewünschte Reflexionsverhalten der Oberfläche eingestellt werden. Die resultierende Überlagerung hat die Form:

$$B_r = B_0 \cdot \Big(k_a + k_d \cdot (\vec{n} \circ \vec{n}_L) + k_s \cdot (\vec{n}_r \circ \vec{n}_B)^m\Big) \tag{4.30}$$

   Die Gewichtungsfaktoren k_a, k_d und k_s können als Reflexionsfaktoren interpretiert werden. Der Faktor k_a ist maßgebend für die Beleuchtung mit einem ungerichteten Strahlungsfeld. Der Fall wird in der Computergraphik als ambiente Beleuchtung bezeichnet. Aus diesem Grunde wurde für den Index der Buchstabe “a” gewählt. Die beiden anderen Faktoren beeinflussen den diffus bzw. spiegelnd reflektierenden Anteil des Oberflächenelementes im Fall einer Bestrahlung mit gerichtetem Licht.

4.3 Farbmodelle

Ein Farbeindruck ist eine Sinnesempfindungen des Menschen, die von Körpern ausgelöst werden, wenn sie mit Licht beleuchtet werden. Die Physik kann zwar die Ursache der Farbe bestimmen, nämlich die spektrale Zusammensetzung der Strahlung. Das eigentliche Farbempfinden ist jedoch rein subjektiv. Bei Farbe wird nicht die physikalische Eigenschaft eines Körpers bewertet, sondern das visuelle System des menschlichen Auges mit in Betracht gezogen.

Das sichtbare Spektrum des menschlichen Auges liegt im Wellenlängenbereich von 400 nm (Violett) bis 700 nm (Rot). Für das Farbesehen sind bestimmte Sehzellen, die so genannten Zäpfchen zuständig, die in der Netzhaut liegen. Es gibt drei verschiedene Arten von Zäpfchen. Alle drei Arten reagieren jeweils spektral unterschiedlich. Neben den Zäpfchen gibt es eine weitere Sorte von Sehzellen, die Stäbchen. Diese reagieren besonders bei geringen Lichtverhältnissen. Sie ermöglichen das Sehen in der Dunkelheit und vermitteln eher einen Schwarz-Weiß-Eindruck. Aufgrund der Tatsache, dass es drei verschieden Farbrezeptoren gibt, kann man mit drei Komponenten Farben hinreichend gut beschreiben.

Neben den physikalischen Größen, wie Wellenlänge und Beleuchtungsstärke (Intensität) zur Beschreibung einer Lichtquelle, verwendet man daher auch die subjektiven Parameter Farbton, Farbsättigung und Helligkeit.

- Farbton
  Der Farbton ist eng gekoppelt mit der physikalischen Größe Wellenlänge. Man kann zwar einer Wellenlänge einen bestimmten Farbton zuordnen, aber es gibt auch Farbtöne, für die keine spektrale Farbe existiert, wie etwa die so genannten unbunten Farben: Weiß, Grau und Schwarz.

- Farbsättigung
  Die Farbsättigung gibt an, wie intensiv der Farbeindruck ist. Man spricht dabei auch von Farbreinheit. Die Farbsättigung ist gekoppelt mit der Überlagerung eines gesamten Wellenlängenbreiches.

- Helligkeit
  Die Helligkeit beschreibt die Intensität der Lichtes und ist damit eng mit der Strahlungsflussdichte des Lichtes gekoppelt.

Bei einem Farbmodell geht man davon aus, dass sich jeder Farbton durch eine Kombination aus drei Grundfarben zusammensetzen lässt. Die Grundfarben können dabei entweder additiv oder subtraktiv überlagert werden. Farbmodelle werden durch ein dreidimensionales Koordinatensystem beschrieben. Die drei Grundfarben bilden dabei die Einheitsvektoren der Koordinatenachsen. Alle Farbmodelle lassen sich durch Koordinatentransformationen ineinander überführen.

4.3.1 Beispiele für Farbmodelle

- Additive Farbmodelle
  - RGB-Farbmodell
    Beim RGB-Farbmodell werden die Farben Rot, Grün und Blau als Grundfarben verwendet. Jeder Punkt des RGB-Einheitswürfels stellt eine Farbe dar. Das Bild 4.7 zeigt der RGB-Einheitswürfel.
    Auf der Diagonalen zwischen den Punkten Weiß und Schwarz liegen alle Grautöne. Entlang der Diagonalen gilt: R = G = B.

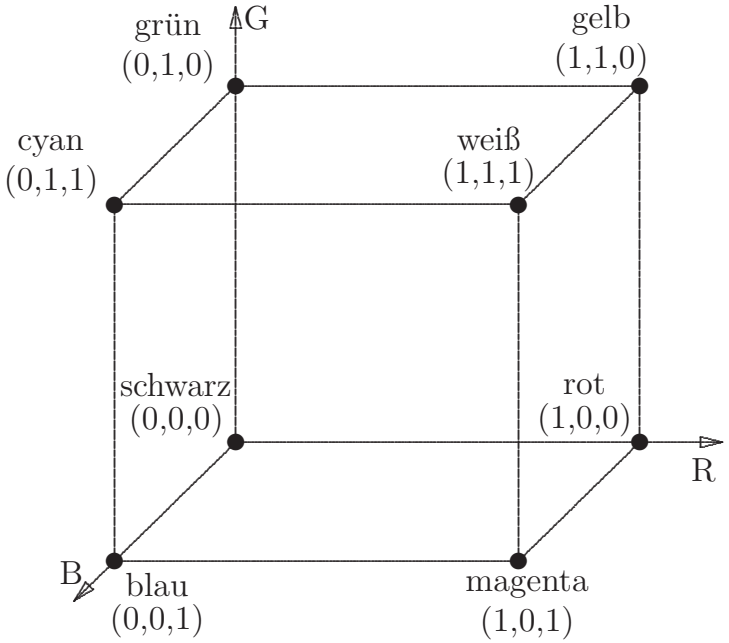

Abbildung 4.7: RGB-Farbwürfel

- HSI-Farbmodell
  Das HSI-Farbmodell ist ein wahrnehmungsorientiertes Modell. Es stützt sich auf die Begriffe Farbton (engl.: Hue), Farbsättigung (engl.: Saturation) und Helligkeit (engl.: Intensity). Das HSI-Farbmodell berechnet sich in zwei Schritten aus den RGB-Modell.

$$\begin{bmatrix} M_1 \\ M_2 \\ I_1 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{aligned} H &= \arctan(\tfrac{M_1}{M_2}) \\ S &= \sqrt{M_1^2 + M_2^2} \\ I &= I_1 \cdot \sqrt{3} \end{aligned} \tag{4.31}$$

Für die Umkehrung gilt:

$$\begin{aligned} M_1 &= S \cdot \sin(H) \\ M_2 &= S \cdot \cos(H) \\ I_1 &= \tfrac{1}{\sqrt{3}} \cdot I \end{aligned} \tag{4.32}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} M_1 \\ M_2 \\ I_1 \end{bmatrix}$$

- YQI-Farbmodell
  Das YQI-Farbmodell wird in der Fernsehtechnik verwendet. Man hat das YQI-Farbmodell in den USA für das NTSC-System eingeführt, um die Kompatibilität zum Schwarz-Weiß-Fernsehen beizubehalten. Es berechnet sich aus den RGB-Modell wie folgt:

$$\begin{bmatrix} Y \\ Q \\ I \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.522 & 0.311 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.33}$$

  Die Umkehrung lautet:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.956 & 0.623 \\ 1.0 & -0.272 & -0.648 \\ 1.0 & -1.105 & 1.705 \end{bmatrix} \cdot \begin{bmatrix} Y \\ Q \\ I \end{bmatrix} \tag{4.34}$$

- YUV-Farbmodell
  Das YUV-Farbmodell ist das für die europäische Fernsehnorm (PAL-System) maßgebende Farbmodell. Es berechnet sich aus den RGB-Farbmodell folgendermaßen:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.146 & -0.288 & -0.434 \\ 0.617 & -0.517 & 0.100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.35}$$

  wobei vereinfacht auch gilt :

$$\begin{aligned} U &= 0.493 \cdot (B - Y) \\ V &= 0.877 \cdot (R - Y) \end{aligned}$$

- Subtraktive Farbmodelle
  - CMY-Farbsystem
    Beim CMY-Farbsystem werden die drei Grundfarben Cyan, Magenta und Gelb (engl.: cyan, magenta, yellow) verwendet. Bei einem subtraktiven Farbmodell entsteht der Farbton durch Subtraktion der Anteile der Grundfarben vom weißen Licht. Weißes Licht durchläuft die Farbfilter Cyan, Magenta und Gelb. Das CMY-Farbmodell entsteht aus dem RGB-Farbmodell durch folgende Vorschrift:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.36}$$

    Die Umkehrung in das RGB-Modell lautet:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \tag{4.37}$$

Das folgende Bild 4.8 zeigt die Reflexion und Absorption von Lichtanteilen beim CMY-Modell.

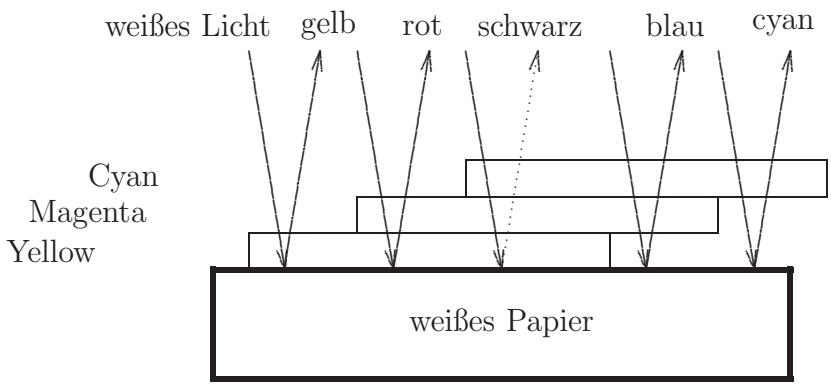

Abbildung 4.8: Reflexion und Absorption von Lichtanteilen beim CMY-Modell

In der Computergraphik wird überwiegend das RGB-Farbmodell verwendet.

4.4 Modelle für Lichtquellen

Lichtquellen werden in der Computergraphik durch ihre Intensität und ihre Abstrahlungscharakteristik beschrieben. Die Farbinformation der Lichtquelle wird mit Hilfe des RGB-Farbmodell angegeben. Für die Intensität ist in der Computergraphik der Buchstabe I gebräuchlich. Die Intensität I setzt sich damit aus den drei Farbkomponenten zusammen:

$$I = \begin{bmatrix} I_R \\ I_G \\ I_B \end{bmatrix} . \tag{4.38}$$

Abkürzenderweise wird häufig nur der Buchstabe I verwendet und nicht die ausführliche Form der Komponentenschreibweise. Diese Abkürzung ist dann zulässig, wenn eindeutig klar ist, dass alle Berechnungen die I betreffen, auf alle drei Komponenten I_R, I_G und I_B anzuwenden sind.

Eine Einteilung von Lichtquellen nach ihren Eigenschaften führt zu den folgenden vier Gruppen.

- Ambientes Licht
  Eine ambiente Lichtquelle wirkt als indirektes Licht, das gleichmäßig in allen Richtungen abgestrahlt wird.

$$I_L = I_a \tag{4.39}$$

- Gerichtetes Licht
  Eine gerichtete Lichtquelle sendet eine parallele Strahlung in die Richtung (θ_L, φ_L) aus.

$$\vec{I}_L = I_g \cdot \delta_0(\theta - \theta_L) \cdot \delta_0(\varphi - \varphi_L) \cdot \vec{e}_L \tag{4.40}$$

  Das Bild 4.9 veranschaulicht parallel ausgerichtetes Licht.

Abbildung 4.9: Parallel gerichtetes Licht

- Punktlicht
  Ein Punktlicht sendet von einem bestimmten Raumpunkt Strahlung in jede Richtung aus. Ein Punktlicht ist durch seine Lage $\vec{r}_L$ im Weltkoordinatensystem und seine Intensität I_p gekennzeichnet. Das Licht wird radial entlang des Einheitsvektors $\vec{e}_R$ abgestrahlt.

$$\vec{I}_L = I_p \cdot \delta_0(\vec{r} - \vec{r}_L) \cdot \vec{e}_R \tag{4.41}$$

  Das Bild 4.10 zeigt die Strahlung eines Punktlichtes.
- Strahler
  Ein Strahler (Spot Light) ist ähnlich wie ein Punktlicht durch seine räumliche Position $\vec{r}_L$ im Weltkoordinatensystem und seiner abgestrahlten Intensität I_s gekennzeichnet. Die Strahlungsrichtung wird jedoch durch einen geraden Kegel mit dem Öffnungswinkel α_s beschränkt. Die Achse des Kegels wird durch den Vektor $\vec{e}_L$ festgelegt. Innerhalb des Öffnungswinkels wird das Licht radial abgestrahlt, außerhalb dagegen

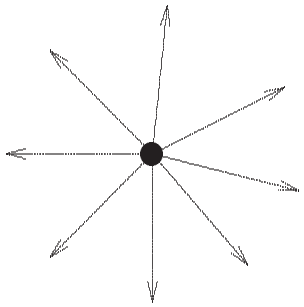

Abbildung 4.10: Punktlicht

nicht. Zusätzlich wird ein Konzentrationsexponent e verwendet, der angibt, wie stark die Intensität mit zunehmendem Winkel abnimmt. Die Intensität eines Strahlers hat die Form:

$$\vec{I}_L = \begin{cases} I_s \cdot \cos^e(\beta) \cdot \delta_0(\vec{r} - \vec{r}_L) \cdot \vec{e}_R & \text{falls} \quad \beta \leq \frac{\alpha_s}{2} \\ 0 & \text{sonst} \end{cases} \tag{4.42}$$

Das Bild 4.11 zeigt den Öffnungswinkel.

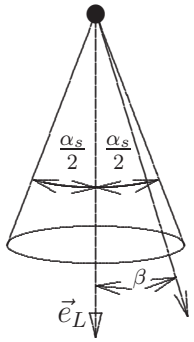

Abbildung 4.11: Strahler

Sehr schöne Lichteffekte erreicht man, wenn bei punktförmigen Lichtquellen die Intensität in Abhängigkeit vom Abstand abgeschwächt wird. Befindet sich eine punktförmige Lichtquelle am Ort $\vec{r}_L$, dann hat die Abschwächungsfunktion im Punkt $\vec{r}$ den Wert:

$$f_{ab} = \min\left(\frac{1}{c_1 + c_2 \cdot |\vec{r} - \vec{r}_L| + c_3 \cdot |\vec{r} - \vec{r}_L|^2} \ , \ 1 \right) . \tag{4.43}$$

Die Faktoren c_1, c_2 und c_3 sind Modellparameter, die für den Einzelfall festgelegt werden müssen.

4.5 Beleuchtungs- und Reflexionsmodell

Im folgenden soll einem Oberflächenelement eines Körper mit Hilfe des Reflexionsmodells von Abschnitt 4.2 eine Farbe zugeordnet werden. Dieser Vorgang wird als Schattierung bezeichnet. Das Oberflächenelement wird mit dem in Abschnitt 4.4 betrachteten Lichtquellen beleuchtet. Die Lichtquellen sollen farbiges Licht ausstrahlen und auch die Oberflächenelement sollen eine Eigenfarbe besitzen.

Zur Berücksichtigung eines Farbmodells wird das Reflexionsmodell von Gl.(4.30) erweitert. Jedem Teilmodell wird eine Eigenfarbe O_a, O_d und O_s zugeordnet. Im Detail definieren die Tripel:

$$\begin{aligned} &\text{ambiente Eigenfarbe} && O_a = \begin{bmatrix} O_{a_R} \\ O_{a_G} \\ O_{a_B} \end{bmatrix} \\ &\text{diffuse Farbe} && O_d = \begin{bmatrix} O_{d_R} \\ O_{d_G} \\ O_{d_B} \end{bmatrix} \\ &\text{spekulare Farbe} && O_s = \begin{bmatrix} O_{s_R} \\ O_{s_G} \\ O_{s_B} \end{bmatrix} \end{aligned} \tag{4.44}$$

die RGB-Komponenten der Eigenfarben.

In Gl.(4.30) wird weiterhin die Strahlungdichte B_0 durch die Intensität I_L ersetzt. Damit resultiert zunächst die Darstellung:

$$I = I_L \cdot \Big(\, k_a \cdot O_a + k_d \cdot O_d \cdot (\vec{n} \circ \vec{n}_L) + k_s \cdot O_s \cdot (\vec{n}_r \circ \vec{n}_B)^m \, \Big) \tag{4.45}$$

Durch die Wahl der Reflexionsfaktoren k_a, k_d und k_s kann die Materialeigenschaft der Objektoberfläche festgelegt werden.

Die Wirkung der verschiedenen, im Abschnitt 4.4 vorgestellten, Lichtquellen auf das Reflexionsmodell von Gl.(4.45) soll im folgenden angegeben werden. Die Teilergebnisse werden anschließend linear zu einem Beleuchtungs- und Reflexionsmodell kombiniert.

- Ambiente Beleuchtung

$$I = k_a \cdot O_a \cdot I_a \tag{4.46}$$

- Gerichtetes paralleles Licht

$$I = k_d \cdot O_d \cdot I_g \cdot (\vec{n} \circ \vec{n}_L) + k_s \cdot O_s \cdot I_g \cdot (\vec{n}_r \circ \vec{n}_B)^m \tag{4.47}$$

- Punktlicht

$$I = k_d \cdot O_d \cdot I_p \cdot (\vec{n} \circ \vec{n}_L) + k_s \cdot O_s \cdot I_p \cdot (\vec{n}_r \circ \vec{n}_B)^m \tag{4.48}$$

- Strahler

$$\begin{aligned} I \quad = \quad & k_d \cdot O_d \cdot I_s \cdot \cos^e(\beta) \cdot (\vec{n} \circ \vec{n}_L) + \\ & k_s \cdot O_s \cdot I_s \cdot \cos^e(\beta) \cdot (\vec{n}_r \circ \vec{n}_B)^m \end{aligned} \tag{4.49}$$

Bei einer VR-Anwendung werden im allgemeinen mehrere Lichtquellen verwendet. In diesem Fall kann wieder das Superpositionsprinzip angewandt werden. Man erhält bei Berücksichtigung der Abschwächungsfunktion von Gl.(4.43) für punktförmige Lichtquellen die Darstellung für die Schattierung einer Oberfläche:

$$\begin{aligned} I \;=\; & \sum_i k_a \cdot O_a \cdot I_{a_i} \;+ \\ & \sum_i k_d \cdot O_d \cdot I_{g_i} \cdot (\vec{n} \circ \vec{n}_{L_i}) \;+\; k_s \cdot O_s \cdot I_{g_i} \cdot (\vec{n}_{r_i} \circ \vec{n}_B)^m \;+ \\ & \sum_i f_{ab_i} \cdot \Big(k_d \cdot O_d \cdot I_{p_i} \cdot (\vec{n} \circ \vec{n}_{L_i}) \;+\; k_s \cdot O_s \cdot I_{p_i} \cdot (\vec{n}_{r_i} \circ \vec{n}_B)^m \Big) \;+ \\ & \sum_i f_{ab_i} \cdot \cos^{e_i}(\beta_i) \cdot \Big(k_d \cdot O_d \cdot I_{s_i} \cdot (\vec{n} \circ \vec{n}_{L_i}) \;+\; k_s \cdot O_s \cdot I_{s_i} \cdot (\vec{n}_{r_i} \circ \vec{n}_B)^m \Big) \,. \end{aligned} \tag{4.50}$$

Formt man Gl.(4.50) weiter um, so erhält man schließlich die Form des Beleuchtungs- und Reflexionsmodells:

$$\begin{aligned} I \;=\; & \sum_{i=1}^{N_A} I_{a_i} \cdot k_a \cdot O_a \;+ \\ & \sum_{i=1}^{N_G} I_{g_i} \cdot \Big(k_d \cdot O_d \cdot (\vec{n} \circ \vec{n}_{L_i}) \;+\; k_s \cdot O_s \cdot (\vec{n}_{r_i} \circ \vec{n}_B)^m \Big) \;+ \\ & \sum_{i=1}^{N_P} f_{ab_i} \cdot I_{p_i} \cdot \Big(k_d \cdot O_d \cdot (\vec{n} \circ \vec{n}_{L_i}) \;+\; k_s \cdot O_s \cdot (\vec{n}_{r_i} \circ \vec{n}_B)^m \Big) \;+ \\ & \sum_{i=1}^{N_S} f_{ab_i} \cdot \cos^{e_i}(\beta_i) \cdot I_{s_i} \cdot \Big(k_d \cdot O_d \cdot (\vec{n} \circ \vec{n}_{L_i}) \;+\; k_s \cdot O_s \cdot (\vec{n}_{r_i} \circ \vec{n}_B)^m \Big) \,. \end{aligned} \tag{4.51}$$

Kapitel 5

Algorithmen zur Bildsynthese

In den folgenden Abschnitten werden Verfahren zur Bildsynthese beschrieben. Grundlage für diese Bildsyntheseverfahren ist das in den vorangegangenen Abschnitten beschriebene Beleuchtungs- und Reflexionsmodell.

Es haben sich drei Bildsyntheseverfahren zur Erzeugung fotorealistischer Bilder entwickelt.

- Rendering
- Raytracing
- Radiosity

Rendering ist eine sehr schnelle Technik für die realitätsnahe Darstellung von virtuellen Welten. Bisher wird ausschließlich Rendering bei VR-Anwendungen eingesetzt, da die beiden anderen Verfahren noch zu aufwendig für Echtzeitanwendungen sind.

Die folgende Tabelle zeigt einen groben Vergleich der Verfahren bezüglich Rechenzeiten und erreichbarer Bildqualität.

Verfahren	Rechenzeit	Qualitätsstufe
Rendering	Millisekunden	befriedigend
Raytracing	Minuten	gut
Radiosity	Stunden	sehr gut

5.1 Rendering

Rendering

Renderingist eine Bezeichnung für eine einfache und schnelle Technik, Polygonflächen farbig zu schattieren (engl.: shading). Man unterteilt die Rendering-Methoden in die folgenden drei Gruppen:

- Flat-Shading

  Flat-Shading

  konstante Schattierung

  flache Schattierung

  Flat-Shading wird auch als konstante Schattierung oder flache Schattierung bezeichnet. Es ist die einfachste und schnellste Art, eine Polygonfläche mit einer Farbe zu schattieren. Beim Flat-Shading geht man von folgenden Voraussetzungen aus:

  1. Die Lichtquelle befindet sich im Unendlichen.
  2. Der Betrachter befindet sich im Unendlichen.
  3. Die Polygonfläche wird als eben angenommen.

  Aufgrund der Annahme, dass die Polygonfläche eben ist, ist der Normalenvektor $\vec{n}$ über die gesamte Polygonfläche konstant. Damit ist auch bei einer unendlichen Entfernung der Lichtquelle das Skalarprodukt $\vec{n} \circ \vec{n}_L$ über die gesamte Polygonfläche konstant und schließlich bei einer unendlichen Entfernung des Betrachters auch das Skalarprodukt $\vec{n}_r \circ \vec{n}_B$. Unter diesen Voraussetzungen muss für jede Polygonfläche nur ein Intensitätswert berechnet werden, der für die gesamte Polygonfläche gültig ist.
  Das Flat-Shading ist zwar sehr schnell, es liefert jedoch keine befriedigende Bildqualität. An unstetigen Übergängen zwischen Polygonen tritt der so genannte Mach-Band-Effekt auf, der sich störend auf die Bildqualität auswirkt. Beim Mach-Band-Effekt wird ein zickzackförmiger Verlauf einer Kante wahrgenommen.

  Das Bild 5.1 zeigt die Polygonfläche mit dem Normalenvektor und dem Richtungsvektor der Lichtquelle.

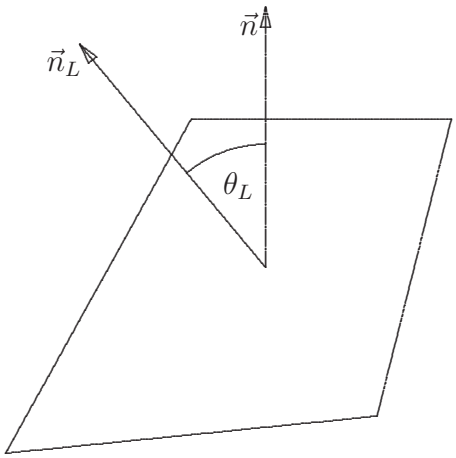

Abbildung 5.1: Flat-Shading

$$I = \rho \cdot I_0 \cdot (\vec{n} \circ \vec{n}_L) \tag{5.1}$$

- Gouraud-Shading

  Gouraud-Shading

  Schattierung durch Intensitätinterpolation

  Schattierung durch Farbinterpolation

  Gouraud-Shading wird auch als Schattierung durch Intensitätinterpolation oder Schattierung durch Farbinterpolation bezeichnet. Der Gouraud-Shading-Algorithmus benötigt die Normalenvektoren in den Polygonecken. Im Bild 5.2 ist ein Polygon mit den Normalenvektoren der Eckpunkte dargestellt.

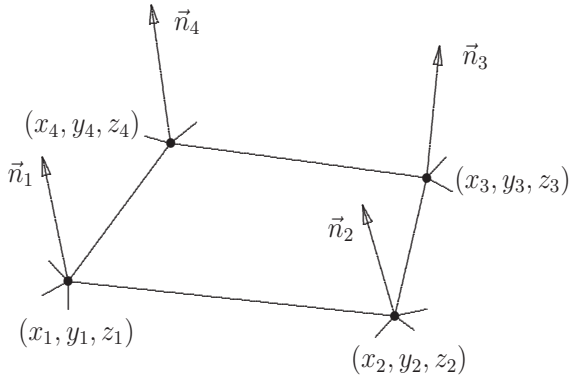

Abbildung 5.2: Polygon mit Normalenvektoren

Für jeden der Eckpunkte des Polygons wird die Intensität mit Hilfe eines Reflexionsmodells ermittelt. Anschließend werden die Intensitätswerte innerhalb des Polygons durch eine lineare Interpolation ermittelt. Das Bild 5.3 zeigt die Interpolation.

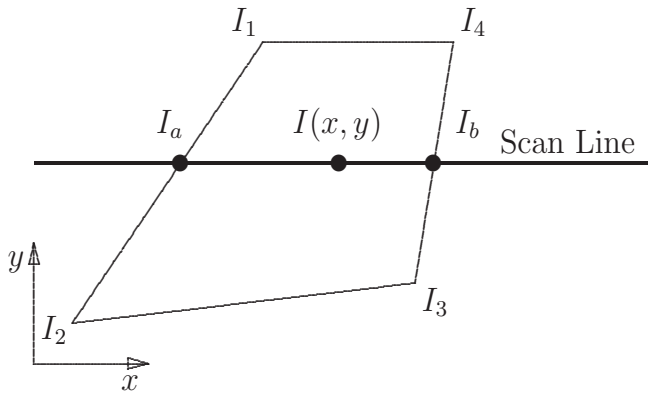

Abbildung 5.3: Interpolation der Intensitätswerte nach Gouraud

Die Interpolation erfolgt nach folgenden Regeln:

1. Berechnung von I_a

$$I_a = \frac{1}{y_1 - y_2} \cdot \Big(I_1 \cdot (y_a - y_2) + I_2 \cdot (y_1 - y_a) \Big) \tag{5.2}$$

2. Berechnung von I_b

$$I_b = \frac{1}{y_4 - y_3} \cdot \Big(I_3 \cdot (y_4 - y_b) + I_4 \cdot (y_b - y_3) \Big) \tag{5.3}$$

3. Berechnung von $I(x, y)$

$$I(x, y) = \frac{1}{x_b - x_a} \cdot \Big(I_a \cdot (x_b - x) + I_b \cdot (x - x_a) \Big) \tag{5.4}$$

Der Vorteil des Gouraud-Shading liegt in der schnellen Berechenbarkeit. Der Mach-Band-Effekt wird weitgehend vermieden. Nachteilig ist, dass aufgrund der linearen Interpolation der Intensitäten keine Glanzlichter möglich sind.

- Phong-Shading

Phong-Shading

Beim Phong-Shadingwird vor der Berechnung der Intensität der Normalenvektor linear interpoliert.

Das Bild 5.4 zeigt die Interpolation des Normalenvektors.

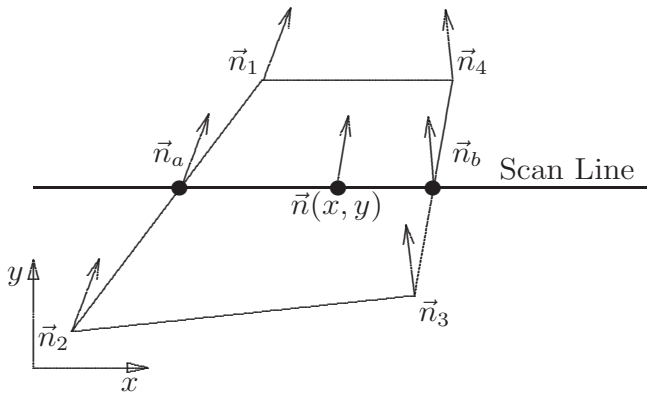

Abbildung 5.4: Interpolation des Normalenvektors nach Phong

Die Interpolationsvorschrift lautet:

1. Berechnung von $\vec{n}_a$

$$\vec{n}_a = \frac{1}{y_1 - y_2} \cdot \Big(\vec{n}_1 \cdot (y_a - y_2) + \vec{n}_2 \cdot (y_1 - y_a)\Big) \tag{5.5}$$

2. Berechnung von $\vec{n}_b$

$$\vec{n}_b = \frac{1}{y_4 - y_3} \cdot \Big(\vec{n}_3 \cdot (y_4 - y_b) + \vec{n}_4 \cdot (y_b - y_3)\Big) \tag{5.6}$$

3. Berechnung von $\vec{n}$

$$\vec{n}(x,y) = \frac{1}{x_b - x_a} \cdot \Big(\vec{n}_a \cdot (x_b - x) + \vec{n}_b \cdot (x - x_a)\Big) \tag{5.7}$$

4. Berechnung der Intensität $I(x,y)$ im Punkt (x,y) mit Hilfe eines Beleuchungs- und Reflexionsmodells und dem berechneten Normalenvektor $\vec{n}(x,y)$

Der Vorteil des Phong-Shadings liegt darin, dass ein qualitativ besserer Bildeindruck entsteht als beim Gouraud-Shading. Beim Phong-Shading sind auch Glanzlichter möglich. Nachteilig ist die etwa dreifache Rechenzeit gegenüber dem Gouraud-Shading.

5.2 Raytracing

Raytracing

Beim Raytracingwerden Wechselwirkungen zwischen Objekten berücksichtigt, damit werden auch Schatten-, Transparenz- und Spiegelungseffekte möglich. Das Prinzip des Raytracing-Verfahrens liegt in der Rückverfolgung der Strahlen, die die Farbe eines Bildpunktes bestimmen. Dazu wird für jeden Punkt der Projektionsebene ein so genannter Primärstrahl erzeugt. Ein Primärstrahl geht vom Projektionszentrum aus durch einen Bildpunkt der Projektionsebene.

Im Detail hat das Verfahren den folgenden Ablauf:

Primärstrahl

1. Wähle einen Primärstrahl, der vom Betrachterpunkt durch einen Bildpunkt der Projektionsebene verläuft.

2. Prüfe, ob der Primärstrahl auf ein Objekt trifft.

3. Wird kein Objekt vom Primärstrahl getroffen, so bekommt der Bildpunkt die Farbe des Hintergrundes zugewiesen. Es wird ein neuer Primärstrahl wird gewählt und bei Punkt 1 wieder begonnen.

4. Wird ein Objekt vom Primärstrahl getroffen, so wird der Schnittpunkt berechnet.

5. Ausgehend vom Schnittpunkt werden die Verbindungen zu den vorhandenen Lichtquellen gesucht. Wird eine Lichtquelle getroffen, so wird anhand des Reflexionsmodells die Umgebungsfarbe ermittelt.

Sekundärstrahlen

6. Vom Schnittpunkt aus werden je nach Reflexionsverhalten der Objektoberfläche so genannte Sekundärstrahlengestartet. Sekundärstrahlen sind Spiegelungs- und Brechungsstrahlen.

7. Falls die Oberfläche reflektierend ist, wird die Richtung des reflektierten Strahls bestimmt. Der Intensitätsanteil der Reflexion wird ermittelt und der Anteil in Betrachterrichtung aufsummiert. Der reflektierte Strahl wird weiter verfolgt, ob dieser auf ein weiteres Objekt trifft und der Ablauf bei Punkt 5 rekursiv fortgesetzt.

8. Falls die Oberfläche transparent ist und keine Totalreflexion erfolgt, so durchdringt ein Sekundärstrahl den transparenten Körper gemäß dem Brechungsgesetz und tritt am anderen Ende wieder aus. Dieser Sekundärstrahl wird weiter verfolgt. Gleichzeitig kann eine Reflexion an diesem Oberflächenpunkt erfolgen. Der Sekundärstrahl in Reflexionsrichtung wird weiter verfolgt. Im Fall einer Totalreflexion wird nur der Sekundärstrahl in der Reflexionsrichtung weiter verfolgt. Für jeden der Fälle wird der Ablauf bei Punkt 5 rekursiv fortgesetzt.

Das Bild 5.5 zeigt die Vorgehensweise bei der Strahlrückverfolgung.

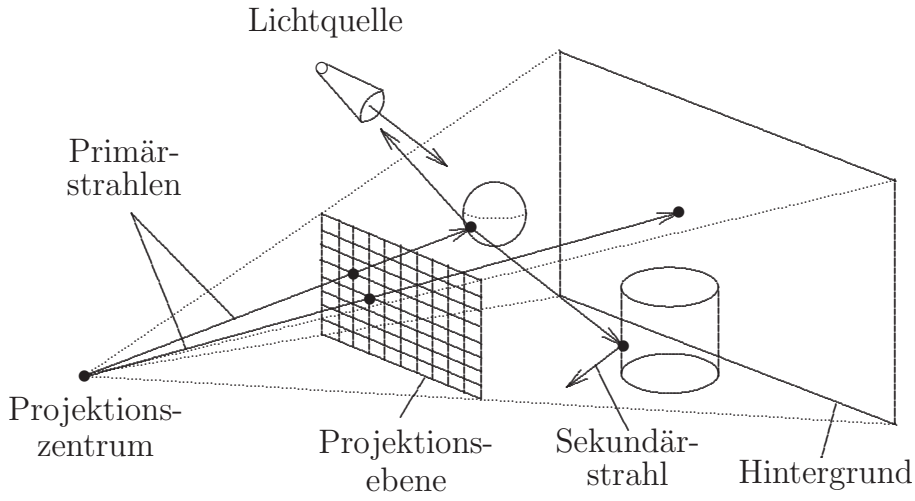

Abbildung 5.5: Strahlrückverfolgung

Das folgende Bild 5.6 zeigt die Brechung nach dem Snellius-Gesetz.

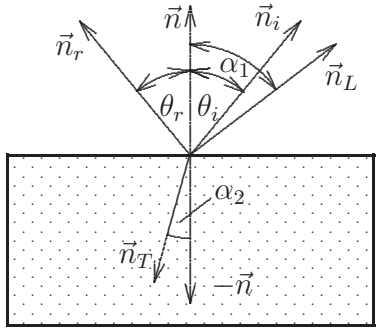

Abbildung 5.6: Brechung

Nach dem Snellius-Brechungsgesetz gilt der Zusammenhang:

$$\frac{\sin(\alpha_1)}{\sin(\alpha_2)} = n \ . \tag{5.8}$$

In Gl.(5.8) ist n der Brechungsindex des Mediums.

Zur Berücksichtigung von Brechungseffekten muss das Reflexionsmodell von Gl.(4.51) um einen Brechungsanteil erweitert werden. Dieser Anteil lautet:

$$k_s \cdot O_s \cdot I_r + k_t \cdot O_t \cdot I_t \ . \tag{5.9}$$

Hierbei ist I_r der Anteil der reflektierten Intensität und I_t der Anteil der transmittierten Intensität. Die Faktoren k_s und k_t. geben die Gewichtung an, welcher Anteil reflektiert und

welcher Anteil transmittiert wird. Die Farbkomponenten für den reflektierten und transmittierten Anteil sind O_r und O_t.

In [13] ist ein einfaches Programm in C-ähnlichem Pseudocode für ein rekursives Raytracing ohne Antialiasing angegeben. Es lautet:

```
Wähle das Projektionszentrum und das Window der View Plane;
for(jede Rasterzeile des Windows){
    for(jedes Pixel der Rasterzeile){
        Ermittle Strahl vom Projektionszentrum durch das Pixel;
        pixel = RT_trace(ray, 1);
    }
}


/* Schneide den Strahl mit Objekten und berechne die Schattierung */
/* am nächsten Schnittpunkt. */

RT_color RT_trace(RT_ray ray, int depth)
{
    Ermittle den nächstliegenden Schnittpunkt mit einem Objekt;
    if(Objekt getroffen){
        Berechne die Normale im Schnittpunkt;
        return RT_shade(nächstliegendes Objekt, Strahl, Schnittpunkt, Normale, depth);
    }
    else
        return HINTERGRUND_FARBE;
}


/* Berechne Schattierung für einen Punkt des Objektes durch */
/* Strahlverfolgung der Schatten-, Reflexions- und Brechungsstrahlen */

RT_color RT_shade(
    RT_object object,                /* Geschnittenes Objekt */
    RT_ray ray,                      /* Einfallender Strahl */
    RT_point point,                  /* Schnittpunkt */
    RT_normal normal,                /* Normale im Schnittpunkt */
    int depth)                       /* Rekursionstiefe */


{ RT_color color;          /* Farbe des Strahls */
RT_ray rRay;               /* Reflexionsstrahl */
RT_ray tRay;               /* Brechungsstrahl */
RT_ray sRay;               /* Schattenstrahl */
RT_color rColor;           /* Farbe des reflekierten Strahls */
RT_color tColor;           /* Farbe des gebrochenen Strahls */

    color = Farbe des ambienten Lichtes;

    for(jede Lichtquelle){
        sRay = Strahl von der Lichtquelle zum Punkt;
        if(Skalarprodukt der Normalen mit Richtung zum Licht positiv){
          Berechne, wieviel Licht von opaken und
```

```
            von transparenten Flächen blockiert wird;
            Skaliere damit die Terme für diffuse und spiegelnde Reflexion,
            bevor sie zur Farbe addiert werden;
        }
    }
    if(depth < maxDepth){
        if(Objektoberfläche reflektierend){
            rRay = Strahl vom Punkt in Reflexionsrichtung;
            rColor = RT_trace(rRay, depth + 1);
            Skaliere rColor mit dem Spiegelungskoeffizienten und
            addiere den Wert zur Farbe;
        }       if(Objektoberfläche transparent){
            tRay = Strahl vom Punkt in Brechungsrichtung;
            if(keine Totalreflexion){
                tColor = RT_trace(rRay, depth + 1);
                Skaliere tColor mit dem Transmissionskoeffizienten und
                addiere den Wert zur Farbe;
            }
        }
    }
    return color;                          /* Farbe des Strahls */
}
```

Der Vorteil des Raytracing-Verfahrens liegt in der Berücksichtigung von Wechselwirkungen zwischen Objekten. Dadurch lassen sich Spiegelungseffekte erzielen. Es ist weiterhin möglich, transparente Körper in die Berechnung mit einzubeziehen. Ebenso ist eine Schattenbildung machbar. Nachteilig ist jedoch der weit höhere Rechenaufwand gegenüber einem Rendering-Verfahren. Der Raytracing-Algorithmus kann rekursiv formuliert werden und ist parallelisierbar. Es ist nur eine Frage der Zeit, bis für Raytracing-Algorithmen spezielle Hardware-Lösungen zur Verfügung stehen. Raytracing-Algorithmen sind dann in Echtzeit durchführbar.

5.3 Radiosity

Radiosity

Radiosity ist ein globales Berechnungsverfahren für die Farbwerte einer dreidimensionalen Szene. Grundlage des Radiosity bilden die beiden Hauptsätze der Thermodynamik. Der erste Hauptsatz ist der Energieerhaltungssatz. Dieser besagt, dass in einem abgeschlossenen System die Energie erhalten bleibt. Der zweite Hauptsatz besagt, dass alle Naturvorgänge so verlaufen, dass die Gesamtentropie aller beteiligten Körper zunimmt. Bei diesem Berechnungsverfahren wird die Strahlung als Energie betrachtet.

Das Radiosity-Verfahren geht von folgenden Voraussetzungen aus.

1. Die dreidimensionale Szene wird zerlegt in eine endliche Anzahl von N Flächenelementen.
2. Die dreidimensionale Szene ist abgeschlossen, so dass der Energieerhaltungssatz angewandt werden kann.
3. Alle Oberflächen sind homogen bezüglich ihrer Farbeigenschaft und strahlen die empfangene Strahlung gleichmäßig ab.
4. Alle Oberflächen reflektieren diffus.
5. Jedes Oberflächenelement wirkt, wenn es bestrahlt wird, selbst als Strahler.
6. Alle Lichtquellen werden wie leuchtende (diffus reflektierende) Oberflächen behandelt. Damit gibt es keinen Unterschied zwischen Lichtquellen- und Körperoberflächen.

Im folgenden sollen die Gleichung hergeleitet werden, die das Radiosity-Verfahren beschreiben.

Jedes Flächenelement A_i wird als opaker, diffuser Lambertscher Strahler betrachtet (Gl.(4.14). Für die Strahlungsflussdichte des Flächenelementes A_i gilt folgende Bilanz:

$$B_i = E_i + \rho_i \sum_{j=1}^{N} B_j \cdot F_{i \to j} \tag{5.10}$$

In Gl.(5.10) ist B_i die auf dem Flächenelement A_i vorherrschende Strahlungsdichte. Sie ist die Summe aus der emittierten Strahlungsdichte E_i, falls das Flächenelement A_i ein Flächenelement einer Lichtquelle ist, und der reflektierten Strahlungsdichte B_j. Die Strahlungsdichte B_j wirkt vom Flächenelement A_j auf das Flächenelement A_i ein, deshalb wird über alle Flächenelemente summiert. Der Anteil B_j wird mit dem Reflexionsfaktor ρ_i gewichtet und ist von den geometrischen Verhältnissen abhängig. Die geometrische Abhängigkeit wird durch den Formfaktor $F_{i \to j}$ berücksichtigt.

Im folgenden soll ein Ausdruck für den Formfaktor gefunden werden. Dazu wird das Bild 5.7 betrachtet.

Für die Strahlungsdichte, die das Flächenelementes A_i in die Richtung (θ_r, φ_r) abstrahlt, gilt nach Gl.(4.10):

$$dB_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = f_r(\theta_i, \varphi_i, \theta_r, \varphi_r) \cdot dD_i(\theta_i, \varphi_i) \ . \tag{5.11}$$

In Gl.(5.11) ist der Ausdruck $f_r(\theta_i, \varphi_i, \theta_r, \varphi_r)$ die Reflexionsfunktion der Oberfläche. Weiterhin gilt mir Gl.(4.7) für die auf den Oberflächenelement A_i von dA_j verursachte infinitesimale Strahlungsflussdichte:

$$dD_i(\theta_i, \varphi_i) = B_j \cdot \cos(\theta_j) \cdot \frac{dA_j}{r^2} \ . \tag{5.12}$$

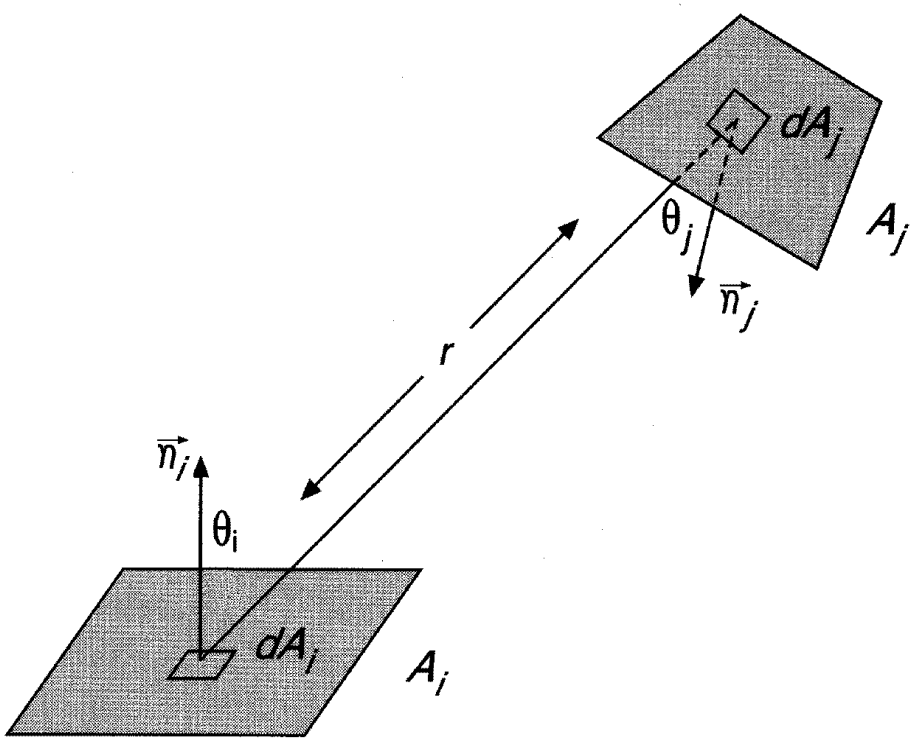

Abbildung 5.7: Berechnung des Formfaktors

Setzt man Gl.(5.12) in Gl.(5.11) ein und berücksichtigt, dass für die Reflexionsfunktion eines Lambertschen Reflektors gilt:

$$f_r = \rho \cdot \frac{1}{\pi} \ , \tag{5.13}$$

so folgt:

$$dB_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \frac{1}{\pi} \cdot \rho_i \cdot B_j \cdot \cos(\theta_j) \cdot \cos(\theta_i) \cdot \frac{dA_j}{r^2} \ . \tag{5.14}$$

Integriert man schließlich Gl.(5.14) auf beiden Seiten, so erhält man:

$$B_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \rho_i \cdot B_j \int_{A_j} \frac{\cos(\theta_j) \cdot \cos(\theta_i)}{\pi \cdot r^2} \, dA_j \ . \tag{5.15}$$

Da alle Punkte des Flächenelementes A_i in die Richtung (θ_r, φ_r) abstrahlen, bildet man den Mittelwert von Gl.(5.15) über die Fläche A_i.

$$B_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \rho_i \cdot B_j \cdot \frac{1}{A_i} \int_{A_j} \int_{A_i} \frac{\cos(\theta_j) \cdot \cos(\theta_i)}{\pi \cdot r^2} \, dA_i \, dA_j \ . \tag{5.16}$$

Berücksichtigt man schließlich noch, dass sich Flächen blockieren können. Bei blockierten Flächen gibt es keine Verbindungsgerade, da z.B. ein anderer Körper zwischen den beiden betrachteten liegt. Diese Eigenschaft kann durch den Faktor H_{ij} berücksichtigt werden. Für ihn gilt:

$$H_{i,j} = \begin{cases} 1 & \text{falls sich die Flächen } A_i \text{ und } A_j \text{ nicht blockieren} \\ 0 & \text{sonst} \end{cases} \ . \tag{5.17}$$

Man erhält damit:

$$B_r(\theta_i, \varphi_i, \theta_r, \varphi_r) = \rho_i \cdot B_j \cdot \frac{1}{A_i} \int_{A_j} \int_{A_i} \frac{\cos(\theta_j) \cdot \cos(\theta_i)}{\pi \cdot r^2} \cdot H_{i,j} \; dA_i \; dA_j \; . \tag{5.18}$$

Vergleicht man letztlich die Ausdrücke von Gl.(5.10) und Gl.(5.18), so erhält man schließlich als Ausdruck für den Formfaktor:

$$F_{i \to j} = \int_{A_j} \int_{A_i} \frac{\cos(\theta_j) \cdot \cos(\theta_i)}{\pi \cdot r^2} \cdot H_{i,j} \; dA_j \; dA_i \; . \tag{5.19}$$

Aus Gl.(5.10) kann ein lineares Gleichungssystem hergeleitet werden.

$$B_i - \rho_i \sum_{j=1}^{N} B_j \cdot F_{i \to j} = E_i \; , \quad i = 1(1)N \tag{5.20}$$

$$\begin{bmatrix} 1 - \rho_1 \cdot F_{1\to1} & -\rho_1 \cdot F_{1\to2} & \cdots & -\rho_1 \cdot F_{1\to N} \\ -\rho_2 \cdot F_{2\to1} & 1 - \rho_2 \cdot F_{2\to2} & \cdots & -\rho_2 \cdot F_{2\to N} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ -\rho_N \cdot F_{N\to1} & -\rho_N \cdot F_{N\to2} & \cdots & 1 - \rho_N \cdot F_{N\to N} \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_N \end{bmatrix} \tag{5.21}$$

Die Lösung des linearen Gleichungsystems (5.21) bildet der Lösungsvektor $[B_1, \; B_2, \; \cdots \; B_N]^T$. Das Gleichungssystem (5.21) hat eine riesige Dimension. Es kann nur iterativ gelöst werden. Üblicherweise besteht ein virtuelle Welt aus zehntausenden von Polygonen. Berücksichtigt man weiterhin, dass für die Farbdarstellung ein RGB-Modell zugrunde liegt, so muss das lineare Gleichungsystem für jede Farbkomponente gelöst werden.

Das Radiosity-Verfahren hat den großen Nachteil, dass es sehr aufwendig ist. Rechenzeiten von mehreren Stunden bis zu Tagen sind hier üblich. Der Vorteil des Verfahrens ist, dass es unabhängig vom Betrachterstandpunkt ist. Sind die Werte B_i einmal berechnet gelten sie für jeden Betrachterstandpunkt. Es muss lediglich noch ein Projektionsverfahren auf die Betrachterebene angewandt werden. Das Radiosity-Verfahren kann dann für VR-Anwendungen eingesetzt werden, wenn eine statische Welt vorliegt. Das bedeutet, dass kein Objekt und keine Lichtquelle der Szene verändert werden darf. Man kann sich lediglich durch die virtuelle Welt bewegen. In diesem Fall kann für die dreidimensionale Szene das Radiosity-Verfahren vorab berechnet werden. Der große Aufwand wird mit einer sehr guten Bildqualität belohnt, bei der auch Spiegelungen und Schattenbildung fast realistisch erscheinen.

Teil III

VRML

Kapitel 6

VRML - Virtual Reality Modelling Language

Virtual Reality Modelling Languageist eine Beschreibungssprache zur Modellierung von virtuellen Welten. Die Abkürzung von Virtual Reality Modelling Language lautet VRML Die englische Aussprache dieser Abkürzung klingt etwa wie “Wirml”. VRML ist für Anwendungen innerhalb des World Wide Web (WWW) gedacht. Damit VR-Anwendungen innerhalb des World Wide Web möglich werden, muss die Visualisierung und Animation der virtuellen Welt beim Client durchgeführt werden und darf nicht dem Server überlassen bleiben. Andernfalls würde die Netzlast gigantisch ansteigen und der Server bei mehreren Clients sofort an die Kapazitätsgrenze gelangen. Sinnvoller ist es, eine Beschreibung der virtuelle Welt nur einmal an den Client zu übertragen und dort alle Berechnungen zur Visualisierung und Animation durchzuführen. Für diese Anwendung ist die plattform-unabhängige Beschreibungssprache VRML gedacht.

Virtual Reality Modelling Language
VRML

VRML ist keine Programmiersprache im herkömmlichen Sinn. Sie besitzt keine Statements, sondern besteht aus Konstrukten und Funktionen der Computergraphik, die interpretierend ausgeführt werden. Die Interpretationsaufgabe übernimmt ein spezielles Plug-In des WWW-Browsers des Clients.

Virtuelle Welten können mit VRML auch Bestandteil eines Hypermedia-Dokumentes sein. Derzeit wachsen Script-Sprachen und Programmiersprachen mit VRML zusammen. VRML öffnet damit für VR-Anwendungen das Tor zum World Wide Web. WWW-Experten prophezeien VRML einen ähnlichen evolutionären Durchbruch, wie er bei Java derzeit stattfindet.

6.1 Historie von VRML

Auf dem ersten internationalen World-Wide-Web-Kongreß, im April 1994 in Genf, wurde von Timothy Berners-Lee, dem Entwickler von Hypertext Markup Language (HTML) und Organisator des Kongresses, zu einer Sitzung zum Thema Virtual Reality Markup Language and the World Wide Web eingeladen. Auf dieser Sitzung wurde die Vorstellung von einem plattform-unabhängigen Standard für VR-Anwendungen im WWW entwickelt. Es wurde der Begriff "Virtual Reality Markup Language" geprägt und später das Wort Markup durch das Wort Modelling ersetzt. Dieser Kongreß gilt allgemein als Geburtsstunde von VRML.

Der erste Aufruf, einen Standard für VRML zu erarbeiten, erfolgte im Anschluß an diesen World-Wide-Web-Kongreß 1994. Dieser Aufruf war nicht nur ein technisches, sondern auch ein gesellschaftliches Experiment, da der Aufruf per Internet erfolgte. Das amerikanische Internet-Magazin WIRED stellte für diese Zwecke einen Mailing-Server zur Verfügung. Durch E-Mail-Austausch sollten Diskussionsbeiträge zur Entwicklung von VRML vorangetrieben werden. Die Beteiligung an der Diskussion war überraschend hoch.

Die Suche nach vorhandenen Sprachen zur Beschreibung von virtuellen Welten führte zum Dateiformat von Open Inventor der Firma Silicon Graphics Inc. (SGI). SGI erklärte sich daraufhin bereit, Teile des Dateiformates von Open Inventor lizenzfrei öffentlich verfügbar zu machen. Der Kern von VRML 1.0 beruht deshalb auf Open Inventor.

Bereits ein halbes Jahr später, auf dem zweiten internationalen World-Wide-Web-Kongreß im Oktober 1994 in Chicago wird VRML 1.0 präsentiert. Innerhalb eines halben Jahres wurde der Vorschlag VRML 1.0 durch weltweite Zusammenarbeit erarbeitet. Ein derartiges schnelles Ergebnis bei Standardisierungsarbeiten gab es bisher noch nie.

VRML Architecture Group

Weitere Diskussionen zu VRML finden auf der Tagung SIGRAPH 95 statt. An alle Interessierten erfolgte der Aufruf an der Weiterentwicklung der Sprache mitzuarbeiten. Zur Koordinierung des Standardisierungsprozesses wurde die VRML Architecture Group (VAG) gegründet. Anfang 1996 rief die VAG öffentlich zur Abgabe von Vorschlägen für VRML 2.0 auf. Die Abstimmung im Internet ergab eine mehrheitliche Entscheidung für den Vorschlag "Moving Worlds" von SGI. Dieser Vorschlag diente als Arbeitspapier. Die wesentliche Neuerung von VRML 2.0 gegenüber VRML 1.0 ist die Möglichkeit der Animation einer Szene. Im August 1996 wurde die erarbeitete Spezifikation auf der Tagung SIGRAPH 96 vorgestellt. Auf der SIGRAPH 96 gab die VAG die Gründung des VRML Consortium bekannt. In diesem Gremium sollen Vertreter von Firmen, Forschungseinrichtungen und Universitäten vertreten sein. Das Gremium hat die Aufgabe, die Weiterentwicklung von VRML fördern, die verschiedenen Meinungen zu bündeln und diese zu artikulieren.

VRML97

ISO/IEC 14772

Die VRML 2.0 Spezifikation wurden unter dem Namen VRML97an die International Standards Organization (ISO) und an die International Electrotechnical Commission (IEC) eingereicht. und 1997 verabschiedet. Inzwischen ist VRML97 ein internationaler Standard. Der internationale Standard VRML97 trägt die Nummer **ISO/IEC 14772**

Die Entwicklung von VRML wird ständig fortgesetzt. Derzeit ist VRML98 in der Diskussion.

6.2 Einführung in VRML

VRML ist eine Beschreibungssprache für virtuelle Welten und dreidimensionale Szenen. Die Beschreibung erfolgt durch Konstrukte und Funktionen in der Form eines festgelegten Dateiformates. Der Ursprung des Dateiformats geht auf Open Inventor von Silicon Graphics zurück.

Die Beschreibungssprache VRML verwendet einen objektorientierten Ansatz und ist für den Benutzer leicht erlernbar.

Im nächsten Abschnitt sollen einige wichtige Begriffe erläutert werden.

6.2.1 Der Szenengraph

Nodes sind die grundlegenden Sprachelemente von VRML. Sie dienen dazu, Objekte und deren Eigenschaften zu beschreiben. Nodes werden beispielsweise zur Gestaltung von geometrischen Körpern, zur Beschreibung von Materialeigenschaften, von Lichtquellen und zur Durchführung geometrischer Transformationen verwendet. Zwischen Nodes können auch Events verschickt werden. Nodes werden hierarchisch in Form einer Baumstruktur, den so genannten Szenengraphen, angeordnet werden. Jeder Node entspricht einer Verzweigung im Szenengraph.

Node

Szenengraphen

Nodes können in zwei Klassen eingeteilt werden.

- Grouping Nodes
  Grouping Nodes fassen andere Nodes zusammen, die zu einem gemeinsamen Objekt gehören. Jeder Grouping Node definiert ein Koordinatensystem. Alle nachfolgenden Child Nodes werden relativ zu diesem Koordinatensystem definiert. In diesem Zusammenhang spricht man von Parent Node und von Child Nodes.
- Child Nodes
  Child Nodes sind eigenständige Nodes. Sie bestehen aus so genannten Fields, die entweder Eigenschaften festlegen oder auf weitere Nodes verweisen. Ein Child Node kann selbst wieder einen Group Node sein.

Mit Hilfe von Grouping Nodes und Child Nodes kann eine Hierarchie im Szenengraph festgelegt werden. Der Szenengraph gibt die Datenstruktur wider, die bei der Visualisierung der 3D-Szene abgearbeitet werden muss. Es handelt sich dabei um einen n-ären Baum. Hinter dieser Datenstruktur lässt sich der objektorientierte Ansatz und die Vererbung von Eigenschaften wiedererkennen.

6.2.2 Spezifikation eines Nodes

Ein Node besteht aus folgenden Elementen.

- Der Typenbezeichnung des Nodes.
- Einer Anzahl von Feldern, (engl.: Fields). Felder können Daten für die Beschreibung eines Objektes oder weitere Child Nodes aufnehmen.

Field

Felder stehen nach dem Namen des Nodes in geschweiften Klammern { und }. Es gibt vier verschiedene Typen von Feldern.

- eventIn
- eventOut

- exposedField
- field

Die Typen “eventIn” und “eventOut” sind zur Verarbeitung von Events und dienen damit zur Dynamisierung der 3D-Szene. Der Typ “exposedField” kann seinen Wert während der Laufzeit ändern. Der Typ “field” ist während der Laufzeit nicht veränderbar.

Ein Feld ist durch seinen Namen und einen Datentyp festgelegt. Die ersten beiden Buchstaben des Names eines Datentypen zeigen an, ob es sich um einen “single-valued” (SF) oder einen “multi-valued” (MF) Parameterwert handelt. Das darauf folgende Wort gibt die eigentliche Bedeutung des Datentypen an. Für jedes Feld gibt es Defaultwerte, die überschrieben werden können. Bei der Angabe der Felder muss keine Reihenfolge eingehalten werden.

Die folgende Tabelle zeigt eine vollständige Spezifikation am Beispiel des Transformation Nodes.

Feldtyp	Datentyp	Feldname	Defaultwerte
Transform {			
eventIn	MFNode	addChildren	
eventIn	MFNode	removeChildren	
exposedField	SFVec3f	center	0 0 0
exposedField	MFNode	children	[ ]
exposedField	SFRotation	rotation	0 0 1 0
exposedField	SFVec3f	scale	1 1 1
exposedField	SFRotation	scaleOrientation	0 0 1 0
exposedField	SFVec3f	translation	0 0 0
field	SFVec3f	bboxCenter	0 0 0
field	SFVec3f	bboxSize	-1 -1 -1
}			

Feldtyp und Datentyp sind nicht Bestandteil der VRML-Syntax, sie dienen lediglich der Spezifikation.

6.2.3 Aufbau des Dateiformat

Derzeit sind die Versionen VRML 1.0 und 2.0 aktuell. Damit der Browser erkennen kann, welche Version vorliegt, muss in der ersten Zeile einer VRML-Datei in Form eines Kommentars die VRML-Version und der verwendete Zeichensatz angegeben werden. Das Kommentarzeichen ist “#”. Es ist bis zum Zeilenende wirksam. Für die beiden Versionen sieht die erste Zeile der VRML-Datei folgendermaßen aus:

```
#VRML V1.0 ascii
#VRML V2.0 utf8
```

Während bei VRML 1.0 generell nur ASCII erlaubt ist, gestattet das Universal Coded Character Set Transformation Format (UTF-8) unterschiedliche Zeichensätze. US-ASCII ist als Subset in UTF-8 enthalten.

Aufbau einer VRML-Datei

Der Aufbau einer VRML-Datei hat die folgende vereinfacht dargestellte Struktur.

```
#VRML V2.0 utf8
Group {
    children [
```

```
        Childnode {
            fields { ...
            }
            node {
                fields { ...
                }
            }
        }
    ]
}
```

Kapitel 7

VRML Tutorial

In diesem Abschnitt soll die VRML Programmierung anhand von einfachen Beispielen erläutert werden.

7.1 Gruppierung einfacher Körper

Die folgenden Beispiele zeigen die Plazierung und Gruppierung primitiver geometrischer Körper im dreidimensionalen Raum. Die folgenden Nodes sind hierfür hilfreich.

- Group Node
- Shape Node
- Transform Node
- Apperance Node
- Material Node
- Box Node
- Cone Node
- Cylinder Node
- Sphere Node

7.1.1 Namensgebung für Nodes

Häufig ist es sehr hilfreich, Nodes einen Namen zuzuordnen. Durch die Namensgebung liest sich ein Programm wesentlich einfacher. Einem Node wird mit Hilfe von Define ein Name zugewiesen.

Syntax:

```
DEF    node-name    node-type
```

Eine Instanz des Nodes wird durch das Schlüsselwort “USE” erzeugt.

Syntax:

```
USE    node-name
```

Die DEF-Anweisung kann zu Beginn eines Programmes stehen oder bei jeder Spezifikation eines Nodes mitten im Programm.

7.1.2 Group Node

Mit Hilfe des Group Node werden zusammengehörige Nodes zu einer Einheit zusammengefügt. Es empfiehlt sich stets eine zusammengehörige Einheit von Objekten mit Hilfe eines Group Nodes zusammenzufassen. Diese Vorgehensweise erleichtert den Aufbau von komplexen Körpern aus primitiven geometrischen Körpern bis hin zum Zusammenfügen von kleineren 3D-Szenen zu einer komplexen 3D-Szene.

Syntax:

```
Group {
    addChildren                     #  eventIn          MFNode
    removeChildren                  #  eventIn          MFNode
    children       [ ]              #  exposedField     MFNode
    bboxCenter      0  0  0         #  field            SFVec3f
    bboxSize       -1 -1 -1         #  field            SFVec3f
}
```

7.1.3 Shape Node

Syntax:

```
Shape {
    appearance NULL    #  exposedField    SFNode
    geometry   NULL    #  exposedField    SFNode
}
```

Der Shape Node besitzt die beiden Felder appearance und geometry. Das Feld appearance beinhaltet den Appearance Node. Dieser spezifiziert das visuelle Aussehen des Körpers. Das Feld geometry ermöglicht die Angabe eines primitiven Körpers mit Hilfe folgender Nodes:

- Box Node,
- Cone Node,
- Cylinder Node,
- Sphere Node.

Der Geometry Node wird schattiert mit Hilfe der Angaben, die der Appearance Node beinhaltet.

7.1.4 Appearance Node

Syntax:

```
Appearance {
    material            NULL    #    exposedField    SFNode
    texture             NULL    #    exposedField    SFNode
    textureTransform    NULL    #    exposedField    SFNode
}
```

Beim Appearance Node kann angegeben werden, ob der Körper eine Materialeigenschaft, eine Textur besitzen soll. Im Fall einer Textur ist es möglich verschiedene Bilderformate, wie z.B. GIF, JPEG, PNG, MPEG zu verwenden. Die Textur kann weiterhin auf einen Körper mit Hilfe von geometrischen Transformationen abgebildet werden. In allen drei Fällen wird auf einen weiteren Node verwiesen.

7.1.5 Material Node

Mit Hilfe des Material Nodes werden die Farb- und Reflexionseigenschaften des Körpers festgelegt.

Syntax:

```
Material {
    ambientIntensity    0.2          #  exposedField    SFFloat
    diffuseColor        0.8 0.8 0.8  #  exposedField    SFColor
    emissiveColor       0 0 0        #  exposedField    SFColor
    shininess           0.2          #  exposedField    SFFloat
    specularColor       0 0 0        #  exposedField    SFColor
    transparency        0            #  exposedField    SFFloat
}
```

Die Felder haben folgende Bedeutung:

- ambientIntensity
  Reflexionskoeffizient für den ambienten Beleuchtungsanteil.
- diffuseColor
  Reflexionskoeffizienten für den diffusen Beleuchtungsanteil für die RGB-Komponenten.
- emissiveColor
  Eigenleuchten in RGB-Komponenten.
- shininess
  Der Parameter shininess ist ein Maß für den Glanz der Oberfläche.
- specularColor
  Reflexionskoeffizienten für den gerichteten Beleuchtungsanteil für die RGB-Komponenten.
- transparency
  Der Parameter transparency ist ein Maß für die Transparenz des Körpers.

7.1.6 Image Texture Node

Syntax:

```
ImageTexture {
    url         [ ]     #  exposedField    MFString
    repeatS     TRUE    #  field           SFBool
    repeatT     TRUE    #  field           SFBool
}
```

Mit Hilfe des Feldes url wird das File spezifiziert, das die Textur beinhaltet. Mit dem Parameter repeatS und repeatT wird angezeigt, ob die Textur in S- bzw. T-Richtung periodisch wiederholt werden soll. Die Koordinaten S und T beziehen sich auf die Koordinatenachsen der Textur.

7.1.7 Geometry Nodes

Geometry Nodes definieren einfache geometrische Körper.

Syntax:

```
Box {
    size  2 2 2    # field    SFVec3f
}
```

Syntax:

```
Cone {
    bottomRadius    1       #  field     SFFloat
    height          2       #  field     SFFloat
    side            TRUE    #  field     SFBool
    bottom          TRUE    #  field     SFBool
}
```

Syntax:

```
Cylinder {
    bottom    TRUE    #  field    SFBool
    height    2       #  field    SFFloat
    radius    1       #  field    SFFloat
    side      TRUE    #  field    SFBool
    top       TRUE    #  field    SFBool
}
```

Syntax:

```
Sphere {
    radius  1    #  field    SFFloat
}
```

7.1.8 Transform Node

Mit Hilfe des Transform Node kann eine Translation, eine Rotation und eine Skalierung auf einen Körper angewandt werden.

Syntax:

```
Transform {
    addChildren                 #  eventIn         MFNode
    removeChildren              #  eventIn         MFNode
    center              0 0 0   #  exposedField    SFVec3f
    children            [ ]     #  exposedField    MFNode
    rotation            0 0 1 0 #  exposedField    SFRotation
    scale               1 1 1   #  exposedField    SFVec3f
    scaleOrientation    0 0 1 0 #  exposedField    SFRotation
    translation         0 0 0   #  exposedField    SFVec3f
    bboxCenter          0 0 0   #  field           SFVec3f
    bboxSize            -1 -1 -1 #  field          SFVec3f
}
```

Der Rotationswinkel muss im Bogenmaß angegeben werden. Der Parameter scaleOrientation wird bei Scherungen benötigt.

7.1.9 Programme

Programm 1 (Einheitswürfel)
Ein Einheitswürfel soll im Koordinatenursprung platziert:

```
#VRML V2.0 utf8
#
# Einheitswuerfel
#
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Box { }
}
```

Die Nodes Appearance und Material werden in diesem Fall nicht näher spezifiziert. Es werden die Default-Values angenommen. Den Einheitswürfel mit Koordinatensystem zeigt das Bild 7.1.

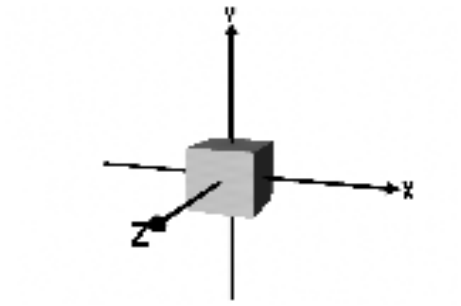

Abbildung 7.1: Einheitswürfel

Programm 2 (Würfel)
Die Abmessungen des Würfels im Koordinatenursprung sollen $(1.0, 3.0, 5.0)$ *Einheiten betragen.*

```
#VRML V2.0 utf8
#
# Wuerfel
#
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Box {
        size 1.0 3.0 5.0
    }
}
```

Der skalierte Würfel hat die in Bild 7.2 dargestellte Form.

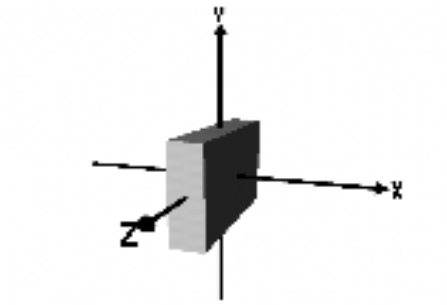

Abbildung 7.2: Skalierter Würfel

Programm 3 (Kegel)
Für einen Kegel mit den Abmessungen Radius = 3.5 und Höhe = 1.5 lautet das Programm:

```
#VRML V2.0 utf8
#
# Kegel
#
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cone {
        bottomRadius 3.5
        height 1.5
    }
}
```

Der Kegel nimmt die in Bild 7.3 dargestellte Form an.

Abbildung 7.3: Kegel

Programm 4 (Verschobener Einheitszylinder)
Der Einheitszylinder soll um den Vektor $(-2.0, 0.0, 0.0)$ *verschoben werden. Das zugehörige VRML-Programm lautet:*

```
#VRML V2.0 utf8
#
# Verschobener Einheitszylinder
#
Transform {
    translation -2.0 0.0 0.0
    children [
        Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Cylinder { }
        }
    ]
}
```

Den verschobenen Zylinder zeigt das Bild 7.4.

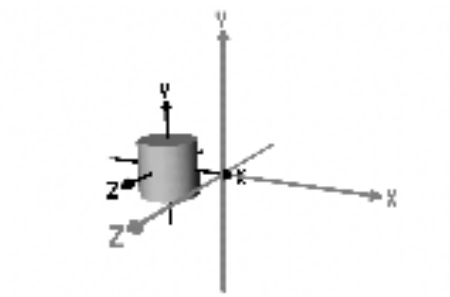

Abbildung 7.4: Verschobener Zylinder

Programm 5 (Rotierter Einheitswürfel)
Der Einheitswürfel soll um 45^o um die x-Achse gedreht werden. Das zugehörige VRML-Programm lautet:

```
#VRML V2.0 utf8
#
# Rotierter Einheitswürfel
#
Transform {
    rotation 1.0 0.0 0.0  0.785
    children [
        Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Box { }
        }
    ]
}
```

Den gedrehten Würfel zeigt das Bild 7.5.

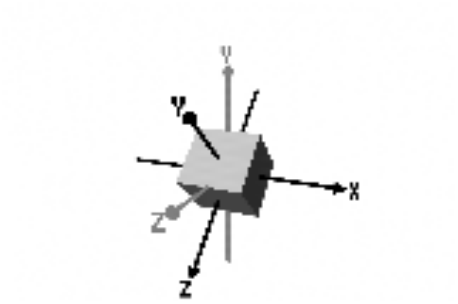

Abbildung 7.5: Rotierter Würfel

Programm 6 (Skalierte Kugel)
Die Einheitskugel soll um den Faktor $s_y = 2.0$ skaliert werden. Das zugehörige VRML-Programm lautet:

```
#VRML V2.0 utf8
#
# Skalierte Kugel
#
Transform {
    scale 1.0 2.0 1.0
    children [
        Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Sphere { }
        }
    ]
}
```

Die skalierte Kugel zeigt das Bild 7.6.

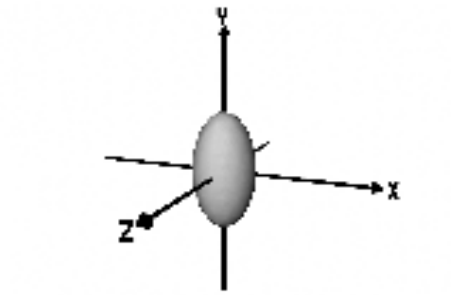

Abbildung 7.6: Skalierte Kugel

Programm 7 (Mehrfache Rotation)
Drei Zylinder, die jeweils um 60^o versetzt sind, sollen überlagert werden. Ein Zylinder hat die Abmessungen Radius = 0.1 und Höhe = 1.0.

```
#VRML V2.0 utf8
#
# Mehrfache Rotation
#
Group {
    children [
    #------------------------------------------- Zylinder 1
        DEF ZYL1 Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Cylinder {
                height 1.0
                radius 0.1
            }
        },
    #------------------------------------------- Zylinder 2
        Transform {
            rotation 1.0 0.0 0.0  1.047
            children USE ZYL1
        },
    #------------------------------------------- Zylinder 3
        Transform {
            rotation 1.0 0.0 0.0  2.094
            children USE ZYL1
        }
    ]
}
```

Die zueinander versetzten Zylinder zeigt Bild 7.7.

Abbildung 7.7: Mehrfache Rotation

Programm 8 (Mehrfach verschachtelte Rotation)
Fünf Zylinder sollen räumlich überlagert werden. Ein Zylinder hat die Abmessungen Radius = 0.1 *und Höhe* = 1.0.

```
#VRML V2.0 utf8
#
# Mehrfach verschachtelte Rotation
#
Group {
    children [
    #-------------------------------------------- Zylinder 1
        DEF ZYL1 Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Cylinder {
                height 1.0
                radius 0.1
            }
        },
    #-------------------------------------------- Zylinder 2
        DEF ZYL2 Transform {
            rotation 1.0 0.0 0.0  1.047
            children USE ZYL1
        },
    #-------------------------------------------- Zylinder 3
        DEF ZYL3 Transform {
            rotation 1.0 0.0 0.0  2.094
            children USE ZYL1
        },
    #-------------------------------------------- Zylinder 4 und 5
        Transform {
            rotation 0.0 1.0 0.0  1.785
            children [
                USE ZYL2,
                USE ZYL3
            ]
        }
    ]
}
```

Die räumlich überlagerten Zylinder zeigt Bild 7.8.

Abbildung 7.8: Mehrfach verschachtelte Rotation

Programm 9 (Schreibtischlampe)
Gegeben ist das VRML-Programm "Schreibtischlampe". Es soll der Szenengraph analysiert werden und die Relation zwischen den Koordinatensystemen angegeben werden.

Das VRML-Programm lautet:

```
#VRML V2.0 utf8
#
# desk lamp
#
Group{
    children [
    #
    #------------------------------------------------------ lamp base
    #
        Shape {
            appearance DEF White Appearance {
                material Material { }
            }
            geometry Cylinder {
                radius 0.1
                height 0.01
            }
        }
    #
    #------------------------------------------------------ base joint
    #
        Transform {
            translation 0.0   0.15 0.0
            rotation    1.0   0.0  0.0  -0.7
            center      0.0 -0.15  0.0
            children [
            #
            #------------------------------------------- lower arm
            #
                DEF LampArm Shape {
                    appearance USE White
                    geometry Cylinder {
                        radius 0.01
                        height 0.3
                    }
```

```
            }
        #
        #---------------------------------------- lower arm
        #----------------------------------- second arm joint
        #
            Transform {
                translation 0.0  0.3  0.0
                rotation    1.0  0.0  0.0  1.9
                center      0.0 -0.15 0.0
                children [
                    #
                    #------------------------------ second arm
                    #
                       USE LampArm
                    #
                    #------------------------------ lamp shade
                    #
                    Transform {
                        translation 0.0  0.075  0.0
                        rotation    1.0  0.0    0.0  -1.25
                        center      0.0  0.075  0.0
                        children [
                            #
                            #--------------------------- shade
                            #
                            Shape {
                                appearance USE White
                                    geometry Cone {
                                        height 0.15
                                        bottomRadius 0.12
                                        bottom FALSE
                                    }
                                }
                            #
                            #---------------------- light bulb
                            #
                            Transform {
                                translation 0.0 -0.05 0.0
                                children Shape {
                                    appearance USE White
                                    geometry Sphere { radius 0.05 }
                                } # end Shape
                            } # end Transform
                        ] # end children
                    } # end Transform
                ] # end children
            } # end Transform
        ] # end children
    } # end Transform
```

Die Schreibtischlampe zeigt Bild 7.9.

Abbildung 7.9: Schreibtischlampe

Der Szenengraph kann aus dem VRML-Programm entnommen werden. Das Bild 7.10 zeigt den Szenengraph für die Schreibtischlampe.

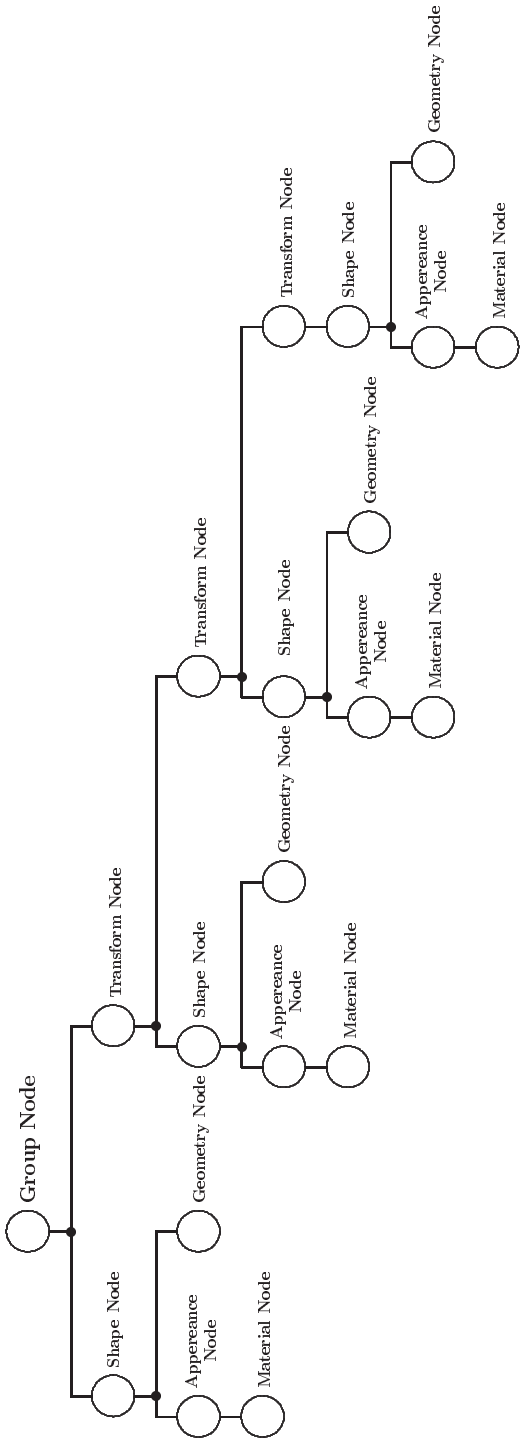

Abbildung 7.10: Szenengraph für Schreibtischlampe

Alle Koordinatensysteme sind relativ zueinander angeordnet. Jeder Transformation Node ist ein Child Node des Vorgängers. Das folgende Bild 7.11 zeigt die Anordnung der Koordinatensysteme für den Aufbau der Schreibtischlampe.

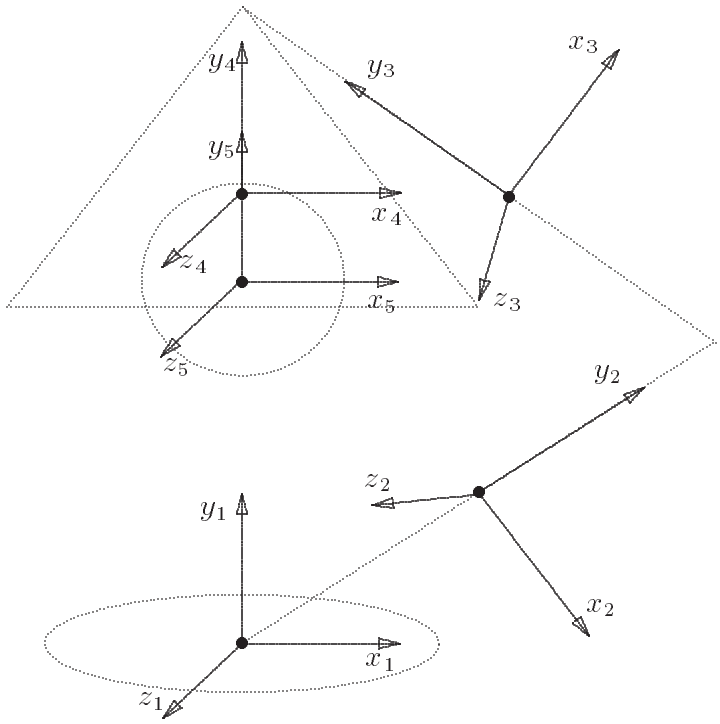

Abbildung 7.11: Koordinatensysteme für Schreibtischlampe

7.2 Animation

VRML bietet die Möglichkeit eine 3D-Szene zu animieren. Die Animation wird wie ein Playback durchgeführt. Diese Technik wird als Keyframing oder als Keyframe Animation bezeichnet. Zur Durchführung der Animation sind zwei Voraussetzungen notwendig:

- Ein Zeitgeber (Clock) der den zeitlichen Ablauf steuert.
- Eine Beschreibung, wie die Animation zu erfolgen hat.

Die Beschreibung, wie die Animation erfolgt wird als Keyframe bezeichnet. Ein Keyframe wird in der Regel periodisch wiederholt. Die Beschreibung, wie sich ein Körper bewegt, wird als Trajektorie vorgegeben. VRML verfügt hierzu über folgende Nodes.

- TimeSensor Node
- PositionInterpolator Node
- OrientationInterpolator Node

Der Kontrollmechanismus für die Animation wird durch Verschicken von Events zwischen Nodes realisiert.

7.2.1 Routing von Events

Mit Hilfe der ROUTE-Anweisung können Events von einem Node zum anderen verschickt werden. Das Bild 7.12 zeigt das Prinzip.

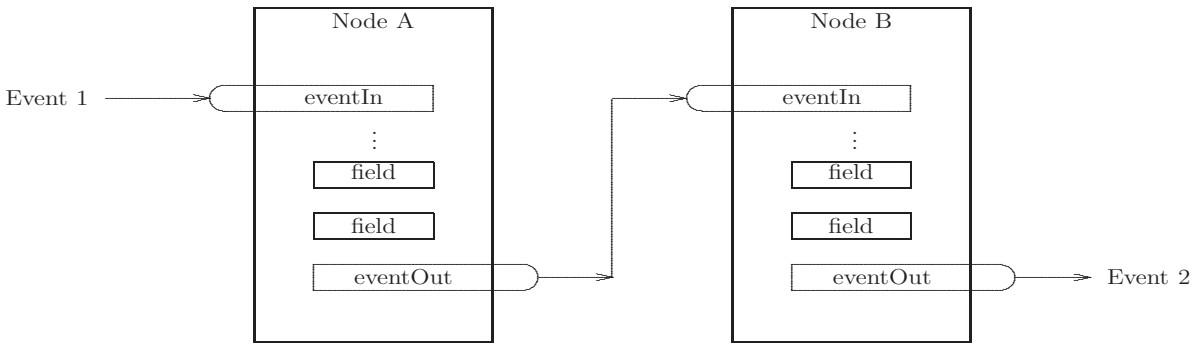

Abbildung 7.12: Routing von Events

Es ist folgende Syntax gültig:

Syntax:

```
ROUTE  NodeName.eventOut_field_changed TO NodeName.set_eventIn_field
```

Für jedes “exposedField” kann die Prefix “set_” bzw. die Suffix “_changed” verwendet werden. Durch diese Namenskonvention können Events zwischen Nodes zu den verschiedensten Feldern verschickt werden. ROUTE-Anweisungen stehen am Ende der VRML-Datei.

7.2.2 TimeSensor Node

Syntax:

```
TimeSensor {
    cycleInterval     1       #  exposedField    SFTime
    enabled           TRUE    #  exposedField    SFBool
    loop              FALSE   #  exposedField    SFBool
    startTime         0       #  exposedField    SFTime
    stopTime          0       #  exposedField    SFTime
    cycleTime                 #  eventOut        SFTime
    fraction_changed          #  eventOut        SFFloat
    isActive                  #  eventOut        SFBool
    time                      #  eventOut        SFTime
}
```

Der TimeSensor Node kann verwendet werden:

- als Zeitgeber zur Animation und Simulation,
- zur Steuerung von periodischen Abläufen,
- zur Steuerung von zeitabhängigen Ereignissen, z.B. als Wecker.

Der TimeSensor Node kann für absolute Zeit oder für Zeitintervalle verwendet werden. Zeitintervalle werden hier häufig auf das Intervall $[0,\ 1]$ normiert. In diesem Zusammenhang spricht man von "Fractional Time". Ein wichtiger Parameter für die Animation ist das Feld "fraction_changed". Dieser Parameter gibt die normierte Zeitdauer von 0.0 bis 1.0 an. Während der Fractional Time wird ein Keyframe durchlaufen. Soll eine Zeitschleife erfolgen, so muss der Parameter "loop" den logischen Wert TRUE annehmen.

7.2.3 PositionInterpolator Node

Der PositionInterpolator Node führt eine lineare Interpolation einer geradlinigen Bewegung durch. Die Syntax lautet:

Syntax:

```
PositionInterpolator {
    set_fraction           #  eventIn         SFFloat
    key             [ ]    #  exposedField    MFFloat
    keyValue        [ ]    #  exposedField    MFVec3f
    value_changed          #  eventOut        SFVec3f
}
```

Die Parameter "key" und "keyValue" werden für die Werte der Trajektorie verwendet. Die Trajektorie wird in der Form:

$$\vec{x}(k) = \begin{pmatrix} x(k) \\ y(k) \\ z(k) \end{pmatrix}$$

angegeben. Der Wert k ist der Parameter "key" und der Wert $\vec{x}(k)$ stellt Parameter "keyValue" dar. Zwischen den Werten wird jeweils linear interpoliert. Der Parameter "set_fraction" ist der Eingabewert für die Interpolation und der Parameter "value_changed" stellt das Interpolationsergebnis dar.

7.2.4 OrientationInterpolator Node

Der OrientationInterpolator Node führt eine lineare Interpolation einer zirkularen Bewegung durch. Die Syntax lautet:

Syntax:

```
OrientationInterpolator {
    set_fraction         #  eventIn         SFFloat
    key           [ ]    #  exposedField    MFFloat
    keyValue      [ ]    #  exposedField    MFRotation
    value_changed  #  eventOut        SFRotation
}
```

Die Trajektorie wird in der Form:

$$\vec{x}(k) = \begin{pmatrix} x(k) \\ y(k) \\ z(k) \end{pmatrix}$$

vorgegeben. Der Wert k ist der Parameter “key” und der Wert $\vec{x}(k)$ stellt Parameter “keyValue” dar. Zwischen den Werten wird jeweils zirkular interpoliert. Der Parameter “set_fraction” ist der Eingabewert für die Interpolation und der Parameter “value_changed” stellt das Interpolationsergebnis dar.

7.2.5 Programme

Programm 10 (Bewegter Einheitswürfel)
Der Einheitswürfel wird entlang eines vorgegebenen Weges bewegt.

```
#VRML V2.0 utf8
#
# Bewegter Einheitswuerfel
#
Group {
    children [
    # Moving box
        DEF Cube Transform {
            children Shape {
                appearance Appearance {
                    material Material { }
                }
                geometry Box { size 1.0 1.0 1.0 }
            }
        },
    # Animation clock
        DEF Clock TimeSensor {
            cycleInterval 4.0
            loop TRUE
        },
    # Animation path
        DEF CubePath PositionInterpolator {
            key [
                0.00, 0.11, 0.17, 0.22,
                0.33, 0.44, 0.50, 0.55,
                0.66, 0.77, 0.83, 0.88,
                0.99
            ]
            keyValue [
                0.0 0.0  0.0,  1.0 1.96 1.0,
                1.5 2.21 1.5,  2.0 1.96 2.0,
                3.0 0.0  3.0,  2.0 1.96 3.0,
                1.5 2.21 3.0,  1.0 1.96 3.0,
                0.0 0.0  3.0,  0.0 1.96 2.0,
                0.0 2.21 1.5,  0.0 1.96 1.0,
                0.0 0.0  0.0
            ]
        }
    ]
}
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_translation
```

Der Einheitwürfel wird periodisch entlang der Trajektorie

$$\begin{pmatrix} x(key) \\ y(key) \\ z(key) \end{pmatrix} = \begin{pmatrix} keyValue_x(key) \\ keyValue_y(key) \\ keyValue_z(key) \end{pmatrix}$$

bewegt.

Programm 11 (Kleines Universum)
Drei Planeten sollen um eine Sonne kreisen.

```
#VRML V2.0 utf8
#
# Universum
#
Group {
    children [
    # Stationary Sun
        Shape {
            appearance DEF White Appearance {
                material Material { }
            }
            geometry Sphere { }
        },
    # Several orbiting planets
        DEF Planet1 Transform {
            translation 2.0 0.0 0.0
            center -2.0 0.0 0.0
            children Shape {
                appearance USE White
                geometry Sphere { radius 0.2 }
            }
        },
        DEF Planet2 Transform {
            translation 3.0 0.0 0.0
            center -3.0 0.0 0.0
            children Shape {
                appearance USE White
                geometry Sphere { radius 0.3 }
            }
        },
        DEF Planet3 Transform {
            translation 4.0 0.0 0.0
            center -4.0 0.0 0.0
            children Shape {
                appearance USE White
                geometry Sphere { radius 0.5 }
            }
        },
    # Animation clocks, one per planet
        DEF Clock1 TimeSensor {
            cycleInterval 2.0
            loop TRUE
        },
        DEF Clock2 TimeSensor {
            cycleInterval 3.5
            loop TRUE
        },
        DEF Clock3 TimeSensor {
            cycleInterval 5.0
            loop TRUE
        },
    # Animation paths, one per planet
```

```
        DEF PlanetPath1 OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [
                0.0 0.0 1.0  0.0,
                0.0 0.0 1.0  3.14,
                0.0 0.0 1.0  6.28
            ]
        },
        DEF PlanetPath2 OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [
                0.0 0.0 1.0  0.0,
                0.0 0.0 1.0  3.14,
                0.0 0.0 1.0  6.28
            ]
        },
        DEF PlanetPath3 OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [
                0.0 0.0 1.0  0.0,
                0.0 0.0 1.0  3.14,
                0.0 0.0 1.0  6.28
            ]
        }
    ]
}
ROUTE Clock1.fraction_changed   TO PlanetPath1.set_fraction
ROUTE Clock2.fraction_changed   TO PlanetPath2.set_fraction
ROUTE Clock3.fraction_changed   TO PlanetPath3.set_fraction
ROUTE PlanetPath1.value_changed TO Planet1.set_rotation
ROUTE PlanetPath2.value_changed TO Planet2.set_rotation
ROUTE PlanetPath3.value_changed TO Planet3.set_rotation
```

Das Bild 7.13 zeigt das konstruierte Universum.

Abbildung 7.13: Universum mit Sonne und drei Planeten

7.3 Interaktion

Die Möglichkeit zur Interaktion ist eine wichtige Voraussetzung für VR-Anwendungen. VRML bietet für Interaktionen mit einer 3D-Szene folgende Sensor Nodes.

- TouchSensor Node
- PlaneSensor Node
- SphereSensor Node
- Cylinder Sensor Node

Die Nodes werden in Zusammenhang mit einer Maus oder einem anderen Eingabegerät verwendet. Sie reagieren auf folgende Ereignisse.

- Bewegung des Cursors über den Sensor (Move)
- Drücken der linken Maustaste (Click)
- Ziehbewegung bei gedrückter linker Maustaste (Drag)

Mit Hilfe der ROUTE-Anweisung werden die Reaktionen der Sensoren zu anderen Nodes verschickt.

7.3.1 TouchSensor Node

Die Bewegung der Maus wird vom TouchSensor verfolgt. Überdeckt sich der Cursor mit einem geometrischen Objekt, dass mit dem TouchSensor belegt wurde, so wird das Ereignis "isOver" ausgelöst.

Syntax:

```
TouchSensor {
    enabled TRUE           #  exposedField    SFBool
    hitNormal_changed      #  eventOut        SFVec3f
    hitPoint_changed       #  eventOut        SFVec3f
    hitTexCoord_changed    #  eventOut        SFVec2f
    isActive               #  eventOut        SFBool
    isOver                 #  eventOut        SFBool
    touchTime              #  eventOut        SFTime
}
```

Die Felder haben folgende Bedeutung:

- hitNormal_changed
  Liefert die Normale am aktuellen Schnittpunkt während der Berührung.
- hitPoint_changed
  Das Feld liefert die Koordinaten des aktuellen Schnittpunktes während der Berührung.
- hitTexCoord_changed
  Das Feld liefert die Texturkoordinaten des aktuellen Schnittpunktes während der Berührung, wenn das Objekt mit einer Textur überzogen ist.
- isActive
  Das Feld isActive nimmt den Wert TRUE an, wenn bei der Berührung die linke Maustaste gedrückt wird.
- isOver
  Das Feld isOver nimmt den Wert TRUE an, wenn eine Berührung stattfindet.
- touchTime
  Das Feld liefert die aktuelle Zeit, solange eine Berührung stattfindet.

Die Koordinaten, die das Feld hitPoint_changed liefert, können während einer Ziehbewegung mit der Maus (Drag) ausgewertet werden.

7.3.2 PlaneSensor Node

Mit Hilfe des PlaneSensors kann die Ziehbewegung mit einer Maus ausgewertet werden und in eine Translation des lokalen Koordinatensystems umgesetzt werden.

Syntax:

```
PlaneSensor {
    autoOffset          TRUE        #  exposedField    SFBool
    enabled             TRUE        #  exposedField    SFBool
    maxPosition         -1 -1       #  exposedField    SFVec2f
    minPosition         0   0       #  exposedField    SFVec2f
    offset              0   0   0   #  exposedField    SFVec3f
    isActive                        #  eventOut        SFBool
    trackPoint_changed              #  eventOut        SFVec3f
    translation_changed             #  eventOut        SFVec3f
}
```

Die Felder haben folgende Bedeutung:

- autoOffset
  Ist autoOffset TRUE, so wird nach beenden der Ziehbewegung das Feld offset mit dem aktuellen Wert überschrieben. Ist autoOffset dagegen FALSE, so bleibt der ursprüngliche Wert des Feldes offset erhalten.
- maxPosition und minPosition
  Die beiden Felder begrenzen den Bereich für die Translation.
- offset
  Das Feld Offset wird zur relativen Translation hinzuaddiert.
- isActive
  Das Feld isActive nimmt den Wert TRUE an, wenn bei der Berührung die linke Maustaste gedrückt wird.
- trackPoint_changed
  Das Feld liefert die Koordinaten des aktuellen Schnittpunktes während der Ziehbewegung.
- translation_changed
  Das Feld liefert die relativen Translationswerte, die aus der Ziehbewegung resultieren. Die Werte werden durch die Parameter maxPosition und minPosition beschränkt.

7.3.3 SphereSensor Node

Beim SphereSensor Node wird die Ziehbewegung mit der Maus in eine rollende Bewegung umgewandelt.

Syntax:

```
SphereSensor {
    autoOffset        TRUE      #  exposedField    SFBool
    enabled           TRUE      #  exposedField    SFBool
    offset            0 1 0 0   #  exposedField    SFRotation
    isActive                    #  eventOut        SFBool
    rotation_changed            #  eventOut        SFRotation
    trackPoint_changed          #  eventOut        SFVec3f
}
```

Die Felder haben folgende Bedeutung:

- autoOffset
  Ist autoOffset TRUE, so wird nach beenden der Ziehbewegung das Feld offset mit dem

aktuellen Wert überschrieben. Ist autoOffset dagegen FALSE, so bleibt der ursprüngliche Wert des Feldes offset erhalten.
- offset
  Das Feld Offset wird zur relativen Rotation hinzuaddiert.
- isActive Das Feld isActive nimmt den Wert TRUE an, wenn bei der Berührung die linke Maustaste gedrückt wird.
- rotation_changed
  Das Feld liefert die relativen Rotationswerte, die aus der Ziehbewegung resultieren.
- trackPoint_changed
  Das Feld liefert die Koordinaten des aktuellen Schnittpunktes während der Ziehbewegung.

7.3.4 CylinderSensor Node

Beim CylinderSensor Node wird die Ziehbewegung mit der Maus in eine rollende Bewegung umgewandelt.

Syntax:

```
CylinderSensor {
    autoOffset          TRUE      #  exposedField    SFBool
    diskAngle           0.262     #  exposedField    SFFloat
    enabled             TRUE      #  exposedField    SFBool
    maxAngle             -1       #  exposedField    SFFloat
    minAngle              0       #  exposedField    SFFloat
    offset                0       #  exposedField    SFFloat
    isActive                      #  eventOut        SFBool
    rotation_changed              #  eventOut        SFRotation
    trackPoint_changed            #  eventOut        SFVec3f
}
```

Die Felder haben folgende Bedeutung:

- autoOffset
  Ist autoOffset TRUE, so wird nach beenden der Ziehbewegung das Feld offset mit dem aktuellen Wert überschrieben. Ist autoOffset dagegen FALSE, so bleibt der ursprüngliche Wert des Feldes offset erhalten.
- diskAngle
  Ist der Winkel zwischen Schnittgeraden und y-Achse bei einer Berührung kleiner als der angegebene Wert, so wird der Schnittpunkt auf eine Scheibe projiziert, ansonsten auf einen Zylindermantel.
- maxAngle und minAngle
  Die beiden Felder begrenzen den Bereich für die Rotation.
- offset
  Das Feld Offset gibt den Offset-Wert der Rotation an. Die relative Rotation wird zu diesem Wert hinzuaddiert.
- isActive nimmt den Wert TRUE an, wenn bei der Berührung die linke Maustaste gedrückt wird.
- rotation_changed
  liefert die relativen Rotationswerte, die aus der Ziehbewegung resultieren.
- trackPoint_changed
  liefert die Koordinaten des aktuellen Schnittpunktes während der Ziehbewegung.

7.3.5 Verschachtelung von Sensoren

Ein Sensor Node ist nur innerhalb der Gruppe wirksam, in der er definiert ist. Werden mehrere Sensoren in geschachtelten Gruppen verwendet, so hat der Sensor der innersten Gruppe im Fall einer Aktivierung die höchste Priorität.

Die Schachtelung der Gruppen hat folgende Gestalt.

```
#VRML V2.0 utf8
#
#
Group {
    children [
        PlaneSensor{ }

        ...

        Group {
            children [
                TouchSensor{ }

                ...

            ]
        }
   ]
}
```

7.3.6 Programme

Programm 12 (Drehbarer Würfel 1)

```
#VRML V2.0 utf8
#
# Drehender Wuerfel
#
Group {
    children [
    # Rotating Cube
        DEF Cube Transform {
            children Shape {
                appearance Appearance {
                    material Material { }
                }
                geometry Box { }
            }
        },
    # Sensor
        DEF Touch TouchSensor { },
    # Animation clock
        DEF Clock TimeSensor {
            enabled FALSE
            cycleInterval 4.0
            loop TRUE
        },
    # Animation path
        DEF CubePath OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [
                0.0 1.0 0.0  0.0,
                0.0 1.0 0.0  3.14,
                0.0 1.0 0.0  6.28
            ]
        }
    ]
}
ROUTE Touch.isOver           TO Clock.set_enabled
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_rotation
```

Das Bild 7.14 zeigt den Würfel.

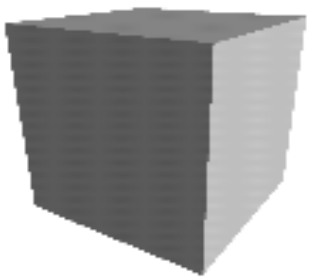

Abbildung 7.14: Drehbarer Würfel

Programm 13 (Drehbarer Würfel 2)
Der Würfel dreht sich, wenn der Cursor auf den Würfel zeigt und die linke Maustaste gedrückt ist.

```
#VRML V2.0 utf8
#
# Drehender Würfel 2
#
Group {
    children [
    # Rotating Cube
        DEF Cube Transform {
            children Shape {
                appearance Appearance {
                    material Material { }
                }
                geometry Box { }
            }
        },
    # Sensor
        DEF Touch TouchSensor { },
    # Animation clock
        DEF Clock TimeSensor { cycleInterval 4.0 },
    # Animation path
        DEF CubePath OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [
                0.0 1.0 0.0  0.0,
                0.0 1.0 0.0  3.14,
                0.0 1.0 0.0  6.28
            ]
        }
    ]
}
ROUTE Touch.touchTime        TO Clock.set_startTime
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_rotation
```

Programm 14 (Verschiebbarer Würfel)
Der Würfel kann bei gedrückter Maustaste verschoben werden.

```
#VRML V2.0 utf8
```

```
#
# Verschiebbarer Wuerfel
#
Group {
    children [
        Transform {
            rotation 1.0 0.0 0.0 -1.57
            children DEF Cube Transform {
                children Shape {
                    appearance Appearance {
                        material Material { }
                    }
                    geometry Box { }
                }
            }
        },
        DEF Sensor PlaneSensor {
            minPosition -2.0 -2.0
            maxPosition  2.0  2.0
        }
    ]
}
ROUTE Sensor.translation_changed TO Cube.set_translation
```

Programm 15 (Würfel und Kegel mit SphereSensor)
Ein Würfel und ein Kegel können mit Hilfe eines SphereSensor gedreht werden.

```
#VRML V2.0 utf8
#
# Würfel und Kegel mit SphereSensor
#
Group {
    children [
        Group {
            children [
                DEF Shape1 Transform {
                    children Shape {
                        appearance DEF White Appearance {
                            material Material { }
                        }
                        geometry Box { }
                    }
                },
                DEF Shape1Sensor SphereSensor { }
            ]
        },
        Group {
            children [
                DEF Shape2 Transform {
                    translation 2.5 0.0 0.0
                    children Shape {
                        appearance USE White
                        geometry Cone { }
                    }
                },
                DEF Shape2Sensor SphereSensor { }
            ]
        }
    ]
}
ROUTE Shape1Sensor.rotation_changed TO Shape1.set_rotation
ROUTE Shape2Sensor.rotation_changed TO Shape2.set_rotation
```

Das Bild 7.15 zeigt den Würfel und den Kegel.

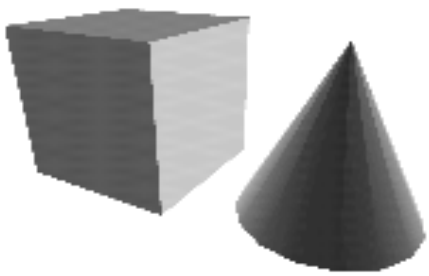

Abbildung 7.15: Würfel und Kegel

Programm 16 (Vier verschiebbare Würfel)
Vier Würfel sollen unabhängig voneinander verschoben werden können.

```
#VRML V2.0 utf8
#
# Vier verschiebbare Würfel
#
Group {
    children [
        Group {
            children [
                DEF Block1 Transform {
                    children DEF BlockShape Shape {
                        appearance Appearance {
                            material Material { }
                        }
                        geometry Box { }
                    }
                },
                DEF Block1Sensor PlaneSensor {
                    offset 0.0 0.0 0.0
                }
            ]
        },
        Group {
            children [
                DEF Block2 Transform {
                    translation 2.5 0.0 0.0
                    children USE BlockShape
                },
                DEF Block2Sensor PlaneSensor {
                    offset 2.5 0.0 0.0
                }
            ]
        },
        Group {
            children [
                DEF Block3 Transform {
                    translation 1.5 2.0 0.0
                    children USE BlockShape
                },
                DEF Block3Sensor PlaneSensor {
                    offset 1.5 2.0 0.0
                }
            ]
        },
        Group {
            children [
                DEF Block4 Transform {
                    translation 0.75 4.0 0.0
                    children USE BlockShape
                },
                DEF Block4Sensor PlaneSensor {
                    offset 0.75 4.0 0.0
                }
```

```
            ]
        }
    ]
}
ROUTE Block1Sensor.translation_changed TO Block1.set_translation
ROUTE Block2Sensor.translation_changed TO Block2.set_translation
ROUTE Block3Sensor.translation_changed TO Block3.set_translation
ROUTE Block4Sensor.translation_changed TO Block4.set_translation
```

Das Bild 7.16 zeigt die vier Würfel.

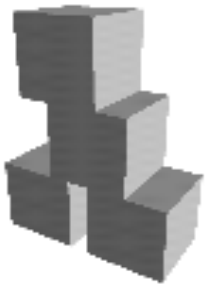

Abbildung 7.16: Vier Würfel

Programm 17 (Einstellbare Schreibtischlampe)
Die Schreibtischlampe kann in der x-z-Ebene verschoben und eingestellt werden.

```
#VRML V2.0 utf8
#
# Einstellbare Schreibtischlampe
#
Group {
  children [
  # Lamp
    DEF MoveLamp PlaneSensor { },
    DEF Lamp Transform {
      children [
      # Lamp base
        Shape {
          appearance DEF White Appearance {
            material Material { }
          }
          geometry Cylinder {
            radius 0.1
            height 0.01
          }
        },
      # Base - First arm joint
        Group {
          children [
            DEF MoveFirstArm SphereSensor {
              offset 1.0 0.0 0.0 -0.7
            },
            DEF FirstArm Transform {
              translation 0.0 0.15 0.0
              rotation    1.0 0.0 0.0  -0.7
              center      0.0 -0.15 0.0
              children [
              # Lower arm
                DEF LampArm Shape {
                  appearance USE White
                  geometry Cylinder {
                    radius 0.01
                    height 0.3
                  }
                },
              # First arm - second arm joint
                Group {
                  children [
                    DEF MoveSecondArm SphereSensor {
                      offset 1.0 0.0 0.0 1.9
                    },
                    DEF SecondArm Transform {
                      translation 0.0 0.3 0.0
                      rotation  1.0 0.0 0.0  1.9
                      center    0.0 -0.15 0.0
                      children [
                      # Second arm
                        USE LampArm,
```

```
                    # Second arm - shade joint
                      Group {
                        children [
                          DEF MoveLampShade SphereSensor {
                            offset 1.0 0.0 0.0 -1.25
                          },
                          DEF LampShade Transform {
                            translation 0.0 0.075 0.0
                            rotation  1.0 0.0 0.0  -1.25
                            center    0.0 0.075 0.0
                            children [
                            # Shade
                              Shape {
                                appearance USE White
                                geometry Cone {
                                  height 0.15
                                  bottomRadius 0.12
                                  bottom FALSE
                                }
                              },
                            # Light bulb
                              Transform {
                                translation 0.0 -0.05 0.0
                                children Shape {
                                  appearance USE White
                                  geometry Sphere {
                                    radius 0.05
                                  }
                                }
                              }
                            ]
                          }
                        ]
                      }
                    ]
                  }
                ]
              }
            ]
          }
        ]
      }
    ]
  }
}
ROUTE MoveLamp.translation_changed   TO Lamp.set_translation
ROUTE MoveFirstArm.rotation_changed  TO FirstArm.set_rotation
ROUTE MoveSecondArm.rotation_changed TO SecondArm.set_rotation
ROUTE MoveLampShade.rotation_changed TO LampShade.set_rotation
```

Das Bild 7.17 zeigt die Schreibtischlampe.

Abbildung 7.17: Bewegbare Schreibtischlampe

7.4 Kontrolle der Viewer-Position

VRML stellt drei Nodes für die Kontrolle der Viewer-Position zur Verfügung. Es handelt sich um die drei Nodes:

- VisibilitySensor,
- ProximitySensor,
- Collision.

7.4.1 VisibilitySensor Node

Der VisibilitySensor Node gibt an, ob ein quaderförmiger Bereich (Bounding-Box) im momentanen Blickfeld liegt oder nicht. Der Sensor liefert ein Ereignis, sobald ein Teil des Bereiches in das Blickfeld eintritt. Nicht berücksichtigt werden mögliche Verdeckungen durch andere Körper. Der Sensor kann beispielsweise verwendet werden, um eine Animation zu starten.

Der VisibilitySensor Node hat folgenden Aufbau:

Syntax:

```
VisibilitySensor {
   center       0 0 0    #  exposedField    SFVec3f
   enabled      TRUE     #  exposedField    SFBool
   size         0 0 0    #  exposedField    SFVec3f
   enterTime             #  eventOut        SFTime
   exitTime              #  eventOut        SFTime
   isActive              #  eventOut        SFBool
}
```

Die Felder haben folgende Bedeutung:

- center
  Das Feld center gibt das Zentrum des quaderförmigen Bereiches an.
- size
  Das Feld size gibt die Kantenlängen des quaderförmigen Bereiches an.
- isActive
  Nimmt den Wert TRUE an, wenn der quaderförmige Bereich sichtbar ist.
- enterTime
  Gibt die aktuelle Zeit für das Eintreten des Ereignisse an (isActive = TRUE).

- exitTime
  Gibt die aktuelle Zeit für die Beendigung des Ereignisses an (isActive = FALSE).

7.4.2 ProximitySensor Node

Der ProximitySensor Node liefert Ereignisse, wenn der Betrachter einen festgelegten quaderförmigen Bereich betritt bzw. ihn verläßt. Der ProximitySensor Node eignet sich zum Starten von Animationen.

Der ProximitySensor Node hat folgenden Aufbau:

Syntax:

```
ProximitySensor {
    center              0 0 0     #  exposedField    SFVec3f
    size                0 0 0     #  exposedField    SFVec3f
    enabled             TRUE      #  exposedField    SFBool
    isActive                      #  eventOut        SFBool
    position_changed              #  eventOut        SFVec3f
    orientation_changed           #  eventOut        SFRotation
    enterTime                     #  eventOut        SFTime
    exitTime                      #  eventOut        SFTime
}
```

Die Felder haben folgende Bedeutung:

- center
  Das Feld center gibt das Zentrum des quaderförmigen Bereiches an.
- size
  Das Feld size gibt die Kantenlängen des quaderförmigen Bereiches an.
- isActive
  Nimmt den Wert TRUE an, wenn der quaderförmige Bereich sichtbar ist.
- position_changed
  Das Feld position_changed liefert die Koordinaten der Betrachterbewegung relativ zum lokalen Koordinatensystem.
- orientation_changed
  Das Feld orientation_changed liefert die Drehwinkel der Betrachterbewegung relativ zum lokalen Koordinatensystem.
- enterTime
  Gibt die aktuelle Zeit für das Eintreten des Ereignisses an (isActive = TRUE).
- exitTime
  Gibt die aktuelle Zeit für die Beendigung des Ereignisses an (isActive = FALSE).

7.4.3 Collision Node

Der Collision Node dient dazu, Kollisionen aufzudecken. In der Realität sind feste Körper nicht durchdringbar. Das Ereignis einer Kollision kann mit diesem Node überwacht werden. Der Versuch durch einen festen Körper hindurch zu navigieren kann dann gestoppt werden. Dadurch wird das Verhalten realistischer.

Bei komplizierten Körpern kann die Berechnung der Kollision sehr aufwendig werden. In Hinblick auf eine zeitliche Optimierung bietet der Collision Node die Möglichkeit, einen einfacheren Hilfsknoten (Proxy Node) anzugeben, der eine ähnliche, aber einfacher umgebende Hülle aufweist.

Der Collision Node stellt einen Grouping Node dar. Eine Kollision findet statt, wenn der Weg des Betrachters, die Bounding-Box des Child Nodes oder die Bounding-Box des Hilfsknotens (Proxy Node) schneidet.

Der Collision Node hat folgenden Aufbau:

Syntax:

```
Collision {
    addChildren                 #  eventIn          MFNode
    removeChildren              #  eventIn          MFNode
    children          [ ]       #  exposedField     MFNode
    collide           TRUE      #  exposedField     SFBool
    bboxCenter        0  0  0   #  field            SFVec3f
    bboxSize         -1 -1 -1   #  field            SFVec3f
    proxy             NULL      #  field            SFNode
    collideTime                 #  eventOut         SFTime
}
```

Die Felder haben folgende Bedeutung:

- addChildren
  Das Feld addChildren dient zum ereignisgesteuerten Eintragen eines weiteren Child Nodes.
- removeChildren
  Das Feld removeChildren dient zum ereignisgesteuerten Löschen eines Child Nodes.
- children
  Das Feld children nimmt alle Child Nodes auf.
- collide
  Das Feld collide nimmt den logischen Wert TRUE an, wenn eine Kollision stattgefunden hat.
- bboxCenter
  Die Koordinaten geben das Zentrum der Bounding-Box an.
- bboxSize
  Die Längenangaben bedeuten die Kantenlängen der Bounding-Box.
- proxy
  Ein hier angegebener geometrischer Hilfsknoten wird nicht dargestellt, sondern nur bei der Berechnung der Kollision verwendet. Die Kollision von Child Nodes wird bei der Berechnung nicht mehr berücksichtigt.
- collideTime
  Ab den Zeitpunkt des Eintreten einer Kollision (collide = TRUE) wird die aktuelle Zeit im Feld collideTime angegeben. Weitere Child Nodes, die ebenfalls kollidieren, senden dieses Ereignisse.

7.5 Lichtquellen

VRML bietet drei Nodes für Lichtquellen zur Auswahl.

- PointLight Node
- DirectionalLight Node
- SpotLight Node

7.5.1 PointLight Node

Syntax:

```
PointLight {
    ambientIntensity   0       #  exposedField    SFFloat
    attenuation        1 0 0   #  exposedField    SFVec3f
    color              1 1 1   #  exposedField    SFColor
    intensity          1       #  exposedField    SFFloat
    location           0 0 0   #  exposedField    SFVec3f
    on                 TRUE    #  exposedField    SFBool
    radius             100     #  exposedField    SFFloat
}
```

Das Abschwächen der Lichtintensität mit dem Abstand wird über den Parameter attenuation berechnet. Der Abschwächungsfaktor f_{ab} berechnet sich nach folgender Formel aus dem Abstand r:

$$f_{ab} = \frac{1}{attenuation[0] + attenuation[1] \cdot r + attenuation[2] \cdot r^2} \tag{7.1}$$

Die Felder haben folgende Bedeutung:

- ambientIntensity
  Beitrag der Intensität zum ambienten Licht
- attenuation
  Festlegung der Abschwächungsfunktion f_{ab} nach Gl.(7.1).
- color
  Farbe des Punktlichtes.
- intensity
  Intensität des Punktlichtes.
- location
  Koordinaten des Punktlichtes.
- on
  Der logische Parameter wirkt als Lichtschalter. Bei TRUE ist die Lichtquelle eingeschaltet.
- radius
  Innerhalb des angegebenen Radius werden alle Objekte beleuchtet.

7.5.2 SpotLight Node

Syntax:

```
SpotLight {
    ambientIntensity    0           #  exposedField    SFFloat
    attenuation         1 0 0       #  exposedField    SFVec3f
    beamWidth           1.570796    #  exposedField    SFFloat
```

```
    color               1 1 1       #  exposedField    SFColor
    cutOffAngle         0.785398    #  exposedField    SFFloat
    direction           0 0 -1      #  exposedField    SFVec3f
    intensity           1           #  exposedField    SFFloat
    location            0 0 0       #  exposedField    SFVec3f
    on                  TRUE        #  exposedField    SFBool
    radius              100         #  exposedField    SFFloat
}
```

Die Felder haben folgende Bedeutung:

- ambientIntensity
  Beitrag der Intensität zum ambienten Licht
- attenuation
  Festlegung der Abschwächungsfunktion f_{ab} nach Gl.(7.1).
- beamWidth
  Das Feld beamWidth entspricht dem Öffnungwinkel des Spotlichtkegels. Der Wert liegt zwischen $[0, \pi/2]$.
- color
  Farbe des Punktlichtes.
- cutOffAngle
  Außerhalb des angegebenen Winkels ist die Intensität Null.
- direction
  Das Feld gibt die Achse des Spotlichtkegels an.
- intensity
  Intensität des Spotlights.
- location
  Koordinaten des Punktlichtes.
- on
  Der logische Parameter wirkt als Lichtschalter. Bei TRUE ist die Lichtquelle eingeschaltet.
- radius
  Innerhalb des angegebenen Radius werden alle Objekte beleuchtet.

Die Intensität des Spotlights wird nach folgender Formel berechnet:

$$I(\alpha) = intensity \cdot \cos^p(\alpha) \tag{7.2}$$

$$p = \frac{0.5 \cdot \log(0.5)}{\log(\cos(beamWidth))} \tag{7.3}$$

Der Winkel α wird zwischen Richtungsvektor und einer beliebigen Strahlungsrichtung bestimmt.

7.5.3 DirectionalLight Node

Syntax:

```
DirectionalLight {
    ambientIntensity   0        #   exposedField SFFloat
    color              1 1 1    #   exposedField SFColor
    direction          0 0 -1   #   exposedField SFVec3f
    intensity          1        #   exposedField SFFloat
    on                 TRUE     #   exposedField SFBool
}
```

Die Parameter haben folgende Bedeutung:

- ambientIntensity
  Beitrag der Intensität zum ambienten Licht
- color
  Farbe des Punktlichtes.
- direction
  Das Feld gibt die Achse des Spotlichtkegel an.
- intensity
  Intensität des Spotlights.
- on
  Der logische Parameter wirkt als Lichtschalter. Bei TRUE ist die Lichtquelle eingeschaltet.

7.6 Facettierte Körper

Neben den primitiven geometrischen Körpern, stellt VRML weitere Nodes bereit, um komplexere Körper zu gestalten. Weitere Möglichkeiten bieten die folgenden Nodes:

- ElevationGrid,
- IndexedFaceSet,
- Extrusion.

7.6.1 ElevationGrid Node

Mit Hilfe des ElevationGrid Node kann eine dreidimensionale Fläche mit äquidistanten Stützstellen dargestellt werden. Die Funktion wird in der folgenden Form angegeben.

$$\begin{aligned} y(i,j) &= heigh(j \cdot xDimension + i)\ , \\ & \quad 0 \leq i \leq xDimension - 1\ , \\ & \quad 0 \leq j \leq zDimension - 1 \end{aligned} \tag{7.4}$$

Syntax:

```
ElevationGrid {
    set_height              #  eventIn         MFFloat
    color           NULL    #  exposedField    SFNode
    normal          NULL    #  exposedField    SFNode
    texCoord        NULL    #  exposedField    SFNode
    height          [ ]     #  field           MFFloat
    ccw             TRUE    #  field           SFBool
    colorPerVertex  TRUE    #  field           SFBool
    creaseAngle     0       #  field           SFFloat
    normalPerVertex TRUE    #  field           SFBool
    solid           TRUE    #  field           SFBool
    xDimension      0       #  field           SFInt32
    xSpacing        1.0     #  field           SFFloat
    zDimension      0       #  field           SFInt32
    zSpacing        1.0     #  field           SFFloat
}
```

Die Felder haben folgende Bedeutung:

- set_height
  Das Feld set_height stellt die veränderten Höhenwerte nach einem empfangenen Ereignis bereit.
- color
  Das Feld verweist auf einen Color Node.
- normal
  Jedem Höhenwert kann eine Normale zugewiesen werden.
- texCoord
  Das Feld verweist auf einen TextureCoordinate Node.
- height
  Das Feld stellt die Höhenwerte bereit. Diese können ereignisgesteuert überschrieben werden.
- ccw
  Beim logischen Wert TRUE sind die Eckpunkte bei Aufsicht entgegen dem Uhrzeigersinn sortiert.

- colorPerVertex
  Ist das color field nicht leer, so werden beim logischen Wert TRUE die Eckpunkte und beim logischen Wert FALSE die Flächenelemente zwischen jeweils benachbarten Eckpunkten eingefärbt.
- creaseAngle
  Ist der angegebene Winkel zwischen Normalen und zwei benachbarten Flächen kleiner als der Parameter creaseAngle, so wird weich schattiert, andernfalls entsteht ein Knick, der ein facettenhaftes Aussehen bewirkt.
- normalPerVertex
  Ist der Parameter mit dem logischen Wert TRUE gesetzt, so werden die Normalen den Eckpunkten und beim logischen Wert FALSE den Flächenelementen zwischen jeweils vier Eckpunkten zugeordnet.
- solid
  Beim logischen Wert TRUE wird für den dargestellten Körper ein Volumen angenommen.
- xDimension
  Anzahl der Wert in x-Richtung.
- xSpacing
  Skalierungsfaktor für die Abstände in x-Richtung.
- zDimension
  Anzahl der Werte in z-Richtung.
- zSpacing
  Skalierungsfaktor für die Abstände in z-Richtung.

7.6.2 IndexedFaceSet Node

Eine sehr häufige Darstellung von komplexen Körpern erfolgt mit Hilfe von Polygonen (engl.: faces). Alle Polygone eines IndexedFaceSet Node werden aus Punkten oder Vektoren konstruiert, deren Koordinaten in einem weiteren Node, den Coordinate Node, enthalten sind. Nach Abschluß jedes Polygons ist zur Trennung vom nächsten anstelle eines Indexes der Wert -1 anzugeben. Der IndexedFaceSet Node hat folgenden Aufbau:

Syntax:

```
IndexedFaceSet {
    set_colorIndex              #  eventIn         MFInt32
    set_coordIndex              #  eventIn         MFInt32
    set_normalIndex             #  eventIn         MFInt32
    set_texCoordIndex           #  eventIn         MFInt32
    color               NULL    #  exposedField    SFNode
    coord               NULL    #  exposedField    SFNode
    normal              NULL    #  exposedField    SFNode
    texCoord            NULL    #  exposedField    SFNode
    ccw                 TRUE    #  field           SFBool
    colorIndex          [ ]     #  field           MFInt32
    colorPerVertex      TRUE    #  field           SFBool
    convex              TRUE    #  field           SFBool
    coordIndex          [ ]     #  field           MFInt32
    creaseAngle         0       #  field           SFFloat
    normalIndex         [ ]     #  field           MFInt32
    normalPerVertex     TRUE    #  field           SFBool
    solid               TRUE    #  field           SFBool
    texCoordIndex       [ ]     #  field           MFInt32
```

```
}
```

Die Felder haben folgende Bedeutung:

- set_colorIndex
  Das Feld set_colorIndex ermöglicht die ereignisgesteuerte Änderung des colorIndex.
- set_coordIndex
  Das Feld set_coordIndex ermöglicht die ereignisgesteuerte Änderung des coordIndex.
- set_normalIndex
  Das Feld set_normalIndex ermöglicht die ereignisgesteuerte Änderung des normalIndex.
- set_texCoordIndex
  Das Feld set_texCoordIndex ermöglicht die ereignisgesteuerte Änderung der Eckpunkte der Polygone.
- color
  Das Feld verweist auf einen Color Node.
- coord
  Das Feld verweist auf einen Coordinate Node mit den Koordinaten von Punkten.
- normal
  Jedem Eckpunkt kann eine Normale zugewiesen werden.
- texCoord
  Der Parameter verweist auf einen TextureCoordinate Node.
- ccw
  Beim logischen Wert TRUE sind die Eckpunkte bei Aufsicht entgegen dem Uhrzeigersinn sortiert.
- colorIndex
  Über das Feld colorIndex können die Farbwerte des verwendeten Color Nodes den Polygonflächen oder Eckpunkten zugeordnet werden.
- colorPerVertex
  Ist das color Feld nicht leer, so werden beim logischen Wert TRUE die Eckpunkte und beim logischen Wert FALSE die Flächenelemente zwischen jeweils benachbarten Eckpunkten eingefärbt.
- convex
  Wird der logische Wert TRUE gesetzt, so ist der Körper nach außen gewölbt (konvex).
- coordIndex
  Die Festlegung der Polygone besteht aus einer Folge von Eckpunkten. Für jeden Eckpunkt wird der Index seiner Koordinaten im Coodinate Node angegeben.
- creaseAngle
  Ist der angegebene Winkel zwischen Normalen und zwei benachbarten Flächen kleiner als der Parameter creaseAngle, so wird weich schattiert, andernfalls entsteht ein Knick, der ein facettenhaftes Aussehen bewirkt.
- normalIndex
  Über das Feld normalIndex können die Normalen des verwendeten Normal Nodes den Polygonflächen zugeordnet werden.
- normalPerVertex
  Ist der Parameter mit dem logischen Wert TRUE gesetzt, so werden die Normalen den Eckpunkten und beim logischen Wert FALSE den Flächenelementen zwischen jeweils vier Eckpunkten zugeordnet.
- solid
  Beim logischen Wert TRUE wird für den dargestellten Körper ein Volumen angenommen.
- texCoordIndex
  Den Eckpunkten der Polygonflächen können die Texturkoordinaten des verwendeten TextureCoordinate Nodes über einen Index zugeordnet werden.

7.6.3 Extrusion Node

Mit Hilfe des Extrusion Nodes können Extrusionskörper erzeugt werden. Der Extrusion Node hat folgenden Aufbau:

Syntax:

```
Extrusion {
    set_crossSection                                # eventIn   MFVec2f
    set_orientation                                 # eventIn   MFRotation
    set_scale                                       # eventIn   MFVec2f
    set_spine                                       # eventIn   MFVec3f
    beginCap          TRUE                          # field     SFBool
    ccw               TRUE                          # field     SFBool
    convex            TRUE                          # field     SFBool
    creaseAngle       0                             # field     SFFloat
    crossSection      [1 1, 1 -1, -1 -1, -1 1, 1 1] # field     MFVec2f
    endCap            TRUE                          # field     SFBool
    orientation       0 0 1 0                       # field     MFRotation
    scale             1 1                           # field     MFVec2f
    solid             TRUE                          # field     SFBool
    spine             [ 0 0 0, 0 1 0 ]              # field     MFVec3f
}
```

Die Felder haben folgende Bedeutung:

- set_crossSection
  Das Feld set_crossSection ändert ereignisgesteuert die Werte crossSection.
- set_orientation
  Das Feld set_orientation ändert ereignisgesteuert die Werte orientation.
- set_scale
  Das Feld set_scale ändert ereignisgesteuert die Werte scale.
- set_spine
  Das Feld set_spine ändert ereignisgesteuert die Werte spine.
- beginCap
  Beim logischen Wert TRUE wird ein Deckel am Beginn des Extrusionskörpers wiedergegeben.
- ccw
  Beim logischen Wert TRUE sind die Eckpunkte bei Aufsicht entgegen dem Uhrzeigersinn sortiert.
- convex
  Wird der logische Wert TRUE gesetzt, so ist der Körper nach außen gewölbt (konvex).
- creaseAngle
  Ist der angegebene Winkel zwischen Normalen und zwei benachbarten Flächen kleiner als der Parameter creaseAngle, so wird weich schattiert, andernfalls entsteht ein Knick, der ein facettenhaftes Aussehen bewirkt.
- crossSection
  Der Querschnitt wird durch einen ebenen Polygonzug definiert. Jeder Eckpunkt wird durch zwei Koordinaten in der xz-Ebene angegeben.
- endCap
  Beim logischen Wert TRUE wird ein Deckel am Ende des Extrusionskörpers wiedergegeben.
- orientation
  Vor jeder Extrusion wird der Querschnitt so rotiert, dass die Normale in Richtung der

Trajektorie zeigt. Pro Eckpunkt der Trajektorie kann eine Rotation angegeben werden oder ein einheitlicher Wert für alle Eckpunkte.

- scale
  Vor jeder Extrusion wird der Querschnitt mit einem Faktor in x- und z-Richtung skaliert. Pro Punkt der Trajektorie kann ein Wertepaar angegeben werden oder ein einheitliches Wertepaar für alle Punkte.
- solid
  Beim logischen Wert TRUE wird für den dargestellten Körper ein Volumen angenommen.
- spine
  Die Trajektorie wird als Polygonzug im Raum angegeben, für jeden Eckpunkt ist ein Vektor mit drei Koordinaten anzugeben.

7.6.4 Programme

Programm 18 (Farbiges Gebirge)
Eine dreidimensionale Funktion soll als farbiges Gebirge visualisiert werden.

```
#VRML V2.0 utf8
#
# Farbiges Gebirge
#
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry ElevationGrid {
        xDimension  9
        zDimension  9
        xSpacing    1.0
        zSpacing    1.0
        solid       FALSE
        creaseAngle 0.785
        height [
            0.0, 0.0, 0.5, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 2.5, 0.5, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.5, 0.5, 3.0, 1.0, 0.5, 0.0, 1.0,
            0.0, 0.0, 0.5, 2.0, 4.5, 2.5, 1.0, 1.5, 0.5,
            1.0, 2.5, 3.0, 4.5, 5.5, 3.5, 3.0, 1.0, 0.0,
            0.5, 2.0, 2.0, 2.5, 3.5, 4.0, 2.0, 0.5, 0.0,
            0.0, 0.0, 0.5, 1.5, 1.0, 2.0, 3.0, 1.5, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 1.5, 0.5,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0,
        ]
    }
}
```

Das folgende Bild 7.18 zeigt das Gebirge.

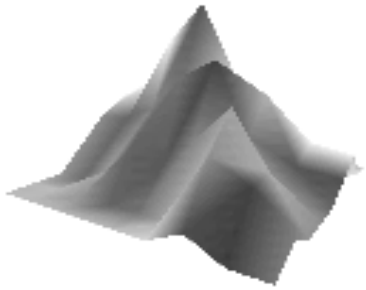

Abbildung 7.18: Farbiges Gebirge

Programm 19 (Pyramide)
Eine Pyramide soll mit Polygonen aufgebaut werden.

```
#VRML V2.0 utf8
#
# Pyramide in Polygondarstellung
#
Transform {
     translation 6 0 0
     children [
          Shape {
               geometry IndexedFaceSet {
                    colorPerVertex FALSE
                    color Color {
                         color [
                              1 0 0,
                              0 1 0,
                              1 1 0,
                              1 0 1,
                              0 1 1
                         ]
                    }
                    coord Coordinate {
                         point [
                                -2,  0, -2
                                 2,  0, -2
                                -2,  0,  2
                                 2,  0,  2
                                 0,  3,  0
                         ]
                    }
                    coordIndex [        # gegen Uhrzeigersinn
                              4, 1, 0, -1,
                              4, 3, 1, -1,
                              4, 2, 3, -1,
                              4, 0, 2, -1,
                              2, 0, 1, 3        # Boden
                    ]
               }
          }
     ]
```

```
}
```

Das folgende Bild 7.19 zeigt die Pyramide.

Abbildung 7.19: Pyramide

Programm 20 (Extrusionskörper)
Eine Vase soll als Extrusionskörper dargestellt werden.

```
#VRML V2.0 utf8
#
# Vase als Extrusionskörper
#
Shape {
    appearance Appearance {
        material Material {
            diffuseColor 1.0 0.8 0.0
        }
    }
    geometry Extrusion {
        creaseAngle 1.57
        endCap FALSE
        solid  FALSE
        crossSection [
        # Circle
             1.00  0.00,   0.92 -0.38,
             0.71 -0.71,   0.38 -0.92,
             0.00 -1.00,  -0.38 -0.92,
            -0.71 -0.71,  -0.92 -0.38,
            -1.00 -0.00,  -0.92  0.38,
            -0.71  0.71,  -0.38  0.92,
             0.00  1.00,   0.38  0.92,
             0.71  0.71,   0.92  0.38,
             1.00  0.00
        ]
        spine [
        # Straight-line
            0.0 0.0 0.0,  0.0 0.4 0.0,
            0.0 0.8 0.0,  0.0 1.2 0.0,
            0.0 1.6 0.0,  0.0 2.0 0.0,
            0.0 2.4 0.0,  0.0 2.8 0.0,
            0.0 3.2 0.0,  0.0 3.6 0.0,
            0.0 4.0 0.0
        ]
        scale [
            1.8  1.8,  1.95 1.95,
            2.0  2.0,  1.95 1.95
            1.8  1.8,  1.5  1.5
            1.2  1.2,  1.05 1.05,
            1.0  1.0,  1.05  1.05,
            1.15 1.15,
        ]
    }
}
```

Das folgende Bild 7.20 zeigt die Vase.

Abbildung 7.20: Extrusionskörper

7.7 Details für eine 3D-Szene

VRML stellt einige Nodes zu Verfügung, um auf einfachem Wege Details in eine 3D-Szene einzubringen, ohne dass der Aufwand zu groß wird. Folgende Nodes sind dabei hilfreich:

- Background Node
- Level of Detail Node (LOD Node)
- Switch Node
- Billboard Node
- Inline Node
- Anchor Node

7.7.1 Background Node

Mit dem Background Node läßt sich das gesamte Umfeld einer 3D-Szene nach eigenen Wünschen gestalten. Anstelle der eintönigen Hintergrundfarbe des VRML-Browsers lassen sich Farben und Farbverläufe für den Himmel und Boden angeben. Während der Boden als Halbkugel mit unendlichem Radius berechnet wird, umschließt der Himmel als vollständige Kugel die gesamte Welt in unendlicher Entfernung. Zusätzlich zum simulierten Himmel und Boden läßt sich eine 3D-Szene mit Texturen umgeben. Insgesamt können sechs verschiedene Texturen eingesetzt werden, die auf jeweils einer der Innenseiten eines unendlich großen Quaders plaziert werden. In Verbindung mit teiltransparenten Texturen kann ein sehr realistisches Umfeld erzeugt werden.

Die Syntax für den Background Node lautet:

Syntax:

```
Background {
     set_bind                  #  eventIn      SFBool
     groundAngle    [ ]        #  exposedField MFFloat
     groundColor    [ ]        #  exposedField MFColor
     backUrl        [ ]        #  exposedField MFString
     bottomUrl      [ ]        #  exposedField MFString
     frontUrl       [ ]        #  exposedField MFString
     leftUrl        [ ]        #  exposedField MFString
     rightUrl       [ ]        #  exposedField MFString
     topUrl         [ ]        #  exposedField MFString
     skyAngle       [ ]        #  exposedField MFFloat
```

```
    skyColor       [ 0 0 0 ]     #  exposedField MFColor
    isBound                      #  eventOut     SFBool
}
```

Die Felder haben folgende Bedeutung:

- set_bind
  Ein empfangenes Ereignis mit dem logischen Wert TRUE bindet den Node, er kommt auf den Stack und wird aktiv. Ein Ereignis mit dem logischen Wert FALSE nimmt den Node vom Stack. Damit können verschiedene Hintergrundszenarien aktiviert werden.
- groundAngle
  Für jede Farbe des Bodens (außer der ersten) wird ein Winkel angegeben, bei dem der Farbverlauf beginnt.
- groundColor
  Das Feld groundColor gibt die Farbstufungen des Bodens an.
- backUrl
  Verweis auf eine Graphikdatei für das hintere Umfeld.
- bottomUrl
  Verweis auf eine Graphikdatei für den Boden.
- frontUrl
  Verweis auf eine Graphikdatei für das vordere Umfeld.
- leftUrl
  Verweis auf eine Graphikdatei für das linke Umfeld.
- rightUrl
  Verweis auf eine Graphikdatei für das rechte Umfeld.
- topUrl
  Verweis auf eine Graphikdatei für den Himmel.
- skyAngle
  Für jede Farbe des Himmels (außer der ersten) wird ein Winkel angegeben, bei dem der Farbverlauf beginnt.
- skyColor
  Das Feld skyColor gibt die Farbstufungen des Himmels an.
- isBound
  Beim Binden des Nodes wird ein Ereignis mit dem logischen Wert TRUE erzeugt.

Beispiel für ein Hintergrundpanorama:

```
Background {
     skyAngle [ 1.57 ]
     skyColor [ 0 0 1, 0.4 0.4 1 ]
     groundAngle [ 1.57 ]
     groundColor [ .8 .8 .8, .01 .025 .001 ]
     frontUrl "panorama1.jpg"
     backUrl  "panorama2.jpg"
     leftUrl  "panorama3.jpg"
     rightUrl "panorama4.jpg"
}
```

7.7.2 Level of Detail Node

Der Level of Detail Node dient der Optimierung des Darstellungsaufwandes. Ein Objekt muss nur dann exakt wieder gegeben werden, wenn sich der Betrachter in der Nähe des

Objektes befindet. Ist das Objekt jedoch weit vom Betrachter entfernt, so genügt eine reduzierte Darstellung. Diese reduzierte Darstellung wird als "Level of Detail" bezeichnet.

Beim Level of Detail Node werden abstandsabhängig unterschiedlich detaillierte Objektversionen festgelegt. Je Größer der Abstand ist, desto rudimentärer wird das Objekt dargestellt.

Die Syntax für den Level of Detail Node lautet:

Syntax:

```
LOD {
   level    [ ]      #  exposedField    MFNode
   center   0 0 0    #  field           SFVec3f
   range    [ ]      #  field           MFFloat
}
```

Die Felder haben folgende Bedeutung:

- level
  Das Feld level enthält die Child Nodes in der Reihenfolge vom präzisesten zum primitivsten. Es wird ein Child Node mehr benötigt, als Werte im Feld range stehen.
- center
  Das Feld center gibt das Zentrum des lokalen Koordinatensystems an. Die Voreinstellung ist der Ursprung des lokalen Koordinatensystems.
- range
  Das Feld range gibt die Liste der Entfernungen zwischen Position des Betrachters und Zentrum des LOD Nodes. Die Entfernungen müssen in aufsteigender Folge angegeben sein.

Beispiel für eine LOD-Anwendung:

```
Group {
    children [
        LOD {
            range [ 20, 40 ]
            level [

                Group {                  # Level 1  nah
                    children [
                        Transform{ ... }
                         ...
                    ]
                 }

                Group {                  # Level 2  halbfern
                    children [
                        Transform{ ... }
                         ...
                    ]
                 }

                Group {                  # Level 3  fern
                    children [
                        Transform{ ... }
                         ...
```

```
                        ]
                    }
                ]
            }
        ]
    }
```

7.7.3 Switch Node

Mit Hilfe des Switch Nodes kann aus mehreren vorhandenen Child Nodes durch Auswahl jeweils einer oder gar keiner wiedergegeben werden. Der Switch Node kann beispielsweise für Szenenwechsel verwendet werden.
Die Syntax für den Switch Node lautet:

Syntax:

```
Switch {
    choice        [ ]    #  exposedField    MFNode
    whichChoice    -1    #  exposedField    SFInt32
}
```

Die Felder haben folgende Bedeutung:

- choice
  Einer von mehreren vorhandenen Child Nodes kann jeweils dargestellt werden. Die Auswahl erfolgt durch das Feld whichChoice.
- whichChoice
  Das Feld whichChoice gibt den Index des darzustellenden Child Nodes an. Ist die Zahl kleiner Null, so erfolgt keine Wiedergabe.

7.7.4 Billboard Node

Ein Billboard Node ist einer Gruppenknoten der sein lokales Koordinatensystem permanent dreht, so dass es auf den Betrachter ausgerichtet ist. Billboard Nodes eigen sich für Texturen. Wird eine Textur auf eine Ebene des lokalen Koordinatensystems des Billboard Nodes gelegt, so ist diese Textur stets dem Betrachter zugewandt dargestellt.

Die Syntax für den Billboard Node lautet:

Syntax:

```
Billboard {
   addChildren                #  eventIn         MFNode
   removeChildren             #  eventIn         MFNode
   axisOfRotation    0  1  0  #  exposedField    SFVec3f
   children         [ ]       #  exposedField    MFNode
   bboxCenter        0  0  0  #  field           SFVec3f
   bboxSize         -1 -1 -1  #  field           SFVec3f
}
```

Die Felder haben folgende Bedeutung:

- addChildren
  Das Feld addChildren dient zum ereignisgesteuerten Eintragen eines weiteren Child Nodes.
- removeChildren
  Das Feld removeChildren dient zum ereignisgesteuerten Löschen eines Child Nodes.
- children
  Das Feld children nimmt alle Child Nodes auf.
- axisOfRotation
  Im lokalen Koordinatensystem wird die Achse angegeben, um die bei der Ausrichtung gedreht werden soll. Ein Spezialfall stellt der Wert 0 0 0 dar. In diesem Fall wird stets auf den Betrachter ausgerichtet.
- bboxCenter
  Das Feld bboxCenter gibt das Zentrum der Bounding-Box an.
- bboxSize
  Das Feld bboxSize gibt die Maße der Bounding-Box an.

Der Billboard Node kann beispielsweise verwendet werden, um Bäume mit einem realistischen Aussehen ohne großen Aufwand zu realisieren.

Beispiel für eine Darstellung eines Baumes:

```
DEF Baum Billboard {                        # Billboard - Anfang
    axisOfRotation 0 1 0                    # Drehung um y-Achse
    children [
        Shape {
            appearance Appearance {
                texture ImageTexture {
                    url "tree.gif"          # GIF-Bild des Baumes
                    repeatS FALSE  repeatT FALSE
                }
            }
            geometry Box { size 1 1 0 }     # Fläche in der xy-Ebene
        }

   ]
}                                           # Billboard - Ende
```

7.7.5 Inline Node

Der Inline Node ermöglicht das Laden einer anderen VRML-Datei. Die Datei kann sich auf irgendeinem Server im Internet befinden. Der Aufbau der 3D-Szene wird an der VRML-Browser übertragen und die VRML-Datei ausgeführt.
Die Syntax für den Inline Node lautet:

Syntax:

```
Inline {
   url          [ ]        # exposedField    MFString
   bboxCenter    0  0  0   # field           SFVec3f
   bboxSize     -1 -1 -1   # field           SFVec3f
}
```

Die Felder haben folgende Bedeutung:

- url
  Das Feld url enthält eine Liste von URLs, die der Reihenfolge nach verwendet werden können. Die erste gefundene VRML-Datei wird ausgeführt.
- bboxCenter
  Das Feld bboxCenter gibt das Zentrum der Bounding-Box an.
- bboxSize
  Das Feld bboxSize gibt die Maße der Bounding-Box an.

7.7.6 Anchor Node

Der Anchor Node ermöglicht einen Hyperlink auf Dateien von Servern im Internet.
Die Syntax für den Anchor Node lautet:

Syntax:

```
Anchor {
   addChildren                 #  eventIn       MFNode
   removeChildren              #  eventIn       MFNode
   children          [ ]       #  exposedField  MFNode
   description       " "       #  exposedField  SFString
   parameter         [ ]       #  exposedField  MFString
   url               [ ]       #  exposedField  MFString
   bboxCenter         0  0  0  #  field         SFVec3f
   bboxSize          -1 -1 -1  #  field         SFVec3f
}
```

Die Felder haben folgende Bedeutung:

- addChildren
  Das Feld addChildren dient zum ereignisgesteuerten Eintragen eines weiteren Child Nodes.
- removeChildren
  Das Feld removeChildren dient zum ereignisgesteuerten Löschen eines Child Nodes.
- children
  Das Feld children nimmt alle Child Nodes auf.
- description
  Die Zeichenfolge des Feldes description wird vom Browser angezeigt.
- parameter
  Die hier angegebene Zeichenfolge kann zur Interpretation durch das wiedergebende Programm (Plug-In) verwendet werden.
- url
  Das Feld url enthält eine Liste von URLs, die der Reihenfolge nach verwendet werden können. Die erste gefundene Datei wird ausgeführt.
- bboxCenter
  Das Feld bboxCenter gibt das Zentrum der Bounding-Box an.
- bboxSize
  Das Feld bboxSize gibt die Maße der Bounding-Box an.

Die Einbindung einer VRML-Datei in ein HTML-Dokument erfolgt über das $< embed > -Tag$. In das HTML-Dokument muss folgende Zeile eingefügt werden:

```
<embed src="http://www.anywhere.com/examples/world.wrl"
height=200 width=250>
```

7.8 Kontrolle des VRML-Viewers

Zur Kontrolle des VRML-Viewers gibt es drei Nodes.

- Viewpoint Node
- NavigationInfo Node
- WorldInfo Node

7.8.1 Viewpoint Node

Der Viewpoint Node ermöglicht Voreinstellung für die Betrachtung der 3D-Szene. Position, Blickrichtung sowie Blickwinkel des Betrachters können voreingestellt werden.

Die Syntax für den Viewpoint Node lautet:

Syntax:

```
Viewpoint {
    set_bind                # eventIn         SFBool
    fieldOfView    0.785398 # exposedField    SFFloat
    jump           TRUE     # exposedField    SFBool
    orientation    0 0 1 0  # exposedField    SFRotation
    position       0 0 10   # exposedField    SFVec3f
    description    ""       # field           SFString
    bindTime                # eventOut        SFTime
    isBound                 # eventOut        SFBool
}
```

- set_bind
  Ein empfangenes Ereignis mit dem logischen Wert TRUE bindet den Node, er kommt auf den Stack und wird aktiv. Ein Ereignis mit dem logischen Wert FALSE nimmt den Node vom Stack. Damit können verschiedene Hintergrundszenarien aktiviert werden.
- fieldOfView
  Das Feld fieldOfView gibt die Größe des Blickwinkels an.
- jump
  Beim logischen Wert TRUE werden die Einstellungen des neuen Viewpoint Nodes übernommen.
- orientation
  Das Feld gibt die Voreinstellung für die Blickrichtung an.
- position
  Das Feld gibt die Position des Betrachters im lokalen Koordinatensystem an.
- description
  Die hier angegebene Zeichenfolge enthält eine Beschreibung des Viewpoint Nodes.
- bindTime
  Das Feld bindTime liefert die aktuelle Zeit, so lange der Node gebunden ist.
- isBound
  Beim Binden des Nodes wird ein Ereignis mit dem logischen Wert TRUE erzeugt.

7.8.2 NavigationInfo Node

Der NavigationInfo Node ermöglicht weitere Voreinstellung für den Betrachter.

Die Syntax für den NavigationInfo Node lautet:

Syntax:

```
NavigationInfo {
    set_bind                        #  eventIn         SFBool
    avatarSize       [0.25, 1.6, 0.75] #  exposedField    MFFloat
    headlight        TRUE           #  exposedField    SFBool
    speed            1.0            #  exposedField    SFFloat
    type             ["WALK", "ANY"]   #  exposedField    MFString
    visibilityLimit  0.0            #  exposedField    SFFloat
    isBound                         #  eventOut        SFBool
}
```

- set_bind
  Ein empfangenes Ereignis mit dem logischen Wert TRUE bindet den Node, er kommt auf den Stack und wird aktiv. Ein Ereignis mit dem logischen Wert FALSE nimmt den Node vom Stack. Damit können verschiedene Hintergrundszenarien aktiviert werden.
- avatarSize
  Das Feld avatarSize beschreibt die Maße des virtuellen Betrachter. Der erste Wert gibt den Mindestabstand für Kollisionen mit anderen geometrischen Objekten an. Der zweite Wert beschreibt die Höhe des Avatar. Diese Größe entspricht der Höhe der Kamera über dem Boden des lokalen Koordinatensystems des Avatar. Der dritte Wert gibt die Höhe des größten Hindernisses an, die der Avatar noch übersteigen kann.
- headlight
  Beim logischen Wert TRUE wird eine Lichtquelle (DirectionalLight) in der Kopfhöhe des Avatar angenommen.
- speed
  Das Feld speed gibt die Navigationsgeschwindigkeit an. Er wird in Einheiten pro Sekunde angegeben.
- type
  Das Feld type gibt die Art der Bewegung des Avatars an. Mögliche Typen sind: WALK, FLY, NONE, EXAMINE.
- visibilityLimit
  Das Feld visibilityLimit gibt die Sichtweite des Avatars an.
- isBound
  Beim Binden des Nodes wird ein Ereignis mit dem logischen Wert TRUE erzeugt.

7.8.3 WorldInfo Node

Der WorldInfo Node hat keinen Einfluss auf die 3D-Szene. Der Node dient lediglich zur Information.

Die Syntax für den WorldInfo Node lautet:

Syntax:

```
WorldInfo {
 info    [ ]   #  field MFString
 title   " "   #  field SFString
```

```
}
```

- info
  Die hier angegebene Zeichenfolge enthält beliebige Informationen über die 3D-Szene, wie z.B. der Name des Autors, Copyright, Beschreibung der Szene etc.
- title
  Die hier angegebene Zeichenfolge wird vom VRML-Browser angezeigt.

7.9 Einbindung von Scripts und Programmiersprachen

Zur Animation und zur Interaktion einer 3D-Szene können Scripts und Programme verwendet werden. VRML bietet für diese Möglichkeit den Script Node.

Liegt ein Script oder ein Programm als externe Datei auf einem Server vor, so muss mittels URL im Feld url darauf zugegriffen werden.
Ein Script kann aber auch Bestandteil des Script Nodes sein. Es muss dann ebenfalls im Feld url angegeben werden. Dabei wird das Schlüsselwort "javascript:" bei JavaScript oder "javabc:" bei Java-Programmen angegeben.

Die Syntax für den Script Node lautet:

Syntax:

```
Script {
    url           [ ]        #  exposedField  MFString
    directOutput  FALSE      #  field         SFBool
    mustEvaluate  FALSE      #  field         SFBool
    #--- And any number of: ---------------------------
    eventIn                  #  eventName     eventType
    field     initialValue   #  fieldName     fieldType
    eventOut                 #  eventName     eventType
}
```

Die Felder haben folgende Bedeutung:

- url
  Im Feld url können mehrere URLs angegeben werden, die auf eine Datei mit einem Script oder Programm verweisen. Die erste gefundene Datei wird ausgeführt. Für inline angegebene JavaScripts wurde eigens das Protokoll "javascript:" definiert und für JavaBytecode das Protokoll "javabc:".
- directOutput
  Wird das Feld directOutput auf den logischen Wert FALSE gesetzt, so ist nur ein Austausch von Ereignissen mit anderen Nodes über die eventIn/Out-Definitionen im Script Node und die ROUTE-Anweisungen möglich. Wird der logische Wert auf TRUE gesetzt, so kann ein Script oder ein Programm direkt auf ein exposedFields zugreifen und ein Ereignisse für die Child Nodes erzeugen.
- mustEvaluate
  Das Verschicken von Ereignissen durch den VRML-Browser kann verzögert stattfinden. Wird das Feld mustEvaluate auf den logischen Wert TRUE gesetzt, so muss das Ereignis sofort verschickt werden.
- eventIn
  Definiert ein Input Event.
- field
  Definiert ein Field
- eventOut
  Definiert ein Output Event.

Programm 21 (Berechnung einer Spiralbewegung)

```
DEF Spiral Script {
   url ‘‘javascript:
         // Move a shape in a spiral path
```

```
         function set_fraction( fraction, eventTime) {
            value_change[0] = radius * Math.sin(turns * fraction * 6.28);
            value_change[1] = fraction;
            value_change[2] = radius * Math.cos(turns * fraction * 6.28);
   }''
field     SFFloat    radius 1.0
field     SFFloat    turns  1.0
eventIn   SFFloat    set_fraction
eventOut  SVec3f     value_changed
}
```

Der resultierende Parameter "value_changed" kann jetzt via ROUTE für eine Bewegung verwendet werden.

Literaturverzeichnis

[1] Andrea L. Ames, David R. Nadeau, und John L. Moreland.
VRML 2.0 Sourcebook.
John Wiles & Sons, Inc., 1. Ausgabe, 1997.
Tutorial VRML 2.0, CD,
ISBN 0-471-16507-7.

[2] Eckhard Amman.
Programmierung animierter Welten
Java, JavaScript and VRML.
Thomson Publishing, 1. Ausgabe, 1997.
Beispiel VRML mit Java,
ISBN 3-8266-0329-x.

[3] Ian Ashdown.
Radiosity A Programmer's Perspective.
John Wiles & Sons, Inc., 1. Ausgabe, 1994.
weiterführendes Lehrbuch,
Diskette mit C-Programmen,
ISBN 0-471-30488-3.

[4] Steve Aukstakalnis und David Blatner.
The Silicon Mirage, The Art and Science of Virtual Reality.
Peachpit Press, Inc., 1992.
leicht lesbar, praktisch,
Anwendungen, Ein- and Ausgabegeräte für VR-Systeme.

[5] Steve Aukstakalnis und David Blatner.
Cyberspace, Die Entdeckung künstlicher Welten.
vgs Verlag, 1994.
leicht lesbar, praktisch,
Anwendungen, Ein- and Ausgabegeräte für VR-Systeme,
deutsche Übersetzung von Silicon Mirage,
ISBN 3-802025-1275-8.

[6] Durand R. Begault.
3-D Sound for Virtual Reality and Multimedia.
Academic Press, Inc., 1. Ausgabe, 1994.
einführendes Lehrbuch, 3D-Akustik,
ISBN 0-12-084735-3.

[7] Sven Bormann.
Virtuelle Realität.
Addison-Wesley Publishing Company, 1. Ausgabe, 1994.
einführendes Lehrbuch, virtuelle Realität,
ISBN 3-89319-707-9.

[8] Frank Eckgold.
Virtual Reality.
Vieweg-Verlag, 1. Ausgabe, 1995.
einführendes Lehrbuch, Computergraphik,
Diskette mit C-Programmen,
ISBN 3-528-05398-4.

[9] José Encarnação, Wolfgang Straßer, und Reinhard Klein.
Graphische Datenverarbeitung, Band 1.
Oldenbourg Verlag, 4. aktuallisierte Ausgabe, 1996.
sehr gutes Lehrbuch, Computergraphik,
ISBN 3-486-23223-1.

[10] José Encarnação, Wolfgang Straßer, und Reinhard Klein.
Graphische Datenverarbeitung, Band 2.
Oldenbourg Verlag, 4. aktuallisierte Ausgabe, 1997.
sehr gutes Lehrbuch, Computergraphik,
ISBN 3-486-23469-2.

[11] Gerald Farin.
Kurven und Flächen im Computer Aided Geometric Design.
Vieweg Verlag, 2. Ausgabe, 1994.
weiterführendes Lehrbuch, Computer Graphik,
ISBN 3-528-16542-1.

[12] James D. Foley, Andries van Dam, Steve K. Feiner, und John F. Hughes.
Computer Graphics, Principles And Practice.
Addison-Wesley Publishing Company, 2. Ausgabe, 1993.
leicht lesbar, umfassendes Lehrbuch,
ISBN 0-201-12110-7.

[13] James D. Foley, Andries van Dam, Steve K. Feiner, John F. Hughes, und Richard L. Philips.
Grundlagen der Computergraphik.
Addison-Wesley Publishing Company, 1. Ausgabe, 1994.
leicht lesbar, umfassendes Lehrbuch,
deutsche Übersetzung von Computer Graphics,
jedoch nicht so umfangreich,
ISBN 0-201-60921-5.

[14] Thomas Haenselmann.
Raytracing.
Addison-Wesley Publishing Company, 1. Ausgabe, 1996.
einführendes Lehrbuch, Computergraphik, CD,
ISBN 3-89319-922-5.

[15] Jed Hartman und Josie Wernecke.
The VRML 2.0 Handbook.
Addison-Wesley Publishing Company, 1. Ausgabe, 1996.
einführendes Lehrbuch, VRML 2.0,
Autoren sind bei Silicon Graphics, Inc.,
ISBN 0-201-47944-3.

[16] Hans-Lothar Hase.
Dynamische virtuelle Welten mit VRML 2.0.
dpunkt Verlag, 1. Ausgabe, 1997.
einführendes Lehrbuch, VRML 2.0, CD,
ISBN 3-920993-63-2.

[17] Roy S. Kalawsky.
Virtual Reality and virtual Environments.
Addison-Wesley Publishing Company, 1. Ausgabe, 1993.
Reprinted 1994,
sehr gutes Lehrbuch, virtuelle Realität,
ISBN 0-201-63171-7.

[18] Jörg Kloss, Robert Rockwell, Kornél Szabó, und Martin Duchrow.
VRML97 Der neue Standard für interaktive 3D-Welten im World Wide Web.
Addison-Wesley Publishing Company, 1. Ausgabe, 1998.
leicht lesbar, einführendes Lehrbuch, CD,
ISBN 3-8273-1187-x.

[19] Michael Kobzan und Murat Bezel.
3D-REndering and Animation.
Markt & Technik Buch- and Software-Verlag, 1. Ausgabe, 1996.
einführendes Lehrbuch, Computergraphik, CD,
ISBN 3-87791-745-3.

[20] Herbert Kopp.
Graphische Datenverarbeitung.
Hanser VErlag, 1. Ausgabe, 1989.
gutes Lehrbuch, Computergraphik,
Vorlesung an der FH Regensburg,
ISBN 3-446-15645-3.

[21] Wolfram Luther und Martin Ohsmann.
Mathematische Grundlagen der Computergraphik.
Vieweg-Verlag, 2. verbesserte auflage Ausgabe, 1989.
einführendes Lehrbuch, Computergraphik,
ISBN 3-528-16302-x.

[22] Stephen N. Matsuba und Bernie Roehl.
VRML Das Kompendium.
Markt & Technik Buch- and Software-Verlag, 1. Ausgabe, 1997.
allgemeine Übersicht, VRML 1.0, CD,
ISBN 3-8272-5157-5.

[23] Mark Pesce.
VRML Cyberspace-Welten erkunden and erschaffen.
Hanser Verlag, 1. Ausgabe, 1997.
einführendes Lehrbuch VRML 1.0,
ISBN 3-446-18889-4.

[24] Les Piegl und Wayne Tiller.
The NURBS Book.
Springer-Verlag, 1. Ausgabe, 1997.
Reprinted 1996,
weiterführendes Lehrbuch, Computergraphik,
ISBN 3-540-61545-8.

[25] Josef Pöpsel, Ute Claussen, Rolf-Dieter Klein, und Jürgen Plate.
Computergraphik Algorithem and Implementierung.
Springer-Verlag, 1. Ausgabe, 1994.
einführendes Lehrbuch, Computergraphik,
Diskette mit Pascal-Programmen,
ISBN 3-540-57248-1.

[26] Franco P. Preparata und Michael I. Shamos.
Computational Geometry.
Springer-Verlag, 1. Ausgabe, 1988.
weiterführendes Lehrbuch, Computergraphik,
ISBN 3-540-96131-1.

[27] Thomas Rauber.
Algorithmen in der Computergraphik.
Teubner Verlag, 1. Ausgabe, 1993.
leicht verständliches Lehrbuch, Computergraphik,
ISBN 3-519-02127-7.

[28] Bernie Roehl, Justin Couch, Cindy Reed-Ballreich, Tim Rohaly, und Geoff Brown.
Late Night VRML 2.0 with Java.
Ziff-Davis Press, 1. Ausgabe, 1997.
Beispiele VRML 2.0 mit Java, CD,
ISBN 1-56276-504-3.

[29] Nadia Magnenat Thalmann und Daniel Thalmann.
Interactive Computer Animation.
Prentice Hall, 1. Ausgabe, 1996.
weiterführendes Lehrbuch, Computergraphik,
ISBN 0-13-518309-x.

[30] John R. Vacca.
VRML.
SYBEX-Verlag, 1. Ausgabe, 1997.
einführendes Lehrbuch, VRML 1.0, CD,
ISBN 3-8155-2015-0.

[31] Johne Vince.
3-D Computer Animation.
Addison-Wesley Publishing Company, 1. Ausgabe, 1992.
Reprinted 1993,
sehr gutes Lehrbuch, Computergraphik,
ISBN 0-201-62756-6.

[32] Johne Vince.
Virtual Reality Systems.
Addison-Wesley Publishing Company, 1. Ausgabe, 1995.
sehr gutes Lehrbuch, virtuelle Realität,
ISBN 0-201-87687-6.

[33] Christopher D. Watkins und Larry Sharp.
Programming in 3 Dimensions.
M & T Publishing, Inc., 1. Ausgabe, 1992.
sehr gutes Lehrbuch, Computergraphik,
Diskette mit C-Programmen,
ISBN 0-13-726613-8.

[34] Alan Watt.
3D Computer Graphics.
Addison-Wesley Publishing Company, 2. Ausgabe, 1993.
sehr gutes Lehrbuch, Computergraphik,
ISBN 0-201-63186-5.

[35] Alan Watt und Mark Watt.
Advanced Animation and Rendering Techniques.

Addison-Wesley Publishing Company, 1. Ausgabe, 1992.
Reprinted 1993,
sehr gutes Lehrbuch, Computergraphik,
ISBN 0-201-54412-1.

[36] Ron Wodaski und Donna Brown.
Virtual Reality für Insider.
SAMS, 1. Ausgabe, 1995.
allgemeine Übersicht, Virtuelle Realität, CD,
ISBN 3-87791-652-x.

Index