**Hochschule für Technik und Wirtschaft Dresden (FH)**
**Fachbereich Informatik/Mathematik**

**Diplomarbeit**

im Studiengang Medieninformatik

Thema:

# Prototypische Entwicklung eines 3-dimensionalen Informations- und Navigationssystems mit Director Shockwave3D

eingereicht von: Ralf Halgasch

eingereicht am: 10.03.2003

Betreuerin: Prof. Dr. Teresa Merino

# Inhaltsverzeichnis

Einleitung....5
1. Einführende Bestandsaufnahme....7
1.1. Vom Anfang des Webs bis zum Aufbruch in die dritte Dimension....7
1.1.1. Ein kurzer Blick in die Geschichte des Webs....7
1.1.2. Die dritte Dimension erobert das Internet....8
1.1.3. Situationsanalyse der Internetnutzung....9
1.2. Anwendungsbereiche....11
1.2.1. E-Commerce....11
1.2.2. E-Learning....14
1.2.3. Entertainment....15
1.2.4. Entwicklung....16
1.2.5. Forschung / Simulation....17
1.3. Kategorisierung von 3D-Webtechnologien....17
1.3.1. Standardbasierte Technologien....17
1.3.2. Proprietäre Plug-in-Lösungen....21
1.3.3. Appletbasierte Viewer....24
1.3.4. VRML Viewer....27
1.3.5. Einschätzung des aktuellen Standes der Web3D-Technologien....28
1.4. Szenenbeschreibung....28
2. Grundlagen Director und 3D-Darstellung....31
2.1. Einführung in die Director-Semantik....31
2.1.1. Die Bühne....32
2.1.2. Die Besetzung....32
2.1.3. Das Drehbuch....33
2.1.4. Der Eigenschaftsinspektor....33
2.1.5. Das Skriptfenster....34
2.1.6. Das Nachrichtenfenster....34
2.2. Lingo – die Programmiersprache von Director....35
2.3. Einführung Vektormathematik....38
2.3.1. Begriffsdefinitionen....38
2.3.2. Vektoren....38
2.3.3. Matrizen....39
2.4. Einführung in Director3D....41
2.4.1. Shockwave3D-Darsteller, Koordinatensystem und Orientierung....42
2.4.2. Definition geometrischer Primitive....46
2.4.3. Oberflächeneigenschaften, Shader und Texturen....47
2.4.4. Transformation von Objekten....48
2.4.5. Hierarchische Strukturen, Parent-Child-Beziehungen....48

2.4.6. Animation der hierarchischen Anordnung ..... 49
2.4.7. Ein unverzichtbares Tool für den Überblick – 3DPI ..... 49
3. Modellierung der Szene ..... 51
3.1. Das Modellierungswerkzeug 3D Studio MAX ..... 51
3.1.1. Der Export einer 3D-Szene ..... 52
3.1.2. Detailgenauigkeit und Geometrieauflösung ..... 52
3.1.3. Beleuchtung und Schattierung ..... 53
3.1.4. Simulation eines Spotlichtkegels ..... 53
3.1.5. Problem der Shockwave-Tiefensortierung ..... 54
3.2. Modellierung in Shockwave3D ..... 55
3.2.1. Texturierung der Fenster ..... 55
3.2.2. Das Kino-Informationsobjekt ..... 56
4. Die Kamerasteuerung ..... 62
4.1. Anforderungen ..... 62
4.2. Machbarkeitsanalyse eines Kollisionserkennungsprinzips ..... 63
4.2.1. Einführung in das Modifierkonzept ..... 63
4.2.2. Der Collision-Modifier ..... 64
4.2.3. Funktionsprinzip der Kameraboundingsphere ..... 64
4.2.4 Auswirkungen der Modi des Collision-Modifiers ..... 65
4.3. Endgültige Realisierung der Kamerasteuerung ..... 65
4.3.1. Die notwendigen Flags und Eigenschaftsvariablen ..... 66
4.3.2. Triggerinitialisierung/ Ereignisgenerierung ..... 66
4.3.3. Die Kamerafahrt ..... 67
4.3.4. Die Steuerung der Kameratransformation ..... 69
4.3.5. Erkennung, Auflösung und Ausnahmen bei Kollisionen ..... 71
5. Übermittlung und Darstellung dynamischer Inhalte ..... 73
5.1. Contentübertragung per XML und asynchrones Scripting ..... 73
5.2. Dynamische Visualisierung textbasierter Inhalte ..... 76
5.2.1. Statischen Text im 3D Studio MAX generieren ..... 76
5.2.2. 3D-Text zur Laufzeit erzeugen ..... 77
5.2.3. Informationsvermittlung unter Nutzung von Texturen ..... 78
6. Darstellungsqualität und Performance ..... 84
6.1. Kantenbildung an ebenen Flächen ..... 85
6.2. Renderperformance in Abhängigkeit der Geometrieauflösung ..... 90
6.3. Verbesserte Darstellungsqualität mittels Antialiasing ..... 91
6.4. Renderperformance mit und ohne Antialiasing ..... 92
6.5. Renderperformance in Abhängigkeit des Abspielmodus ..... 92
6.6. Zusammenfassung ..... 94

7. Schlussbetrachtungen....95
7.1. Was bringt Director MX?....95
7.2. Fazit....97
Anhang....98
Listings....99
Abbildungsverzeichnis....110
Tabellenverzeichnis....113
Abkürzungsverzeichnis....114
Literaturverzeichnis....115
Verzeichnis der Internet- Links....116
Glossar....121
Selbständigkeitserklärung....125

# Einleitung

Diese Diplomarbeit hat die Zielstellung, die Möglichkeiten der 3D-Darstellung im Internet aufzuzeigen. Zur Einführung in die Thematik erfolgt ein kurzer Abriss der historischen Entwicklung des Internets, eine Erläuterung der anfänglichen Entwicklung standardbasierter 3D-Technologien und eine ausschnittsweise Darstellung der momentanen Internetnutzung. Anschließend werden die verschiedenen Einsatzmöglichkeiten der Web3D-Technologien und ausgewählte, relevante Web3D-Lösungen vorgestellt. Um die Einführung in die Problematik abzurunden, folgen zum Ende des theoretischen Teils Einschätzungen bezüglich aktueller Technologiekonzepte und deren Vor- und Nachteile. Im Hinblick auf das in den nachfolgenden Kapiteln vorgestellte Director3D-Konzept soll auch das oft verwendete Organisationsprinzips von 3D-Szenen (der Szenegraph) dargestellt werden.

Im Rahmen des praktischen Teils der Diplomarbeit soll unter Verwendung von Director Shockwave3D ein Prototyp für ein 3-dimensionales Informations- und Navigationssystem entwickelt werden. Dabei geht es um eine möglichst vollständige Analyse der verwendeten Technologie, ihrer Umsetzung und der Probleme, die im Zusammenhang mit einer Onlinepräsentation von 3D-Inhalten mit Director Shockwave entstehen. Ziel der Diplomarbeit war es nicht, ein möglichst detailgetreues Abbild des darzustellenden Objektes mit all seinen Räumlichkeiten und Angeboten zu realisieren.

Damit auch Leser, die mit dem Autorenwerkzeug Director keine oder nur geringe Erfahrungen haben, den Ausführungen folgen können, erfolgt im Kapitel 2 eine Vorstellung des Director-Konzeptes. Dazu gehört eine kurze Einführung in vektormathematische Grundlagen und eine Vorstellung der 3D-Technologie von Director.

Im Vorfeld der Diplomarbeit wurde in Zusammenarbeit mit der Wildstyle Network GmbH lange überlegt, welche Aufgabenstellung hinsichtlich einer gewünschten 3D-Realisierung mit Director diplomgerecht formuliert werden könnte. Zu diesem Zeitpunkt erschien uns die Visualisierung des Entertainmentkomplexes Wandelhof Schwarzheide als lohnenswertes und anspruchvolles Ziel. Im Kapitel *3. Modellierung der Szene* und im Abschnitt *7.2 Fazit* wird die Wahl des zu realisierenden Objektes dann kritischer hinterfragt und ausführlicher behandelt.
Für die Neukonzeption der Webpräsentation des Wandelhofes sollte die Möglichkeit analysiert werden, dem Besucher der Seite einen Mehrwert dahingehend anzubieten, dass er online den virtuellen Freizeitkomplex begehen und gewünschte Informationen holen kann.

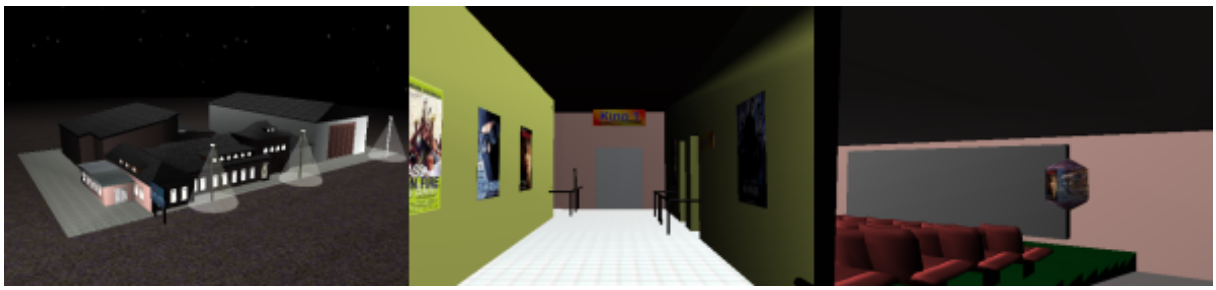


Abb. 0.7.1 3D-Ansichten des Entertainmentkomplexes

Neben der Visualisierung der Außenansicht und der Inneneinrichtung war es geplant, Informationen zum aktuellen Kino- und Diskoprogramm sowie zu Veranstaltungen anzubieten. Diese dynamischen Informationen sollten so in die Anwendung integriert werden, dass sie jederzeit geändert werden können, ohne den Shockwavefilm anpassen zu müssen.

Der Freizeit- und Entertainmentkomplex Wandelhof repräsentiert eine gewachsene Struktur von mehreren Gebäuden, deren gemeinsames Merkmal ihre völlig unterschiedliche architektonische Gestaltung ist. Es handelt sich also nicht um einen der heute typischen, durchgestylten, am Reißbrett entworfenen Spaßtempel. Der Komplex bietet eine Großraumdiskothek, ein Tanzcafé, einen großen und drei kleine Kinosäle, eine Weinbar, eine Kellerbar mit einer kleineren Disko, eine Bowlingbahn und Möglichkeiten für Billard und Dart.

Da die Modellierung der gesamten Räumlichkeiten den Rahmen der Diplomarbeit sprengen würde und der Focus der Arbeit auch nicht auf einer vollständigen Replikation des Originals lag, wurden nur einige Teile des Komplexes (Außenansicht, Kinogang, großer Kinosaal) erstellt, die dann die Basis für die technologische Betrachtung der Shockwave3D-Technologie bildeten.

Neben der 3D-Darstellung des Wandelhofkomplexes wurde eine konventionelle 2D-Präsentation entworfen, die vom Nutzer für weitergehende Textinformationen, für Präsentation einer Diashow, als Navigationshilfe sowie als Feedback-Möglichkeit genutzt werden kann.

Abb. 0.7.2 2D-Navigationsplan, Diashow

Diese Präsentation wurde nur teilweise realisiert, hat thematisch keine weitere Bedeutung für die Diplomarbeit und wurde nur der Vollständigkeit halber hier aufgeführt. In den nachfolgenden Kapiteln wird sie höchstens punktuell erwähnt.

Hinsichtlich der Systematik der Fachtermini wird innerhalb der Diplomarbeit die deutsche Rechtschreibung verwendet. Ausnahme hiervon bilden Markennamen und die Codebeispiele, die der originalen Lingosyntax entsprechen.

# 1. Einführende Bestandsaufnahme

## 1.1. Vom Anfang des Webs bis zum Aufbruch in die dritte Dimension

Es gibt wahrscheinlich neben den Erfindungen des Telefons und Fernsehens keine Entwicklung, die das Informationsverhalten der Gesellschaft in den Industrieländern so verändert, wie dies mit dem rasanten Siegeszug des Internets der Fall ist.
Der bekannteste Vertreter des Internets, das World Wide Web hat dazu sicherlich den entscheidenden Anteil beigetragen.

### 1.1.1. Ein kurzer Blick in die Geschichte des Webs

Der Name World Wide Web, kurz WWW oder Web, entstand 1990 am Kernforschungszentrum CERN [www01] in Genf aus einem Projekt zur ortsunabhängigen Darstellung von wissenschaftlichen Daten auf vernetzten Computern. Im gleichen Jahr war der erste Web-Server unter info.cern.ch erreichbar. Der erste benutzbare, im Textmodus arbeitende Browser wurde 1991 veröffentlicht. Ein Jahr später existierten bereits die ersten Browser (Erwise und Viola), die eine grafische Oberfläche verwendeten. Zum Durchbruch des Web 1993 verhalf die Einführung des Web-Browsers Mosaic, der von Marc Andreessen und Eric Bina am NCSA [www02] der University of Illinois entwickelt wurde. Mit der Freigabe der WWW-Technologie im selben Jahr seitens des CERN konnte diese Technik ohne irgendwelche Patentrechte oder Copyrightgebühren frei verwendet werden.

HTML, die Seitenbeschreibungssprache des Webs wurde als Anwendung von SGML (Standard Generalized Markup Language) normiert. Marc Andreesen, einer der Entwickler des Mosaic-Browsers gründete 1994 die Firma Netscape [www03]. Der von der Firma weiter entwickelte Mosaic-Browser, der sich als Netscape Navigator bzw. später als Netscape Communicator etablierte, erreichte schnell einen „Quasi-Status“. Mit diesem Browser führte Netscape weitere Funktionalitäten wie farbige Schriften und Hintergründe, Tabellen, Framesets und die Plug-in-Technologie ein. Die fast Monopolstellung, die der Netscape-Browser 1996 mit einem Marktanteil von ca. 90% inne hatte, wurde 1997 von Microsoft [www04] mit dem Internet Explorer 4 durchbrochen. Mit der Integrierung des Internet Explorers in das Betriebssystem gelang es Microsoft die Marktanteile umzudrehen. Heute besitzt der Internet Explorer mit über 80% Marktanteil die weiteste Verbreitung aller Browsersysteme.

Um den mit dem „Browserkrieg“ einhergehenden willkürlichen, nicht kompatiblen Entwicklungen der Browserhersteller entgegenzuwirken, wurde im Oktober 1994 das World Wide Web Consortium [www05], kurz W3C gegründet. Das W3C sieht seine Aufgabe als Standardisierungs- und Koordinationsgremium für das World Wide Web. „Seine Aufgabe ist es, Webtechnologien zu initiieren, zu entwickeln, ihre Entwicklungen zu koordinieren und zu standardisieren.“ [WEI01] Aktuelle Browser sollen die von der Organisation verabschiedeten Standards so umsetzen, dass jedem Internetnutzer eine einheitliche Darstellung und Funktionalität der von ihm aufgesuchten Internetseiten gewährleistet werden kann.

Als eine der bedeutendsten Entwicklungen der letzten Jahre hat sich, neben der Standardisierung von XML, die Visualisierung von 3D-Inhalten im Web herausgestellt. Die Nutzung von 3D-Inhalten im Web, die daraus resultierenden Vorteile, Anforderungen und Realisierungsstrategien sollen nachfolgend näher analysiert werden.

### 1.1.2. Die dritte Dimension erobert das Internet

Bis zur Einführung der Plug-in-Technologie waren Browser nicht in der Lage, 3D-Objekte und komplexe Szenen, die in einer Webseite eingebunden waren, darzustellen. Mit Hilfe dieser Technologie war es ab 1994 möglich, die Browser durch zusätzliche Bibliotheken zu erweitern und somit neue Funktionalitäten zu realisieren.
Auf der ersten internationalen WWW-Konferenz 1994 in Genf wurde von Mark Pesce ein von ihm und Toni Parisi entwickelter Prototyp einer 3D-Schnittstelle für das Web vorgestellt [www06]. Im Ergebnis dieser Präsentation wurde der Notwendigkeit Rechnung getragen, eine gemeinsame Sprache zur Spezifikation von 3D-Szenenbeschreibungen zu entwickeln. Diese Sprache wurde als Virtual Reality Modelling Language (VRML) definiert und basierte auf dem OpenInventor ACSII File Format von Silicon Graphics, Inc., kurz SGI [www07]. Die Spezifikation von VRML wurde 1995 vorgestellt. Zugleich wurde die VRML Architecture Group, kurz VAG, gegründet, die zukünftige VRML-Spezifikationen festlegen sollte. Es erfolgte die Veröffentlichung des ersten VRML-fähigen Browsers (WebSpace Navigator) durch SGI in Kooperation mit TGS [www08]. Microsoft lizenzierte das Plug-in WorldView der Firma Intervista und erweiterte damit den Internet Explorer um die VRML-Darstellung. Netscape integrierte 1996 den von der Firma Paper aufgekauften VRML-fähigen Browser in deren eigenes Produkt. Mit der Unterstützung der VRML Technik durch die beiden verbreiteten Browser war die Grundlage für eine große Akzeptanz des VRML-Konzeptes gelegt worden.

VRML 1.0 ermöglichte einfache Interaktionen, es fehlte jedoch die Unterstützung von Objektanimationen. Deshalb wurde von der VAG die Definition einer weiterführenden Spezifikation initiiert. Grundlage für die neue VRML 2.0 Spezifikation bildete das von SGI, WorldMaker, Sony [www09] und anderen entwickelte, auf VRML 1.0 basierende Moving Worlds. VRML 2.0 wurde zum ersten Mal 1996 auf der SIGGRAPH vorgestellt. Ende 1997 wurde nach einigen kleinen Änderungen VRML 2.0 in VRML97, nunmehr durch das VRML Consortium, umbenannt und als ISO/IEC Standard veröffentlicht.

Der Nachfolger des VRML97-Formats, anfänglich mit VRML NG bezeichnet, wurde 1999 in X3D (Abkürzung für Extensible 3D) umbenannt und in seiner endgültigen Version auf der SIGGRAPH 2001 öffentlich vorgestellt. Mit dem zunehmenden Interesse an 3-dimensionalen Darstellungen im Web und dem damit einhergehenden Angebot an neuen Formaten wurde das VRML Consortium seinem Namen nicht mehr gerecht. Neben VRML standen mittlerweile mit Java 3D [www10] und dem anstehenden MPEG-4 Format weitere Möglichkeiten für eine 3D-Visualisierung im Web zur Verfügung. Daher erfolgte eine Umbenennung in Web3D Consortium [www11]. Auch der Name, der seit 1997 stattfindenden VRML-Konferenz, wurde geändert – in Web3D Konferenz. Das Web3D Consortium setzt sich aus einer Reihe verschiedener Gruppen, die sich mit den einzelnen Teilaspekten der 3D-Grafik im Web beschäftigen, zusammen. Diese Teilbereiche beziehen sich zum Beispiel auf Themen wie VRML- Streaming, GeoVRML und Web3D-MPEG. Aufgabe dieser gemeinnützigen Organisation ist es, offene Standards für die 3D-Webdarstellung und Übertragung zu entwickeln und zu fördern.

Nach der anfänglich breiten Nutzung des VRML-Formats folgten in den letzten Jahren eine Reihe von Entwicklungen, die nicht mehr auf dieses standardisierte Format oder dessen Nachfolger X3D aufsetzen. Vielmehr versuchen Firmen, wie Adobe [www12], Macromedia [www13] oder Viewpoint [www14], ihre eigenen Formate als „Quasi"-3D-Standard für das Web zu etablieren. Diese auf der Plug-in-Technologie basierenden Systeme werden im Abschnitt *1.3.2. Proprietäre Plug-in-Lösungen* näher untersucht.

### 1.1.3. Situationsanalyse der Internetnutzung

Kennzeichnend für die bisherige Entwicklung war eine fast ausschließliche Fokussierung auf die 2-dimensionale Informationsdarstellung. Mangelnde Performance und damit einhergehende Qualitätseinbußen in der Echtzeitdarstellung von 3D-Objekten, das Fehlen marktdurchdringender Standards und die unzureichende Unterstützung von Web3D-Entwicklungswerkzeugen führten bisher dazu, dass sich weder ein entsprechend großer Markt für die 3D-Visualisierung im Web bilden konnte, noch dass es einen kommerziell bedeutsamen Bedarf an 3D-Web Tools gab.
Doch gerade in dieser Hinsicht erfolgen momentan große Veränderungen. Diese sind zum einen an der stetig anwachsenden Rechen- und Grafikleistung der Consumer-PCs und zum anderen an einer zunehmenden Verbreitung breitbandiger Internetzugänge festzustellen. Daneben haben sich in den letzten Jahren mit OpenGL und DirectX [www15] Standards zur Programmierung der Grafik-Hardware etabliert. Mit diesen Voraussetzungen gewinnt der potentielle Markt für 3D-Webinhalte eine immer größer werdende Bedeutung.

Um die Marktrelevanz für 3D-Webdarstellungen einschätzen zu können, soll nachfolgend eine kurze Zusammenfassung über den aktuellen Stand der Internetnutzung, über die Zugangsperformance und über die Hauptprobleme der Web3D-Darstellung gegeben werden.
Einige Auszüge, die aus einer Internationalen Vergleichstudie zur Informationsgesellschaft des Bundesverbandes Informationswirtschaft, Telekommunikation und Neue Medien e.V. stammen [www16], sollen die Nutzung und die Bedeutung des Internets bezogen auf das Jahr 2002 verdeutlichen.

- Weltweit wird das Internet von über 600 Millionen Menschen genutzt, 100 Millionen mehr als im Jahr zuvor.
- Jeder zweite Deutsche (41 Millionen) wird das Internet im Jahr 2003 nutzen.
- Weltweit laufen bereits 54 Millionen Internet-Zugänge über Breitband-Anschlüsse.
- Deutschland hat weltweit mit die leistungsfähigste Telekommunikations-Infrastruktur (25 Millionen ISDN-Kanäle, 3,2 Millionen DSL-Anschlüsse).
- Bis zum Jahr 2005 sollen sich die DSL-Anschlüsse auf 6,3 Millionen verdoppeln.
- Der Umsatz des elektronischen Handels stieg deutschlandweit auf 87,8 Milliarden € pro Jahr an.

3D-Darstellungen erfordern trotz ihrer verwendeten Kompressionsalgorithmen und der Möglichkeit 3D-Inhalte zu streamen mehr Übertragungsbandbreite als dies bei konventionellen HTML-Webseiten der Fall ist. Mit der fast flächendeckenden Verbreitung schneller Zugangsmöglichkeiten per ISDN oder DSL sind gerade bei privaten Nutzern die Grundlagen für eine breite Marktdurchdringung der Web3D-Technik gegeben.

| | Bit/s | kB/s |
|---|---|---|
| 56k-Modem | 56000 | 6,84 |
| ISDN 1-Kanal | 64000 | 7,81 |
| DSL-768 | 768000 | 93,75 |

Tabelle 1 Ausgewählte, maximal erreichbare Downloadraten

Eine statistische Untersuchung zur Internetnutzung in Europa, die durch Electronic Commerce Info Net [www17] veröffentlicht wurde, zeigte unter anderem auch die verschiedenen Anteile nach dem jeweiligen Nutzertyp auf. Nimmt man den News-Freak und den Klicker unter der Annahme heraus, dass sich für diese Gruppe keine gezielte Web3D-Anwendung ableiten lässt, so ergibt sich für die anderen Nutzergruppen, die mehr als 60% der Befragten umfassen, ein potentieller Markt für 3-dimensionale Darstellungen im Web. Sicherlich wird nicht jeder Internetnutzer eine 3D-Darstellung benötigen, aber für viele Nutzerkategorien existieren bereits sinnvolle Web3D-Anwendungen (siehe Abschnitt *1.2 Anwendungsbereiche*).

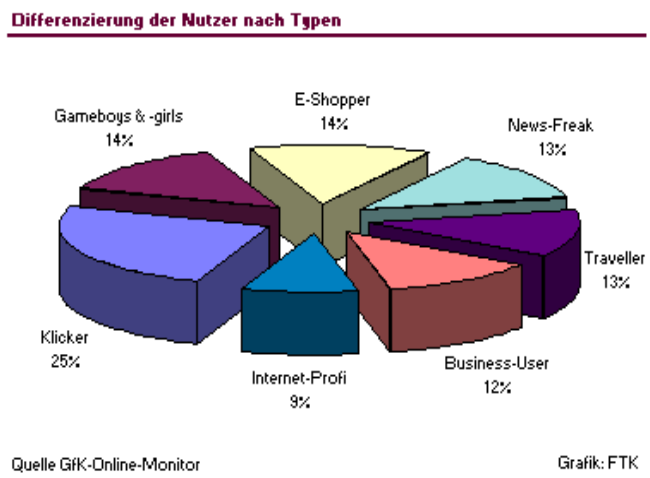



Abb. 1.1 Relative Verteilung der Internet-Nutzertypen [www18]

- Gameboys & -girls sind überwiegend jung und surfen zum Vergnügen
- E-Shopper kaufen häufiger online ein und sind in der Mehrzahl männlich
- Travellers interessieren sich für touristische Angebote, ein Großteil davon sind Frauen
- Business-User sind eher männlich und nutzen das Internet für berufliche Zwecke
- Internet-Profis sind meist jünger und nutzen das Netz sehr intensiv und überwiegend für private Zwecke
- Klicker sind sporadische Nutzer, die dem soziodemographischen Bundesdurchschnitt entsprechen
- Der News-Freak ist etwas älter und nutzt das Netz hauptsächlich zur Informationsrecherche

## 1.2. Anwendungsbereiche

Die Anwendungsbereiche für 3D-Content lassen sich in verschiedene Kategorien untergliedern. Dabei sind Übergänge zum Teil fließend und die Lösungen bedienen z. T. auch mehrere dieser Anwendungsfelder.

### 1.2.1. E-Commerce

Beim elektronischen Geschäftsmodell wird das herkömmliche „Top-Down-Modell“ (Hersteller – Händler – Konsument) gegen das „Bottom-Up-Modell“ (Kunde – Händler – Hersteller) getauscht. Der Kunde wird wesentlich stärker als bisher zum bestimmenden Faktor, der Anbietermarkt wandelt sich in einen Konsumentenmarkt. Entsprechend erlangt die Art und Weise einer Produktpräsentation eine entscheidende Bedeutung. Durch interaktive 3D-Darstellungen kann ein umfassender und realistischer Eindruck eines Produktes vermittelt werden. Der Nachteil des nichtsinnlichen Einkaufserlebnisses kann somit zum Teil relativiert werden.
Die Bedeutung interaktiver Web3D-Darstellungen im E-Commerce-Bereich kann beispielhaft an der Aussage von Monika Heidemann vom Fraunhofer-Institut für Graphische Datenverarbeitung in Darmstadt festgemacht werden: „Die Darstellung von dreidimensionalen Bildern im Internet ist ein Wachstumsmarkt und spielt eine Schlüsselrolle beim weiteren Erfolg des eCommerce. Zahlreiche Firmen bieten schon heute auf ihren Web-Seiten 3D-Modelle zum Konfigurieren ihrer Produkte, zur Auswahl von Ersatzteilen oder einfach nur zur Produktpräsentation an.“ [www19]

Nach Aussage des deutschen Business-Portals 3D Economy [www20], welches die kommerziellen Aspekte von Web3D kommuniziert, bieten sich verschiedene Vorteile bei der Nutzung dreidimensionaler Darstellungen im Web:

- fotorealistische Darstellung von Produkten und Prozessen in 3D
- Veranschaulichung komplexer Sachverhalte durch Animation
- hoher Identifikationsgrad durch Produktnähe
- virtuelle Produktkonfiguration durch Interaktion
- multimediale Zusatzinformationen integrierbar
- hoher Persistenzgrad durch Interaktion und längere Verweildauer

Außerdem werden auf der Webseite zwei Aussagen getroffen, die in ihre Absolutheit zu relativieren sind:

- schnelle Übertragung der Inhalte durch geringere Dateigrößen (Vektor-Datenformat)
- Installation keines oder lediglich eines kleinen Plug-Ins erforderlich

Die erste Aussage behält dann ihre Gültigkeit, wenn ein vergleichbarer Mehrwert durch eine Lösung realisiert werden würde, die auf dem Prinzip von QuickTimeVR [www21] basiert. Bei dieser Pseudo-3D-Technologie handelt es sich um keine echte, zur Laufzeit generierte 3D-Darstellung. QuickTimeVR und entsprechende Nachahmerprodukte [www22],[www23] basieren auf Fotos, die aus verschiedenen Perspektiven aufgenommen wurden. Die sehr gute visuelle Qualität ist hierbei mit dem Nachteil des großen Downloadvolumens verbunden.

Der Vorteil relativ kleiner Dateigrößen bei „richtigen“ Web3D-Lösungen durch ihre deskriptive, vektorielle Szenenbeschreibung kann jedoch bei intensiver Nutzung von Texturen schnell verloren gehen.

Auch der zweiten Aussage kann nur unter Vorbehalt zugestimmt werden. Je nach verwendeter 3D-Technik kann die Installation eines Plug-ins, welches mehrere Megabytes umfassen kann, erforderlich sein. Es gibt zwar Bestrebungen der Anbieter von Web3D-Technologien, ihre Plug-ins mit den Installationspaketen diverser Browser zu bündeln und offline per CD zu vertreiben, aber von einer marktdurchdringenden Verbreitung, noch dazu verschiedener Plug-ins, kann nicht ausgegangen werden. Außerdem erfolgen, nachvollziehbar am Beispiel des Macromedia Shockwave-Players, in unterschiedlichen zeitlichen Abständen automatische Updates des Plug-ins.
Bei Web3D-Darstellung mittels appletbasierter Viewer ist zwar die Installation eines Plug-ins nicht nötig, dafür müssen jedoch die Java-Class-Dateien, welche für die 3D-Funktionalitäten benötigt werden, jedes Mal mit geladen werden, wenn diese nicht mehr im Browser-Cache vorhanden sind.

Insgesamt lassen sich im E-Commerce zwei grundlegende Darstellungskonzepte unterscheiden:

- Es wird ein dreidimensionales, virtuelles Einkaufszentrum, eine Shopping Mall repräsentiert, in der sich der Nutzer bewegen kann. Dieser virtuelle Komplex dient als Rahmen für die eigentlichen Produktpräsentationen. Mit diesem Konzept verbunden ist eine möglichst große Fensterdarstellung der Webapplikation und erhöhter Aufwand für die Textdarstellung innerhalb der Szene für Navigation und Information (siehe auch Abschnitt 5.2 Dynamische Visualisierung textbasierter Inhalte). Außerdem ergibt sich ein weiterer Implementierungsaufwand für Kamerasteuerung, Kollisionserkennung, zusätzliche Interaktionsmöglichkeiten und ggf. für die persönliche Identifizierung des Nutzers mit einem angebotenen Avatar (3D-Figur als Repräsentant eines Gesprächsteilnehmers/ Spielers).
- Im Gegensatz dazu erfolgt bei einer Produktpräsentation die Visualisierung eines einzelnen Objektes ohne den entsprechenden äußeren Rahmen. Hier ist eine möglichst große Detailgenauigkeit von Bedeutung. Dem Nutzer werden Standardinteraktionsmöglichkeiten für Objektdrehung und Kamerazoom angeboten. Sollen Funktions- und Designdetails dargestellt werden, so sind zusätzliche Informationen und Auslöseereignisse zu implementieren.

Der Prozess des E-Commerce kann in weitere Subkategorien unterteilt werden, die nachfolgend näher beschrieben werden sollen.

#### 1.2.1.1. Presales

Für den Marktforschungsbereich bieten 3D-Produktpräsentationen wesentliche Vorteile gegenüber herkömmlichen Online-Anwendungen. Die Testpersonen können die Produkte drehen, wenden und von allen Seiten „begreifen“. Der so vermittelte Eindruck entspricht dem Produkterleben in der Realität; ein Effekt, der mit Fotos nicht zu erzielen ist. Durch vordefinierte Animationen können auch 3D-Präsentationen geschaffen werden, die der User nicht beeinflussen kann. Wenn gewünscht, kann so ein für alle Testpersonen identischer visueller Eindruck vorgegeben werden.

#### 1.2.1.2. Marketing & Sales

Weitreichende Möglichkeiten ergeben sich für die Präsentation von Produkten abseits unmittelbarer E-Commerce-Anwendungen.

- Produkte können bereits in frühen Entwicklungsphasen interaktiv präsentiert werden. CAD-Daten oder Zeichnungen ermöglichen die Erstellung von 3D-Web-Darstellungen. Somit können lange vor der Markteinführung Produkte im Web dargestellt und die Kundenresonanz in die weitere Produktentwicklung eingebunden werden.
- Auch hinsichtlich der Informationsvermittlung, die letztendlich zur Kaufentscheidung führen soll, bieten sich neue Möglichkeiten. Es können zur Sensibilisierung des Kunden Funktionserklärungen an einem animierten 3D-Modell erfolgen, visuelle Modifikationen in Form von verschiedenen Farbkombinationen oder eine Darstellung des Produkts mit unterschiedlichem Zubehör bzw. unterschiedlicher Ausstattung realisiert werden.
- Mit gelungenen, attraktiven Web3D-Präsentationen kann die Imagepflege, der Aufbau einer Marke unterstützt werden.

#### 1.2.1.3. Service

Mit der Implementierung von interaktiven 3D-Darstellungen in elektronischen Bedienungsanleitungen ergeben sich (ähnlich wie dies im Abschnitt *1.2.1.2. Marketing & Sales* unter dem Punkt Kundensensibilisierung angedeutet wurde) eine Reihe von Vorteilen und Verbesserungen hinsichtlich Präsentationsqualität und Verständlichkeit beim Endnutzer. Funktionen und Zusammenhänge, die sich nur kompliziert oder gar nicht darstellen lassen, können nun im Kontext visuell erschlossen werden. Allerdings wird hier auch relativ schnell das Prinzip der webbasierten 3D-Darstellung verlassen. Gewöhnlich werden Bedienungsanleitungen auf einem Offlinemedium (CD-ROM) beigelegt. Außerdem ist in diesem Fall eine Darstellung im Browser zwar denkbar, aber nicht zwingend erforderlich. Stattdessen kann eine Anleitung auch als ausführbare Datei realisiert werden. Ein weiterer Vorteil der Offline-Distribution besteht darin, dass benötigte Module für die 3D-Darstellung nicht heruntergeladen und installiert werden müssen (diese Aussage bezieht sich auf eine mögliche, mit Macromedia Director erstellte, 3D-Offline-Präsentation).


Abb. 1.2 Bedienungsanleitung einer Kamera [www24]

### 1.2.2. E-Learning

Eine Ergänzung zum Präsenzlernen stellt das E-Learning mit dem Vorteil der räumlichen und zeitlichen Unabhängigkeit des Lernvorgangs dar. Für den Einsatz von 3-dimensionalen Elementen in diesem Segment existieren verschiedene Möglichkeiten.

Anwendungsbereiche von 3D im Bereich des E-Learning können z. B. Avatare sein, die als virtuelle Begleiter eingesetzt werden, um den Nutzer beim Lernen am Computer zu unterstützen. Die Begleiter können in Lernanwendungen alle denkbaren Hilfs- und Unterstützungsfunktionen übernehmen. Ihr mittlerweile fast lebensechtes Aussehen spricht den Lernenden gefühlsmäßig an und soll ihn gleichzeitig motivieren.

Ein weiteres mögliches Einsatzgebiet betrifft das handlungs- und erlebnisorientiertes Lernen mit 3D-Modellen (s. Abb. 1.3). Dabei ist mit handlungsorientiert gemeint: 3D-Modelle können mit Hilfe von Maus und Tastatur erforscht werden. Es können auch solche Objekte untersucht werden, deren Erforschung in der Realität nicht möglich wäre. 3D-Modelle eignen sich z. B. hervorragend, um die menschliche Anatomie zu erkunden. Mit Hilfe von 3D können also Objekte erlebbar, begreifbar und im Normalfall Verborgenes sichtbar gemacht werden. Nicht zuletzt können mit 3-dimensionalen Visualisierungen (s. Abb. 1.4) naturwissenschaftliche Vorgänge und komplexe Zusammenhänge technologischer Funktionen verständlicher dargestellt sowie abrufbarer abgelegt werden.

Des Weiteren können 3D-Darstellungen von virtuellen Klassenräumen in Lernanwendungen die so genannte Raummetapher umsetzen. So wird der Eindruck eines geschlossenen Raumes, in dem man sich zum Lernen aufhält, erzeugt und die reale Lernsituation nachgeahmt. Mit virtuellen Räumen kann der Lernprozess um spielerische und entdeckende Elemente erweitert werden. Der Lernende kann im Sinne des handlungsorientierten Ansatzes den virtuellen Raum mit der Maus erforschen.

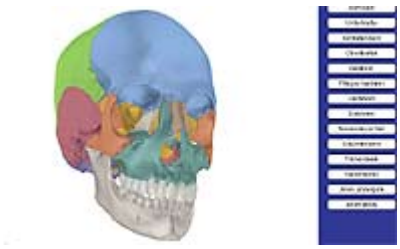


Abb. 1.3 Interaktive 3D-Darstellung eines Schädels [www25]


Abb. 1.4 Darstellung der ISS-Raumstation [www26]

### 1.2.3. Entertainment

Im Bereich webbasierter Unterhaltung gibt es verschiedene Anwendungsgebiete für die dreidimensionale Darstellung. Von besonderer Bedeutung, auch im Hinblick auf eine weitere Verbreitung von Web3D-Anwendungen, sind hierbei Spielkonzepte, speziell in kooperativen virtuellen Welten. Eine oft realisierte Variante stellen dabei so genannte First-Person-3D-Shooter dar, bei denen man sich in einer virtuellen Umgebung bewegt, diese aus dem Blickwinkel der gesteuerten Person wahrnimmt und innerhalb der Szenerie verschiedene „Aufgaben löst“. Dieser Bereich stellt hohe Anforderungen hinsichtlich Detailgenauigkeit, Visualisierungsqualität und Renderperformance an die verwendete Web3D-Plattform, da für den Nutzer die Möglichkeit besteht, relativ einfach Vergleiche zwischen webbasierten 3D-Spielen und inhaltlich ähnlich gelagerten PC-Spielelösungen zu ziehen. Häufig werden solche und ähnliche Spielkonzepte für Marketingaktionen entwickelt und genutzt. Einige Anbieter von Web3D-Technologien, u. a. CyCo Systems [www27] und The Groove Alliance [www28] haben sich auf das Marktsegment des 3D-Online-Gamings spezialisiert.

Eine andere Variante im Entertainmentbereich stellen Avatare dar. Bei einem Avatar handelt es sich um einen virtuellen Charakter, der einem Teilnehmer einer virtuellen Welt (Chatroom, Lerngemeinschaft) zugeordnet werden kann und somit zu einer Verringerung der Anonymität beiträgt (s. Abb. 1.5). Avatare können mittels Mimik und Gestik eine begrenzte visuelle Kommunikation ermöglichen und eröffnen damit eine zusätzliche Ebene der Kommunikation. Beispiele hierfür bieten u. a. Adobe Atmosphere [www29] und Pulse3D [www30] an.


Abb. 1.5 Beispiel eines 3D-Chatsystems [www31]

Als dritte Möglichkeit für den Einsatz von Web3D-Technologien bietet sich deren Nutzung für Comics und Sketche in Form von Charakteranimationen an. Hierbei spielen Details und Interaktionsfähigkeit nur eine untergeordnete Rolle. So werden beispielsweise auf der Website des amerikanischen Senders NBC Sketche mit Jay Leno, dem Moderator der sehr erfolgreichen Tonight Show [www32] in Form einer 3D-Charakteranimation visualisiert.

Weitere Ansätze für Web3D-Anwendungen im Entertainmentbereich, deren Marktakzeptanz momentan untersucht wird, lassen sich im Umfeld von Sportevents finden. Auch gibt es bereits 3D-Visualisierungen im Web, die virtuelle Führungen durch Museen, bekannte Bauwerke und Hotels ermöglichen. Jedoch wird auf eine eingehendere Darstellung dieser Versuche verzichtet.

### 1.2.4. Entwicklung

Mit den realistischen Darstellungsmöglichkeiten des 3D-Renderings können Designentscheidungen, Diskussionen über zu implementierende Features und vor allem die Auswahl von Ausstattungs- sowie Designvarianten (s. Abb. 1.6) schnell und effektiv unterstützt werden. Daher ist es nicht verwunderlich, dass innerhalb der Produktentwicklung für die unterschiedlichsten Entscheidungsphasen immer mehr Vorab-Darstellungen verwendet werden. Dreidimensionale Modelle, die für die Generierung von Prävisualisierungen, das heißt für eine frühe Nutzungsphase auf Basis vorhandener CAD-Daten erstellt wurden, können für weitere Anwendungen wie Marketing- & Aftersalesprozesse verwendet werden. Durch diese Wiederverwendbarkeit wird eine deutlich stärkere Kosteneffizienz erreicht.

Abb. 1.6 3D-Rendering als Hilfsmittel für Designentscheidungen [www20]

Ein weiteres Einsatzgebiet von 3D-Webtechnologien ergibt sich im Umfeld von Architektur und urbaner Entwicklungsplanung. Seit einigen Jahren gibt es schon leistungsfähige CAD-Lösungen, die eine dreidimensionale Visualisierung von Häusern, Landschaften sogar ganzen Stadtteilen ermöglichen. Man kann die Häuser begehen, deren Räume und Inneneinrichtung gestalten und somit einen realitätsnahen Bezug zur zukünftigen Entwicklung knüpfen. Bisher waren diese dreidimensionalen Darstellungen an leistungsfähige Workstations (u. a. *Silicon Graphics*) gebunden und verstanden sich als Offlinemedium. Mit Verbesserung der Grundlagen für eine flüssige Darstellung von 3D-Objekten ergeben sich jedoch die Voraussetzungen für eine Präsentation von Architekturentwürfen und Projekten im Web. Somit kann ein breiteres Publikum erreicht werden. Öffentliche aber auch private Bebauungspläne können im Vorfeld besser diskutiert und abgeschätzt werden (s. Abb. 1.7).

Abb. 1.7 Beispiel eines 3D-Entwurfes für ein Bebauungsprojekt [www20]

### 1.2.5. Forschung / Simulation

Der Einsatz von Web3D-Technologien in der Forschung ist in verschiedenen Bereichen denkbar. So können dreidimensionale Visualisierungen (ggf. in Explosionsdarstellung) dazu genutzt werden, das Verständnis und die Zusammenarbeit lokal verteilter wissenschaftlicher Mitarbeiter zu unterstützen. Das Einsatzgebiet ist jedoch nicht auf die alleinige Nutzung in der Forschung beschränkt, vielmehr bietet sich dieses Konzept auch in der Entwicklung neuer Produkte oder im Ersatzteilwesen an.
Ein Anbieter, der sich auf webbasierte 3D-Kollaborationswerkzeuge spezialisiert hat, ist RealityWave mit seiner VizStream-Lösung [www33].
Eine andere vorstellbare Nutzung von Web3D-Technologien im Forschungsbereich ist die Möglichkeit, Simulationen, d. h. Veränderungen über einen bestimmten Zeitraum hinweg, mittels komplexer Animationen zu visualisieren oder Ergebnisse von Untersuchungen dreidimensional zu präsentieren (s. Abb. 1.8).

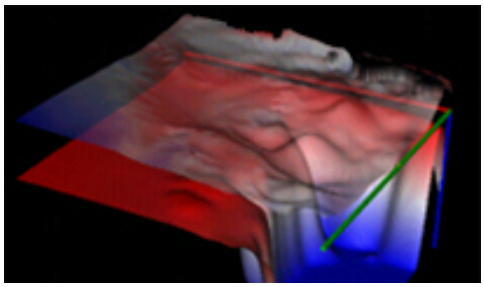


Abb. 1.8 Beispiel einer 3D-Reliefdarstellung [www20]

## 1.3. Kategorisierung von 3D-Webtechnologien

Eine treffende Definition des bereits mehrfach verwendeten Begriffs Web3D findet man in einer Pressemitteilung zum ersten deutschen 3D-eCommerce-Tag 2001 in Paderborn: „Der Begriff „Web3D“ bezeichnet alle offenen, aber auch firmeneigenen Technologien, die interaktive 3D-Grafiken im Web darstellen können. Experten sprechen auch schon vom „zweiten Web“, einer neuen Generation des World Wide Web, die durch Web3D-Technologien ermöglicht wird. Zu diesen Technologien gehören z. B. VRML, Java 3D und MPEG-4, sowie X3D, der zukünftige Standard, der derzeit vom Web3D Consortium spezifiziert wird.“ [www34]

Im folgenden Kapitel soll eine Auswahl aktueller, offener bzw. firmeneigener Web3D-Technologien näher beleuchtet, die verschiedenen Hersteller, die verwendeten (proprietären) Standards und deren Markrelevanz untersucht werden. Die vorgestellten Aussagen stützen sich auf die Quellen [www20], [www35],[LAM02],[GÖT02] und [RUK01].

### 1.3.1. Standardbasierte Technologien

Damit Portabilität und Interoperabilität von Web3D-Applikationen gewährleistet werden können, sollten benötigte Schnittstellen, Protokolle, Verfahren und Formate basierend auf der Grundlage technischer Spezifikationen realisiert werden. Diese Spezifikationen können durch die von Normungsorganisationen wie der ISO/IEC durchgeführten öffentlichen Verfahren in eine Norm überführt werden.

Die beiden nachfolgenden Kapitel beschreiben die offiziellen Standards für eine interne Repräsentation einer 3D-Szene, das heißt, es werden die Formate beschrieben, die die 3D-Informationen kapseln. Für die Visualisierung der Daten sind entsprechende Plug-ins oder Java-Applikationen (siehe Abschnitt *1.3.4. VRML Viewer*) notwendig.

Außer den beiden nachfolgend beschriebenen Formaten gibt es momentan mit der Weiterentwicklung des MPEG-4-Standards ein weiteres Format, welches theoretisch in der Lage ist, 3D-Inhalte im Web wiederzugeben. Allerdings wird auf eine weitere Darstellung verzichtet, da momentan noch keine Implementierungen für interaktive 3D-Grafik basierend auf diesem Format existieren. Weiterführende Aussagen zu diesem Format, den geplanten Weiterentwicklungen und die Implementierung von X3D innerhalb des Formates können bei Interesse unter [RUK01] und [GÖT02] nachgelesen werden.

#### 1.3.1.1. VRML / VRML97

Innerhalb der letzten acht Jahre hat sich das auf OpenInventor basierende VRML-Format zu dem am weitesten verbreiteten und am meisten genutzten 3D-Format entwickelt. Dabei handelt es sich um ein plattformunabhängiges 3D-Austauschformat, das als Exportformat von den meisten am Markt erhältlichen 3D-Modellierungswerkzeugen unterstützt wird. Die historische Entwicklung von VRML als der (bisherige) Standard der 3D-Darstellung im Web wurde bereits im Abschnitt *1.1.2. Die dritte Dimension erobert das Internet* beschrieben. VRML 2.0 war der Nachfolger von VRML 1.0. Nach der Verabschiedung als ISO/IEC-Standard 1997 wurde es in VRML97 umbenannt. Das VRML97-Format unterstützt im Gegensatz zu VRML 1.0 Animationen und die Kompression mittels gzip-Algorithmus. Außerdem ermöglicht es die Darstellung von zwei und drei Dimensionen, Text, multimedialen Komponenten und die Einbindung von Grafik, Skripten sowie die Möglichkeit der Vernetzung. Bedeutende Merkmale von VRML97 sind:

- VRML-Dateien besitzen das Dateiformat *.wrl und können mit jedem einfachen Texteditor bearbeitet werden. Sie können Verweise auf andere Dateien wie zum Beispiel andere 3D-Szenen (VRML-Dateien), Bilder, Texturen oder Filme enthalten. Diese Links können sowohl lokal als auch extern sein.
- Eine VRML97-Datei besteht aus einer Menge von Objekten oder Knoten, Feldern, Parametern usw. Dabei können Knoten weitere Knoten enthalten, die so hierarchische Strukturen bilden (siehe Abschnitt *1.4. Szenenbeschreibung*). Unter Knoten versteht man verschiedene Objekte, die für die Beschreibung der Szene und ihr Verhalten benötigt werden. Solche Knoten können z. B. geometrische Primitive, Gruppierungen, Transformationen oder Lichtquellen sein. Insgesamt existieren im Sprachumfang von VRML97 54 verschiedene Knotenarten. Daneben werden in dieser Norm auch 20 eigenständige Feldtypen spezifiziert, die vergleichbar mit unterschiedlichen Variablentypen anderer Hochsprachen sind. Beispiele für Feldtypen sind SFBool, SFColor, MFColor oder SFFloat.
- VRML97 beinhaltet Möglichkeiten zur Nutzerinteraktion und Animation von Objekten.
- Seit VRML97 können Skripte auch auf Java [www36], JavaScript und ECMAScript basieren. Mittels der Scripting-Schnittstelle und dem External Authoring Interface (kurz EAI) kann eine Verbindung zwischen VRML und Skripten bzw. Browsern hergestellt werden.
- Das bisher verwendete Codierungsformat wurde von ASCII (VRML 1.0) in UTF-8 (VRML97) geändert.

Zusätzliche und weiterführende Informationen zu VRML97 findet man in der VRML97-Spezifikation [www37].

### 1.3.1.2. X3D

VRML gilt mittlerweile als überholt. Um möglichst alle Aspekte der dreidimensionalen Darstellung im Internet umfassend zu berücksichtigen, erweiterte sich das VRML Consortium 1998 zum Web3D Consortium. Dieses verkündete im Februar 1999 die Einführung von XML zur Formulierung einer Beschreibungssprache. Ergebnis dieser Bemühungen ist die DTD eXtensible 3D (X3D), die im August des gleichen Jahres auf der SIGGRAPH in Los Angeles vorgestellt wurde.

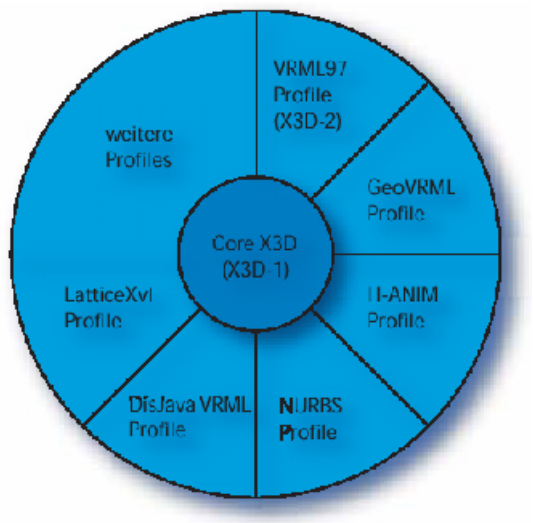
Die Entwicklung ist jedoch in verschiedenen Punkten nicht abgeschlossen. Es existiert noch keine endgültige Spezifikation des streamingfähigen binären Dateiformats und der einzige bisher veröffentlichte X3D-Player befindet sich in der Betaphase.

Bei diesem als Xj3D bezeichneten Player [www38] handelt es sich um eine auf dem Open-Source-Prinzip basierende Entwicklung, die Jave 3D als Visualisierungstechnologie verwendet. Koordiniert und vorangetrieben wird die Realisierung durch die Browser Working Group [www39].

Mit der Entwicklung von X3D sind mehrere Ziele verbunden:

- Rückwärts-Kompatibilität mit VRML97 bezüglich bestehender Anwendungen, Viewer und Tools
- Realisierung eines gegenüber VRML97 kompakteren X3D-Kerns (Core X3D), der grundlegende Funktionalitäten für die Realisierung von Web3D-Anwendungen, bezogen auf die Anwendungsbereiche E-Commerce, Unterhaltung und verteiltes Lernen, implementiert
- Unterstützung von anderen Codierungen (Schwerpunkt der Implementierung XML)
- Entwicklung eines Erweiterungsmechanismus für die Integration neuer Funktionalitäten. Es werden erweiterte Komponenten verwendet, die das schlanke Core-Profil ergänzen (s. Abb. 1.9)

Momentan sind sechs Profile in der X3D-DTD aufgeführt:




- *VRML97* Profil (X3D-2)
  die eigentliche Umsetzung von VRML97
- GeoVRML Profil
  dient zur geografische Visualisierung
- H-Anim Profil
  dient zur Darstellung von Menschen
- NURBS Profil
  Verwendung von NURBS
- DisJava VRML Profil
  zur Nutzung von verteilten interaktiven Simulationen
- LatticeXvl Profil zur kompakten Übertragung komplexer Geometrien

Abb. 1.9 Erweiterte Profile der X3D-Architektur [GÖT02]

Das X3D-Format besitzt folgende Merkmale:

- Der X3D-Kern besteht im Gegensatz zu VRML97 nur aus 26 Knotenarten und 23 Feldtypen. Damit ist X3D hinsichtlich einer Visualisierung leichter zu implementieren als ein VRML97-Viewer. Falls nötig, kann der X3D-Kern um zusätzliche Profile erweitert werden.
- Das Scene Authoring Interface (kurz SAI) ermöglicht die Kommunikation zwischen einem externen Applet innerhalb einer HMTL-Seite und der Szene. Dabei handelt es sich um ein komfortables API (Application Programming Interface), das den Zugriff auf den Szenegraphen gestattet und außerdem die Möglichkeit bietet, Verhaltensweisen für die Szene zu definieren.
- Die Kodierung des Szenegraphen erfolgt im XML-Format und die Struktur wird per XML-Grammatik definiert. Mit Hilfe einer X3D-DTD ist eine einfache und schnelle Validierung der generierten X3D-Datei möglich.
- Ein großer Nachteil von VRML war das Fehlen eines binären Datei-Formats. Dies soll zukünftig bei dem X3D-Format geändert werden. Geplant ist ein binäres Datei-Format, welches zu der Struktur des UTF-8-Formates kompatibel ist, weniger Speicherplatz für die Organisation der Daten benötigt und streamingfähig ist.

Der folgende Szenegraph (s. Abb. 1.10), der im X3D-Format (Listing 42 Pseudo-Code einer X3D-Szene [GÖT02]) beschrieben ist, soll als abschließendes Beispiel für das X3D-Konzept dienen.

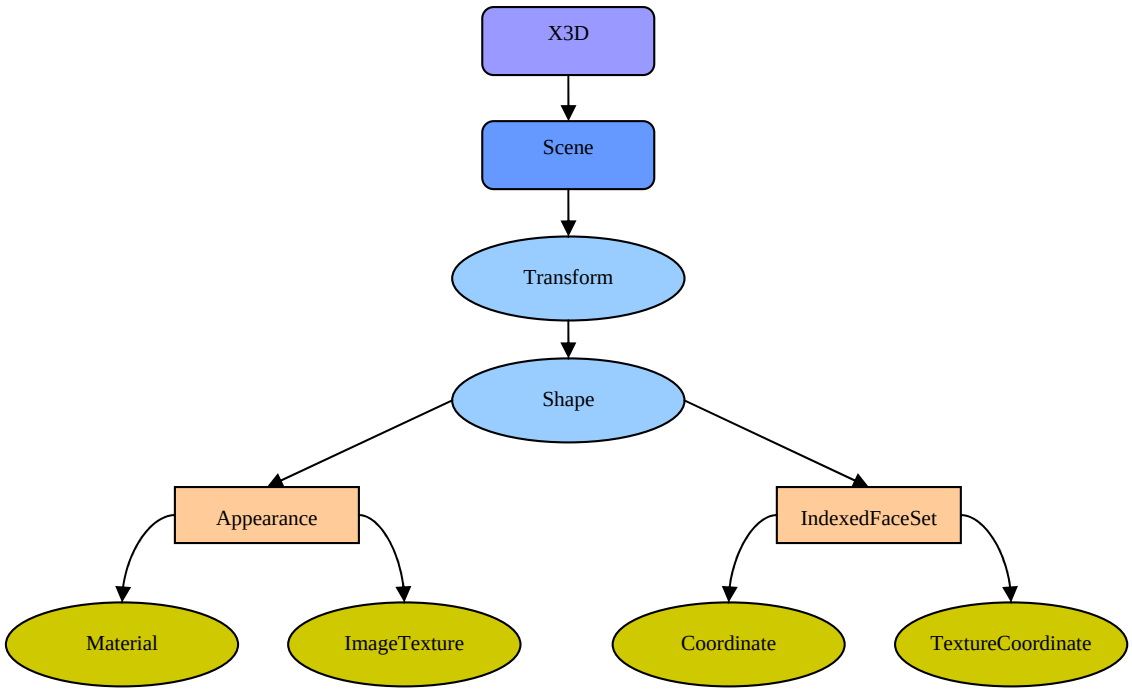



Abb. 1.10 Allgemeiner Aufbau einer X3D-Szene nach [GÖT02]

### 1.3.2. Proprietäre Plug-in-Lösungen

Es gibt eine fast nicht mehr überschaubare Anzahl von herstellerspezifischen Lösungen für die dreidimensionale Darstellung. Aus diesem Grund wird in der nachfolgenden Analyse der Focus auf marktrelevante bzw. interessante Entwicklungen gelegt.

#### 1.3.2.1. Viewpoint VET

| | |
|---|---|
| Hersteller | Viewpoint, gegründet aus MetaCreation / Metastream (2000) |
| URL | http://www.viewpoint.com |
| Plattformen | MS Windows, MacOS |
| Installationsdateigröße | ca. 1,3 MB |
| Zusammenfassung<br>  <br><br>Abb. 1.11 Quelle [www14] | Viewpoint bietet mit seiner Viewpoint Experience Technology (VET) eine Rich Media Plattform, die mit dem integrierten Viewpoint Media Player (VMP) die unterschiedlichsten multimedialen Formate wiedergeben kann (3D-Welten, Flash, QTVR). Dieses Plug-in bietet neben Cult3D und Director Shockwave3D die beste Darstellungsqualität. Entsprechend gibt es für dieses Web3D-Format auch ähnlich viele gute Referenzen. Andere Hersteller, wie z. B. Adobe mit seinem Produkt Atmosphere, nutzen für ihre Lösungen das VET-Konzept. Für die interne Repräsentation verwendet Viewpoint das XML-Datenformat. Der Player selbst ist modular aufgebaut und lädt bei Bedarf fehlende Komponenten automatisch nach. Ein interessantes Feature ist die so genannte Hyperview-Technologie. Sie ermöglicht Animationen über Fenstergrenzen hinweg. So bewegt sich zum Beispiel, wie in Bild links zu sehen ist, ein Fahrzeug über den gesamten Bildschirm. Der Media Player ist recht weit verbreitet. Er wurde bisher weltweit ca. 25 Millionen mal, davon 8 Millionen mal in Deutschland installiert. Unter den Firmen, die sich der VE-Technology bedienen, sind solch renommierte Firmen vertreten, wie z. B. BMW, die Deutsche Telekom oder AOL. Letztgenannte Firma vertreibt das Plug-in seit der AOL-Zugangssoftware 7.0. |

Tabelle 2 Zusammenfassung Viewpoint Experience Technology

### 1.3.2.2. Pulse3D

| | |
|---|---|
| Hersteller | Pulse, gegründet 1994, USA |
| URL | http://www.pulse3d.com |
| Plattformen | MS Windows, MacOS, PocketPC |
| Installationsdateigröße | ca. 0,4 MB |
| Zusammenfassung  Abb. 1.12 Quelle [www40] | Die Stärken von Pulse3D liegen in der Unterstützung von Charakteranimationen und Sprachsynchronisationen, in einer guten Darstellungsqualität und der Möglichkeit, Inhalte streamen zu können. Das Plug-in nutzt für die Wiedergabe entweder den Softwarerenderer oder setzt auf vorhandene Hardwareunterstützungen (DirectX, OpenGL) auf. Die Software unterstützt Antialiasing und kann eng mit dem Realplayer und QuickTime integriert werden. Sie beinhaltet eine eigene Programmiersprache, Pulsescript, mit der zur Laufzeit dynamische Änderungen an der Szene realisiert werden können. Die Referenzliste im Web3D-Bereich ist allerdings nicht sehr umfangreich. Viele Lösungen sind über den Status eines Prototyps nicht hinaus gekommen, andere sind bereits wieder eingestellt worden oder nur sehr schwer zu finden. Eine detaillierte Darstellung dieser Web3D-Lösung mit der Beschreibung einer Charakteranimation inklusive Sprachsynchronisation kann in [NAU01] nachgelesen werden. |

Tabelle 3 Zusammenfassung Puls3D-Technologie

### 1.3.2.3. Cycore Cult3D®

| | |
|---|---|
| Hersteller | Cycore, gegründet 1996, Schweden |
| URL | http://www.cult3d.com |
| Plattformen | MS Windows, MacOS, Linux, Solaris, HP-UX,AIX |
| Installationsdateigröße | ca. 1,2 MB |
| Zusammenfassung  Abb. 1.13 Quelle [www41] | Der Viewer unterstützt deaktivierbares Antialiasing und bietet, auch dank der Unterstützung von Transparenzen und Schattenwurf, eine sehr gute visuelle Qualität. Die Renderengine ist komplett im Plug-in realisiert und greift nicht auf eine Hardwarebeschleunigung zurück. Stärken dieser Technologie sind neben der Visualisierungsqualität das kompakte (eigene) Datenformat, das Kompression und Streaming erlaubt, die breite Unterstützung verschiedener Plattformen sowie die große Anzahl guter Referenzen. Erstellt werden können 3D-Objekte mit professionellen 3D-Modellierungswerkzeugen wie 3D Studio MAX oder Maya, für die Export-Plug-ins existieren. Die im Cult3D-eigenen Format (*.c3d) exportierten 3D-Szenen werden anschließend im Cult3D-Designer bearbeitet und um interaktive Verhaltensweisen erweitert. Die Publikation der Cult3D-Applikation erfolgt im eigenen verschlüsselten, szenegraphbasierten Dateiformat (*.co). Cult3D wird erfolgreich vor allem für kommerzielle Produktpräsentation eingesetzt. |

Tabelle 4 Zusammenfassung Cult3D-Technologie

### 1.3.2.4. Macromedia Director Shockwave3D

| | |
|---|---|
| Hersteller | Macromedia, USA |
| URL | http://www.macromedia.com/software/director/ |
| Plattformen | MS Windows, MacOS |
| Installationsdateigröße | ca. 4,5 MB |
| Zusammenfassung<br>  <br>Abb. 1.14 Aus selbst-entwickelter Testszene | Director bietet mit der aktuellen Version 8.5.1/ MX eine der umfassendsten Implementationen von 3D-Funktionen aller am Markt befindlichen Plug-ins. Als Dateiformat für 3D-Szenen verwendet Director das W3D-Format. Fast alle kommerziellen 3D-Editoren bieten mittlerweile ein Export-Plug-in für das W3D-Format an. Leider können Szenen, die in Director per Lingo erstellt oder manipuliert worden sind, standardmäßig nicht in eine W3D-Datei geschrieben werden (es gibt allerdings erste Versuche, dies über Xtras zu realisieren). Director3D unterstützt Keyframe- und Knochenanimationen, Partikelsysteme, unterschiedliche Interaktionsformen, komplexe Beleuchtungsmodelle und Multitexturing.<br>Das Plug-in verwendet Multi-Resolution Meshes, mit denen die Auflösung der Geometriedaten der Modelle zur Laufzeit skaliert werden kann. Dies kann entweder in Abhängigkeit zur Kameraentfernung automatisch erfolgen oder aber per Lingo um die Renderanforderungen an die Hardwarevoraussetzungen anzupassen.<br>Das gegenteilige Prinzip wird mit der Subdivision Surfaces Technik (SDS) realisiert. Zur Laufzeit werden durch weitere Oberflächenunterteilungen Polygone zu einem Modell hinzugefügt, um es komplexer erscheinen zu lassen. Die Datei, die die ursprüngliche Geometrie beinhaltet kann somit relativ kompakt gehalten werden.<br>Für Standard-3D-Anforderungen, wie sie sich zum Beispiel bei einer Produktpräsentation ergeben, stellt Director vorgefertigte Skripte zur Verfügung, mit denen Objekt- und Szenenmanipulationen ohne Programmierkenntnisse realisiert werden können.<br>Das Plug-in besitzt eine große Verbreitung [www42] und ermöglicht, auch dank entsprechender Grafikhardwareunterstützung mittels OpenGL und DirectX, eine sehr gute Darstellungsqualität. Antialiasing für 3D-Szenen kann seit Version 8.5.1. aktiviert werden. Mit der eigenen Programmiersprache Lingo hat man die volle Kontrolle über die Szene. Director Shockwave3D eignet sich gleichermaßen für den Einsatz bei Offlinemedien wie CD-ROMs oder DVDs, als auch für die dreidimensionale Darstellung im Internet. Allerdings ist die Größe des Plug-ins eher von nachteiliger Wirkung. Außerdem werden keine Schattenmaps unterstützt; diese müssen bei Bedarf durch entsprechende Texturen simuliert werden. |

Tabelle 5 Übersicht Macromedia Director3D

### 1.3.2.5. Adobe Atmosphere

| | |
|---|---|
| Hersteller | Adobe, USA |
| URL | http://www.adobe.com/products/atmosphere/ |
| Plattformen | MS Windows |
| Installationsdateigröße | ca. 4,5 MB |
| Zusammenfassung<br>  <br><br>Abb. 1.15 Quelle [www29] | Bei dieser Software handelt es sich um eine Betaversion, die Anfang 2001 veröffentlicht wurde. Scheinbar ist Adobe nicht gewillt, das Produkt weiter zu entwickeln. Das Plug-in unterstützt Antialiasing, benutzt nur den Softwarerenderer und basiert auf dem Viewpoint-Datenformat. Insgesamt zeichnet sich die Technologie durch eine gute Darstellungsqualität aus, besonders erwähnenswert ist der relativ einfach gestaltete Editor, mit dem eigene 3D-Welten generiert werden können. Vorstellbare Einsatzszenarien wären virtuelle Chatsysteme, Community-Plattformen und architektonische Präsentationen.<br>Die Usability der Kamerasteuerung ist wenig geeignet für große Szenen (Maus vorwärts schieben für Bewegung in Blickrichtung).<br>Adobe hätte mit seinem Bekanntheitsgrad am Markt durchaus Chancen, dieses Produkt als Nischenlösung zu etablieren. Allerdings ist im Hinblick auf eine fast zweijährige Zeitspanne, in der keine Final-Version veröffentlicht wurde, davon auszugehen, dass die Produktentwicklung eingestellt wurde. Eine offizielle Aussage konnte dazu jedoch nicht gefunden werden. |

Tabelle 6 Übersicht zu Adobe Atmosphere

### 1.3.3. Appletbasierte Viewer

Diese Viewer werden als Applet realisiert. Somit können die 3D-Darstellungen mit jedem Browser auf jeder Plattform visualisiert werden, die eine Java 1.1-konforme Virtual Machine unterstützen. Die Appletlösungen benutzen in der Regel ein eigenes Format für die Speicherung und Anzeige von 3D-Elementen auf einer Website, sie unterstützen jedoch meistens (zumindest in den nachfolgenden Lösungen) Szenenmaterial, das im VRML-Format vorliegt.

Die für 3D-Szenen notwendigen Funktionen hinsichtlich Rendering, Picking, Kollisionserkennung etc. sind in eigenen Class-Dateien gekapselt, die jedes Mal, wenn sie nicht mehr im Cache des Browsers vorhanden sind, mit geladen werden müssen.

#### 1.3.3.1. 3Danywhere

| | |
|---|---|
| Hersteller | 3Di |
| URL | http://www.3danywhere.com |
| Plattformen | JavaApplet, entsprechend plattformunabhängig |
| Installationsdateigröße | Applet wird jedes Mal mit heruntergeladen |
| Zusammenfassung<br>  <br>Abb. 1.16 Produktpräsentation mit 3Danywhere [www43] | 3Danywhere bietet insgesamt, auch dank der Unterstützung von Antialiasing, eine gute Darstellungsqualität. Das Applet hat eine eigene Renderengine und unterstützt keine hardwarebasierten Rendermodi. Die Darstellungsgeschwindigkeit ist dadurch nicht besonders hoch, für einfache Szenen jedoch ausreichend. Interaktivität muss per Java oder Javascript manuell implementiert werden. Allerdings können mit dem zugehörigen Editor auch Funktionen zur Interaktivität erstellt werden.<br>3Danywhere nutzt das VRML97-Format. Somit kann jedes VRML97-Export-fähige Modellierungswerkzeug für die Erstellung von Modellen und Animationen verwendet werden. |

Tabelle 7 Zusammenfassung der Web3D-Technologie 3Danywhere

#### 1.3.3.2. Shout3D

| | |
|---|---|
| Hersteller | Shout Interactiv |
| URL | http://www.shout3d.com |
| Plattformen | JavaApplet, entsprechend plattformunabhängig |
| Installationsdateigröße | Applet wird jedes mal mit heruntergeladen |
| Zusammenfassung<br>  <br>Abb. 1.17 Quelle [www44] | Shout3D ist komplett in Java programmiert und nutzt seinen eigenen Renderer, der standardmäßig keine 3D-Hardwarebeschleunigung unterstützt. Es bietet aktivierbares Antialiasing und eine insgesamt gute Darstellungsqualität, wobei die Renderperformance nicht sehr groß ist. Interaktive Funktionalitäten müssen per Java oder JavaScript programmiert werden. Standardmäßig werden einige Basis-Applets mitgeliefert, die einfache Interaktionen abdecken oder als Grundlage für Weiterentwicklungen dienen können. Shout3D nutzt für die Informationsübertragung sein eigenes Datenformat (*.s3d) oder das VRML-Format. Die Technologie ist schon sehr lange am Markt und besitzt dank guter Referenzen einen entsprechenden Bekanntheitsgrad. |

Tabelle 8 Zusammenfassung der Web3D-Technologie Shout3D

#### 1.3.3.3. Sun Microsystems Java 3D

Im Gegensatz zu den beiden ersten appletbasierten Web3D-Technologien ist es bei der Verwendung von Java 3D für die Visualisierung nicht ausreichend, dass eine Java 1.1-konforme Virtual Machine im System vorhanden ist. Für die Darstellung wird zusätzlich die Java 3D-RunTime-Version benötigt.

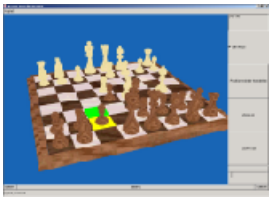
| | |
|---|---|
| Hersteller | Sun Microsystems, USA |
| URL | http://www.java.sun.com/products/java-media/3D/index.html |
| Plattformen | MS Windows, Mac OS, Linux, Solaris, HP-UX,AIX |
| Installationsdateigröße | ca. 1,2 MB |
| Zusammenfassung<br>  <br>Abb. 1.18 Aus Java 3D-Beleg | Java 3D wird zumeist in Wissenschaft, Forschung und Lehre verwendet. Haupteinsatzgebiete sind dabei Simulation, Animation oder die 3-dimensionale-Datenvisualisierung.Bei dieser Web3D-Technologie handelt es sich um ein 3D-Grafik-API, welches auf der objektorientierten Programmiersprache Java basiert. Ein Vorteil von Java 3D ist, dass die Programmierung auf einem relativ hohen Abstraktionslevel erfolgen kann. Aufwendige 3D-Programmierungen auf Low-Level-Ebene werden durch die Verwendung der Java 3D-API vereinfacht, der Entwickler kann sich auf die Realisierung der Szene unter Verwendung der Java 3D-Funktionen konzentrieren. Für die interne Repräsentation der 3D-Objekte wird das Prinzip des Szenegraphen (siehe Abschnitt *1.4. Szenenbeschreibung*) verwendet. Die mit der Java 3D-API erstellten Applets können problemlos im Web präsentiert werden. Die für die Betrachtung der Java 3D Szenen benötigte Laufzeitumgebung ist standardmäßig nicht in den Browsern implementiert. Sie muss separat herunter geladen und installiert werden.<br>Java 3D nutzt kein eigenes Format für die Repräsentation der Szeneninformationen. Allerdings existieren entsprechende Importlösungen (Runtime-Loader), um extern erstellte Szenen im VRML- oder Alias-Wavefront OBJ-Datenformat integrieren zu können. Durch den zu VRML97 ähnlich aufgebauten Szenegraphen wird Java 3D gern als Rendering-Engine für die Entwicklung von 3D-Viewern verwendet (u. a. X3D/ VRML97-Viewer Xj3D [www37]). Die 3D-Formate werden per Loader-Klassen importiert, anschließend interpretiert, der Szenegraph wird entsprechend aufgebaut und die Szene visualisiert.<br>Standardmäßig setzt Java 3D auf das vom Betriebssystem zur Verfügung gestellte OpenGL-API auf. Bei Windows-Plattformen besteht zusätzlich die Möglichkeit, Direct3D (DirectX) zu nutzen. Auf Macintosh-Computern kann stattdessen auf QuickDraw3D zurückgegriffen werden. Bestehen keine Möglichkeiten für hardwareunterstütztes Rendering, so verwendet Java 3D einen eigenen Softwarerenderer. |

Tabelle 9 Übersicht zu Java 3D

### 1.3.4. VRML Viewer

In [LAM02] werden VRML-Viewer dahingehend klassifiziert, dass sie zwar auch als Plug-ins realisiert sind, aber im Gegensatz zu den in Abschnitt *1.3.2. Proprietäre Plug-in-Lösungen* vorgestellten Lösungen auf VRML basieren und kein eigenes proprietäres Format mit sich bringen.

#### 1.3.4.1. Cortona

| | |
|---|---|
| Hersteller | Parallelgraphics, gegründet 1998, Irland |
| URL | http://www.parallelgraphics.com |
| Plattformen | MS Windows, MacOS, PocketPC, Java |
| Installationsdateigröße | ca. 1MB |
| Zusammenfassung<br>  <br>Abb. 1.19 Quelle [www45] | Die Darstellungsqualität ist durch Antialiasing und eine Reihe erweiterter Darstellungsmöglichkeiten sehr gut. Neben dem Software-Rendermodus werden auch hardwareunterstützte Rendermodi angeboten (DirectX, OpenGL). Der Viewer benutzt das VRML97-Format für die interne Datenrepräsentation. Mittels Java und Javascript kann die Funktionalität erweitert werden (um z. B. Interaktivität zu implementieren).<br>Für das Erstellen der Szenen kann jedes 3D-Modellierungswerkzeug verwendet werden, welches in der Lage ist, in das VRML97-Format zu exportieren. Außerdem bietet der Hersteller eigene Authoringsoftware für verschiedene Spezialgebiete (Architektur) an. Da Cortona auf dem VRML97-Standard basiert, wird das Plug-in in Zukunft mit Sicherheit auch den neuen X3D-Standard unterstützen. |

Tabelle 10 Übersicht zum Cortona VRML-Viewer

#### 1.3.4.2. Blaxxun Contact

| | |
|---|---|
| Hersteller | blaxxun interactive AG, gegründet 1995, Deutschland |
| URL | http://www.blaxxun.de |
| Plattformen | MS Windows |
| Installationsdateigröße | ca. 1,5 MB |
| Zusammenfassung<br>  <br><br>Abb. 1.20 Quelle [www46] | Die Firma Blaxxun positioniert sich mit seinem Produkt Blaxxun Contact in Richtung virtuelle Communities und 3D-Chatlösungen. Außerdem sieht die Firma ihr 3D-Konzept als Ergänzung zum Portfolio ihrer bisherigen E-Learning-Produkte. Blaxxun ist schon relativ lange im W3D-Marktsegment tätig und wirkte u. a. bei der Standardisierung von X3D mit. Daher ist davon auszugehen, dass dieser neue Standard von Blaxxun unterstützt und implementiert wird.<br>Für die Generierung von Szenen kann jedes 3D-Modellierungswerkzeug verwendet werden, welches in der Lage ist in das VRML97-Format zu exportieren. |

Tabelle 11 Zusammenfassung der Web3D-Technologie Blaxxun Contact

### 1.3.5. Einschätzung des aktuellen Standes der Web3D-Technologien

Es gibt verschiedene Kriterien, die für oder gegen die Verwendung der einen oder anderen Web3D-Technologie sprechen.
Plug-in-Lösungen, wie sie durch Director Shockwave3D, Viewpoint oder Cult3D repräsentiert werden, bieten die beste Darstellungsqualität und Renderperformance. Nachteilig für diese Lösungen ist jedoch die Tatsache, dass ihre proprietären Technologien standardmäßig nicht in den Browsern integriert sind. Deshalb müssen diese Plug-ins einmalig herunter geladen und installiert werden. Dies geschieht jedoch automatisch und benutzerfreundlich. Allerdings können Downloadgrößen von mehr als 4MB (Shockwave) auch ein Grund sein, Webinhalte, die auf dem entsprechenden Plug-in beruhen, nicht zu akzeptieren. Dies relativiert sich jedoch auch schon wieder im Hinblick auf die Verbreitung breitbandiger Internetzugänge und die Tatsache, dass dem Nutzer klar sein sollte, dass der Mehrwert einer 3D-Darstellung mit einem größeren Downloadvolumen verbunden ist.

Entscheidendes Argument gegen eine Plug-in-Lösung können neben der begrenzten Plattformunabhängigkeit vor allem jedoch administrative Restriktionen bei der Zielgruppe sein. Eine Reihe von Firmen und Institutionen untersagen in ihren EDV-Richtlinien die Installation und Nutzung fremder Plug-ins aus Wartungs- und Sicherheitsgründen (Beispiel: Sparkassenverband Deutschland). In solchen Fällen können nur Lösungen eingesetzt werden, die auf offenen Standards basieren und zur Darstellung Java-Applets verwenden.

Die getesteten appletbasierten Viewer konnten zum Teil nicht überzeugen, der Download der benötigten 3D-Klassen dauerte, bezogen auf die einmalige Installation eines Plug-ins, recht lange, die Darstellungsqualität war nicht mit den oben genannten Lösungen vergleichbar und auch die Renderperformance war wenig akzeptabel.
Vorteil dieser Lösungen ist das Prinzip des „Write Once, Run Everywhere“. 3D-Darstellungen können mit jedem Browser auf jeder Plattform visualisiert werden, die eine Java 1.1-konforme Virtual Machine unterstützen.

## 1.4. Szenenbeschreibung

Eine Reihe von 3D-Technologien, so zum Beispiel X3D, VRML und Java 3D, nutzen zur Beschreibung einer Szene einen Szenegraphen. Das bedeutet, dass alle Informationen, welche die Szene definieren, in einer hierarchischen Baumstruktur angeordnet werden. Die verschiedenen Technologien, die eine virtuelle 3D-Welt beschreiben, indem sie einen Szenegraphen verwenden, nutzen das gleiche Funktionsprinzip einer hierarchisch strukturierten Datenkapslung. Allerdings bestehen erhebliche Unterschiede innerhalb der Syntax, die zur Beschreibung des Graphen verwendet wird. Aufgrund der besseren Dokumentation soll das Prinzip des Szenegraphen detailliert am Beispiel von Java 3D beschrieben werden. Bei Details, die zur Syntax, zum Prinzip des Szenegraphen von Director vergleichbar sind, wird entsprechend darauf eingegangen.
Grundsätzlich ist ein Szenegraph aus Knoten (Gruppenknoten, Blattknoten), Kanten und Komponenten aufgebaut. Knoten (Nodes) sind grundlegende Datenelemente des Szenegraphen, welche alle Details der 3D-Welt definieren. Dabei dienen Gruppenknoten zur Gruppierung von Knoten. Blattknoten wiederum bilden den Abschluss eines Zweiges des Szenegraphen. Kanten beschreiben hierarchische Verknüpfungen (Eltern-Kind-Beziehungen) zwischen den Knoten des Graphen. Dabei darf jeder Knoten immer nur einen Elternknoten besitzen.

Komponenten beschreiben die Eigenschaften der Knoten, zum Beispiel das Aussehen, das Material sowie die Geometrie. Sie werden mittels Referenzen in den Szenegraphen eingebunden. Eine Referenz stellt (in Java 3D) eine Assoziierung zwischen einer Komponenten- Instanz und einem Knoten des Graphen dar. Dadurch kann auf eine Komponente von mehreren Knoten aus verwiesen werden (siehe Abb. 1.21).
Ein Group-Node hat dabei ein Parentelement und beliebig viele Childelemente. Im Gegensatz dazu hat ein Leaf-Node ein Parentelement und keine Childelemente.

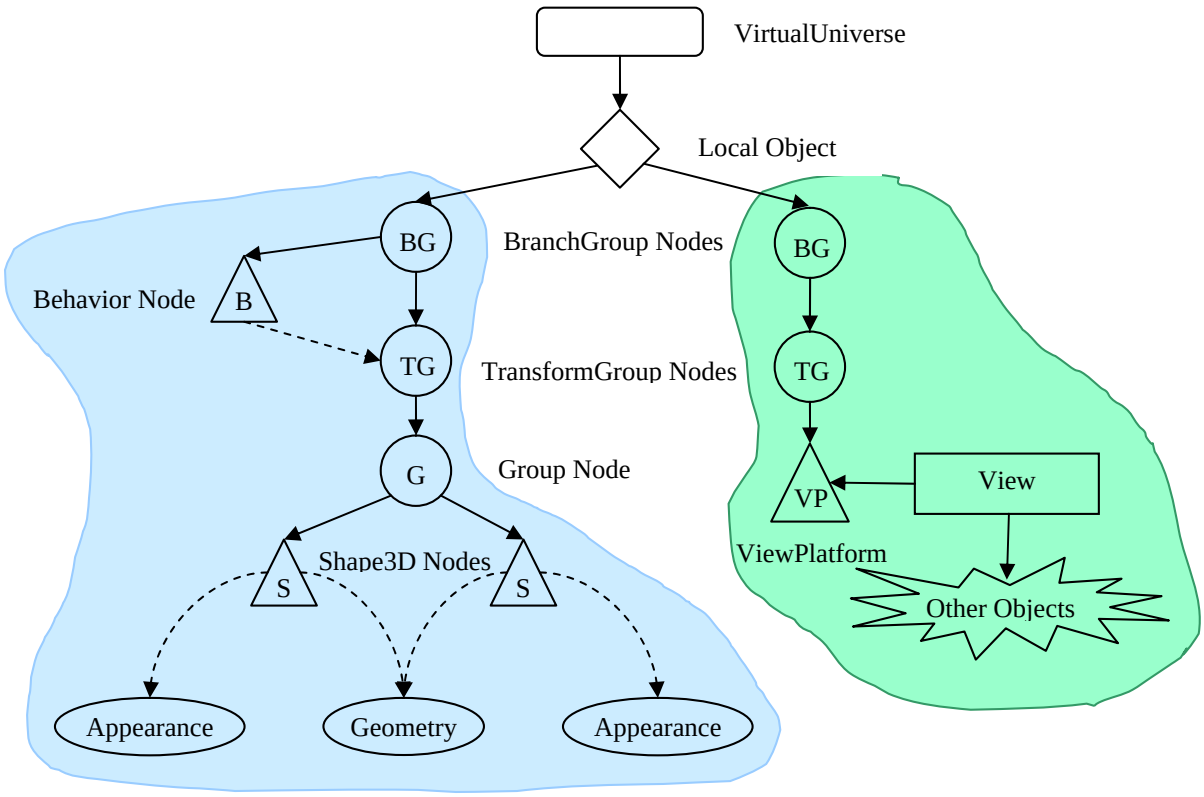



Abb. 1.21 Allgemeiner Aufbau eines Java 3D-Szenegraphen

Das oberste Element einer Szene, auch als Rootelement bezeichnet, wird in Java 3D als VirtualUniverse bezeichnet (in Director als „World“). Hierarchisch unter dem Wurzelobjekt befindet sich eine Art „Startpunkt“, das Locale Object. Für komplexe, austauschbare Szenen, die verschiedene Ansichten beinhalten, könnten auch weitere Locale Objects definiert werden. Unter dem Locale Object teilen sich zwei verschiedene Zweige auf. Der in Abb. 1.21 dargestellte linke hellblaue Zweig beinhaltet den Aufbau der Szene (Geometrien, Aussehen, Positionen, Hierarchien, Beleuchtung sowie Verhalten von Objekten) und wird als ContentBranch bezeichnet. Der rechte hellgrüne Zweig definiert den Blick auf die Szene, das heißt, wie die Welt dargestellt wird (z. B. Kameraeigenschaften und -position). Dieser Zweig wird als ViewBranch bezeichnet. Die Wurzelelemente dieser beiden Zweige sind jeweils so genannte BranchGroup-Objekte. Bei diesen Objekten handelt es sich um einfache Wurzelknoten, die untergeordnete Elemente zu einem kompilierbaren Subgraphen zusammenfügen. Mit ihnen können Teile des Szenegraphen zur Laufzeit eingehängt und auch wieder entfernt werden.

In Director3D erfolgt keine solch strikte Trennung in einen View- und einen Contentzweig. Stattdessen wird standardmäßig allen Kameras, Lichtern, Modellen und Gruppen (Director3D-Semantik siehe Abschnitt *2.4.1. Shockwave3D-Darsteller, Koordinatensystem und Orientierung*) der Knoten „World“ zugeordnet. Diese Zuordnung lässt sich jedoch für die Erstellung hierarchischer Strukturen anpassen (siehe *Abschnitt 2.4.5. Hierarchische Strukturen, Parent-Child-Beziehungen*).

Die Renderengine von Director3D verwendet für die Berechnung der Szene alle Objekte, deren oberstes Element der World-Knoten ist. Knoten inklusive ihrer Subelemente können mit dem Befehl `#removeFromWorld` vom Rendervorgang ausgeschlossen werden. Umgekehrt können einzelne Elemente und ganze Zweige per `#addToWorld` wieder in den Renderprozess eingebunden werden.

TransformGroup-Knoten dienen der örtlichen Verschiebung bzw. Positionierung, Skalierung und Drehung von Objekten. Mit diesen Knoten können zur Laufzeit alle Elemente, die Kinder von ihnen sind, verändert werden.

In Director wird explizit nicht von solchen (übergeordneten) TransformGroups gesprochen. Stattdessen beinhaltet jeder Knoten seine eigene Transformationsmatrix (`node(whichNode).transform`).

Gewöhnlich werden 3D-Objekte in Java 3D durch Shape3D-Knoten beschrieben. Diese Blattknoten referenzieren entsprechende Appearance- und Geometry-Komponenten, die das Aussehen bzw. die Geometrie des Objektes definieren.

In Director3D ist das Modell das Äquivalent zum Objekt, die geometrische Form und das Erscheinungsbild werden in einer Modellressource definiert, wobei diese wiederum eine Referenz auf einen Shader beinhaltet, welcher die optischen Materialeigenschaften beschreibt.

Um Objekten ein bestimmtes Verhaltensmuster zuzuweisen, werden in Java 3D von der Klasse Behavior abgeleitete Klassen verwendet. Mit ihnen kann auf Ereignisse reagiert und der Szenegraph manipuliert werden (Transformationen, Animationen, Interaktionen (Picking), Lichtquellen (de-)aktivieren usw. ).

In Director wird die Funktionalität auch durch Behaviors (Verhalten) erweitert. Allerdings sind die Behaviors keine Elemente des Szenegraphen. Sie wirken von „außen“ auf die Elemente des Graphen und nutzen dafür entsprechende Referenzen (z. B. `member(which3DMember).model(whichModel)`).

# 2. Grundlagen Director und 3D-Darstellung

Um die Ausführungen hinsichtlich der praktischen Umsetzung des Diplomthemas verstehen zu können, auch wenn nur geringe oder gar keine Kenntnisse in Handhabung von Director vorhanden sind, soll in diesem Kapitel eine systematische Einführung erfolgen.

Der Begriff Director wird in der weiteren Abhandlung als verkürzte Form von Director®8.5 Shockwave® Studio für 3D verwendet. Er bezeichnet das Autorenwerkzeug von Macromedia, mit dem der Prototyp realisiert wurde.

Shockwave ist die Distributionstechnologie von Director-Produktionen, die im Internet präsentiert werden. Diese Technik ist gekennzeichnet durch die Nutzung verschiedener Kompressionsverfahren, die es ermöglichen, Richmedia-Content in relativ kleinen Dateien im Internet anzubieten. Ein weiterer Vorteil dieser Technologie ist die Möglichkeit, Shockwave-Filme zu streamen, d. h. sie können schon abgespielt werden, ohne dass die Datei komplett geladen wurde.

## 2.1. Einführung in die Director-Semantik

Bevor die 3D-Philosophie von Director näher untersucht wird, sollen hiermit allgemeine Begriffe, die die Arbeitsweise von Director beschreiben, erläutert werden. Abb. 2.1 zeigt die wichtigsten Fenster des Entwicklungswerkzeuges. Ihr Verwendungszweck und ihre Einordnung in das Director-Konzept wird in den nachfolgenden Unterkapiteln beschrieben.

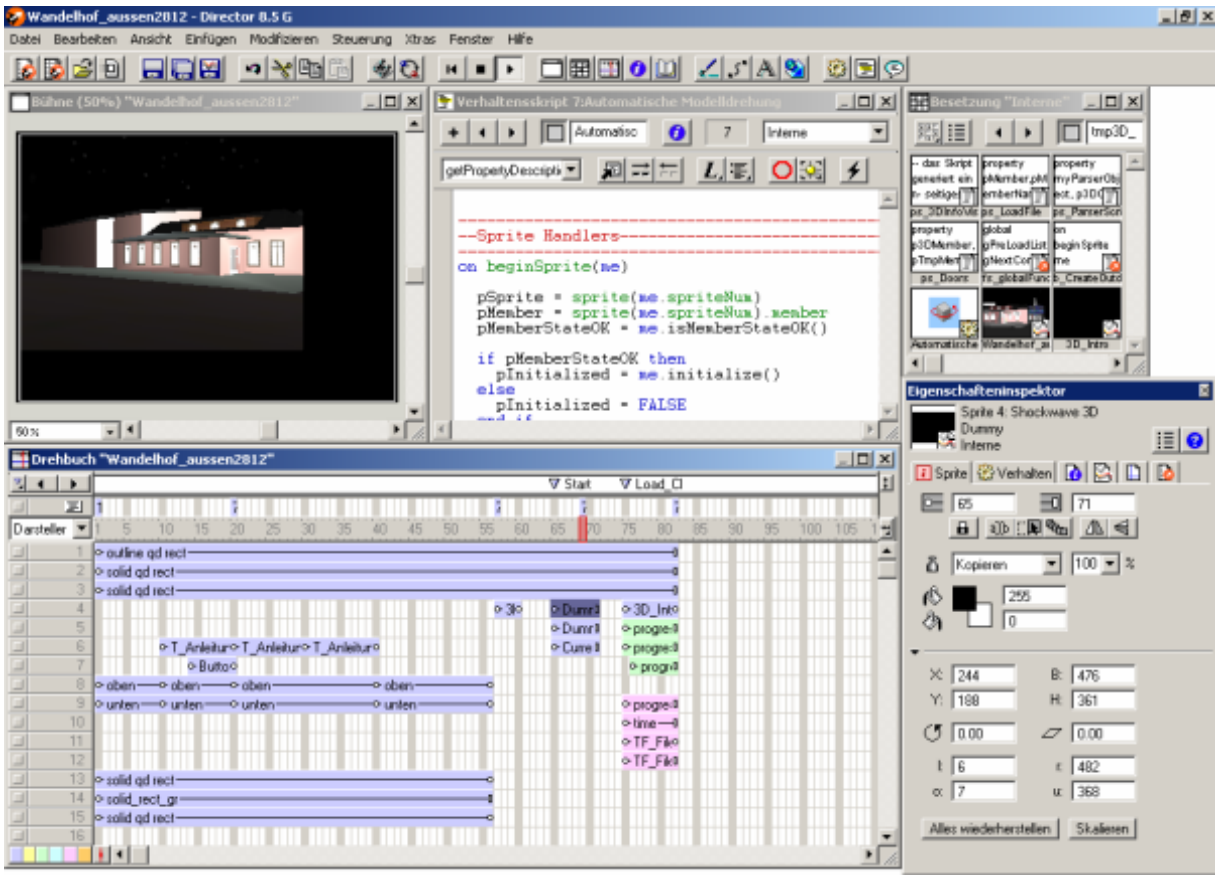



Abb. 2.1 Der Director-Arbeitsbereich

### 2.1.1. Die Bühne

Director bedient sich bei den Bezeichnungen seiner Komponenten der Theater-Metapher. Eine Director-Datei wird auf Grund ihres zeitlichen Charakters als Film bezeichnet. Der sichtbare Teil einer zukünftigen Präsentation wird Bühne genannt. In Abb. 2.2 wird die Dimension der Bühne durch die roten Winkel begrenzt. Der graue Hintergrund ist die Leinwand. Auf ihr können weitere Elemente platziert werden, die jedoch während der Präsentation unsichtbar bleiben.


Abb. 2.2 Director-Arbeitsbereich – die Bühne

### 2.1.2. Die Besetzung

Alle benötigten Medienelemente und Skripte werden in Besetzungen (s. Abb. 2.3) verwaltet. Bei diesen Medienelementen, die als Darsteller bezeichnet werden, kann es sich z. B. um 3D-Szenen, Bilder, digitale Videos oder Sounds handeln. Von einem Darsteller können mehrere Instanzen auf der Bühne/ Leinwand erzeugt werden. Diese Instanzen, sie werden als Sprites bezeichnet, sind durch unterschiedliche, typspezifische Attribute definiert. Die zwei wichtigsten davon, Position und zeitlicher Gültigkeitsbereich, werden gewöhnlich im Bühnen- bzw. Drehbuchfenster festgelegt.

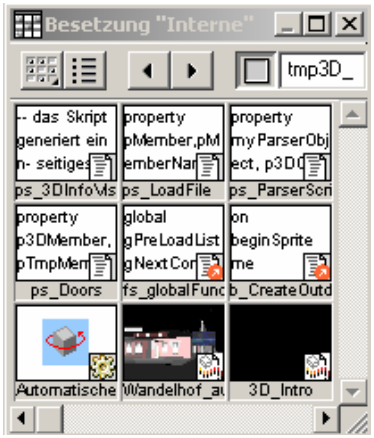



Abb. 2.3 Director-Arbeitsbereich – die Besetzung

### 2.1.3. Das Drehbuch

Das Drehbuch (s. Abb. 2.4) stellt die einzelnen Sprites bezogen auf zeitliches Verhalten und ihre Tiefenanordnung dar. Dabei entspricht Spritekanal 1 dem Hintergrund. Das zeitliche Verhalten, auch als Gültigkeitsbereich bezeichnet, beschreibt den Anfangszeitpunkt, die Dauer und somit auch den Endzeitpunkt der Existenz eines Sprites. Die Darstellung basiert auf einer Timeline, deren einzelne Segmente als Frames bezeichnet werden. Die Abspielgeschwindigkeit wird in Frames pro Sekunde angegeben.

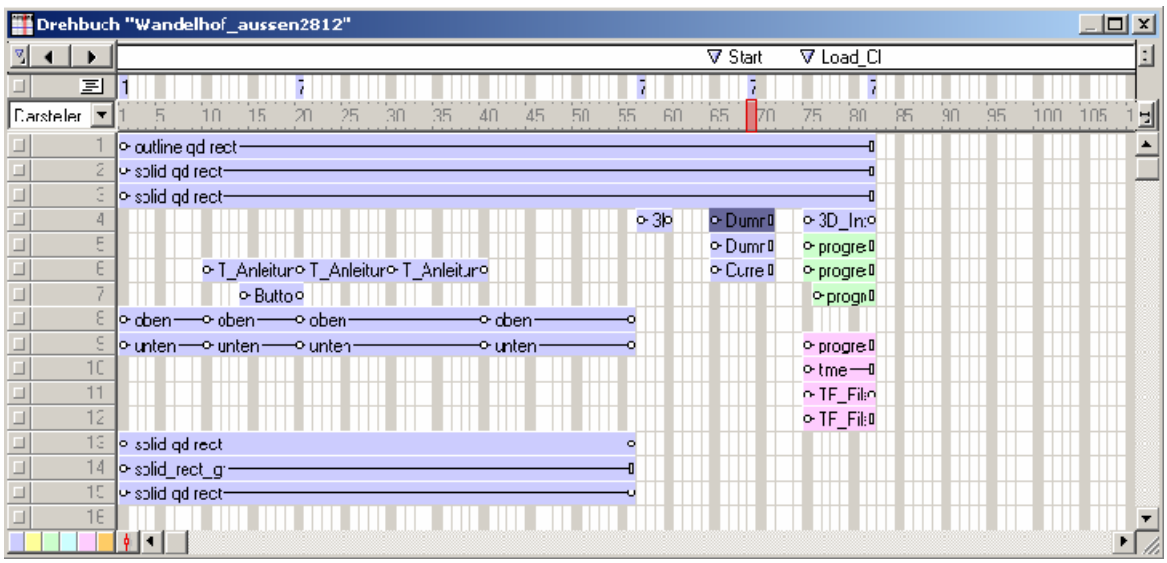



Abb. 2.4 Director-Arbeitsbereich – das Drehbuch

### 2.1.4. Der Eigenschaftsinspektor

Der Eigenschaftsinspektor (s. Abb. 2.5) dient dem Darstellen und Manipulieren verschiedener Eigenschaften. Er ändert abhängig vom darzustellenden Kontext automatisch seinen Inhalt. Das heißt, mit ihm können Sprite-, Film- und darstellerspezifische Eigenschaften angezeigt und geändert werden.

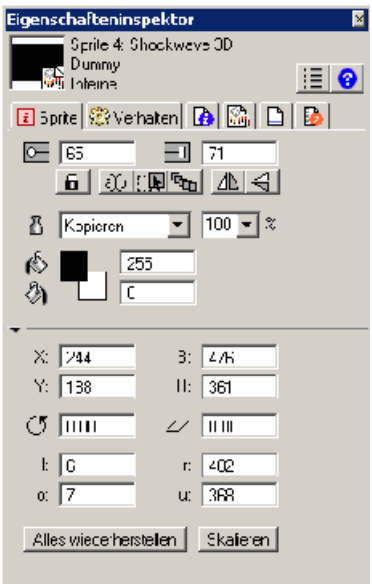



Abb. 2.5 Director-Arbeitsbereich – der Eigenschaftsinspektor

### 2.1.5. Das Skriptfenster

Director besitzt mit Lingo eine eigene, sehr umfangreiche Skriptsprache. Mit dieser Programmiersprache ist es möglich, fast alle Eigenschaften von Sprites und Darstellern zu ändern, Interaktivität zu integrieren und somit auch die Navigation innerhalb eines Filmes zu ermöglichen. Komplexe Projekte lassen sich nur mit intensiver Nutzung von Lingo realisieren. Allein für die 3D-Darstellung, die den Schwerpunkt innerhalb der praktischen Realisierung darstellt, stehen mehr als 300 neue Lingobefehle zur Verfügung, mit deren Verwendung die volle Kontrolle über eine 3D-Szene möglich ist. Im Skriptfenster (Abb. 2.6) werden Quellcodes eingegeben, auf korrekte Syntax hin geprüft und für die Fehlersuche eventuell benötige Haltepunkte gesetzt.

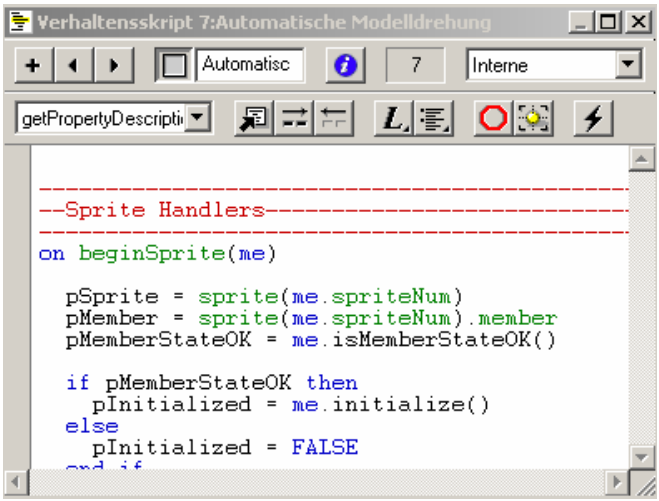



Abb. 2.6 Director-Arbeitsbereich – das Skriptfenster

### 2.1.6. Das Nachrichtenfenster

Das Nachrichtenfenster (Abb. 2.7) dient zur Ausgabe von Kontroll- und Debuginformationen während der Entwicklungsphase. Mit Hilfe des `put`-Befehls können Werte in Strings umgewandelt im Nachrichtenfenster ausgegeben werden. In umgekehrter Richtung ist es möglich, Lingobefehle im Nachrichtenfenster einzugeben und damit temporäre Veränderungen herbeizuführen.

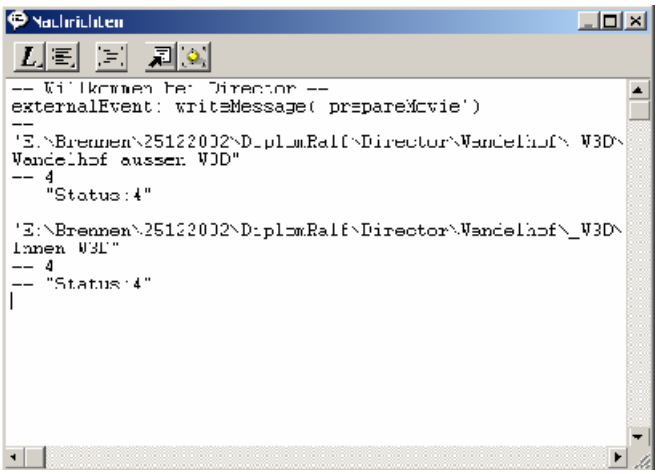



Abb. 2.7 Director-Arbeitsbereich – das Nachrichtenfenster

Außer den genannten, wichtigsten Fenstern gibt es noch eine Reihe weiterer Fenster, die z. B. für die Erstellung und Manipulation von Darstellern oder für die Fehlersuche (Debugging) genutzt werden.

## 2.2. Lingo – die Programmiersprache von Director

Mit der Entwicklung komplexer interaktiver Anwendungen ergibt sich zwangsläufig die Notwendigkeit, Steuerung und Funktionalität von Director-Filmen mit den verschiedensten Lingofunktionen anpassen und erweitern zu können.
Für die Realisierung des Prototyps wurden Lingoelemente verschiedener Kategorien verwendet:

- NetLingo (Abfrage des Netz- und Downloadstatus, Download verknüpfter Medien und Initfiles)
- ImagingLingo (Generierung dynamischer Texturen zur visuellen Vermittlung veränderlicher Informationen, Textur des Sternenhimmels)
- 3D-Lingo (Erstellung von Szeneelementen, Navigationssteuerung, Interaktionskontrolle)
- Kontroll- und Navigationsbefehle (typische Lingobefehle und Kontrollstrukturen für die Film und Ablaufsteuerung)
- Objektorientierte Lingoprogrammierung – Verwendung von Parentskripten zur Erzeugung von Instanzen mit dem Ziel der Funktionskapselung und der Wiederverwendbarkeit, genutzt für FPS-Counter, Downloadsteuerung (Loaderobjekt) und Kinoinformationsobjekt)

Grundlage für das Verhalten eines Filmes sind so genannte Nachrichten. Dabei unterscheidet man zwischen Nachrichten, die vom Betriebssystem generiert werden (`mouseDown`) und solchen, die der Kernel von Director selbst erzeugt (`enterFrame`). Alle Nachrichten können über ihren eindeutigen Namen identifiziert und in entsprechenden Skripts unter Nutzung von EventHandlern abgefangen und interpretiert werden.

Die unterschiedlichsten Objekte können Nachrichten generieren und empfangen:

- der Film selbst
- das Drehbuch
- die Sprites im Drehbuch
- die Darsteller in den Besetzungen

Die Eigenschaften der Eventhandler, ihr Typ, ihre Zuordnung und ihre Positionierung im Film bestimmen grundlegend ihr Verhalten auf Nachrichten (s. Abb. 2.8).

Die Hierarchie der Eventebenen bestimmt, welches Skript die Nachricht zuerst erhält. Nachfolgende Darstellung verdeutlicht diesen Zusammenhang.

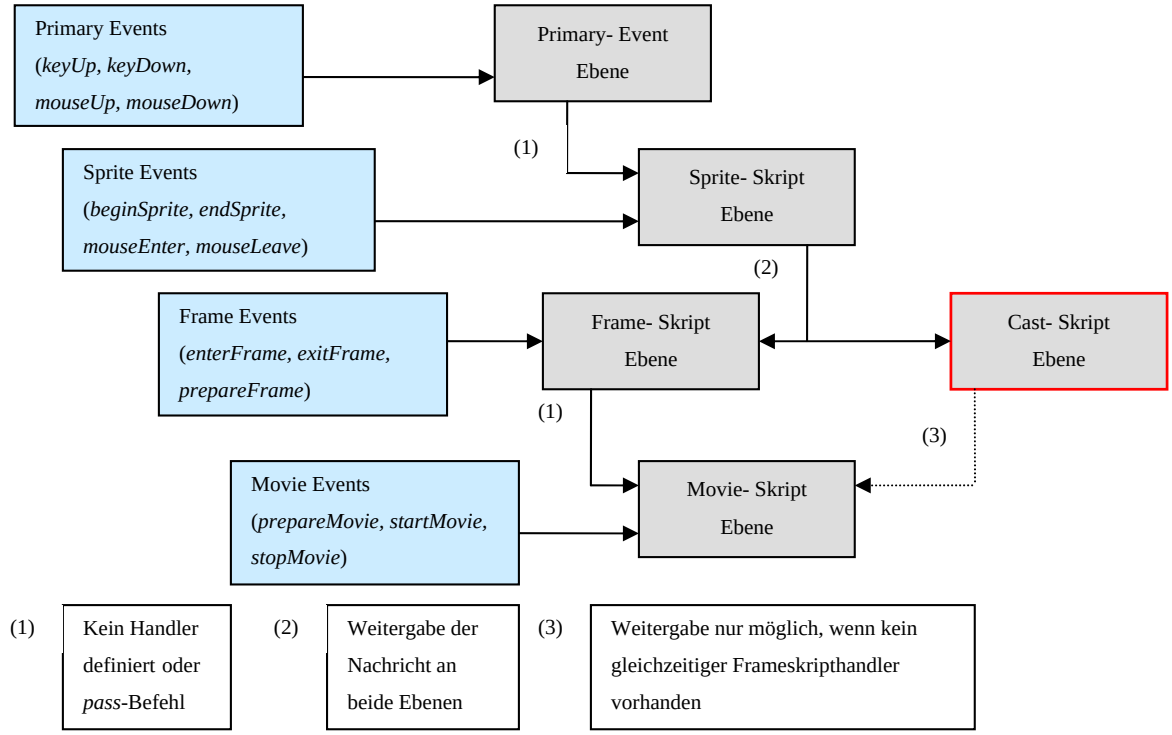



Abb. 2.8 Nachrichtenhierarchie in Director

Basierend auf dieser Hierarchie gibt es 4 unterschiedliche Skriptarten, die sich hinsichtlich ihres Gültigkeitsbereiches und der handhabbaren Nachrichten unterscheiden.

- Primary Event-Handler
- Darstellerskripte
- Verhaltensskripte
- Filmskripte

Primary Event-Handler haben die höchste Priorität. Hierbei handelt es sich um vier reservierte Skripte, die nur für den Film ihrer Deklaration Gültigkeit besitzen.

| Nachricht | Primary-Event-Handler |
|---|---|
| `keyUp` | `the keyupscript` |
| `keyDown` | `the keydownscript` |
| `mouseUp` | `the mouseupscript` |
| `mouseDown` | `the mousedownscript` |

Tabelle 12 Mögliche Nachrichten für Primary-Event-Handler

Im eigentlichen Sinn handelt es sich nicht um Skripte, sondern um Eigenschaftsvariablen. Sie können den Namen eines Filmskriptes oder ein komplettes Lingo-Skript enthalten.
Nachfolgendes Beispiel zeigt die Initialisierung eines Primary-Event-Handlers:

```
on startMovie
  the keydownscript= "alert(" && QUOTE & "Event KeyDown" & QUOTE && ")"
end
```

Listing 1 Initialisierung eines Primary-Event-Handlers

Verwendung fand das Prinzip des Primary-Event-Handlers für den Aufruf einer 2D-Director-Präsentation aus der 3D-Szene heraus, wenn eine entsprechende Taste gedrückt wird. Dieser Directorfilm wird in einem neuen Browserfenster geöffnet und soll in seiner endgültigen Version einen Navigationsplan, eine Diashow, zusätzliche Informationen in Textform und weitere Funktionen bieten. Geplant war die Taste F1 für den Aufruf, aber auf Grund der Tastaturkonkurrenz zwischen Browser und Film wird die Nachricht von beiden interpretiert und der Browser öffnet sein eigenes Hilfefenster.
Da es nach Aussage der DirectorList [www47] keinen Workaround für dieses Problem gibt, wurde für den Aufruf die Taste [h] verwendet.

Darstellerskripte haben in den letzten Jahren aufgrund der größeren Flexibilität von Verhaltensskripten an Bedeutung verloren. Es ist heute eher ein Zeichen schlechten Stils, Skripte, die an Darsteller gebunden sind, zu verwenden. Ihr Verwendungszweck kann vollständig durch die Nutzung von Verhaltensskripten abgedeckt werden.

Verhaltensskripte können im Skriptkanal des Drehbuches abgelegt oder an ein Sprite gebunden werden. Sie stellen die am häufigsten verwendete Skriptart dar. Sie bilden somit kleine logische Funktionsbausteine, welche, so sie an ein Sprite gebunden sind, nur auf Ereignisse reagieren, die im spezifischen Zusammenhang mit diesem Sprite stehen (`mouseEnter, beginSprite`). Skripte, die im Skriptkanal angebracht sind, können jedoch nicht (da sie an kein Sprite gebunden sind) auf solche spritetypischen Nachrichten reagieren.

Im Gegensatz zu den Verhaltensskripten reagieren Filmskripte global. Das heißt, sie sind während der gesamten Abspieldauer immer aktiv. Nachrichten, die nicht von einem Skript der vorherigen Ebene der Nachrichtenhierarchie behandelt wurden, werden an die nächste Event-Ebene weitergereicht und erreichen so am Ende der Hierarchie die Movie-Ebene.
Eine ganze Reihe von Nachrichten kann nur in diesen Skripten ausgewertet werden, da sie nicht im Zusammenhang mit Sprites oder dem Drehbuch generiert wurden.
Zu diesen besonderen Nachrichten gehören u. a. `prepareMovie` und `startMovie`.

## 2.3. Einführung Vektormathematik

Bevor das neue 3D-Konzept von Director vorgestellt wird, sollen nachfolgend noch einige Definitionen und mathematische Grundlagen zum besseren Verständnis beschrieben werden.

### 2.3.1. Begriffsdefinitionen

Als Scheitelpunkt (`vertex`) wird ein Punkt im 3D-Raum bezeichnet. Diese Punkte werden durch ihre Ortsvektoren (vom Koordinatenursprung zum Punkt selbst) beschrieben. Mit zwei Scheitelpunkten (`vertices`) lässt sich eine Kante eines Polygons definieren. Drei verschiedene Scheitelpunkte beschreiben drei Kanten und somit eine Dreiecksfläche (`face`). Flächen, die auf mindestens 3 Scheitelpunkten basieren, werden auch als Polygone bezeichnet. In 3D Studio MAX besteht ein Polygon aus 2 Dreiecksflächen und somit aus 4 Scheitelpunkten.
Director nutzt dagegen nur Dreiecksflächen. Das heißt, im Kontext von Director3D bezeichnen Polygone und Dreiecksflächen die gleiche geometrische Grundform. Mehrere zusammengehörige Polygone beschreiben ein Gitternetz (`mesh`). Objekte basieren auf mindestens einem Gitternetz.

### 2.3.2. Vektoren

Vektoren sind gerichtete Größen, die durch ihre Länge, ihre Richtung und ihren Betrag gekennzeichnet sind. Sie können als eindimensionale Matrizen, in Zeilen- oder Spaltenschreibweise angegeben werden. Auf Kennzeichnung für die transponierte Darstellung wird jedoch meistens verzichtet.

$$v = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad oder \quad v^T = \begin{pmatrix} x & y & z \end{pmatrix}$$

Der Ortsvektor $\overrightarrow{0P}$ ist dem Punkt P im 3D-Raum zugeordnet, d. h. sein Ursprung liegt im Koordinatenursprung und sein Ende entspricht dem Punkt P.
Zwei Vektoren werden miteinander addiert/ subtrahiert, indem die einzelnen korrespondierenden Werte addiert bzw. subtrahiert werden. Das Ergebnis ist wieder ein Vektor.

$$\begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \pm \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} = \begin{pmatrix} ax \pm bx \\ ay \pm by \\ az \pm bz \end{pmatrix}$$

Der Betrag (die Länge) eines Vektors kann über den räumlichen Lehrsatz des Pythagoras ermittelt werden.

$$\left| \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \right| = \sqrt{(ax)^2 + (ay)^2 + (az)^2}$$

Von einem Einheitsvektor (normierten Vektor, Normalenvektor) wird gesprochen, wenn der Betrag des Vektors gleich eins ist ($|a| = 1$). Soll ein Vektor normiert werden, so werden seine Vektorkomponenten durch den Betrag des Vektors dividiert.
Die Multiplikation von Vektoren wird für die Skalierung eines Vektors und für die Berechnung des Winkels zwischen zwei Vektoren benötigt.

$$s \cdot \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} = \begin{pmatrix} s \cdot bx \\ s \cdot by \\ s \cdot bz \end{pmatrix} \quad \begin{matrix} s > 1 \; positive\; Skalierung, Verlängerung\; des\; Vektor \\ 0 < s < 1 \; negativeSkalierung, Verkürzung\; des\; Vektors \\ s < 0 \; inverser, (skalierter)\; Vektor \end{matrix}$$

Werden zwei Vektoren miteinander multipliziert, so erhält man ein Skalar (Maßzahl).

$$\begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \cdot \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} = ax \cdot bx + ay \cdot by + az \cdot bz$$

Anwendung findet die Multiplikation zweier Vektoren bei der Berechnung des Winkels zwischen diesen Vektoren (Skalarprodukt, inneres Produkt zweier Vektoren).

$$\begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \cdot \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} = |a| \cdot |b| \cdot \cos \angle(a,b)$$

$$\cos \angle(a,b) = \frac{ax \cdot bx + ay \cdot by + az \cdot bz}{\sqrt{(ax)^2 + (ay)^2 + (az)^2} \cdot \sqrt{(bx)^2 + (by)^2 + (bz)^2}}$$

Mit dem Vektorprodukt (äußeres Produkt, Kreuzprodukt) zweier nicht kollinearer Vektoren ( $\vec{a}, \vec{b}$ ) lässt sich ein dritter Vektor ( $\vec{c}$ ) ermitteln, der senkrecht zu der von den beiden Vektoren aufgespannten Ebene steht.

$$\vec{c} = \vec{a} \times \vec{b} = \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \times \begin{pmatrix} bx \\ by \\ bz \end{pmatrix} = \begin{pmatrix} aybz - azby \\ azbx - axbz \\ axby - aybx \end{pmatrix}$$

Anwendung fanden die oben genannten Formeln der Vektorrechnung zum Beispiel bei der Berechnung der Korrekturrotation des Kino-Informationsobjektes (siehe Abschnitt *3.2.2.2. Rotation des Kino-Informationsobjektes*).

### 2.3.3. Matrizen

Matrizen werden zur Manipulation von Vektoren verwendet. Damit eine einheitliche Beschreibung aller geometrischen Transformationen in einer 4x4 Matrix möglich ist, wird ein Vektor um eine vierte Komponente (w), die in der Regel immer auf 1 gesetzt wird, erweitert. Diese Erweiterung der Darstellung nennt man homogene Koordinaten.

Vektoren werden durch Matrizen manipuliert, indem sie mit dieser multipliziert werden. Das Ergebnis dieser Operation ist wieder ein Vektor.
Der Vorteil in der Verwendung von Matrizen liegt darin, dass diese miteinander kombiniert, d. h. miteinander multipliziert werden können. Werden mehrere Transformationen hintereinander ausgeführt, so erhält man durch die Multiplikation der Matrizen eine resultierende Matrix, welche die gesamten Transformationen beschreibt. Diese Matrix kann für die Manipulation aller betroffenen Vektoren verwendet werden. Mit der Zusammenfassung einer Reihe von Transformationen in einer Zielmatrix und der Anwendung dieser auf eine große Anzahl von Vektoren lässt sich entsprechend viel Rechenzeit einsparen.

Für die Matrizenoperationen  werden 6 verschieden Matrizentypen benötigt.

Grundlagen für die Matrizenoperation ist eine Matrix, die keine Änderungen an einem Vektor oder an einer Matrix hervorruft wenn sie angewendet wird. Diese Matrix wird als Einheitsmatrix (Abb. 2.9) bezeichnet.

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Abb. 2.9 Die Einheitsmatrix

Die Verschiebung (Translation) eines Vektors (Abb. 2.10) kann als Vektoraddition aufgefasst werden.

$$M = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Abb. 2.10 Die Translationsmatrix

Eine Skalierung (Abb. 2.11) verschiebt den durch den Vektor definierten Punkt auf der Geraden, die durch den Koordinatenursprung und den Endpunkt des Vektors definiert wurde.

$$M = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Abb. 2.11 Die Skalierungsmatrix

Eine Rotationsmatrix beschreibt eine Drehung um einen bestimmten Winkel um den Ursprung. In Abhängigkeit der Rotationsachse gibt es verschiedene Rotationsmatrizen (Abb. 2.12):

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha x) & -\sin(\alpha x) & 0 \\ 0 & \sin(\alpha x) & \cos(\alpha x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M = \begin{pmatrix} \cos(\alpha y) & 0 & \sin(\alpha y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha y) & 0 & \cos(\alpha y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M = \begin{pmatrix} \cos(\alpha z) & -\sin(\alpha z) & 0 & 0 \\ \sin(\alpha z) & \cos(\alpha z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Abb. 2.12 Matrizen für Rotation um x-,y- und z-Achse

Im Abschnitt *2.4.4. Transformation von Objekten* wird kurz auf die interne Verwendung von Matrizen und deren Manipulation in Director sowie auf die Relevanz für das Projekt eingegangen.

## 2.4. Einführung in Director3D

Mit Director® 8.5 Shockwave® Studio für 3D hat Macromedia das meistverbreitete Autorenwerkzeug für die Erstellung von interaktiven Online-/Offlinemedien um die Fähigkeit der skalierbaren 3D-Echtzeitdarstellung erweitert. Die implementierte 3D-RenderEngine basiert dabei auf der von Intel Architecture Labs [www48] entwickelten Intel Internet 3D Graphics-Technologie [www49].

Zum besseren Verständnis der weiteren Ausführungen soll im folgenden Abschnitt die grundlegende Funktionalität von Director3D dargestellt werden.

Innerhalb des Projektes gibt es zwei verschiedene Situationen mit Elemente, die nicht in einem externen 3D-Modellierungswerkzeug (z. B. 3D Studio MAX) erstellt und anschließend exportiert wurden, um sie in Director zu verwenden.

In der Preloadersequenz, bevor also die eigentliche 3D-Szene angezeigt werden kann bzw. in Situationen, in denen Szenenelemente nachgeladen werden sollen, wird ein würfelähnliches Objekt (s. Abb. 2.13) dargestellt.

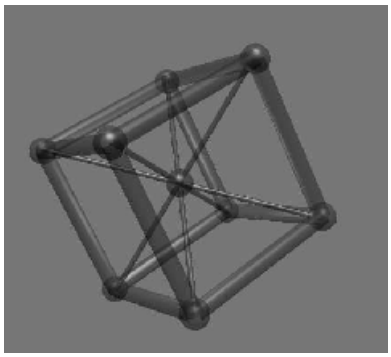


Abb. 2.13 Das Preloaderelement

Dieses besteht aus einer Reihe von Zylindern und Kugeln, die per 3D-Lingo generiert, im Raum angeordnet, hierarchisch gruppiert und animiert wurden. Die Vorgehensweise, um ein solches Objekt mit den neuen 3D-Lingobefehlen zu erstellen, soll in den folgenden Kapiteln als Einführung in die Shockwave3D-Welt veranschaulicht werden.

Ein zweiter Anwendungsfall im Projekt für die Generierung eines komplexen 3D-Objektes per Lingo wird im Abschnitt *3.2.2. Das Kino-Informationsobjekt* ausführlich dargestellt.

### 2.4.1. Shockwave3D-Darsteller, Koordinatensystem und Orientierung

Für die 3-dimensionale Visualisierung in Director wurde ein neuer 3D-Darstellertyp eingeführt. Dieser stellt die 3D-Welt dar und kapselt in sich alle Komponenten, die zur Beschreibung der 3D-Welt benötigt werden. Nachfolgende Abbildung (Abb. 2.14) soll zur weiteren Erläuterung der Zusammenhänge zwischen den einzelnen Komponenten und ihrer funktionellen Bedeutung genutzt werden.

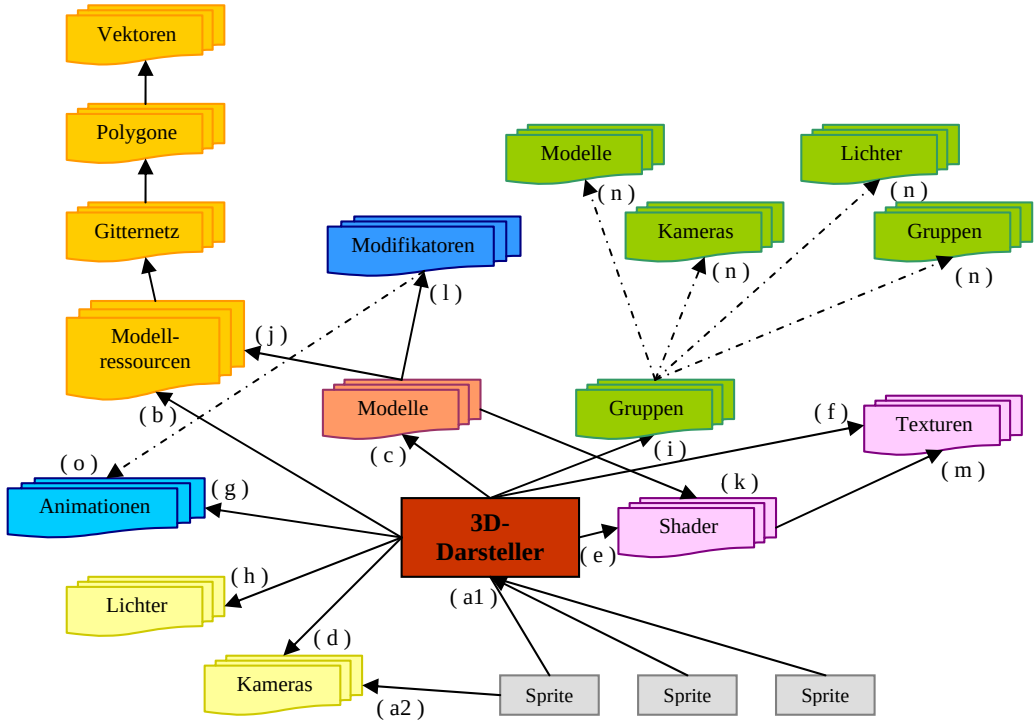



Abb. 2.14 Internes Organisationsprinzip einer 3D-Darstellung

Die einzelnen Komponenten, die für eine 3D-Darstellung in Director benötigt werden, können hinsichtlich ihrer Grundfunktion teilweise entsprechend zusammengefasst werden. Deshalb wurden funktional zusammengehörige bzw. voneinander abhängige Elemente durch eine gemeinsame farbliche Hervorhebung dargestellt. Die Pfeile stellen Referenzierungen zwischen den einzelnen Komponenten dar, deren Realisierungen per Lingo in Tabelle 13 jeweils mit einem Code-Beispiel angegeben werden.

| | |
|---|---|
| Sprite | Ein Sprite ist ein zeitlicher Repräsentant eines (3D-) Darstellers. Im 3D-Kontext stellt es einen Blick durch eine Kamera mit einem definierten Blickwinkel auf die Szene des 3D-Darstellers dar. |
| | (a1) `sprite(Index).member = member(3D_MemberName)`<br>(a2) `sprite(Index).camera = member(3D_MemberName).camera[Index]` |
| 3D-Darsteller | Ein 3D-Darsteller ist das zentrale Element der Director3D-Philosophie. Er kapselt alle Komponenten, die zur Beschreibung einer 3D-Szene benötigt werden. |
| | Der Zugriff erfolgt über den Namen oder den Index der Komponente.<br>(b) `member(3D_MemberName).modelresource[Index]`<br>(c) `member(3D_MemberName).model[Index]`<br>(d) `member(3D_MemberName).camera[Index]`<br>(e) `member(3D_MemberName).shader[Index]` |

| | |
|---|---|
| | (f) `member(3D_MemberName).texture(textureName)`<br>(g) `member(3D_MemberName).motion(motionName)`<br>(h) `member(3D_MemberName).light(lightName)`<br>(i) `member(3D_MemberName).group(groupName)` |
| Modelle | Modelle sind Instanzen von Modellressourcen. Nur sie sind in der Shockwave-3D Welt sichtbar. Zu ihrer Beschreibung werden Modellressourcen, Shader und ggf. Animationen benötigt. |
| | (j) `model(modelName).modelresource`<br>(k) `model(modelName).shaderList`<br>(l) `model(modelName).modifier[Index]` |
| Modellressourcen | Modellressourcen beschreiben die Geometrie von Objekten und bilden die Grundlage für das Erstellen von Modellen. Sie basieren auf Gitternetzen, die sich wiederum aus Polygonen, die mittels Vektoren definiert werden, zusammensetzen. |
| Lichter | Lichter beleuchten die Szene und sorgen somit für einen räumlichen Eindruck. Sie können aufhellen und entgegen der Realität auch abdunkeln. |
| Kameras | Kameras sind der „Blick“ in die 3D-Welt. Sie zeigen einen Ausschnitt aus der gesamten Szene. Zusammen mit der Beleuchtung bestimmen sie, was wie wahrgenommen wird. |
| Shader | Shader definieren das Aussehen der Modelloberfläche (Farbe, Glanzlichter, etc.). Die Darstellungseigenschaften eines Shaders werden durch Renderstyle (`#fill, #wire, #point`) und Shadertyp (`#standard, #painter, #engraver, #newsprint`) bestimmt. |
| | (m) `shader(shaderName).textureList[Index]` |
| Texturen | Texturen bestimmen im Zusammenspiel mit Shadern das Aussehen von Modellen. Sie basieren auf Bildobjekten und können innerhalb eines Shaders kombiniert werden. |
| Gruppen | Gruppen dienen der hierarchischen Zusammenfassung von mehreren Elementen. Folgende Knotenelemente können zusammengefasst werden: Modelle, Kameras, Lichter oder andere Gruppen. |
| | (n) `group(groupName).child(Index)` |
| Modifikatoren | Modifikatoren erweitern die Funktionalität in einer 3D-Szene. Sie ermöglichen Knochen- und Schlüsselbildanimationen, Kollisionserkennung, Geometrie- und Detailveränderungen sowie alternative Rendermodi (`#inker, #toon`) im zeitungs- oder comicähnlichen Stil. |
| | (o) `model(modelName).keyframePlayer.playList` |
| Animationen | Animationen ermöglichen Veränderungen eines Modells über die Zeit. Schlüssel-bildanimationen erzeugen Positions-, Ausrichtungs- und Skalierungsveränderungen. Knochenanimationen dienen der komplexen Bewegung verknüpfter 3D-Objekte mit Hilfe eines hierarchischen Skeletts – individuelle Teile eines 3D-Objektes werden zusammen realitätsnah animiert |

Tabelle 13 Komponenten eines 3D-Darstellers und ihre Funktionen

Dem Entwickler offeriert Director zwei verschiedene Möglichkeiten einen 3D-Darsteller und somit die Basis für eine virtuelle Welt zu generieren:

- Erstellung der Szene mit einem externen 3D-Werkzeug; anschließender Export in das W3D- oder OBJ-Format (OBJ-Dateien müssen über ein entsprechendes Director-Xtra noch in das W3D-Format konvertiert werden) und nachfolgender Import der Datei (eingebettet oder extern verknüpft) in Director.
- Erstellung eines 3D-Darstellers im Shockwave3D-Fenster; indem eine Kameratransformation eingestellt wird oder per Lingo-Befehl (`pMember = new(#shockwave3D)`. Anschließende Generierung/ Manipulation der Szenenkomponenten mittels verschiedener 3D-Lingo Befehle.

Es bietet sich an, beide Varianten gemeinsam zu verwenden, d. h. die Szene wird mit einem 3D-Werkzeug modelliert und kann dann durch 3D-Lingo-Befehle vollständig kontrolliert werden.

Wird eine W3D-Datei extern verlinkt, so ergeben sich einige Probleme, die im Aufbau einer solchen Datei begründet sind. Eine W3D-Datei besteht aus einem Kopfteil, dem so genannten Initial Loader Segment (kurz ILS) und einem Rumpfteil, der die eigentlichen Geometrie-, Textur- und Animationsdaten enthält. Wenn ein verknüpfter 3D-Darsteller in den Speicher geladen wird, so verändert sich die Darstellereigenschaft `state` während des Vorgangs (s. Abb. 2.15).

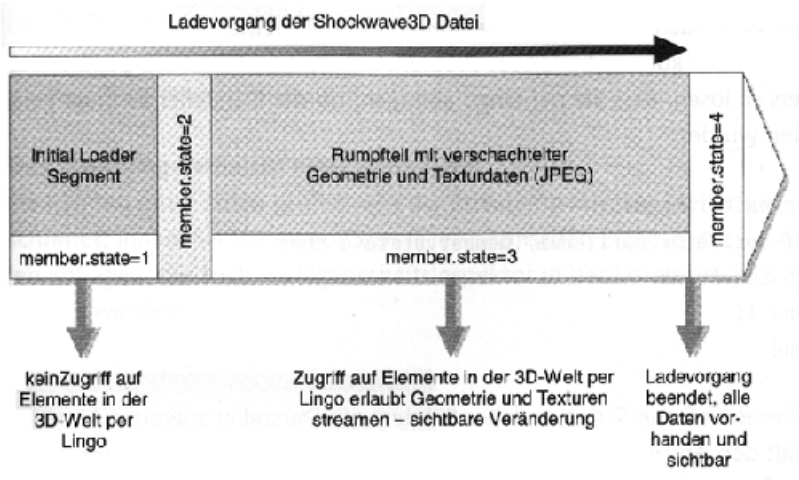



Abb. 2.15 Aufbau eines W3D- Files [BIE02]

Erst wenn der Wert von `member.state` größer bzw. gleich 2 ist (Ladevorgang des Kopfteils beendet), können Lingobefehle auf Szenenelemente angewendet werden. Gestreamt werden kann nur der Rumpfteil. Wird er geladen, so wird die Szene nach und nach aufgebaut.

Um den Rendervorgang eines 3D-Darstellers anzustoßen, muss eine Instanz des Darstellers im Drehbuch existieren. Eine solche Instanz wird als Sprite bezeichnet. Sie ist ein zeitlicher Repräsentant des Darstellers und stellt einen Blick durch eine Kamera mit einem definierten Blickwinkel auf die Szene des 3D-Darstellers dar. Die Erstellung eines Sprites, das auf einem Shockwave3D-Darsteller beruht, kann wie bei jedem anderen Darstellertyp auch, zur Entwurfszeit erfolgen, indem man den Darsteller auf die Bühne zieht und seine Position sowie die zeitliche Präsenz festlegt oder indem der Darsteller eines Sprites zur Laufzeit mit folgendem Befehl gewechselt wird.

```
sprite( me.spriteNum ).member = member("Preloader")
```

Listing 2 Zuordnung eines Darstellers zu einem Sprite

Ein Beispiel für eine solche dynamische Zuordnung ist der Einsatz von Dummysprites, die die Position und das zeitliche Verhalten vorgeben und erst zur Laufzeit den eigentlichen 3D-Darsteller zugeordnet bekommen. Diese Methode ermöglicht es, die Downloadgröße des Films gering zu halten und 3D-Darsteller zur Laufzeit zu erzeugen, nachzuladen und zuzuordnen. Das Prinzip wird im Projekt dahingehend verwendet, dass nur ein Dummysprite im Drehbuch als Referenz für das eigentliche 3D-Sprite existiert. Dieses kann jedoch erst dann eingesetzt werden, wenn der Downloadvorgang des zugehörigen 3D-Darstellers erfolgreich beendet wurde.

Zum besseren Überblick über die Orientierung der Szene im Raum kann sich der Entwickler die Welt- und Modellachsen anzeigen lassen.

```
Member("new3D_Member").debugFlags = 130
```

Listing 3 Anzeige der Koordinatenachsen in einem 3D-Darsteller

Wurde der 3D-Darsteller mit Director erzeugt und besitzt die Kamera ihre Standardposition und -orientierung, zeigt die positive x-Achse (rot) nach rechts, die y-Achse (grün) nach oben und die positive z-Achse (blau) auf den Betrachter.

Die beiden nachfolgenden Darstellungen (Abb. 2.16 und 2.17) sollen die Orientierung bzw. die positiven Drehrichtungen verdeutlichen.

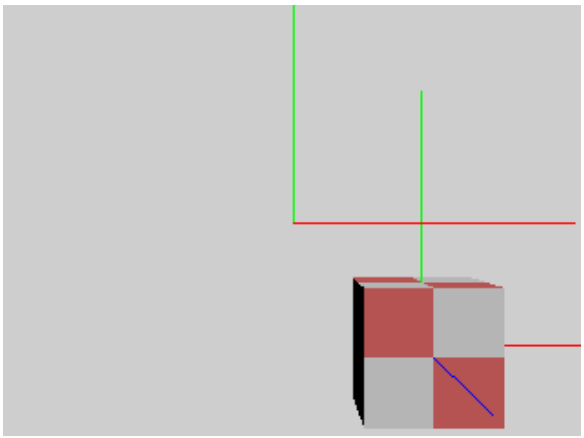


Abb. 2.16 Koordinatensystem und Ausrichtung

Jedes Modell besitzt ein eigenes, lokales Koordinatensystem. Wenn ein Modell nicht rotiert wurde, so stimmen die Ausrichtungen der Achsen des lokalen Koordinatensystems mit denen des Weltkoordinatensystems, kurz WKS, überein. Im Beispiel der Abb. 2.17 wurde der Würfel in positiver Richtung der x- und in negativer Richtung der y-Achse verschoben. Aufgrund der räumlichen Verzerrung ist die lokale z-Achse im Gegensatz zu der des WKS sichtbar. Die z-Achse des WKS ist nur als Punkt sichtbar, da der Orientierungsvektor der Standardkamera (`vector(0,0,-1)`) der negierten z-Achse des WKS entspricht. Mit der Initialposition (`vector(0,0,250)`) der Standardkamera blickt diese somit entlang der z-Achse auf das Zentrum des Weltkoordinatensystems.

Director verwendet standardmäßig für die 3D-Darstellung ein rechtshändiges Koordinatensystem, d. h. die z-Achse zeigt nach vorn. Daraus resultiert die Definition der positiven Drehrichtung, welche für die Rotation der Objekte entscheidend ist.

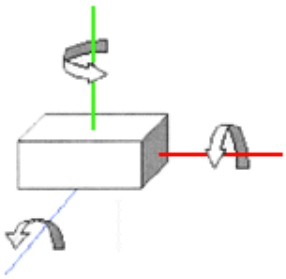


Abb. 2.17 Definition positive Drehrichtung

Darsteller, die auf einer exportierten W3D-Datei basieren, haben natürlich die Orientierung der Szene innerhalb des Erstellungswerkzeuges übernommen.

Director nutzt für die 3D-Darstellung nur eine quantitative Maßeinheit. Verwendete Größen werden in Einheiten, ohne weitere Zusätze wie Zentimeter oder Meter, angegeben (beim Export in 3D Studio MAX erhält man jedoch eine Fehlermeldung, wenn die Basiseinheit nicht Zoll ist).

### 2.4.2. Definition geometrischer Primitive

Für die Modellierung einer 3D-Szene zur Laufzeit bietet Director folgende Grundkörper an:

- Ebene
- Quader
- Kugel
- Zylinder
- Partikelsystem
- 3D-Text
- Gitternetz

Basis aller geometrischen Objekte einer Szene bilden die Modellressourcen. Sie definieren den geometrischen Typ, die Geometrie selbst und damit verbunden die notwendigen Eigenschaften der Primitive, z. B. Höhe, Breite oder Anzahl der Polygone.

Das Preloaderobjekt im Projekt besteht aus Kugeln, die alle auf einer Modellressource basieren und Zylindern, die Instanzen zweier verschiedener Ressourcen sind, die sich hinsichtlich Radius und Länge unterscheiden. Die oberen und unteren Seiten der Zylinder werden nicht dargestellt.

```
mrCylinder=pMember.newModelResource("mrCylinder",#cylinder,
#front)
mrCylinder.topradius=1
mrCylinder.bottomradius=1
mrCylinder.topCap=False
mrCylinder.bottomCap=False
mrCylinder.height=2.0*sqrt(3*length*length)
```

Listing 4 Modellressource für raumdiagonale Zylinder

```
mrSphere=pMember.newModelResource("mrSphere",#sphere,#front)
mrSphere.radius=5
mrSphere.resolution=50
```

Listing 5 Modellressource für die Kugeln

Sind die Modellressourcen definiert, lassen sich darauf aufbauend die Modelle erstellen. Modelle sind Instanzen von Modellressourcen mit jeweils bestimmten Positionen im Raum. Sie sind die sichtbaren Elemente einer Szene. Erzeugt wird ein Modell wie folgt:

```
mCylinder1=pMember.newModel("mCylinder1", mrCylinder)
```

Listing 6 Erstellung eines Modells

### 2.4.3. Oberflächeneigenschaften, Shader und Texturen

Shader bestimmen das visuelle Erscheinungsbild eines Modells, sie definieren die optischen Eigenschaften der Oberflächen von Objekten. Jedes Modell besteht aus mindestens einem Gitternetz, eine Kugel besitzt standardmäßig ein (gewölbtes) Gitternetz, wohingegen sich ein Quader aus sechs Gitternetzen zusammensetzt. Diese Gitternetze besitzen jeweils eine Referenz auf einen möglicherweise gemeinsam genutzten Shader. Ein Shader kann bis zu 8 Texturen beinhalten, die miteinander kombiniert werden können.
Die Einstellungen hinsichtlich der Texturen, des Verhaltens auf Beleuchtung (diffuse/ hochglänzende Reflektionen) und die Art des Renderns (gefüllte Oberfläche, Drahtgitter- oder Punktdarstellung) bestimmen das Aussehen des Modells. Mit der Instanzierung erhält jedes 3D-Modell einen Verweis auf den Standardshader mit dessen Standardtextur (rot-weißes Schachbrettmuster).

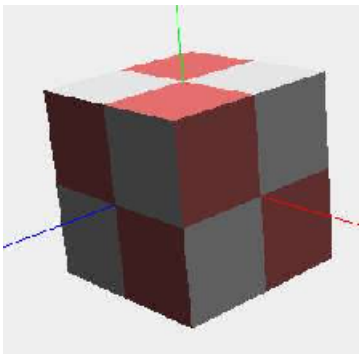


Abb. 2.18 Würfel mit Standardshader

Für das Preloaderobjekt wurde ein einheitlicher Shader für alle Elemente definiert. Als Textur wird ein so genanntes Imageobjekt per Lingo erzeugt und mit einem dunklen Grauton gefüllt. Dieses Imageobjekt wird anschließend verwendet, um eine neue Textur zu definieren. Dem Shader kann diese Textur entsprechend zugeordnet werden.

```
gridImg=image(32,32,24,0)
gridImg.fill(0,0,31,31,rgb(50,50,50))
tx_plane=pMember.newTexture("tex1",#fromImageObject,gridImg)
shd1.textureModeList[1]=#wrapPlanar
shd1=pMember.newShader("shaderCubeElements",#standard)
shd1.texture=tx_plane
shd1.blend=50
```

Listing 7 Textur- und Shadergenerierung

Die Zuordnung des soeben definierten Shaders zu einem Modell erfolgt nach folgender Syntax:

```
pMember.model("CenterSphere").shader=shd1
```

Listing 8 Zuordnung eines Shaders zu einem Modell

### 2.4.4. Transformation von Objekten

Die Transformation von Objekten realisiert Director intern mit Transformationsmatrizen, die sich mit entsprechendem Aufwand auch manuell manipulieren lassen. Allerdings ist das Arbeiten mit solchen Matrizen wenig intuitiv. Deshalb existieren entsprechende Funktionen für Rotation, Translation und Skalierung.
Innerhalb des Projektes ergab sich nicht die Notwendigkeit, Transformationsmanipulationen auf der Basis von manuellen Matrizenveränderungen zu realisieren. Alle Transformationen konnten mit den nachfolgend dargestellten Funktionen programmiert werden.

```
member(whichCastmember).node(whichNode).translate\
        (xIncrement,yIncrement,zIncrement{,relativeTo})
member(whichCastmember).node(whichNode).translate\
        (translateVector{,relativeTo})
```

Listing 9 Allgemeine Form der Translation

```
member(whichCastmember).node(whichNode).rotate(xAngle,yAngle,zAngle{,relativeTo})
member(whichCastmember).node(whichNode).rotate(rotationVector{,relativeTo})
member(whichCastmember).node(whichNode).rotate(position,axis,angle{,relativeTo})
```

Listing 10 Allgemeine Form des Rotationskommandos

```
member(whichCastmember).node(whichNode).scale(xScale,yScale,zScale)
member(whichCastmember).node(whichNode).scale(uniformScale)
```

Listing 11 Allgemeine Form des Skalierungsbefehls

Die räumliche Anordnung der Elemente des Preloaderobjektes wurde realisiert, indem zuerst eine Liste der normierten Ortsvektoren der Translation für jedes zu verschiebende Element erstellt wurde. Anschließend wurden die Modelle erzeugt und die notwendigen Translationen auf dieser Liste basierend zugeordnet (siehe Listing 43 Erstellung und Translation der Szenenmodelle).

### 2.4.5. Hierarchische Strukturen, Parent-Child-Beziehungen

Parent-Child-Beziehungen sind dann sehr nützlich, wenn mehrere Modelle eine logische, hierarchisch gegliederte Einheit bilden. Ohne diese Funktionalität müssten die Animationen der einzelnen Elemente „per Hand“ relativ zu ihren übergeordneten Elementen berechnet werden. Stattdessen werden hierarchische Ketten definiert, die es erlauben, dass die Kindelemente automatisch den Bewegungen ihrer Parents relativ folgen.
Das Standard-Parent für jedes mit Lingo erstellte Objekt ist der oberste Knoten der Hierarchie – die Welt. Dieses Parent wird als `group("World")` referenziert. Eine hierarchische Zuordnung erfolgt wie im nachfolgenden Beispiel beschrieben:

```
-- Zuordnung eines Parents zu einem Modell
pMember.model("Sphere1").parent=pMember.model("CSphere")
-- oder einem Modell,Camera, Licht ein untergeordnetes Objekt zuordnen
pMember.camera("Kamera01").addChild(pMember.model("Boundingsphere"))
```

Listing 12 Erstellung einer Parent-Child-Beziehung

Im Beispiel des Preloaders wird allen geometrischen Primitiven die mittlere Kugel als Parent zugeordnet, wobei diese aber als Parent die Welt behält.

### 2.4.6. Animation der hierarchischen Anordnung

Director bringt mit seiner Bibliothek eine Reihe von vorgefertigten Standardverhalten mit. Darunter befinden sich u. a. auch Verhalten für eine automatische Modelldrehung. Diese lassen sich per Drag&Drop einem Sprite, das einen 3D-Darsteller als Member besitzt, zuordnen. Dialogbasiert können Einstellungen hinsichtlich des zu rotierenden Modells, der Rotationsachse und des Rotationswinkels vorgenommen werden. Die vorgefertigten Behaviors bieten ihren Quellcode offen an. Das versetzt den Entwickler in die Lage, deren Funktionalität einzusehen und gegebenenfalls an seine Bedürfnisse anzupassen. Die Funktionsweise des Skriptes lässt sich verkürzt so darstellen: Im `enterFrame`-Handler wird entsprechend den vorgenommenen Voreinstellungen die Funktion für die Rotation des Modells (oder aller Modelle) aufgerufen. Dabei erfolgt die Rotation bezogen auf das objekteigene Koordinatensystem um die festgelegte Achse mit dem definierten Rotationsoffset. In der Beispielszene wird nur die mittlere Kugel animiert. Auf Grund der hierarchischen Anordnung der restlichen Modelle erfolgt deren Bewegung relativ (bezogen auf die mittlere Kugel), d. h. sie kreisen mit ihrem originalen Abstand und ihrer Ausrichtung um das Rotationszentrum.

### 2.4.7. Ein unverzichtbares Tool für den Überblick – 3DPI

Ein ausgesprochenes Defizit von Director3D ist ein Tool, das Übersichts- und Einstellungsmöglichkeiten für all jene Elemente bietet, die in einem 3D-Darsteller eingebettet sind. Mit den Bordmitteln von Director ist eine räumliche Analyse nur sehr umständlich und wenig anschaulich zu realisieren. Nur wenige Eigenschaften sind mit dem Eigenschaftsinspektor bzw. über das Shockwave3D-Fenster einstellbar. Die anderen Informationen können nur über das Nachrichtenfenster abgefragt und gesetzt werden. Diese Vorgehensweise ist recht umständlich, wenig übersichtlich und meistens nicht ausreichend. Einfache Abfragen, wie zum Beispiel die Auflistung aller Modelle und ihrer Namen, sind damit nicht realisierbar. Abhilfe schafft der 3D-Property-Inspector, kurz 3DPI (Abb. 2.19) genannt – ein unverzichtbares Tool für die Analyse aller Elemente eines 3D-Darstellers.

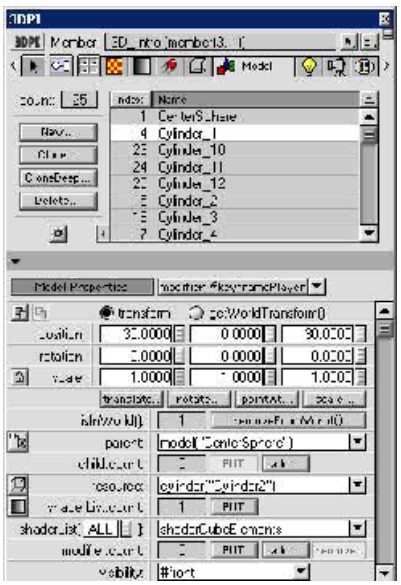



Abb. 2.19 Übersicht Modelleigenschaften im 3DPI

Dieses Shareware Xtra wurde von Ulla Gusenbauer entwickelt und wird auch von ihr gepflegt. Die aktuelle Version kann unter [www50] heruntergeladen werden. Das Werkzeug fasst die verschiedenen Paletten eines 3D-Darstellers in Karteireitern (s. Abb. 2.20) zusammen.


Abb. 2.20 Die einzelnen Kategorien des 3DPI

Somit ist eine übersichtliche Anordnung folgender Elemente garantiert:

- Modellressourcen
- Modelle
- Shader
- Texturen
- Animationen
- Kameras
- Lichter
- Gruppen und
- Havokfunktionen

Hinzu kommen noch Reiter für das Picken von Szenenelementen, eine 3D-Spriteübersicht sowie 3D-Darstellerübersicht und eine Auflistung der Eigenschaften des aktuellen Renderers. Alle Eigenschaften der 3D-Elemente eines Darstellers, die per Lingo gelesen werden können, werden von 3DPI aufgelistet. Zusätzlich können verschiedene Informationen über die Grafikkarte abgefragt werden (z. B. unterstützte Rendermodi, Hersteller). Neben den Eigenschaften, die Nur-Lese-Charakter besitzen, können die meisten Eigenschaften gelesen und gesetzt werden. Einstellungen, die mit 3DPI vorgenommen werden, können am 3D-Darsteller und dessen Instanz auf der Bühne sofort verifiziert werden.

Zusätzlich bietet das Tool eine Reihe von objektspezifischen Funktionen an. Dazu gehören zum Beispiel:

- Erzeugen
- Löschen
- Ausrichten
- Klonen von Objekten
- Hinzufügen und Manipulieren von Modifikatoren

Als sehr nützlich erweist sich die Möglichkeit, Änderungs- und Auflistungsbefehle in Lingosyntax im Nachrichten-Fenster auszugeben. Diese Ausgabe kann in angepasster Form in eigene Skripte übernommen werden, um Änderungen an der Szene permanent vorzunehmen. Ohne dieses sehr hilfreiche Werkzeug lassen sich umfangreiche Projekte, wie die Visualisierung des Entertainmentkomplexes Wandelhof Schwarzheide, nicht realisieren.

# 3. Modellierung der Szene


Abb. 3.1 Ansicht Haupteingang Wandelhof Schwarzheide

Für die Modellierung des Komplexes Wandelhof Schwarzheide (s. Abb. 3.1) konnte nicht wie geplant auf bereits digitalisiertes Material in Form von CAD-Zeichnungen, 3D-Architekturentwürfen oder ähnliches zurückgegriffen werden. Der Betreiber war nicht in der Lage, entsprechendes Material zur Verfügung zu stellen. Zu diesem Zeitpunkt war aber die Aufgabenstellung für die Diplomarbeit schon festgelegt und bestätigt. Somit ergab sich ein zusätzlicher, nicht geplanter und nur schwer einschätzbarer Mehraufwand für eine nachträgliche Digitalisierung der Architektur und Inneneinrichtung. Viele Details, Größenverhältnisse und räumliche Eindrücke sind nur schwer reproduzierbar.

Basis für die nachträgliche Digitalisierung bildeten ein Grundriss des Erdgeschosses, der die Elektroinstallation repräsentiert und keine Höheninformationen beinhaltet sowie eine Reihe von Bildern, die zu einem späteren Zeitpunkt zur Unterstützung der Modellierung aufgenommen wurden. Außerdem stellte der Betreiber später noch Videos bereit, die mit einer einfachen Handkamera produziert waren. Die Qualität der Videos war jedoch so schlecht, dass nur wenige Informationen in die weitere Modellierung einfließen konnten.

Auf Grund fehlender detaillierter Pläne wurde die Szene mit 3D Studio MAX [www51] subjektiv nachmodelliert. Dabei wurde die Erstellung der Szene von 4 Zielen bestimmt:

- Erstellung einer kompletten Außenansicht unter Beachtung gegebener Größenverhältnisse
- Implementierung wichtiger Details, die eine Wiedererkennung fördern
- strukturierte Modellierung für Aufteilung der Gesamtszene in verschieden Teilszenen und damit möglicher Export in separate W3D-Files
- optimale Darstellungsqualität bei minimaler Downloadgröße

## 3.1. Das Modellierungswerkzeug 3D Studio MAX

Die optimale Modellierung einer Szene für die Verwendung in Shockwave erfordert viel Erfahrung und bringt eine Reihe von potentiellen Fehlerquellen mit sich. Im Umgang mit 3D Studio MAX sind hinsichtlich der Verwendung von exportierten Dateien in Shockwave3D einige Besonderheiten zu beachten, die, so sie für das Projekt von Bedeutung waren, nachfolgend beschrieben werden.

### 3.1.1. Der Export einer 3D-Szene

Der Hersteller von 3D Studio MAX bietet mittlerweile ein eigenes W3D-ExportPlug-in an, das es ermöglicht, eine 3D-Szene direkt in das W3D-Format zu exportieren. Der Umweg über das Wavefront-/ OBJ-Format, wie es in [GRO00] noch beschrieben ist, entfällt damit.

Einschränkungen, die dem W3D-Format zu Grunde liegen, führen dazu, dass eine Reihe von Funktionen und Objekten, die 3D Studio MAX unterstützt, nicht exportiert werden können. Dies ist bei der Modellierung entsprechend zu beachten.

Die Funktion *Auswahl exportieren* exportiert die gesamte Szene, aber nicht die ausgewählten Elemente. Außerdem werden alle Objekte einer Szene, die sichtbaren und die verdeckten, exportiert. Das bedeutet: Objekte, die benötigt werden, aber vorerst unsichtbar sein sollen, müssen per Lingo-Befehl vom Rendervorgang ausgeschlossen werden.

```
Member("3Dmem").model("hiddenModel").removeFromWorld()
```

Listing 13: Entfernen eines Modells aus der RenderList

2D-Objekte können nicht exportiert werden, d. h. die Kamerafahrt der Introsequenz entlang eines vorgegebenen Pfades mittels eines Position-Path-Controllers konnte auf diese Weise nicht realisiert werden. Die Animation der Kamerafahrt hätte aus diesem Grund als Keyframeanimation durchgeführt werden müssen. Auf das aufwendige Setzen der Keyframes wurde deshalb auch im Hinblick auf die Downloadgröße verzichtet. Stattdessen erfolgt die Steuerung und Animation der Kamera komplett per Lingo.

Rechenintensive Operationen (Volumenlicht, Schatten und Partikel) werden vom Exporter nicht unterstützt. Jedoch können Partikelemitter per Lingo erstellt werden.

Objekte, die als Gruppe zusammengefasst und selbst keine Gruppe sind, werden zu einem Objekt vereint, das bedeutet, sie können in Director nicht mehr getrennt manipuliert werden.

Für korrekte Knochenanimationen (hierarchische Animationen) müssen die an die Knochen gebundenen Objekte einzeln gruppiert und damit deren Schwerpunkte auch neu positioniert werden (vgl. dazu auch [NAU01]).

### 3.1.2. Detailgenauigkeit und Geometrieauflösung

Als eines der größten Probleme hat sich das Erreichen einer Balance zwischen der Detailgenauigkeit und der Render- sowie der Downloadperformance herausgestellt. Bei der Entwicklung des Prototyps hat sich gezeigt, dass die Polygonanzahl der Szene nicht mehr als 10000 betragen sollte (siehe auch Kapitel *6. Darstellungsqualität und Performancebetrachtungen*).

Für die Einschätzung der Polygonanzahl der Szene und einzelner Objekte steht in 3D Studio MAX das Dienstprogramm Polygonzähler (s. Abb. 3.2) zur Verfügung.


Abb. 3.2 Hilfsmittel Polygonzähler in 3D Studio MAX

Ein kurzes Beispiel soll diesen Sachverhalt verdeutlichen. In der Szene existieren über 30 Außenfenster, die als Instanzen erzeugt wurden. Hinsichtlich der Downloadgröße ergeben sich damit keine Probleme. Aber jedes Fenster bestand ursprünglich aus 130 Polygonen. Somit würde nur für die Modellierung der Fenster ein Drittel der maximalen Polygonanzahl verwendet werden. Um dieses Problem zu lösen, wurden einfache Ebenenobjekte für die Fenster verwendet, die zur Laufzeit per Lingo eine Fenstertextur erhielten.

### 3.1.3. Beleuchtung und Schattierung

Bei großflächigen Polygonen fällt eine typische Kantenbildung auf. Außerdem wird keine gleichmäßige, kreisförmige Beleuchtung der Fläche erzeugt. Diese Probleme sind materialunabhängig und können während der Modellierungsphase nur dadurch vermieden werden, dass geometrische Primitive mit mehreren Segmenten erstellt werden. Eine korrekte Darstellungsqualität konnte bei den in diesem Beispiel gewählten Größenverhältnissen mit 10 Breiten- und Längensegmenten des Basisquaders erreicht werden (Qualitätsunterschiede s. Abb. 3.3 bis Abb. 3.5). Dies führt jedoch zu einer größeren Exportdatei und damit zu längeren Downloadzeiten.

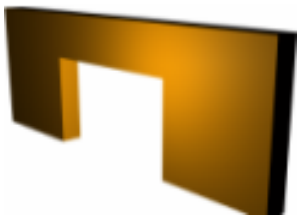


Abb. 3.3 3D Studio MAX-Renderausgabe

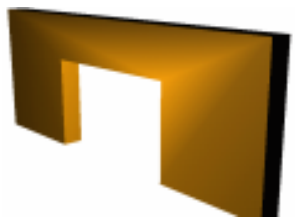


Abb. 3.4 ohne Segmentierung

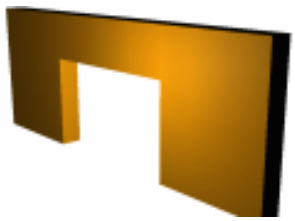


Abb. 3.5 10x10x1 Segmente

Eine Alternative zur Verwendung zusätzlicher Segmente ist die Erhöhung der Geometrieauflösung per Lingo. Auf die Ursache dieser Darstellungsprobleme wird in Abschnitt *6.1. Kantenbildung an ebenen Flächen* noch detaillierter eingegangen.

### 3.1.4. Simulation eines Spotlichtkegels

Drei Einschränkungen von Shockwave3D führen dazu, dass der Lichtkegel der Straßenlaternen nicht mit einem Spotlicht und aktiviertem Volumenlicht dargestellt werden kann:

- Volumenlicht wird unterstützt.
- Die runde Form des Lichtflecks auf dem Boden ist nur kompliziert zu generieren (siehe Abschnitt *6.1. Kantenbildung an ebenen Flächen*).
- Jedes weitere Licht in der Szene reduziert die Renderperformance. Mehr als 8 Lichter werden nicht unterstützt.

Aus diesen Gründen wurde der Lichtkegel (Abb. 3.6) durch einen Kegel mit halbtransparenten, weiß-gelben Material simuliert. Der Lichtfleck besteht aus einem flachen Zylinder mit dem gleichen Material.

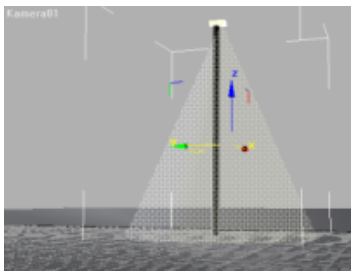


Abb. 3.6 Laterne mit simulierten Lichtkegel in 3D Studio MAX

Für die Realisierung der Kollisionserkennung der Kamerasteuerung bedeutete dies, dass die Lichtkegel kein Kollisionsereignis auslösen durften, damit man durch den Lichtkegel hindurchgehen kann. Entsprechend musste die Kamerasteuerung um eine Liste der Modelle erweitert werden, bei denen eine Kollision mit der Kamera ignoriert werden sollte.

Der Nachteil der Lösung mit dem simulierten Lichtkegel zeigt sich, wenn die Kamera durch diesen hindurch bewegt wird. Dann fallen Helligkeits- und Farbtonunterschiede auf, sobald die Kamera den Mantel durchdringt (sich in den Kegel hinein und wieder heraus bewegt). Dieser Unterschied lässt sich jedoch nicht gänzlich vermeiden.

### 3.1.5. Problem der Shockwave-Tiefensortierung

Als eines der größten Probleme stellte sich eine geeignete Modellierung der Wände heraus. Flächen, die in der gleichen Tiefenebene liegen oder sich in der Tiefenanordnung nur wenig unterscheiden, können in Shockwave3D nicht korrekt gerendert werden, es treten „Flackereffekte“ auf. Mal wird die eine Fläche zuletzt gerendert, dann die andere (Prinzip des Maleralgorithmus: male zuerst den Hintergrund und dann die Objekte in der Reihefolge, wie sie von der Kamera entfernt sind, beginnend mit der größten Entfernung). Da der Renderer nur eine begrenzte Genauigkeit für die Ermittlung der Tiefensortierung hat, ist eine eindeutige Festlegung, welche Fläche sich vor welcher befindet, nicht immer möglich. Sie erfolgt dann bei jedem Renderzyklus nach dem Zufallsprinzip. Sichtbar wird das Problem der Tiefensortierung jedoch erst dann, wenn die abwechselnd dargestellten Dreiecksflächen unterschiedliche Schattierungen aufweisen. Dies tritt zum Beispiel dann auf, wenn die Quader, welche die Wände bilden, über Eck angeordnet sind (Abb. 3.7).

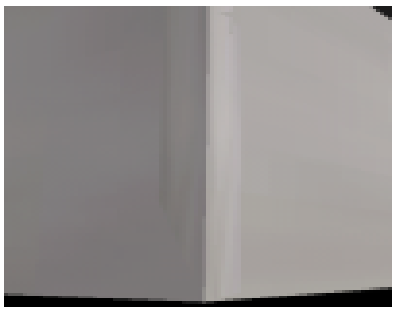


Abb. 3.7 Darstellungsfehler an Wandecken

Aufgrund der Verwendung des Gouraudshadings und der unterschiedlichen Polygongrößen werden die Polygone unterschiedlich schattiert (siehe auch Abschnitt *6.1. Kantenbildung an ebenen Flächen*). Durch ihren helleren Farbton unterscheiden sich die Stirnseiten von den Frontflächen der Quader. Ändert sich auch noch die Perspektive, so kommt es zum Flackereffekt, da der Renderer nicht in der Lage ist, eine eindeutige Tiefensortierung, wie oben beschrieben, durchzuführen.

Für die Lösung dieses Problems bieten sich mehrere Ansätze an:

- Stirnflächen der Quader entfernen
- höhere Geometrieauflösung zur korrekten Schattierung
- Verwendung von C- und L-Extrusionen für die Erstellung gewinkelter Wände

Aber auch bei Elementen, die in unterschiedlicher Tiefe angeordnet sind, kann es zu Darstellungsfehlern in Shockwave3D kommen, wenn nicht ein Mindestabstand zwischen den Flächen eingehalten wird. Die Ausprägung dieser Darstellungsfehler ist von der Perspektive abhängig, d. h. je nach Blickwinkel der Kamera werden Teile der hinten liegenden Fläche mehr oder weniger sichtbar falsch gerendert.

Abb. 3.8 Problem der sichtbaren Flächen und Kanten

Im Beispiel der obigen Darstellung (s. Abb. 3.8) sollte eine Fläche, bestehend aus 2 Dreiecksflächen, als Fensterhintergrund dienen. Erst nachdem die Fläche stärker unterteilt wurde, erfolgte eine korrekte Darstellung. Das gleiche Ergebnis konnte auch ohne Segmentierung erzielt werden, indem die Fläche einen größeren Abstand zu der Fensterfront erhielt.

## 3.2. Modellierung in Shockwave3D

Es gibt verschiedene Gründe dafür, dass nicht die komplette Szene in 3D Studio MAX erstellt und eins zu eins verwendet werden konnte. Neben der Visualisierung von Textinformationen (beschrieben im Abschnitt *5.2 Dynamische Visualisierung textbasierter Inhalte*) ist auch die Realisierung eines Objektes mit veränderlicher Geometrie, wie dies beim Infoobjekt im Kinosaal notwendig ist, nur per Lingo zu verwirklichen.
Außerdem wurden, um die Funktionalität einer Alphatextur zu testen und die Downloadgröße gering zu halten, die Materialeigenschaften der Fenster erst zur Laufzeit definiert (Abschnitt *3.2.1. Texturierung der Fenster*).

### 3.2.1. Texturierung der Fenster

Für die Fenster sind während der Modellierung nur Dummyebenen in der Größe der Fenster erstellt worden. Diese besitzen alle die gleichen Materialeigenschaften, d. h. sie verwenden in Shockwave3D alle den gleichen Shader.
Um das Fensteraussehen etwas realistischer zu gestalten und die Möglichkeit von Alphatexturen zu testen, wurde für jedes Fenster eine zweite Ebene für den Beleuchtungshintergrund erstellt (Listing 44 Duplizierung der Fensterebenen). Außerdem wurden 3 Shader angelegt; einer für die Fensterebenen und jeweils einer für einen beleuchteten bzw. unbeleuchteten Hintergrund (Listing 45 Definition der Fenstertexturen).

Für die Textur wurde ein Bitmapdarsteller mit einem 1-Bit-Alphakanal verwendet. Die Farbe des Alphakanals für den Fensterrahmen ist weiß, d. h. die Textur wird an dieser Stelle gezeichnet. Für die Glasflächen ist die Alphainformation schwarz; das Objekt wird an diesen Stellen transparent.

Da die Ebene für die Hintergrundbeleuchtung die gleiche Position wie die Fensterebene hat, würden sich die beiden Ebenen überdecken. Indem der invertierte Alphakanal verwendet wird, können auf der Hintergrundebene die Bereiche des Fensterrahmens transparent gestaltet werden (Listing 46 Erzeugen und Zuweisen der Alphakanalmasken).

### 3.2.2. Das Kino-Informationsobjekt

#### 3.2.2.1. Erstellung des Objekts

Neben dem Preloaderelement wurde ein zweites Element komplett per Lingo generiert. Dabei handelt es sich um ein rotierendes, prismenähnliches Objekt, dessen Mantelflächen als Textur-Bitmaps von Kinoplakaten verwenden. Damit ist die Funktion des Objekts auch schon fast erklärt. Der Nutzer soll darauf klicken und erhält, je nach dem auf welche Mantelfläche er geklickt hat, weitere Informationen zum Film. Das Objekt wird zur Laufzeit per Lingo in Abhängigkeit verschiedener Parameter erzeugt und mit Betreten des großen Kinosaals gerendert. Zwecks Kapselung und Wiederverwendbarkeit ist die gesamte Funktionalität unter Verwendung eines Parentskriptes realisiert worden. Es wird eine Instanz des Parentskriptes `ps_3DInfoVisualisation` erzeugt (Behavior `b_Cinema`, Funktion `xmlParsing_successful`), wenn die XML-Datei, die die Informationen zu den aktuellen Kinofilmen beinhaltet, erfolgreich geparst wurde (siehe Abschnitt *5.1 Contentübertragung per XML & Asynchrones Scripting*). Damit das Objekt die Nachricht `stepFrame` erhält, wird die Instanz der `actorList` (siehe Listing 14) hinzugefügt. Es wird automatisch an alle in der `actorList` enthaltenen Referenzen auf Skriptobjekte die Nachricht `stepFrame` gesendet, wenn der Abspielknopf ein Bild betritt oder die Bühne aktualisiert wird. Der Vorteil liegt darin, das jede Skriptinstanz eine mit dem `enterFrame`-Handler vergleichbare Funktion besitzen kann. Der Code kann damit übersichtlicher und verständlicher gestaltet werden, potentielle Fehlerquellen werden reduziert, da die Property-Variablen jeder Instanz ihren eigenen Gültigkeitsbereich besitzen.

Nachfolgende Tabelle zeigt die Parameter, die als Liste dem Konstruktor (siehe Listing 14) übergeben werden.

| | |
|---|---|
| Name des 3D-Darstellers | `p3DMemberName` |
| Höhe der Mantelflächen | `pRadius` |
| Außendurchmesser des Prismas | `pHeight` |
| Liste der geparsten XLM-Informationen | `pParsedList` |
| Position des Infoobjektes im Raum | `pPosition` |

Tabelle 14 Generierungsparameter des Infoobjektes

```
p3DObj = new( Script "ps_3DInfoVisualisation", aPropList )
(the actorList).add( p3DObj )
```

Listing 14 Instanzierung eines Parentskriptes

Es können sinnvoll Objekte von 3 bis 8 Mantelflächen erstellt werden. Somit ist man hinsichtlich der Anzahl der zu bewerbenden Filme flexibel. Die Eckpunkte der Mantelflächen sind zugleich auch Eckpunkte der Basis der n-seitigen Pyramiden, die das Objekt nach oben bzw. unten abschließen (s. Abb. 3.9).

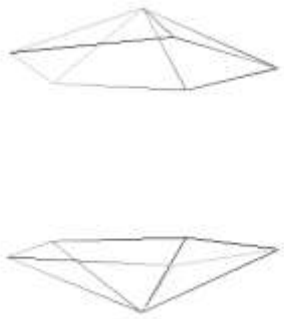


Abb. 3.9 n-seitige Pyramiden als Abschluss

Diese Eckpunkte liegen auf zwei Kreisen, die durch den Außendurchmesser des Prismas definiert werden. Insgesamt besteht das Infoobjekt aus n* 4 Polygonen und n*2 +2 Scheitelpunkten (s. Abb. 3.10).

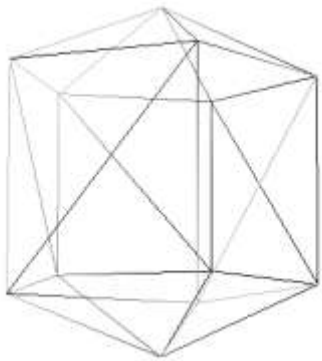


Abb. 3.10 Drahtgittermodell des 5-seitigen Infoobjektes

Die Geometrie des Objektes besteht aus einem Gitternetz, dessen Eigenschaften in der Funktion `createGeometry()` definiert werden.

Zuerst sind einige Initialisierungen vorzunehmen, d. h. es wird u. a. die Anzahl der benötigten Polygone und Scheitelpunkte berechnet.

```
lAngle = 360.0 / pCountPics  -- Winkel zw. 2 Punkten auf dem Umkreis und dem Mittelpunkt
lNumFaces = pCountPics * 4
lNumVertices = pCountPics *2 +2
```

Listing 15 Initialisierung für Meshgenerierung

Anschließend wird ein Gitternetzprimitiv erstellt, dessen Eigenschaften für die Polygone und Scheitelpunkte auf die berechneten Werte gesetzt werden. Außerdem wird festgelegt, das keine NormalList und keine ColorList explizit definiert wird. Für die Positionierung der Textur werden 4 Texturkoordinaten verwendet.

```
p_mrMesh = p3DMember.newMesh("mrMesh", lNumFaces, lNumVertices , 0, 0, 4 )
```

Listing 16 Meshressource erstellen

Als Nächstes erfolgt die Definition der VertexList, die alle Scheitelpunkte beinhaltet.

```
-- Höhe der Kegelspitzen
lZ = integer ( pPropList.pHeight / 1.2 )-- Mitte oben
lVertexList.add( vector( 0.0, 0.0, lZ )) -- Mitte unten
lVertexList.add( vector( 0.0, 0.0, -lZ ))
lZ = pPropList.pHeight / 2

-- Eckpunkte der rechteckigen Mantelflächen
repeat with i = 0 to pCountPics -1
  lX = integer ( cos( i * lAngle * PI / 180.0 ) * lRadius )
  lY = integer ( sin( i * lAngle * PI / 180.0 ) * lRadius )
  lVertexList.add( vector( lX, lY , lZ ))
  lVertexList.add( vector( lX, lY , -lZ ))
end repeat
p_mrMesh.vertexList = lVertexList
```

Listing 17 Berechnung und Zuordnung der VertexList

Nachdem die VertexList alle benötigten Scheitelpunkte beinhaltet, können diese den einzelnen Flächen über ihren Index in der VertexList zugeordnet werden.

```
-- Anzahl der Seitenflächen ohne Kegelflächen
lCount = pCountPics
-- Generierung der Kegelflächen
repeat with i = 1 to lCount
  if i < lCount then
   p_mrMesh.face[i].vertices = [ 2* i + 2 , 2, 2* i + 4] -- oberer Kegel
   p_mrMesh.face[i+ lCount ].vertices = [ 2* i + 1, 2* i + 3 , 1] – unterer Kegel
  else
    p_mrMesh.face[i].vertices = [ 2* i + 2 , 2, 4 ] -- oberer Kegel
    p_mrMesh.face[i+ lCount].vertices = [ 2* i + 1, 3 , 1 ] -- unterer Kegel
  end if
end repeat
```

Listing 18 Zuordnung der Scheitelpunkte zu den Kegelflächen

Nach dem gleichen Muster verläuft auch die Zuordnung der Scheitelpunkte zu den Mantelflächen. Für die Definition der Sichtbarkeit der Flächen werden die Normalen benötigt. Dabei bestimmt die Reihenfolge der Scheitelpunkte eines Polygons die Normalenausrichtung und somit die Sichtbarkeit einer Fläche bei der Modelleigenschaft `visibility = #front` bzw. `#back`. Für die automatische Generierung der Normalen gibt es zwei Varianten:

```
p_mrMesh.generateNormals(#smooth)
p_mrMesh.build()
pPrismModel = p3DMember.newModel("md_3DInfoObject")
pPrismModel.resource = p_mrMesh
```

Listing 19 Automatische Erstellung der Normalist

Mit dem Parameter `#smooth` wird für jeden an einem Polygon beteiligten Vertex eine Normale berechnet. Im Gegensatz dazu wird bei Verwendung des Parameters `#flat` nur eine Normale pro Fläche ermittelt (Abb. 3.11).

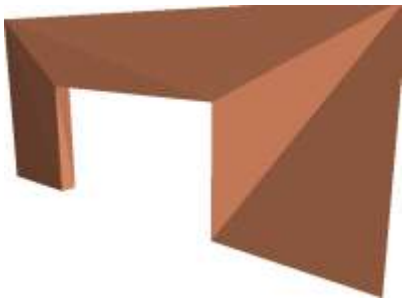


Abb. 3.11 Beispiel für Flatshading

Sind alle Eigenschaften des Meshprimitivs gesetzt, so wird eine renderfähige Modellressource mit dem Befehl `build()` erstellt. Mit dieser Ressource kann dann ein Modell erzeugt werden.
Damit das Infoobjekt nicht mit dem Standardshader und seiner Standardtextur dargestellt wird, wurden die Bitmaps für die Texturen mittels `PreLoadNetThing` geladen, jeweils einem Bitmapdarsteller zugeordnet und Texturen erstellt, die auf diese Darsteller zurückgreifen. Außerdem wurde die Transparenz der Shader auf 65% reduziert. Da die Flächen beidseitig gerendert werden, scheinen die Rückseiten leicht durch. Wird eine Fläche per Maus gepickt, so wird deren Transparenz auf 100% gesetzt (s. Abb. 3.12).


Abb. 3.12 Das fertig texturierte Infoobjekt

Die darzustellenden Informationen sind in einer XML-Datei strukturiert. Diese werden mit Hilfe des XML-ParserXtras und dem Parentskript `ps_ParserScript` in eine Liste übertragen. Die Liste enthält dann u. a. auch die Information über die URL's der Texturen der Mantelflächen (siehe Abschnitt *5.1. Contentübertragung per XML und asynchrones Scripting*).

#### 3.2.2.2. Rotation des Kino-Informationsobjektes

Solange die Maus sich nicht über dem Objekt befindet, rotiert das Objekt um seine lokale z-Achse. Der notwendige Befehl wird am Ende der Funktion `stepFrame` ausgeführt.
Wenn die Maus sich über einer Mantelfläche befindet, soll das Objekt immer eine Korrekturrotation ausführen. Der Winkel der Korrekturrotation wird dabei so gewählt, dass die aktivierte Fläche frontal zur Kamera ausgerichtet wird.

Für die korrekte Ausrichtung des Kino-Informationsobjektes sind eine Reihe vektormathematischer Operationen nötig. Nachfolgende Darstellung des Funktionsprinzips, welche einen Blick von oben auf die Szene darstellt, soll das Verständnis für die Problematik unterstützen. Zur Vereinfachung wird die z-Komponente der einzelnen Koordinaten ignoriert, d. h. auf 0 gesetzt.

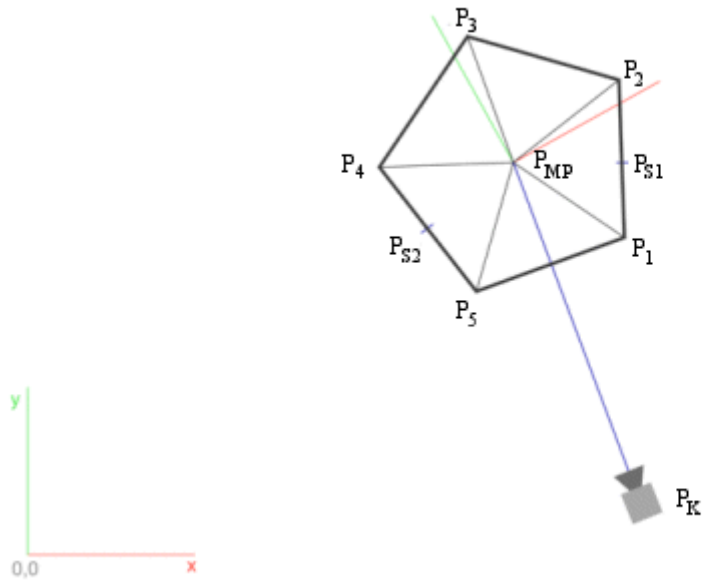



Abb. 3.13 Prinzipskizze Rotation des Kinoinformationsobjektes, Draufsicht

Das Sprite stellt einen Blick durch die Kamera dar, deren Koordinaten mit PK bezeichnet sind. Die blaue Linie repräsentiert den Vektor von der Kameraposition zum Zentrum des Infoobjektes ( $\overrightarrow{P_{MP}} - \overrightarrow{P_K}$ ). Dieser muss nicht unbedingt mit dem Vektor der Blickrichtung der Kamera identisch sein. Wie in der Prinzipskizze erkennbar wird, stimmen das WKS und das LKS in ihren Ausrichtungen nicht überein.

Nachfolgend soll der Algorithmus zur korrekten Flächenausrichtung zur Kamera hin beschrieben werden. Wenn nicht anders definiert, wird von der Prämisse ausgegangen, dass die Fläche, die in der Draufsicht durch die Punkte P4 und P5 repräsentiert wird, zur Kamera hin gedreht werden soll.

Für die Lösung des Problems wurde auf eine Berechnung der absoluten Rotation in Hinblick auf die freie Kameraposition und die variable Anzahl der Mantelflächen verzichtet. Stattdessen soll die notwendige relative Rotation ermittelt werden.

Der Korrekturwinkel lässt sich unter oben genannter Voraussetzung mit den Punkten $P_{S2}$, $P_4$ und $P_5$ beschreiben, wobei $P_{S2}$ mit $\overrightarrow{P_{S2}} = \overrightarrow{P_5} + \frac{1}{2}\overrightarrow{P_5P_4}$ definiert werden kann.

Entweder wird der Winkel zwischen den Vektoren (siehe Abschnitt *2.3.2. Vektoren*) manuell berechnet oder man verwendet die von Director mitgelieferte Funktion für die Berechnung eines Winkels zwischen zwei Vektoren.

```
angle=vector1.angleBetween(vector2)
```

Listing 20 Winkelberechnung zwischen 2 Vektoren

Somit müssen nur noch die benötigten Vektoren $\overrightarrow{P_{MP}\,P_K}$ *und* $\overrightarrow{P_{MP}\,P_{S2}}$ berechnet und in oben genannter Funktion verwendet werden.

Aus dieser Vorgehensweise ergibt sich jedoch folgendes Problem. Der berechnete Winkel gibt keine Aussage über den notwendigen Drehsinn der Korrekturrotation.

Als Lösung dieses Problems bietet sich die Generierung einer Flächennormalen der Fläche an, die durch die beiden zuvor verwendeten Vektoren aufgespannt wird. Da die z- Komponente der Vektoren auf 0 gesetzt wurde, liegt die Fläche in der xy-Ebene und die ermittelte Flächenormale entspricht somit entweder dem Vektor(0,0,1) oder dem dazu invertierten Vektor. Die Berechnung der Normale könnte manuell unter Nutzung der Formel des Kreuzproduktes (vgl. Abschnitt *2.3.2 Vektoren*) ermittelt werden. Diese Formel ist jedoch schon in nachfolgender 3D-Lingofunktion enthalten.

```
normalVector = vector1.cross(vector2)
```

Listing 21 Normalenvektor einer Fläche ermitteln

Werden `vector1` und `vector2` vertauscht, so wird der ermittelte Vektor invertiert.

Mit diesem Hilfsmittel lässt sich nun bestimmen, ob das Objekt im Uhrzeigersinn (für die Fläche, die durch die Punkte $P_1, P_2$ repräsentiert wird) oder entgegen dem Uhrzeigersinn (für die Fläche mit den Punkten $P_4, P_5$) gedreht werden muss.

# 4. Die Kamerasteuerung

## 4.1. Anforderungen

Die Steuerung der virtuellen Kamera in der Szene stellt auf Grund der Komplexität der Anforderungen ein besonderes Problem dar.

Director liefert standardmäßig in seiner Bibliothek schon verschiedene Verhaltsskripte für eine einfache Kamerasteuerung mit. Allerdings zeichnete sich schon bald ab, dass mit diesen Skripten die der Szenennavigation zu Grunde liegenden Anforderungen nicht realisiert werden konnten.

Deshalb musste ein neues Kameraverhalten entwickelt werden bzw. eine existierende Lösung anderer Director-Entwickler angepasst werden. Eine intensive Recherche im Internet lieferte ein einziges Beispiel, welches sich als brauchbare Basis für die weitere Entwicklung der Kamerasteuerung herausstellte [www52]. Folgende Vorgaben waren bei der Umsetzung zu realisieren:

| | |
|---|---|
| Kamerabewegung in Blickrichtung | • bei gedrückter linker Maustaste, mit zunehmender Änderungsgeschwindigkeit |
| Kamerabewegung entgegen Blickrichtung | • bei gedrückter rechter Maustaste |
| Rotation Kamera um z- Achse der Welt für Schwenk links/ rechts | • wenn keine Maustaste und nicht die [Ctrl]-Taste gedrückt, Maus außerhalb des Mittelbereiches des 3D-Sprites |
| Rotation um lokale x- Achse für Blick nach oben/ unten | • wenn [Ctrl]-Taste gedrückt und Maus außerhalb des Mittelbereiches des 3D-Sprites |
| Kollisionserkennung mit Unterbindung der Kamerabewegung | • ignorieren von Elementen, die in einer Positiv-Liste zusammengefasst sind (Bsp. Lichtkegel), Auflösung der Kollision bei allen andern Modellen |
| Picken von Szenenelementen und Triggerauslösung | • Feststellung, auf welches Szenenelement geklickt wurde, Triggerauslösung mit Hilfe Picken in negativer z-Richtung |

Tabelle 15 Auflistung der zu realisierenden Funktionalitäten bezüglich der Kamerasteuerung

## 4.2. Machbarkeitsanalyse eines Kollisionserkennungsprinzips

Für die Kollisionserkennung gibt es theoretisch mindestens zwei verschiedene Lösungsansätze. Der erste Ansatz wird als Technologiestudie nachfolgend beschrieben. Die letztendliche Realisation wird dann im Abschnitt *4.3. Endgültige Realisierung der Kamerasteuerung* dargelegt.

### 4.2.1. Einführung in das Modifierkonzept

Die erste Variante der Kollisionserkennung bedient sich der Funktionalität des Collisionmodifiers. Modifier, auch Modifikatoren genannt, sind in C++ programmierte Xtras. Mittels dieser Xtra-Technologie besteht für Drittanbieter die Möglichkeit, die 3D-Funktionalität von Director zu erweitern, indem sie spezifische Modifikatoren entwickeln. Macromedia hat die bereits mitgelieferten Modifikatoren in dem Xtra Shockwave3D Asset gekapselt. Modifikatoren ähneln Verhalten, das heißt, sie können auf Modelle eines 3D-Members angewendet werden. Nachfolgende Übersicht (Tabelle 16) zeigt die Modifikatoren, die im Umfang von Director enthalten sind:

| Modifikatortyp | Funktionalität |
|---|---|
| `#collision` | • erkennt und behebt Kollisionen von Modellen |
| `#bonesPlayer` | • verwaltet Animationen von Teilen des Modells, die als Knochen bezeichnet werden |
| `#keyFramePlayer` | • verwaltet keyframebasierte Animationen des gesamten Modells |
| `#lod` | • entfernt Details dynamisch aus Modellen bei zunehmender Kameraentfernung (automatisches Standardverhalten),<br>• Reduzierung der Geometrieauflösung auch manuell per Lingo möglich |
| `#inker` | • fügt Silhouetten, Falten und Begrenzungskanten einem Modell hinzu |
| `#meshdeform` | • steuert verschiedene Größen einer Gitternetzstruktur eines darauf basierenden Modells |
| `#sds` | • fügt geometrische Details zu Modellen hinzu und synthetisiert bei abnehmender Kameraentfernung zusätzliche Details um Kurven zu glätten |
| `#toon` | • cartoonartige Wiedergabe der Modelloberfläche |
| `#havok` | • Havok-Physics-Xtra, mit dem sich realistisch physikalische Kräfte, Stöße und Kollisionen simulieren lassen [www53] |

Tabelle 16 Übersicht der Standardmodifikatoren

### 4.2.2. Der Collision-Modifier

Soll der Modifier Collision für die Kollisionserkennung verwendet werden, so wird dieser wie folgt an das Modell gebunden, das potentiell kollidieren kann.

```
member(whichMember).model(whichModel).addModifier(#collision)
```

Listing 22 Zuordnung des Collision-Modifiers zu einem Modell

Anschließend können die benötigten Attribute des Modifikators gesetzt werden. Wenn explizit auf das Kollisionsereignis reagiert werden soll, wird die Callback-Technik eingesetzt, d. h. es wird der Funktion `registerForEvent` der Funktionsname als Symbol übergeben.

Mit der Verwendung des Befehls

```
member(whichMember).registerForEvent((#collideWith,#handlerName,\
                                      scriptObjekt,pMember.model(collisionModel))
```

Listing 23: Initialisierung des Collision-Modifiers

wird festgelegt, welcher Eventhandler in welchem Objekt aufgerufen werden soll, wenn eine Kollision des Modells (`modal(collisionModel)`) mit einem anderen Objekt erkannt wird. Bei der Verwendung von `#collideAny` statt `#collideWith` wird auf jede Kollision in einer Szene reagiert.

Für die Verwendung des Collision-Modifiers können verschiedene Attribute abgefragt und gesetzt werden.

| Eigenschaft | Beschreibung |
|---|---|
| `Collision.enabled` | Erfolgt eine Kollisionsdetection am Modell? |
| `Collision.resolve` | Soll die Kollision am Modell behoben werden? |
| `Collision.mode` | Welche Geometrie soll für den Kollisionserkennungsmodus verwendet werden (`#sphere, #box, #mesh`)? |

Tabelle 17 Eigenschaften des Collision-Modifiers und deren Bedeutung

### 4.2.3. Funktionsprinzip der Kameraboundingsphere

Da eine Kollisionserkennung nur zwischen Modellen funktioniert, muss ein Dummyobjekt für die Kamera eingeführt werden. Als ein solches Objekt fungiert die Boundingsphere (`Model("camera_sphere")`). Der minimale Abstand zwischen Kamera und einem Kollisionsobjekt lässt sich somit über den Radius der Kameraboundingsphere definieren. Für eine korrekte Funktionsweise der Kamerasteuerung ist eine Parent-Child Zuordnung von Kamera und Dummykugel notwendig. Entsprechend ist also bei der Initialisierung die Transformationsmatrix der Kugel mit der der Kamera zu überschreiben. Diese Vorgehensweise gewährleistet, dass Kamera und Dummykugel die gleiche Position und Ausrichtung innerhalb der Szene haben. Einen entscheidenden Einfluss auf das Kollisionsverhalten hat die Definition, welches Objekt Parent und welches Objekt Child ist.

Wenn die Kamera als Parent fungiert und ihre Transformation direkt per Lingo verändert wird, funktioniert dies nur so lange richtig bis eine Kollision auftritt. Der Kollisionmodifier erkennt eine Kollision zwischen Dummykugel und einem anderen Objekt und korrigiert die Transformation der Kugel. Die Transformation der Kamera bleibt jedoch unverändert, Kamera und Dummykugel divergieren. Deshalb muss die Kugel gesteuert und die Kamera als Child der Kugel definiert werden.

### 4.2.4 Auswirkungen der Modi des Collision-Modifiers

Entscheidend für den Auslösemoment der Kollisionserkennung ist die Festlegung des Parameters `collision.mode`. Der einfachste Modus ist dabei `#sphere`. Hier wird eine Kugel als Kollisionsobjekt verwendet, deren Radius automatisch so gewählt wird, dass das Modell komplett eingehüllt ist. Bei einem Quader (z. B. einer Mauer) entstehen so unterschiedlichste Abstände zwischen Kamera und Objekt, wenn ein Kollisionsevent auftritt.

Die Verwendung des Attributes `#box` löst zwar dieses Problem, aber beiden Modi ist gemeinsam, dass sie eine Boundingsphere bzw. Boundingbox über das gesamte Modell legen. Da eine Gruppierung von geometrischen Elementen, die in einem externen 3D-Werkzeug definiert wurden, als eine Modellressource importiert wird und somit auch nur eine Instanz bildet, wird die Kollisionserkennung über das gesamt Volumen ausgeführt.

Nachfolgendes Beispiel soll die Aussage bezüglich der Kollisionserkennung verdeutlichen:

Vier Wände umschließen einen Raum und sind in einer Gruppe zusammengefasst. Ein Objekt in einem solchen Raum würde in diesem Fall immer ein Kollisionsevent auslösen. Ebenso kann bei den beiden oben genannten Modi keine Fallunterscheidung hinsichtlich möglicher Öffnungen (z. B. Türen, entstanden durch boolesche Operationen) realisiert werden. Als theoretischer Ausweg für diese beiden Probleme bietet sich die Verwendung des Attributes `#mesh` für den Parameter `collision.mode` an. Leider steigt die Prozessorbelastung in diesem Modus derart stark an, dass eine Reaktion der Kamera auf Benutzerinteraktionen nicht mehr in Echtzeit erfolgen kann (Basis Pentium 700 Mhz, Szene mit 4 Quadern, einer Kugel, `Collision.mode=#mesh`).

Das bedeutet, dass eine Kamerasteuerung, basierend auf der Kollisionserkennung mittels des Kollisionsmodifiers, nicht realisiert werden kann.

## 4.3. Endgültige Realisierung der Kamerasteuerung

Als zweiter Lösungsansatz bot sich die Adaptierung der Kamerasteuerung aus dem Beispiel *tunnels.dir* [www52] von Tom Higgins an. Die Kollisionserkennung und -auflösung erfolgte auch hier unter Verwendung einer Boundingsphere für die Kamera. Diese transparente Kugel und ein Kameralicht wurden der Hauptkamera als Childelemente zugeordnet.

```
pCamera = pMember.camera(pCameraName)
pCamera.fieldOfView = 45
-- create the camera's bounding sphere
mr = pMember.newModelResource("camera_sphere",#sphere)
mr.radius = pCameraSphere_Radius
pCameraSphere = pMember.newModel("camera_sphere",mr)
pMember.shader("DefaultShader").texture = void
member.shader("DefaultShader").blend = 0
-- create a light to carry with the camera
camLight = pMember.newLight("camera_light",#point)
camLight.color = rgb(170,170,170)
camLight.attenuation = vector(1.0,0.5,0.0)
-- make the sphere and light children of the camera
pCamera.addChild(pCameraSphere,#preserveParent)
pCamera.addChild(camLight,#preserveParent)
```

Listing 24 Vorbereitung Kamerasteuerung

Die vorhandene Lösung wurde um die Funktionalität einer `IgnorCollisionList` (enthält Namen der Modelle, bei denen eine Kollision ignoriert wird), zusätzliches Antialiasing und eine Triggerauslösung erweitert. Außerdem wurde der sensitive Bereich für Kameratransformationen eingeschränkt, der Kameraschwenk für den Blick nach oben/ unten implementiert und die Steuerung der Kamera über Treppen und schiefe Ebenen ermöglicht. Nachfolgend soll die Implementierung der Kamerasteuerung mit den genannten zusätzlichen Features beschrieben werden.

### 4.3.1. Die notwendigen Flags und Eigenschaftsvariablen

Für die Realisierung der Steuerung über verschiedene Funktionen hinweg ergibt sich die Notwendigkeit, eine Reihe von Status- und Eigenschaftsvariablen zu setzen und auszuwerten. Dies erfolgt mittels unten aufgeführter Propertyvariablen (Tabelle 18). Weiter nicht aufgeführte Variablen dienen der Bestimmung des Translationsoffsets und des Rotationswinkels. Andere sind Listen für die Interpolation oder kapseln die Modellnamen der Elemente, die nicht für die Kollisionserkennung verwendet werden sollen. Außerdem gibt es eine Reihe von Properties, die als Referenzen zur verkürzten Schreibweise genutzt werden.

| Generelle Flags | |
|---|---|
| `pControlIsOn` | Kamerasteuerung aktiviert |
| Antialiasingflags | |
| `pAASupported` | 3D-Antialiasing von Renderengine unterstützt |
| `pAllowAA` | Flag, ob AA aktiviert werden kann/ soll |
| Maus- und Tastaturflags | |
| `pMouseDown` | linke Maustaste gedrückt |
| `pRightMouseDown` | rechte Maustaste gedrückt |
| `the controlDown` | Systemvariable, ist [Strg]-Taste gedrückt |
| `the shiftDown` | Systemvariable, ist [Shift]-Taste gedrückt |
| Kameraflags | |
| `pShowAnim` | boolescher Wert, Flag, ob Intro-Animation gezeigt werden soll |
| `pCamAnimFlag` | boolescher Wert, Flag, ob Intro-Animation noch läuft |
| `pStartCamPathInterpol` | boolescher Wert, Flag, ob Kameraposition interpoliert werden soll |

Tabelle 18 Notwendige Eigenschaftsvariablen für die Kamerasteuerung

### 4.3.2. Triggerinitialisierung/ Ereignisgenerierung

Für die Kameraanimation und -steuerung wird die kontinuierliche Generierung von Timerereignissen sowie die Behandlung der Nachricht in einem entsprechenden Handler benötigt. Bei diesem Handler handelt es sich um die Funktion `animateCamera` bzw. `controlCamera`.

```
pOrbitCamera.registerScript(#timeMS,#animateCamera,me,0,40,250)
```

Listing 25 Timer unter Verwendung eines Timerskriptes initialisieren

Mit der Funktion wird ein Timer initialisiert, der periodisch die Callbackfunktion `animateCamera` aufruft, die sich in der gleichen Skriptinstanz befindet (Skriptobjekt `me`). Bei den letzten drei Parametern handelt es sich um den Beginn, die Periodendauer (40ms) und die Anzahl der zu generierenden Timerereignisse. Die Funktion `registerScript` kann auch für die Kollisionserkennung genutzt werden Eine ähnliche Funktion existiert mit `registerForEvent()`. Der einzige Unterschied in den Funktionen liegt in dem Objekt, an das die aufzurufende Funktion bei dem spezifizierten Ereignis (im Beispiel `#timeMS` ein Zeitereignis) gebunden wird.

```
pMember.registerForEvent(#timeMS,#controlCamera,me,lStart,50,0)
```

Listing 26 Registrierung eines Timer-Skriptes bei einem 3D-Darsteller

Im ersten Fall wird ein Script an den Node `pOrbitCamera` gebunden und ein objektspezifischer Timer erzeugt. Mit der zweiten Variante wird das Skript an den Darsteller gebunden und es reagiert auf Zeitereignisse, die im Darsteller auftreten. Für das Beispiel der Kamerasteuerung mit seinen unabhängigen, zeitlich getrennt auftretenden Timerereignissen bleibt es gleich, welche Art der Callbackfunktionsgenerierung verwendet wird. Eine sinnvolle Trennung der Einsatzgebiete ergibt sich, wenn statt auf Timer- auf Kollisionsereignisse reagiert werden muss.
Hier kann die Trennung zwischen objektgebundenen und globalen, darstellergebundenen Callbackmechanismen sinnvoll und notwendig sein (siehe Abschnitt *4.2 Machbarkeitsanalyse Kollisionserkennungsprinzip*).

### 4.3.3. Die Kamerafahrt

Zu Beginn der Szene soll die Kamera einen 360°-Umlauf mit einer erhöhten Perspektive über dem Gebäudekomplex durchführen. Dabei soll die Blickrichtung der Kamera stets auf das Zentrum der Szene gerichtet sein. Diese Kamerafahrt baut dabei nicht auf einer importierten Animation auf, sondern wird komplett per Lingo realisiert.
Notwendig für die Initialisierung der Animation sind die Kamera `Orbit` als Startposition, die Endposition der Kamera (`Kamera01`) und ein Repräsentant für das Rotationszentrum – das Modell `Sky`. Darauf aufbauend wird der Startwinkel der Kamera auf der Kreisbahn sowie deren Radius und Offset zum Koordinatenursprung berechnet. Zur Vereinfachung wurde davon ausgegangen, dass die Kreisbahn in der xy-Ebene liegt (Listing 47 Initialisierung für kreisförmige Kamerafahrt).

Die Animation der Kamera erfolgt in der CallBack-Funktion `animateCamera` (Listing 48 Transformation der Orbitkamera auf Kreisbahn), welche zyklisch durch einen Timer aufgerufen wird (siehe auch vorhergehender Abschnitt). Die Kamerafahrt unterteilt sich in zwei verschiedene Abschnitte; die kreisförmige Translation der Kamera und die lineare Interpolation der Position zwischen der Endposition nach der Kreisbewegung und der Kamerastartposition der Hauptkamera (s. Abb. 4.1). Bei zuletzt genanntem Abschnitt wird die Transformation der Hauptkamera der Szene approximiert.

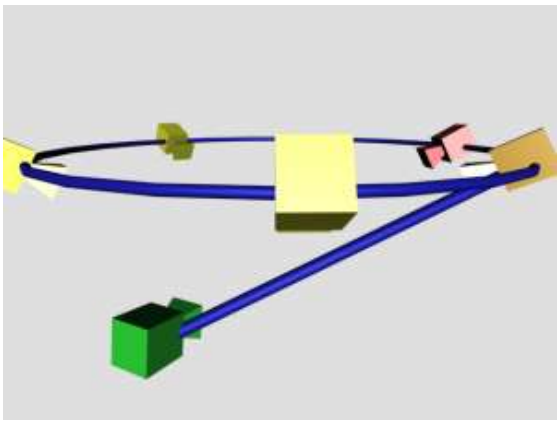


Abb. 4.1 Prinzip Kameraanimation

Die rote Kamera ist die Startposition (entspricht der anfänglichen Transformation der Orbitkamera). Die gelben Kameras repräsentieren den Schwenk der Kamera um das Rotationszentrum und die orangene Kamera stellt die Endposition der Kamera für die Kreisbewegung und zugleich die Anfangsposition für den `PositionPathInterpolator` dar. Die grüne Kamera – sie entspricht der Anfangsposition der Hauptkamera in der Szene – steht für die Endposition des Interpolationsvorganges. Damit die Kamera immer einen Großteil der Szene einfängt wird die Ausrichtung der Kamera nach jeder Positionsänderung korrigiert. Dazu wird die Property `pCamOrbitLookAt`, die per getPropertyDescriptionList bei der Zuordnung des Skriptes zum 3D-Sprite definiert wurde, mit nachfolgendem Befehl genutzt:

```
pOrbitCamera.pointAt(pCamOrbitLookAt,vector(0,0,1))
```

Listing 27 Ausrichtung der Kamera auf einen Punkt im Raum

Anstatt die Kameraposition mittels trigonometrischer Funktionen manuell zu berechnen, hätte man die Kamera auch als Child an ein Dummyobjekt, das sich im Rotationszentrum befindet, binden können. Dieses Dummyobjekt hätte man um die z-Achse rotiert. Die Kamera würde somit durch ihre relative Bewegung eine Kreisbahn um das Dummyobjekt vollziehen. Eine zweite Alternative wäre die Verwendung des Befehls

```
pOrbitCamera.rotate(0.0,0.0,pAngle,pDummy)
```

Listing 28 Relative Rotation um ein Objekt

Hier würde die Rotation um die z-Achse des Dummyobjektes mit dem Winkel `pAngle` relativ zum Objekt `pDummy` erfolgen.

```
if pCamOrbitAngle < -0.7 * PI then
  pCamAnimInfo = [#initT: pOrbitCamera.transform.duplicate(), #finalT:\
  pMember.camera("Kamera01").transform.duplicate(),#count: 0]
  pStartCamPathInterpol = TRUE
```

Listing 29 Abbruchbedingung & Initialisierung

Die Abbruchbedingung für die kreisförmige Kamerabewegung wurde empirisch ermittelt. Wird der Rotationswinkel, der bei jedem Funktionsdurchlauf um den Wert PI/60 dekrementiert wurde, kleiner als der Referenzwert, so erfolgt die Initialisierung der Propertyliste für die Kamerainterpolation mit der Anfangs- und Endposition der Kamera sowie mit dem Anfangswert für die prozentuale Interpolation. Außerdem wird ein Flag gesetzt, welches anzeigt, dass die Kameraposition interpoliert werden soll.

Die Interpolation der Kamera erfolgt linear, indem die Kamera mit dem Differenzvektor aus Start- und Endposition zu Letzterer hin verschoben (und ggf. rotiert) wird. Wie stark die Verschiebung zur endgültigen Position hin erfolgt, wird durch den 3. Parameter bestimmt (Listing 49 Positionpathinterpolation).

### 4.3.4. Die Steuerung der Kameratransformation

Damit die Usability der Kamerasteuerung wie in Abschnitt *4.1 Anforderungen* beschrieben, gewährleistet werden kann, sind eine Reihe von Bedingungen abzufragen. Die Funktion `controlCamera` wird im Normalfall vom Timer alle 50 ms aufgerufen. Die theoretische Anzahl von knapp 20 Funktionsaufrufen wurde bei komplexen Szenen, bei eingeschaltetem Antialiasing oder im Software-Rendermodus deutlich unterschritten, so dass ein akzeptables Reaktionsverhalten auf Benutzerinteraktionen nicht mehr gegeben war. Für die Umsetzung der Kamerasteuerung wurde die Bühne in drei Bereiche aufgeteilt, die bestimmen, wie auf die Mausposition reagiert wird. Nachfolgende Grafik (s. Abb. 4.2) soll die Funktion der Bereiche verdeutlichen.

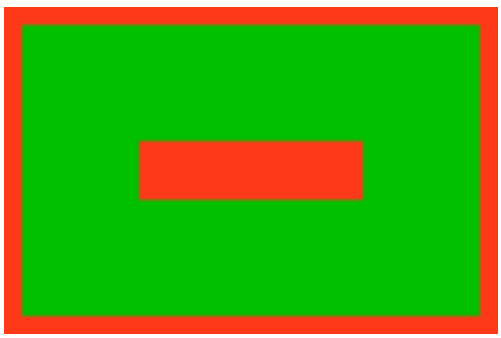


Abb. 4.2 Funktionsschema maussensitiver Bereiche

Das grüne Rechteck symbolisiert das 3D-Sprite mit seiner Position, Höhe und Breite auf der Bühne (roter, äußerer Bereich entspricht der Bühne). Der Registrierungspunkt eines 3D-Sprites ist dabei sein Mittelpunkt.
Im inneren, roten Bereich erfolgt ebenso wie außerhalb des 3D-Sprites keine Aktion auf ein Mausevent. Dies wird über folgenden Funktionsausschnitt gewährleistet:

```
lDiffrel_H = (-2.0 * (the mouseH - pSprite.locH ) / pSprite.width)
lDiffrel_V = (-2.0 * (the mouseV - pSprite.locV ) / pSprite.height)
if abs(lDiffrel_H ) < 1 AND abs(lDiffrel_V) < 1 then
            …
if abs( lDiffrel_H ) > 0.2 then
            …
```

Listing 30 Definition aktiver Bereiche

Damit die Kamera auch bei Treppen und Rampen eine konstante Höhe über dem Boden behält, muss die anfängliche Höhe ermittelt werden. Dies erfolgt mittels eines Pickstrahls, der von der Kameraposition aus in negative z-Richtung ausgesendet wird. Als Ergebnis erhält man eine Propertyliste des ersten gepickten Elements. Im Pickmodus `#detailed` enthält die Liste u. a. auch den Abstand zwischen dem Ursprung des Pickstrahls und dem gepickten Element. Die ermittelte Höhe wird in späteren Translationen als Vergleichs- und Korrekturwert verwendet.

```
lTmpList = pMember.modelsUnderRay(pCamera.transform.position, vector(0,0,-1),1, #detailed)
if lTmpList.count > 0 then
   pCamOldDistance = lTmpList[1].distance
```

Listing 31 Ermittlung der Höhe der Kamera

Die Drehung der Kamera für den Kameraschwenk (Pan) bzw. den Blick nach oben und unten (Tilt) verwendet zwei unterschiedliche Varianten für die Rotation eines Objektes; den Befehl für die Rotation des Objektes und die absolute Angabe der Rotationskomponente der Transformationsmatrix.

```
pCamera.rotate(pCamera.worldPosition,vector(0,0,1),l_rotangle,#world)
-- bzw.
pCamera.transform.rotation.x = …
```

Listing 32 Relative und absolute Rotation der Kamera

Bei der Kamerasteuerung für den Blick nach oben und unten ist außerdem eine Begrenzung des Rotationswinkels notwendig.

Für die Translation sollte eine zunehmende Geschwindigkeit bis zu einem Grenzwert bei gedrückter Maustaste realisiert werden. Wird die Taste wieder losgelassen, so wird der temporäre Richtungsoffset auf seinen Initialwert gesetzt. Die Bewegung der Kamera erfolgt entlang der lokalen z-Achse. Damit die z-Achse der Kamera in der xy-Ebene der Welt liegt, muss die Rotationskomponente der x-Achse vor der Translation auf 90° gesetzt werden. Nach erfolgter Translation wird der ursprüngliche Winkel für den Blick nach oben/ unten wiederhergestellt.
Ist die Verschiebung der Kamera abgeschlossen, erfolgt eventuell eine Korrektur der Höhe der Kamera über dem Boden. Die Verwendung der verschachtelten Repeatschleifen ergibt sich aus der Tatsache, dass die Lichtkegel der Laternen durch semitransparente Kegel simuliert werden. Mit der ursprünglichen Funktion zur Höhenkontrolle ohne eine `IgnoreCollisionList` ist die Kamera an den Mantelflächen der Kegel „hochgerutscht“. Durch die Implementierung einer Liste, welche die Modellnamen enthält, die für die Kollisionserkennung und die Höhenberechnung der Kamera ignoriert werden sollen, ist es nunmehr möglich, durch die simulierten Lichtkegel hindurch zu gehen. Außerdem erfolgt mit Hilfe der Liste, die die gepickten Modelle unter der Kamera beinhaltet, der Test, ob die Kamera sich über einem Dummyobjekt für die Auslösung spezieller, ortsgebundener Funktionen befindet (Türen öffnen und schließen, Teilszenen nachladen) (Listing 50 Korrektur der Höhe der Kamera und Aufruf Triggertest).
Nach Beendigung der Transformationen der Kamera ist ein Test auf Kollision und wenn nötig die Auflösung der Kollision notwendig. Der Algorithmus und seine Implementierung werden im nächsten Kapitel beschrieben.

### 4.3.5. Erkennung, Auflösung und Ausnahmen bei Kollisionen

Nachfolgender Programmablaufplan beschreibt den Algorithmus der implementierten Kollisionserkennung und deren Auflösung (Abb. 4.3).

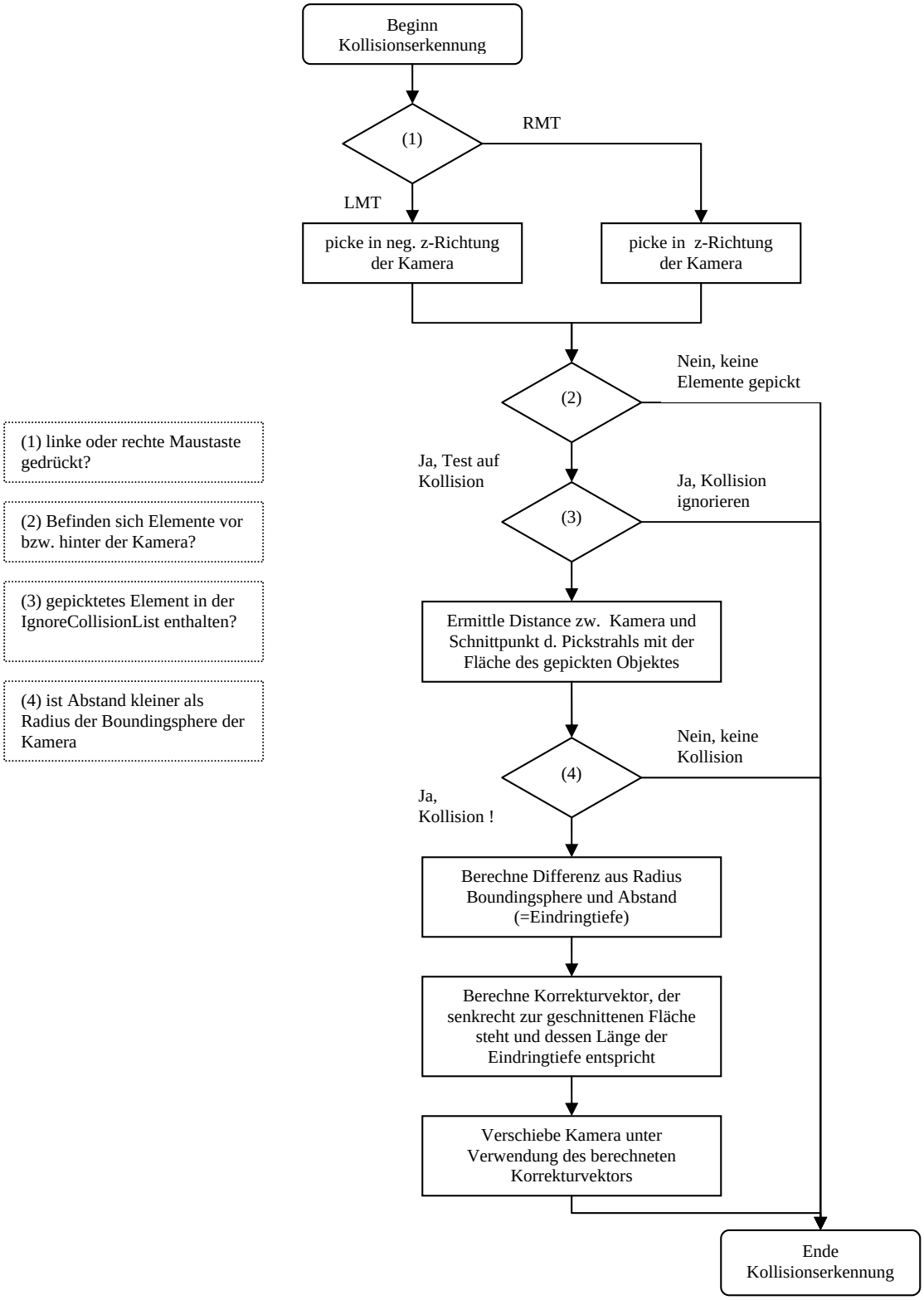



Abb. 4.3 Schema Kollisionserkennung und -auflösung

Wie im Abschnitt *4.2.3. Funktionsprinzip der Kameraboundingsphere* wird auch in der endgültigen Realisierung der Kamerasteuerung eine BoundingSphere zur Kollisionserkennung verwendet. Der Radius der Kugel bestimmt wieder den Kollisionsbereich.

Kamera und Dummykugel werden mit der gleichen Position und Ausrichtung initialisiert. Welches Objekt Parent ist und welches somit per Lingo transformiert wird, spielt hierbei im Gegensatz zum ersten Lösungsansatz keine Rolle.

Nach erfolgter Translation oder Rotation wird ein Pickstrahl von der Kameraposition in Kamerablickrichtung (bzw. entgegengesetzt, je nach letzter Transformation) ausgesendet (Listing 51 Test, ob Objekte vor bzw. hinter der Kamera sind). Als Ergebnis dieser Funktion erhält man ein Liste von Eigenschaftslisten (für jedes vom Pickstrahl geschnittene Modell wird eine Propertyliste erstellt). Diese Liste wird in der Funktion `checkForCollision` interpretiert (Listing 52 Test auf Kollision, ggf. Auflösung).

Damit Objekte, die in der `IgnoreCollisionList` enthalten sind, nicht in den Test auf eine mögliche Kollision einbezogen werden, erfolgt ein Abgleich des Namens des potentiellen Kollisionspendants mit der `IgnoreCollisionList`. Bei einem positiven Vergleich wird die Kollisionsauflösung abgebrochen. Ansonsten wird die Entfernung zwischen der Kamera und dem gepickten Objekt berechnet und mit dem Radius der Boundingsphere verglichen.

Wenn der Abstand zwischen Kamera und dem Objekt geringer ist als der Radius der Dummykugel, wird die Differenz zwischen den beiden Werten als Betrag der notwendigen Verschiebung ermittelt. Anschließend wird der Korrekturvektor berechnet. Dazu wird der normierte Weltvektor des Punktes ermittelt, an dem der Pickstrahl das Gitternetz des gepickten Objektes geschnitten hat. Aus dem Produkt des Betrages der notwendigen Korrektur und dem ermittelten normierten Weltvektor ergibt sich der Korrekturvektor. Mit diesem Vektor erfolgt eine entsprechende Translation und die Kollision ist somit aufgehoben.

# 5. Übermittlung und Darstellung dynamischer Inhalte

Für die Informationsübertragung wurden in dem Projekt zwei verschiedene Varianten der Contentübertragung realisiert. Innerhalb der 3D-Szene wurde XML zur Informationsübertragung genutzt. Alternativ kann eine Informationsübertragung auch mit einer konventionellen Textdatei erfolgen, die mittels `getNetText` geladen und anschließend interpretiert werden kann. Dieses einfache Prinzip der Informationsübertragung wurde im PocketCommunicator verwendet, um die Initialisierungsdatei zu laden, die die Bildstruktur für die Diashow beinhaltet. Im folgenden Kapitel soll die innovativere Variante der Informationsübermittlung per XML beschrieben werden. Die daran anschließenden Kapitel beschäftigen sich mit der Frage der Visualisierung dieser Informationen oder, allgemeiner ausgedrückt, mit dem Problem der Textdarstellung in 3D-Szenen.

## 5.1. Contentübertragung per XML und asynchrones Scripting

In Hinblick auf die Pflege der dynamischen Inhalte in der Website und im Shockwave-Informationssystem stellt sich die Frage, wie die beiden Präsentationswerkzeuge eine gemeinsame Datenbasis nutzen könnten. Für das Beispiel des Kinoprogramms wurde versucht, einen gemeinsamen Technologieansatz exemplarisch zu implementieren. Dabei wurde XML zur hierarchischen Beschreibung einer Dokumentstruktur verwendet. Der Vorteil in der Verwendung von XML liegt in der Trennung von Inhalt und Formatierung. Shockwave kann Inhalte eines XML-Dokumentes anders darstellen, als dies beim Browser der Fall ist. Da es keine bestehende Definition der XML-Struktur für ein Dokument gibt, welches das Kinoprogramm beinhaltet, wurde zu Testzwecken folgende XML-Datei definiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file represents a fragment of a movie database -->
<Film-O-Thek>
    <Film>
    <Titel>Genug</Titel>
    <Genre>"Thriller"</Genre>
    <Original>Enough</Original>
    <Regisseur>Michael Apted</Regisseur>
    <Origin>USA 2002</Origin>
    <Darsteller>Jennifer Lopez, Juliette Lewis, Bill Campbell</Darsteller>
    <Musik>David Arnold</Musik>
    <Inhalt>Hier kommt der Inhalt
    </Inhalt>
    <Laenge>140 Min.</Laenge>
    <ImageName>showPlakat.jpg</ImageName>
  </Film>
  …
</Film-O-Thek>
```

Listing 33 Auszug XML-Datei Kinoprogramm

Das Laden und Parsen der XML-Datei erfolgt unter Verwendung des XML-Parser-Xtras. Dieses liest XML-Dokumente ein und konvertiert sie je nach Dokumentstruktur in eine unterschiedlich tief gestaffelte Liste von Propertylisten.

Allerdings ist in einer endgültigen Realisierung der Shockwave3D-Präsentation zu entscheiden, ob die Informationsübertragung per XML erfolgen soll, da das benötigte Xtra nicht im normalen Installationsumfang des Shockwave-Plug-ins enthalten ist und somit auf der Nutzerseite nachgeladen werden muss.

Wird die XML-Datei per URL aus dem Internet geladen, so erfolgt dieser Vorgang wie bei allen Downloadvorgängen auch asynchron. Das Abschicken eines Befehls und die Erledigung liegen dabei zeitlich mehr oder weniger weit auseinander. Daher muss eine gestartete NetLingo-Aktion zyklisch auf ihren Status, ihr Ergebnis hin überprüft werden (Prinzip des asynchronen Scriptings). Die Überprüfung des Status kann entweder in einem `exitFrame`-Handler oder eleganter unter Verwendung eines Parentskriptes mit eingebautem CallBack-Mechanismus erfolgen (siehe auch [GIL00] S. 399ff.). Eine solche Callbackfunktion wird automatisch aufgerufen, wenn ein Netzvorgang beendet wurde. Dieses Prinzip wurde auch für das Laden eines XML-Files genutzt. Dazu wird eine Instanz des XML-Parser-Xtras erzeugt und dieser beim Aufruf der Funktion `parseURL` als Argumente die zur parsende URL, der Name der Callbackfunktion und die Instanz, die die aufzurufende Funktion enthält, übergeben.

```
parserObject = new(xtra "XMLParser")
parserScriptObject = new(script "ps_ParserScript", parserObject, p3DObj )
if the runMode contains "Plug-in" then
  errorCode=parserObject.parseURL(the moviepath & pURL_FilmInfo, #parseDone,
                                  parserScriptObject)
…
```

Listing 34 Initialisierung XML-Parser unter Nutzung des CallBack-Mechanismus

In der als Symbol übergebenen Funktion `parseDone` erfolgt, nachdem die XML-Datei geladen und geparst wurde, die Umwandlung der geparsten Liste in eine vereinfachte Liste von Propertylisten (Listing 55 Umwandlung der XML-Propertyliste). Dabei werden die Informationen, die durch ein `<Film>`-Tag umschlossen sind, als eine Propertyliste zusammengefasst, deren Propertynamen aus den Tag-Namen der untergeordneten Attribute generiert werden. Die somit für jeden Film erstellten Propertylisten werden in einer gemeinsamen Liste gespeichert. Der Zugriff und die beschriebene Vereinfachung einer geparsten Liste ist jedoch nur möglich, wenn die hierarchische Struktur der XML-Datei bekannt ist.

Innerhalb der XML-Datei sind u. a. auch die Namen der JPG-Dateien, die für die Kinoplakate und das Kino-Informationsobjekt als Texturen verwendet werden, definiert. Diese Dateien müssen vor ihrer Verwendung als Textur geladen und als Darsteller importiert werden. Da neben dem Laden von Texturdateien auch Szenenelemente per zusätzlicher W3D-Files nachgeladen werden sollen, bot es sich an, die benötigten Netz-Funktionalitäten in einem allgemeinen Loaderscript (realisiert als Parentscript) zu implementieren.
Ein Vorteil der objektorientierten Realisierung des Loaderobjektes ist die Möglichkeit, dass dieses Objekt für den Download jeder benötigten Datei verwendet werden kann und durch die Nutzung des CallBack-Mechanismus flexibel einsetzbar ist. Die Statusüberprüfung des Downloadvorganges erfolgt gekapselt innerhalb des Objektes in der `stepFrame`-Funktion. Somit sind auch parallele Downloads durch mehrere Instanzen des Loaderscripts möglich.

Das Loaderobjekt funktioniert prinzipiell so, dass für die Initialisierung des Objektes der Name der aufzurufenden Callbackfunktion und eine URL der zu ladenden Datei übergeben wird (Listing 53 Auszug aus dem Parentskript des Loaderobjektes).
Diese URL dient als Argument für die Lingo-Funktion `preLoadNetThing`, die eine ID zurückliefert, mit deren Hilfe der Downloadstatus in der Funktion `stepFrame` verifiziert werden kann.
Ist der Downloadvorgang erfolgreich abgeschlossen, so wird per CallBack-Mechanismus eine Funktion aufgerufen, die als Symbol dem Konstruktor des Loaderobjektes mit übergeben wird.
In dieser Funktion kann dann spezifisch auf die Beendigung des Downloadvorgangs reagiert werden (Erzeugen von Texturen bei JPG-Download bzw. Ergänzung der Szene durch zusätzliche 3D-Informationen aus der geladenen W3D-Datei).

Während die Erstellung der Texturen und das Nachladen von Szenenkomponenten relativ unkompliziert sind, stellt die Darstellung der per XML übertragenen Textinformationen hinsichtlich der erreichbaren Visualisierungsqualität ein größeres Problem dar. Auf die in diesem Zusammenhang auftretenden Probleme und möglichen Lösungsansätze soll im nächsten Abschnitt eingegangen werden.

## 5.2. Dynamische Visualisierung textbasierter Inhalte

Die Anwendung soll dem Besucher der Website neben einer räumlichen Repräsentation des Entertainmentkomplexes auch spezielle aktuelle Informationen bieten. Außerdem ergibt sich während der Navigation durch die Szene in verschiedenen Situationen die Notwendigkeit, dass Elemente über das Internet nachgeladen werden müssen und der Nutzer über den Status des Downloadvorganges informiert werden soll. In beiden Fällen ist also eine Informationsdarstellung in Textform sinnvoll bzw. zwingend erforderlich. Ein weiterer Anwendungszweck für dynamisch generierten Text ist die Darstellung der Bildwiederholrate innerhalb der Szene für Diagnose- und Optimierungsvorgänge.

Da die gesamte Bühnengröße vom 3D-Darsteller ausgefüllt wird, steht kein weiterer Platz für eine konventionelle 2-dimensionale Textdarstellung unter Verwendung zusätzlicher Sprites zur Verfügung. Aus Performancegründen werden 3D-Darsteller standardmäßig mit der Einstellung `directToStage` abgespielt, d. h. sie werden vor allen anderen Ebenen der Bühne gerendert. Somit besteht keine sinnvolle Möglichkeit mit zusätzlichen Textdarstellern, die in Spritekanälen über dem Spritekanal des 3D-Members angeordnet sind, zu arbeiten. Daraus folgt die Notwendigkeit, die Textdarstellung in die Szene zu integrieren. Fünf unterschiedliche Strategien der Textdarstellung in einer 3D-Szene sollen nachfolgend näher beschrieben werden.

### 5.2.1. Statischen Text im 3D Studio MAX generieren

Für statische Textelemente einer Szene, deren Textinhalt konstant bleibt, bietet sich theoretisch die Generierung mit einem Modellierungswerkzeug an. Vorteil bei dieser Art der Erstellung von Textelementen ist die Möglichkeit der visuellen Einschätzung von Größen und Positionen in Bezug auf andere geometrische Elemente der Szene. Diesem Vorteil steht jedoch das Problem der stark anwachsenden Dateigröße der Szene und die große Anzahl der generierten Polygone gegenüber.

Textelemente basieren in 3D Studio MAX auf Splines. Damit sie gerendert und korrekt exportiert werden können (der W3D-Exporter unterstützt keine Splines), müssen sie zuvor extrudiert werden.

Entscheidend für die Größe der Exportdatei ist neben der Anzahl der Zeichen der gewählte Schriftfont. Schriftarten, die viele geschwungene Linien und komplizierte Geometrien nutzen, führen zu einer wesentlich größeren Anzahl von Scheitelpunkten in einem extrudierten Textobjekt als dies bei der Verwendung einfach gestalteter Schriften der Fall ist.
Als Beispiel diente der Schriftzug Shockwave3D (Schriftgröße und Extrusionsbetrag haben keinen Einfluss auf die Anzahl der Scheitelpunkte). Es wurde mit Geometriequalität 50 und ohne Material- und Shadereinstellungen exportiert.

| Schriftart | Anzahl Scheitelpunkte, Exportgröße |
|---|---|
| Staccato 222 BT (handschriftlicher Charakter) | 12634 Scheitelpunkte 322 kB |
| Arial | 2500 Scheitelpunkte 59kB Exportfile |

Tabelle 19 Anzahl der Scheitelpunkte bei unterschiedlichen Schriftfonts

### 5.2.2. 3D-Text zur Laufzeit erzeugen

Director bietet die Möglichkeit, 2-dimensionalen Text zu extrudieren und somit einen räumlichen Eindruck des Textes zu vermitteln. Die notwendigen Einstellungen lassen sich mit dem Eigenschaftsinspektor oder zur Laufzeit per Lingo vornehmen.
Mit der Erstellung eines 3D-Schriftzuges wird im Hintergrund ein neuer Shockwave3D-Darsteller inklusive Standardkamera und Standardbeleuchtung erzeugt. Ein solches 3D-Textelement ist jedoch auf maximal 70 Zeichen beschränkt. Je nach verwendetem Schriftfont und daraus resultierender Polygonanzahl reduziert sich die Anzahl der maximal darstellbaren Zeichen noch einmal deutlich. Diese Art der Textdarstellung stellt die ungünstigste Variante hinsichtlich zu erwartender Prozessorlast und vermitteltem Informationsgehalt dar.

Es ist jedoch möglich (mittels `cloneModelFromCastmember`) das Modell, welches den 3D-Schriftzug repräsentiert, in einen anderen 3D-Darsteller zu kopieren. Dabei werden die Modellressource, die Shader und die verwendeten Texturen ebenso kopiert. Auf diese Art können Szenenelemente und 3D-Text in einem Darsteller zusammen kombiniert werden (s. Abb. 5.1).


Abb. 5.1 Eine Szene mit importiertem 3D-Schriftzug

Diese Methode eignet sich nur für die Visualisierung geringer Textmengen. Die Änderung des Textes ist relativ einfach möglich. Es muss nur der Inhalt des Textdarstellers geändert werden. Der 3D-Textdarsteller wird daraufhin automatisch aktualisiert. Damit die Veränderung auch in der eigentlichen Szene dargestellt werden kann, ist ein Löschen des Modells und der damit verbundenen Modellressourcen sinnvoll (in Hinblick auf Speicherbedarf, Performance, Vermeidung doppelter Namen). Anschließend kann der Befehl `cloneModelFromCastmember` wieder verwendet werden.
Als problematisch kann sich die Positionierung und Rotation des Objekts erweisen. Der Ursprung eines 3D-Textdarstellers ist nämlich immer seine linke untere Ecke.

### 5.2.3. Informationsvermittlung unter Nutzung von Texturen

Die beiden der Vollständigkeit halber oben genannten Funktionen konnten aus dargestellten Gründen nicht genutzt werden. Stattdessen wurden für die Textdarstellung drei verschiedene Varianten der Verwendung von Texturen in Anspruch genommen.

#### 5.2.3.1. Overlay-Funktion

Diese Kamerafunktion fügt am Ende der Überlagerungsliste eine weitere Überlagerung in Form einer Textur hinzu. Vorstellbar ist dies wie die Verwendung einer bemalten Glasscheibe, die vor die Kamera gehalten wird. Die Funktion erwartet eine Textur, welche auf einem Bitmap-Darsteller basiert, der mit Imaging-Lingo generiert oder importiert wurde. Natürlich kann auch eine schon in einem 3D-Darsteller vorhandene Textur verwendet werden.

```
wrld = member ("3D_Scene")
lTmpImage = new (#bitmap)
lTmpImage.image = member("Text").image
l_texture = wrld.newTexture("tooltip", #fromCastMember, lTmpImage)
wrld.camera[1].addOverlay(l_texture, point(100,100),0)
```

Listing 35 Hinzufügen einer Overlay-Textur zu einer Kamera

Bei der Verwendung von Texturen ergeben sich hinsichtlich der Verzerrung einige Probleme, die es zu beachten gilt. Intern verwendet Director Texturen, deren Breite und Höhe immer einer Zweierpotenz entsprechen. Falls Bitmaps für Texturen verwendet werden, deren Maße nicht einer Zweierpotenz entsprechen, so erfolgt intern eine automatische Skalierung auf die nächst größere oder kleinere Zweierpotenz. Eine fast quadratische Bitmap mit der Größe von 95x97 Pixel würde so zu einer Textur von 64x128 führen.

Eine weitere Ursache für mögliche Verzerrungen findet sich in der Geometrie der zu texturierenden Fläche. Da die Fläche nur im seltensten Fall der Fläche der Textur entspricht, muss die Textur während des Mappings skaliert werden. Dabei kommt es zu unvermeidbaren Verzerrungen, die besonders auffällig sind, wenn das Seitenverhältnis der Fläche deutlich von dem der Textur abweicht.

Die einfachste Variante, eine Textur basierend auf einer Textinformation zu erstellen, zeigt nachfolgendes Beispiel:

```
member(whichTextmember).text = aString
member ("3D_Scene").texture("Test").member = member(whichTextmember)
```

Listing 36 Einfachste Methode der Erstellung einer Textur basierend auf einem Textdarsteller

Bei der Nutzung der Overlay-Funktion erfolgt standardmäßig keine weitere Skalierung. Das Image der Textur wird an der angegebenen 2D-Position dargestellt (s. Abb. 5.2).

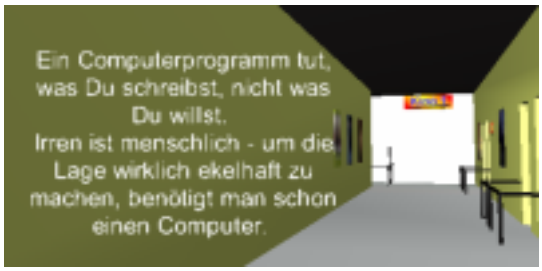



Abb. 5.2 Textdarstellung mittels Overlay-Technik

#### 5.2.3.2. Schrifttextur auf Objekten

Die Nutzung von Texturen zur Informationsvermittlung wird u. a. auch bei der Übermittlung des Preload-Status auf den Schiebtüren im Kinogang verwendet. Es wird auch hier wieder die Bildinformation eines Textdarstellers zur Erzeugung einer Textur verwendet. Diese Textur wird jedoch nicht als Overlay verwendet, sondern dem Shader eines Objektes zugeordnet. Die prinzipielle Vorgehensweise soll vereinfacht mit Hilfe des Beispiels aus Abb. 5.3 erläutert werden.


Abb. 5.3 Beispiel Multitexturing

Bei der Testszene soll eine veränderliche Textinformation auf einem Quader dargestellt werden. Normalerweise nutzen alle 6 Seiten einen gemeinsamen Shader. Dies würde jedoch an den schmalen Seiten zu einer deutlichen Stauchung des Textes führen. Da ein Quader aus 6 Seiten und somit aus 6 Gitternetzen besteht, und jedem Gitternetz ein eigenen Shader zugeordnet werden kann, wird für die schmalen Seiten ein Shader nur mit der Hintergrundtextur definiert. Für die vier anderen Seiten wird ein zweiter Shader angelegt, welcher 2 Texturen miteinander kombiniert. Die erste Textur simuliert wieder das Hintergrundmaterial des Quaders und die zweite dient zur Visualisierung der Textinformationen. Die teilweise Transparenz der Textur wird durch die Nutzung der Alphakanals des verwendeten Bitmapdarstellers bestimmt.

Da jedoch standardmäßig eine Textur beim Mapping so skaliert wird, dass die zu mappende Fläche komplett von der Textur abgedeckt wird, würde die Qualität einer Textdarstellung unter Umständen entsprechend leiden (Abb. 5.4).

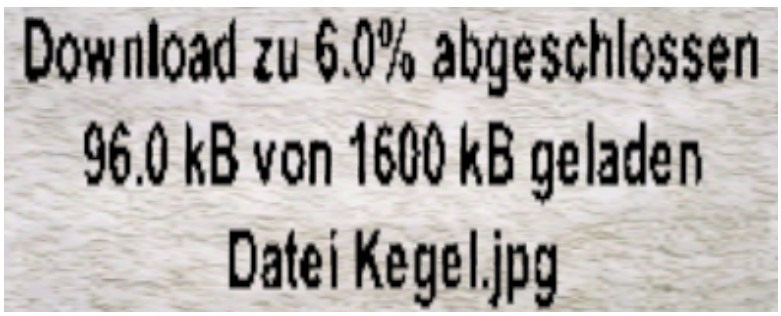



Abb. 5.4 Schlechte Visualisierungsqualität einer Textinformation

Um dieses Problem zu lösen, gibt es zwei Möglichkeiten:

**Variante 1:**

Die Textur der Textinformation wird skaliert (verkleinert) und ohne Wiederholungen neu positioniert (s. Abb. 5.5)

```
theShader = member(1, 1).shader("sh_Box")
theShader.textureTransformList[2].scale = vector( 0.5000, 0.5000, 1.0000 )
theShader.textureTransformList[2].position = vector( 0.2400, 0.2500, 0.0000 )
theShader.textureRepeatList[2] = FALSE
```

Listing 37 Skalierung ohne Wiederholungen und Neupositionierung einer Textur


Abb. 5.5 Verbesserte Darstellungsqualität durch Skalierung der Textur

**Variante 2:**

Die bestmögliche erreichbare Qualität der Textdarstellung (s. Abb. 5.6) war mit größerem Aufwand hinsichtlich Programmierumfang und längeren Ausführungszeiten der genutzten Algorithmen verbunden. Das Testsystem benötigte für die Erstellung einer neuen Textur mehr als 200ms, die automatische Drehung des Modells stockte deshalb sichtbar.


Abb. 5.6 Bestmögliche erreichbare Darstellungsqualität, realisiert mit Imaging-Lingo

Der Funktion zur Erstellung einer Textur wird eine Reihe von Parametern als Property-Liste übergeben:

| Property-Name | Verwendung |
|---|---|
| `p3DMemberName` | Name des 3D-Darstellers |
| `pWidth` | Breite der zu generierende Texturbitmap |
| `pHeight` | Höhe der zu generierende Texturbitmap |
| `pBitmapName` | Darstellername der Bitmap |
| `pSourceName` | Name des Textdarsteller, welcher die zu visualisierende Information beinhaltet |

Tabelle 20 Benötigte Parameter für die Texturerstellung

Der komplette Quellcode für die Erstellung einer Textur, basierend auf dem Image eines Textdarstellers unter Verwendung seiner Alphainformation ist im (Listing 56 Imaging-Lingo für optimale Darstellungsqualität von Textinformation) beschrieben.
Folgende Schritte werden durch den Algorithmus definiert:

- Extraktion der Image-Informationen des Textdarstellers
- Erstellung eines neuen Image mit übergebener Breite und Höhe
- Hintergrund des Image als weiß definieren, notwenig für Transparenz
- Alphainformation des Textdarstellers extrahieren
- Berechnung des Ziel-Rechtecks, damit Text mittig im neuen Image platziert wird
- Kopieren der Textinformationen unter Beachtung der Alphainformationen mittig in das neu erstellte Image
- Überprüfung und ggf. Anpassung des Formates des Image an Potenzen zur Basis 2
- Zuordnung des Image zu einem Bitmapdarsteller
- Zuordnung des Bitmapdarstellers zu einer Textur

Die Nutzung dieses Algorithmus ist dann sinnvoll, wenn eine bestmögliche Darstellungsqualität notwendig ist, die Textinformation nicht als Overlay realisiert werden soll oder kann und für den Zeitraum der Berechnung keine Perspektivänderungen oder Animationen erfolgen.

#### 5.2.3.3.Teile einer Textur unter Nutzung von Texturkoordinaten verwenden

Das Prinzip der Texturanpassung mittels der Manipulation der Texturkoordinaten der Polygone fand im Prototypen in zweierlei Form Anwendung:

- Definition der Textur bei der Darstellung des Downloadstatus auf der Doppeltür vor dem Kino (s. Abb. 5.7)
- Definition der gewählten Bitmapausschnitte für die Anzeige der Bildwiederholrate (s. Abb. 5.8)

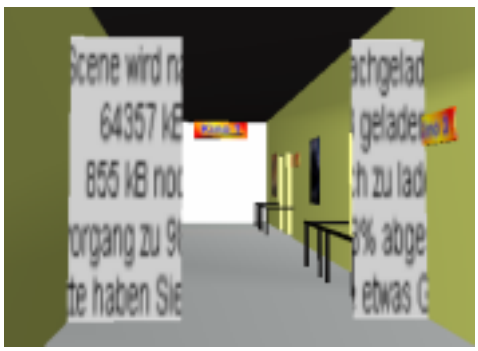


Abb. 5.7 Anzeige Downloadstatus auf Doppeltür

Beim ersten Beispiel wurde per Imaging-Lingo eine Textur, basierend auf einer Textinformation erstellt (siehe Abschnitt 5.2.3.2 Schrifttextur auf Objekten). Damit der Eindruck einer geteilten, gemeinsam genutzten Textur auf den beiden Türen entstand, musste die linke Hälfte der Textur dem linken Modell und die rechte Hälfte dem rechten Modell über eine entsprechende Manipulation der Texturkoordinaten zugeordnet werden. Das Prinzip und die Vorgehensweise soll nachfolgend an dem zweiten realisierten Beispiel erklärt werden.

Für die Performanceanalyse sollte eine FPS-Anzeige (Framerate, Bilder pro Sekunde) realisiert werden. Dazu wurde eine bestehende Lösung (von Timothy Strelchun), welche mehrere Movieskripts nutzte, angepasst und mittels Parentskripts gekapselt. Leitmotiv bei der Verwendung dieses Konzeptes war die Vorgabe, dass die Bildwiederholrate nicht durch zusätzliche CPU-Belastungen für zyklischen Operationen zur Bild- und damit zur Texturmanipulation reduziert werden sollte.


Abb. 5.8 Anzeige der erreichten FPS-Rate innerhalb der 3D-Szene

Grundprinzip dieses Lösungsansatzes ist die Erstellung jeweils einer Modellressource für jeden benötigten Zeichentyp. Dieser Modellressource wird ein Bitmapausschnitt und zwar der, der dem Zeichen entspricht, als Textur zugeordnet. Dazu ist eine Anpassung der Texturkoordinaten der einzelnen Ebenen notwendig. Für die Anzeige der Bildwiederholrate werden somit insgesamt nur 14 Modellressourcen (dargestellt als regulärer Ausdruck [0-9].[0-9][FPS]) mit entsprechend vielen Texturressourcen benötigt.


Abb. 5.9 Ausschnitt der Bitmap für die Texturen

Damit nur ein Teil der gesamten Bitmap (s. Abb. 5.9) als Textur verwendet werden kann, wird bei der Instanzierung eine Liste initialisiert, die den Zeichenumfang und die zeichenspezifischen Texturkoordinaten definiert.

Die Modellressourcen basieren auf einem Ebenen-Primitiv, dessen Geometrie nach Anwendung des `meshDeform`-Modifiers manipuliert werden kann (notwendig für die Anpassung der Texturkoordinaten). Ihre Breite und Höhe wird durch das darzustellende Zeichen bestimmt. Deren Größe wird innerhalb einer Rect-Variable festgelegt.

```
characterDefinitionList = [ [ #ACHAR:".", #ARECT:Rect( 161,  0, 169, 20 ) ], \
                            [ #ACHAR:"0", #ARECT:Rect( 180,  0, 196, 20 ) ], \
                            [ #ACHAR:"1", #ARECT:Rect( 197,  0, 210, 20 ) ], \
                            [ #ACHAR:"2", #ARECT:Rect( 210,  0, 226, 20 ) ], \
```

Listing 38 Auszug aus der Definition der darzustellenden Zeichen

Die Koordinaten der einzelnen Zeichen, die hier bitmapspezifisch sind, werden benötigt um die Texturkoordinaten einer jeden Ebenen-Modellressource anzupassen. Texturkoordinaten bestimmen, wie eine Textur über ein Gitternetz gelegt wird. Bei Modellressourcen vom Typ `#mesh` oder bei einem Modell, an dem der Modifizierer `meshDeform` angebracht ist, können die Texturkoordinaten per Lingo verändert werden.

Die `textureCoordinateList` dient dabei als Hilfsmittel. Sie enthält dimensionslose Wertepaare, deren Definitionsbereich von 0 bis 1 ist. Damit kann das Format jeder Bitmapgrafik und jeder einzelne Bildpunkt repräsentiert werden (das Wertepaar [1,1] entspricht der gegenüberliegenden Ecke des Wertepaars [0,0]). Die Texturkoordinaten eines Polygons werden über die Indizes der Punkte in der `textureCoordinateList` definiert.

```
mrMesh.textureCoordinateList = [[0,0],[1,0],[1,1],[0,1]]
mrMesh.face[1].texcoords = [1,2,3]
```

Listing 39 Definition der textureCoordinateList und Zuweisung der Texturkoordinaten

Abbildung 5.10 und Abbildung 5.11 zeigen beispielhaft einmal die komplette Bitmap als Textur und zum anderen eine Texturierung, die nur die linke Hälfte der Bitmap mit dem Befehl (`textureCoordinateList = [[0,0],[0.5,0],[0.5,1],[0,1]]`)verwendet.

Abb. 5.10 komplette Bitmap als Textur

Abb. 5.11 Teil der Bitmap als Textur

Die Änderung der Darstellung der FPS-Rate erfolgt, indem zyklisch alle 1500ms die Bildwiederholrate bestimmt und der darzustellende String der Funktion `setTextureFontString` als Argument übergeben wird. Der String wird in seine einzelnen Zeichen separiert und entsprechend jeweils ein Modell, basierend auf den bereits definierten Zeichen-Modellressourcen, erstellt. Die einzelnen Modelle werden spezifisch ihrer Breite so verschoben, dass sie nahtlos aneinander gereiht sind, zu einer Gruppe zusammengefasst und im Raum positioniert.

# 6. Darstellungsqualität und Performance

Wie schon öfter angedeutet, ist die Entwicklung virtueller 3D-Darstellungen immer geprägt von der Gegensätzlichkeit von Darstellungsqualität und Renderperformance. Verbesserungen in der Darstellungsqualität durch höhere Geometrieauflösung, größere Texturen oder durch Antialiasing gehen in unterschiedlich starker Ausprägung zu Lasten der Bilderwiederholrate. Im Umkehrschluss lassen sich höhere Bildwiederholraten nur durch Reduzierung der Anzahl der Modelle in einer Szene oder durch Abstriche an der Darstellungsqualität erzielen.
Die Performance einer Directoranwendung lässt sich mittels der erreichten Bildwiederholrate quantifizieren. Je geringer diese Rate ist, umso mehr ruckelt eine Animation, Bewegungen von Kamera oder Objekten erfolgen nicht mehr flüssig. Ein weiteres Problem ergibt sich bei zu geringen Bildwiederholraten dahingehend, dass auf Interaktionen nicht mehr in Echtzeit vom System reagiert wird. In Director kommt ein weiteres Symptom hinzu, wenn die benötigte Rechenleistung die vorhandene übersteigt: Ereignisse, wie zum Beispiel eine Änderung der Mauskoordinaten, werden nicht erkannt. Dieses Phänomen trat bei der Steuerung der Kamera immer dann auf, wenn die Szene zu komplex oder verschiedenste Teilaspekte der Szenensteuerung noch nicht optimiert waren. Das gesamte Projekt war davon gekennzeichnet, eine Balance zwischen der Darstellungsqualität und dem Reaktionsverhalten auf Nutzerinteraktionen zu finden.

Für die Darstellungsgeschwindigkeit ist neben der Leistungsfähigkeit der CPU auch die Grafikkarte entscheidend. Director bietet vier verschiedene 3D-Render-Engines (`#directX7_0, #directX5_2, #openGL, #software`). Drei davon greifen auf hardwareimplementierte Funktionen von Grafikkarten zurück. Leider gibt es eine Reihe von weit verbreiteten 3D-Grafikkarten, deren Hardwareunterstützung von Director nicht fehlerfrei verwendet werden kann (eine entsprechende Liste bietet Macromedia unter [www54] an). Bei diesen Grafikkarten muss auf ein hardwareunterstütztes Rendering verzichtet werden. Stattdessen sollte der langsamere Softwarerenderer verwendet werden.

Um den Einfluss der verschiedenen Rendermodi, unterschiedlicher Geometriekomplexitäten und verschiedener Hardwareausstattung analysieren zu können, wurden diverse Tests entworfen. Als Hardwareplattform dienten 3 verschiedene PCs mit Windows 2000 als Betriebssystem.

| System 1 | System 2 | System 3 |
|---|---|---|
| AMD Duron 700 MHz | AMD Athlon 600 | Intel PIII M 800 MHz |
| ATI Radeon 7500 | 3Dfx Vodoo 3/3000 | ATI Rage128 mobility |

Tabelle 21 Relevante Hardwareausstattung der 3 Testsysteme

Verwendet wurde eine Testszene mit folgenden Eigenschaften:

- 15470 sichtbare Polygone
- 30 Modelle
- 2 Lichter
- Texturspeicherbedarf dekomprimiert 1152 kB

## 6.1. Kantenbildung an ebenen Flächen

Das Problem der nicht korrekten Darstellung beleuchteter Flächen bei geringer geometrischer Auflösung wurde in Abschnitt *3.1.3. Beleuchtung und Schattierung* beschrieben. Nachfolgend sollen die Ursachen für die dargestellten Probleme erläutert werden. Für die Ursachenfindung und die Entwicklung von Lösungsansätzen wurden zwei Testszenen verwendet. Die erste Szene enthält ein Objekt, hervorgegangen aus einer booleschen Operation zweier nicht segmentierter Quader und außerdem ein Punktlicht in Höhe der rechten oberen Ecke der Öffnung. Bei diesem Testmodell soll im Folgenden untersucht werden, wieso es zu diesen Darstellungsfehlern kommt und mit welchem zusätzlichen Aufwand eine akzeptable Visualisierungsqualität erreicht werden könnte. Mit Hilfe der zweiten Testszene soll (als Ergänzung zum Abschnitt *3.1.4. Simulation eines Spotlichtkegels*) erläutert werden, mit welchen Methoden eine korrekte Darstellung einer Spotlichbeleuchtung (ohne Volumenlicht) realisiert werden kann.

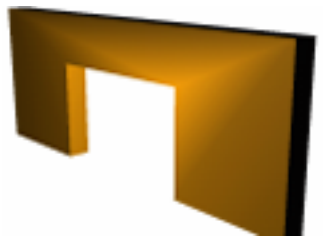


Abb. 6.1 Ansicht nach W3D-Export

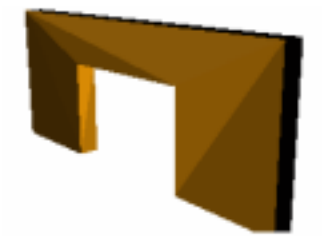


Abb. 6.2 Flatshading in Director3D

Abbildung 6.1 zeigt die Anordnung der Dreiecksflächen der Vorderseite unter Nutzung des Flatshadingmodus (jede Dreiecksfläche erhält einen Farbwert, der sich aus der mittleren Helligkeit der Fläche ergibt) noch deutlicher als dies in Abbildung 6.2 der Fall ist.

Neben dem verwendeten Shadingmodus bestimmen die Flächennormalen, die Position der Kamera, die Position und Farbe des Lichtes sowie die Materialeigenschaften die Schattierung von Objekten. Die erste Vermutung lag darin, dass die Normalen der Dreiecksflächen einer Fläche voneinander abweichen Das Testobjekt besteht, ohne Segmentierung, aus 16 Eckpunkten, 28 Dreiecken und umfasst eine Vertex- und Normalist von 48 Elementen. Jeder Eckpunkt wird von Dreiecksflächen verwendet, die in drei verschiedenen Ebenen liegen. Somit enthält die Vertexlist 16 verschiedene Scheitelpunkte, die jeweils dreimal vorkommen, sich jedoch in ihren zugeordneten Normalen unterscheiden. Für die weitere Betrachtung soll zur Vereinfachung nur die der Kamera zugewendete Fläche analysiert werden. Diese Fläche liegt in der xz-Ebene und besteht aus 6 Dreiecksflächen. Damit man Zugriff auf die Geometriedaten eines importierten Modells erhält, muss diesem der `meshDeform`-Modifier zugeordnet werden.

```
pMember.model("Quader01").addModifier(#meshDeform)
```

Listing 40:Zuordnung des meshDeform-Modifiers zu einem Modell

Mit einem einfachen Skript (Listing 57 Bestimmung des gepickten Polygons und dessen Scheitelpunkte) lassen sich der Index des gepickten Polygons und darauf aufbauend die zugehörigen Scheitelpunkte bestimmen. Mit Hilfe des 3DPI können die den Scheitelpunkten zugeordneten Normalen untersucht werden (ohne dieses Tool müssten die Normalen mit Hilfe des Kreuzproduktes manuell bestimmt werden). Dabei fällt auf, dass die Normalenvektoren nicht dem erwarteten Vektor (0,-1,0) entsprechen. Die Werte unterscheiden sich in der dritten Kommastelle vom Normalenvektor der xz-Ebene.

Eine Ursache für diese Abweichung ist nicht eindeutig bestimmbar. Der Fehler kann beim Export entstanden oder aber erst mit der Anwendung des `meshDeform`-Modifiers erzeugt worden sein.

Um den Einfluss der Abweichung der Normalenvektoren zu verifizieren, wurde ein Referenzobjekt, basierend auf einer Gitternetzressource, ohne Textur per Lingo erstellt (Listing 58 Erstellung Referenzobjekt für Untersuchung der Darstellungsqualität). Die Vertexpunkte des neuen Objektes stimmen mit denen des importierten, geboolten Objektes überein. Nachdem die Normalen des Referenzobjektes generiert wurden (`mrMesh.generateNormals(#smooth)`, konnte festgestellt werden, dass die Normalen der Polygone dem erwarteten Vektor(0,-1,0) entsprechen. Trotzdem ist die fehlerhafte Darstellung der beleuchteten Fläche auch an diesem Objekt feststellbar, d. h. die geringe Abweichung der Normalen des importierten Objektes sind nicht die Ursache für die Darstellungsprobleme.

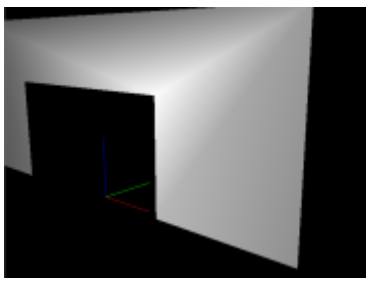


Abb. 6.3 Fehlerhafte Darstellung des Referenzobjektes

Damit blieb als mögliche Ursache nur der verwendete Shadingmodus. Shockwave verwendet als Beleuchtungsmodell entweder den Flat-Shading oder den Smooth-Shading-Algorithmus (respektive Gouraud-Shading, Intensity Interpolation Shading, Color Interpolation Shading oder auch Vertex Lighting)

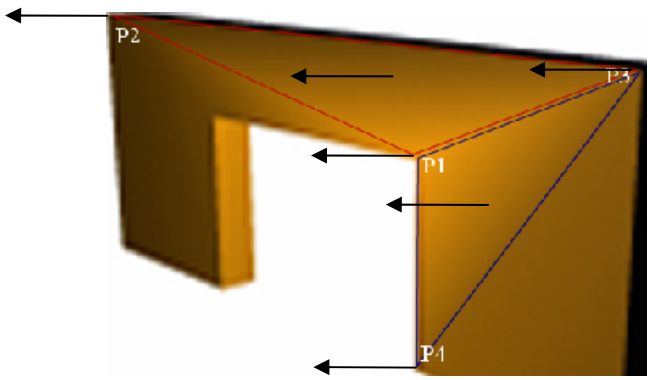



Abb. 6.4 Gouraud-Shading am Beispiel des geboolten Objektes

Bei diesem Algorithmus werden die Scheitelpunktnormalen basierend auf den Flächennormalen der angrenzenden Polygone gemittelt (im Beispiel des Referenzobjektes entsprechen auch die gemittelten Scheitelpunktnormalen dem Vektor (0,-1,0) da alle Flächennormalen konstant sind). Aus der Richtung des einfallenden Lichts und den ermittelten Scheitelpunktnormalen, kann der Lichteinfallswinkel berechnet werden. Davon abhängig ist dann der Einfluss der Lichtquelle auf die Scheitelpunktfarbe. Nur für die Scheitelpunkte (in Abb. 6.4, also für Punkte $P_1P_2P_3$ bzw. $P_1P_3P_4$ usw.) werden die Farbwerte unter Berücksichtigung der Lichtquellen berechnet. Die Farbwerte der Punkte, die auf einer Kante zwischen zwei Scheitelpunkten (z. B. $P_1P_3$) liegen, werden, basierend auf den Scheitelpunktfarbwerten, interpoliert.

Sind alle Punkte (repräsentiert durch $P_a$ und $P_b$) auf den Kanten berechnet worden, wird das Innere der Polygone ($P_i$) durch scanline-orientierte Interpolation bestimmt (s. Abb. 6.5)

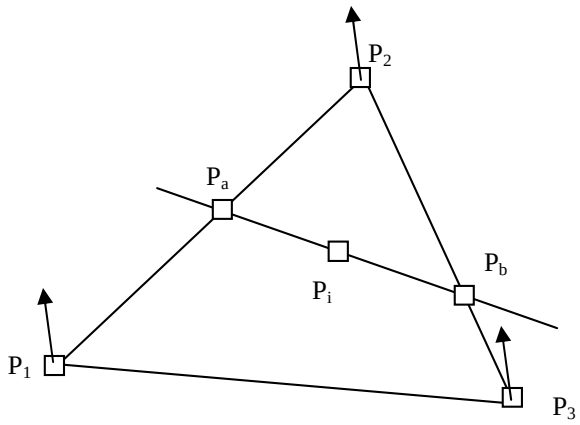



Abb. 6.5 Prinzip der Kanten- und Flächeninterpolation

Um das Problem der Kantenbildung zu minimieren, gibt es eigentlich nur zwei Möglichkeiten: Entweder die Fläche wird gleichmäßig beleuchtet (nicht sinnvoll realisierbar) oder ihre geometrische Auflösung muss erhöht werden. Dies kann entweder durch Segmentierung während der Modellierungsphase (siehe Abschnitt *3.1.2. Detailgenauigkeit und Geometrieauflösung*) oder zur Laufzeit per Lingo erfolgen. Für den zuletzt genannten Lösungsansatz wird der SDS-Modifier (Sub-Division Surfaces, Oberflächenunterteilung) verwendet.. Allerdings kann dieser Modifier nicht auf Modelle angewendet werden, die aus einer booleschen Operation einer C- oder L-Extrusion mit einem Quader (für Türen- und Fensteröffnungen) entstanden sind (betrifft einige Wände des Wandelhof-Komplexes). In diesem Fall erfolgt eine nicht gewollte Veränderung der geometrischen Form des Modells (siehe nachfolgende Abbildungen).

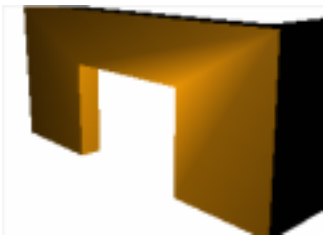


Abb. 6.6 Geboolte L-Extrusion ohne SDS-Modifier

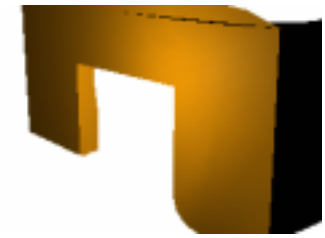


Abb. 6.7 Fehlerhafte Geometrie nach Anwendung SDS-Modifier

Basieren die zu manipulierenden Objekte jedoch auf Quaderprimitiven, die ggf. mit booleschen Operationen verändert wurden (s. Abb. 6.8), so kann der SDS-Modifier problemlos angewendet werden. Pro Wiederholung des SDS-Algorithmus wird im Modus `#uniform` eine Dreiecksfläche in jeweils 4 neue Dreiecksflächen unterteilt. Bei 3 Wiederholungszyklen des SDS-Modifiers wird eine akzeptable Darstellungsqualität erreicht. Die Testfläche wird mit einer gleichmäßig abnehmenden, kreisförmigen Schattierung gerendert (s. Abb. 6.9). Da das Modell aus 28 Dreiecksflächen besteht, besitzt es nach Anwendung des SDS-Modifiers mit 3 Wiederholungen $28*4^3=1792$ Teilflächen. Erst wenn die Fläche der einzelnen Dreiecksflächen eine „gewisse Maximalgröße“ nicht übersteigt, erfolgt eine korrekte Visualisierung. In dem gewählten Testbeispiel ist dafür eine 64-fache Menge der ursprünglichen Geometrieinformationen notwendig.

Da in einer Szene mehrere Modelle (Wandelemente) gleichzeitig in ihrer Geometrieauflösung verändert werden mussten, war eine optimale Darstellung mit Hilfe des SDS-Modifiers auf Grund der damit verbundenen Verschlechterung der Renderperfomance (auf dem Test- und Referenzsystem) nicht realisierbar. Auf die Entwicklung eines Algorithmus für eine selektive Erhöhung der Geometrieauflösung einzelner Modelle in Abhängigkeit der Kameratransformation wurde verzichtet, da die Bildwiederholrate auch ohne Anwendung des SDS-Modifiers teilweise schon weniger als 10 FPS betrug.

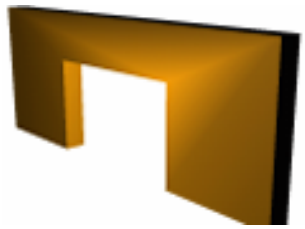


Abb. 6.8 Testmodell ohne SDS-Modifier

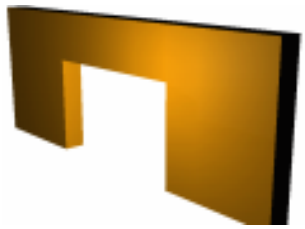


Abb. 6.9 Testmodell nach Anwendung des SDS-Modifiers

Folgende Beispielszene soll im zweiten Beispiel zur Simulation eines typischen Spotlichteffekts als Grundlage dienen:

- Ebene 300x300, 4 Breiten- und Längensegmente, 32 Polygone
- Würfel 30x30x30, keine Segmentierung, 12 Polygone
- Spotlicht, FallOff 45° Höhe150, zentriert über Objekten

Betrachtet werden sollen die Quantität der geometrischen Auflösung und die Qualität der Form der beleuchteten Fläche. Pro Wiederholung des SDS-Algorithmus wird ein Polygon in 4 neue Polygone unterteilt.

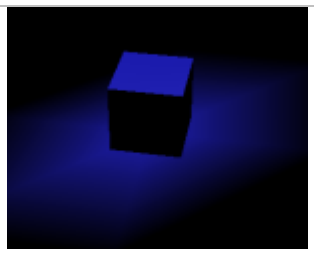
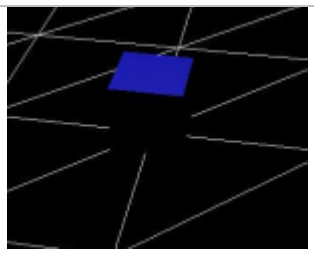
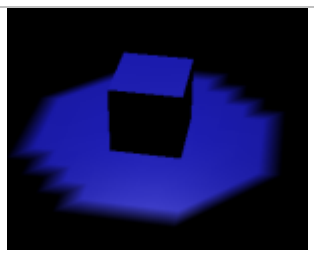
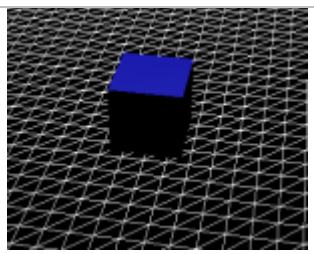
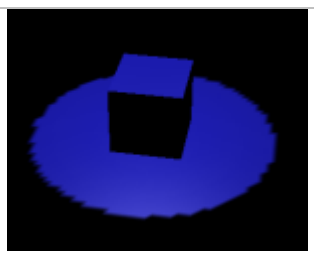
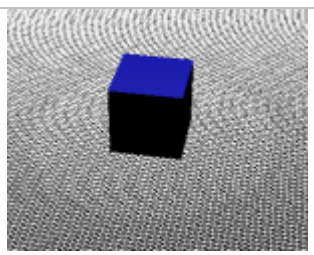
| | | |
|---|---|---|
| Kein SDS Modifier, 32 Polygone |  |  |
| 1 Wiederholung 128 Polygone | | |
| 3 Wiederholungen 2048 Polygone |  |  |
| 4 Wiederholungen 8192 Polygone | | |
| 5 Wiederholungen, 32768 Polygone |  |  |

Abb. 6.10 Testreihe zur Simulation eines typischen Spotlichtes

Erst bei einer sehr großen Anzahl von Polygonen und der damit verbundenen relativ kleinen Polygonfläche wird eine annähernd spotlicht-typische Schattierung erreicht. Allerdings ist das Verhältnis zwischen der erreichten Darstellungsqualität und der damit einhergehenden Performanceverschlechterung als sehr ungünstig einzuschätzen.

Es gibt jedoch schon bei der Modellierung Möglichkeiten, solche gekrümmten Schattierungsflächen, wie nachfolgendes Beispiel zeigt, zu unterstützen:

- Zylinder, obere Fläche beleuchtet,
- Resolution 20: 20x2+2 Polygone bilden obere Fläche
- Radius 45, entspricht etwa dem erwarteten Spotlichtkreis

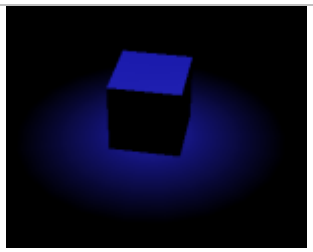
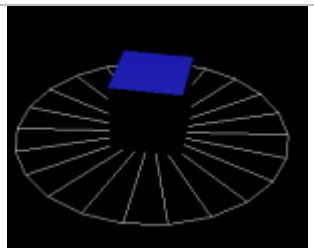
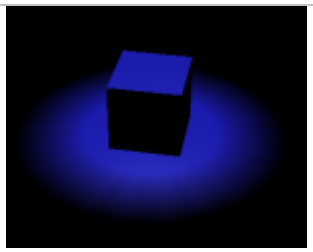
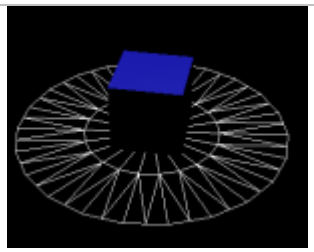
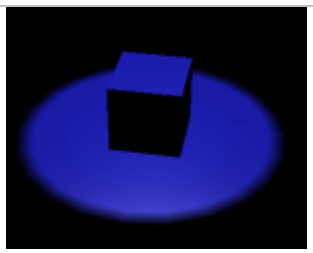
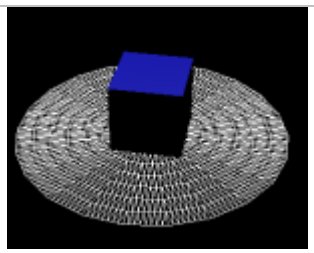
| | | |
|---|---|---|
| Kein SDS Modifier, 22 Polygone |  |  |
| 1 Wiederholung 88 Polygone |  |  |
| 3 Wiederholungen: 1408 Polygone |  |  |

Abb. 6.11 Zweite Testreihe zur Spotlichtsimulation mit veränderter Basisgeometrie

Wie in diesem Beispiel zu sehen ist, kann im Verhältnis zur ersten Testszene mit einer relativ geringen Geometrieauflösung eine sehr gute Nachahmung des Spotlichtes realisiert werden. Der Modellierungsaufwand wird allerdings bei einer solchen Lösung größer. Die Fläche des Spotlichtes muss durch ein einzelnes Zylinder-Modell repräsentiert werden um per SDS-Modifier die Geometrieauflösung zur Laufzeit erhöhen zu können. Eventuell ist es sogar erforderlich, eine Teilfläche des Bodens, welche sich unter dem „Spotlichtzylinder“ befindet, per boolscher Operation zu entfernen um Probleme in der Tiefensortierung zu vermeiden (siehe Abschnitt *3.1.5. Problem der Shockwave-Tiefensortierung*). Dieser Mehraufwand, die trotzdem erhöhte Polygonanzahl und die Notwendigkeit den Spotlichtkegel zu imitieren, haben zu der in Abschnitt *3.1.4. Simulation eines Spotlichtkegels* beschriebenen letztendlichen Lösung geführt.

## 6.2. Renderperformance in Abhängigkeit der Geometrieauflösung

Da im vorhergehenden Kapitel eine Erhöhung der Geometrieauflösung als eine Möglichkeit der Verbesserung der Visualisierungsqualität dargestellt wurde, soll bei dem nachfolgenden Vergleich eine Tendenz des Einflusses der Geometrieauflösung bei verschiedenen Rendermodi abgeleitet werden. Als Vergleichsgröße diente hierzu die erreichte durchschnittliche FPS-Rate.

Die Testumgebung war wenig praxisnah, da die Ergebnisse im Autorenmodus entstanden, was zur Folge hatte, dass eine signifikante CPU-Belastung (ca. 30%, ohne dass der Film abgespielt wird) durch die Entwicklungsumgebung allein schon verursacht wurde. Die unten abgebildete Testreihe (Abb. 6.12) wurde unter Verwendung von System 1 generiert. Die Anzahl der Polygone wurde unter Nutzung des SDS-Modifiers variiert und mit Hilfe des Mesh-Modifiers verifiziert.

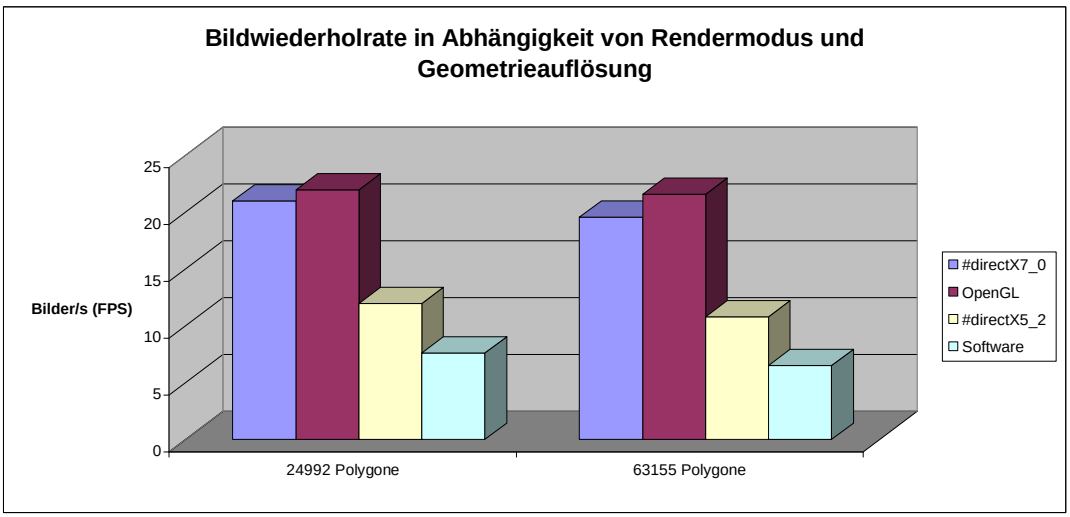



Abb. 6.12 Maximale Bildwiederholrate bei veränderter Polygonanzahl

Obwohl in der zweiten Testreihe die Anzahl der Polygone 2½ mal so groß wie in der ersten Variante waren, fiel die Bildwiederholrate nur relativ gering ab. Ursache dafür ist wahrscheinlich die Tatsache, dass Director automatisch eine Polygonreduzierung durchführt, wenn die vorgegebene Framerate (hier 30 FPS) nicht erreicht wird. Ansonsten zeigt das Diagramm die fast typische Renderleistung der einzelnen Modi, allerdings mit der Ausnahme, dass der `#openGL`-Modus eine ungewöhnlich gute Performance aufweist. Interessant war die Erkenntnis, dass die CPU-Belastung nicht anstieg, wenn das Objekt im Raum transformiert wurde.

## 6.3. Verbesserte Darstellungsqualität mittels Antialiasing

Antialiasing für 3D-Darsteller und die zugehörigen Befehle werden erst mit der Director-Version 8.5.1 unterstützt. Daher ist bei der Initialisierung des Filmes die Abfrage der Version und das Setzen der entsprechenden Flags notwendig (Listing 59 Abfrage auf Antialiasingsupport). Die Prozessorbelastung für Szenen mit aktiviertem Antialiasing ist so hoch, dass Macromedia die Benutzung nur bei statischen Szenen empfiehlt. Das Ein- und Ausschalten des Antialiasings ist in einer separaten Funktion realisiert (Listing 60 Aktivieren bzw. Deaktivieren des Antialiasings). Damit kann das (De-)Aktivieren der Kantenglättung von verschiedenen Funktionen in unterschiedlichen Skripten mit Hilfe des `sendSprite`-Befehls angestoßen werden. Der Befehl wird auf einer 3D-Spritereferenz ausgeführt und glättet alle sichtbaren Kanten. Die Beschränkung des Antialiasings auf einzelne Modelle ist nicht möglich.

Abb. 6.13 Vergleich Szenenausschnitt ohne / mit Antialiasing

Auch mit aktiviertem Antialiasing bleiben die typischen treppenartigen Artefakte erkennbar (s. Abb. 6.13). Im Hinblick auf die Verschlechterung der Renderperformance und des Echtzeitverhaltens auf Nutzerinteraktionen ist im Allgemeinen von der Nutzung des Antialiasings abzuraten. Diese Aussage soll mit der nachfolgenden Untersuchung belegt werden.

## 6.4. Renderperformance mit und ohne Antialiasing

Für die Testuntersuchungen wurde wiederum das System 1 verwendet. Die 3D-Testgeometrie bestand aus über 15000 Polygonen. Der OpenGL-Treiber der Grafikkarte unterstützte keine Kantenglättung. Daher sind für diesen Rendermodus keine Werte erfasst worden. Die Werte wurden im Projektormodus gewonnen.

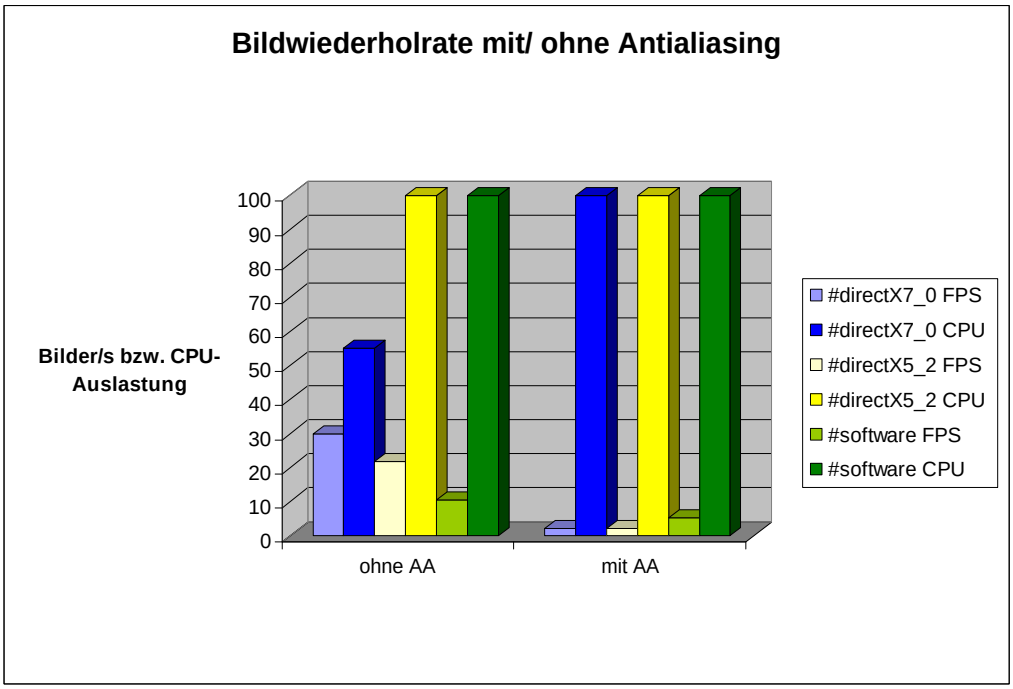



Abb. 6.14 Bildwiederholraten mit/ ohne Kantenglättung

Die Daten ohne Antialiasing entsprechen tendenziell den Ergebnissen der vorhergehenden Untersuchungen. Wird jedoch die Kantenglättung aktiviert, so reduziert sich die Rendergeschwindigkeit auf ein Minimum von ca. 2 FPS bei hardwareunterstütztem Rendermodus. Im Modus `#software` ist die Bildwiederholrate zwar mehr als doppelt so groß (ca. 5,3 FPS), jedoch trotzdem nicht akzeptabel, wenn es sich bei der Szene um eine dynamische Darstellung mit entsprechender Nutzerinteraktion handelt.

## 6.5. Renderperformance in Abhängigkeit des Abspielmodus

Die letzte hier dargestellte Testreihe bestand darin, zu vergleichen, welche Performance in welchem Abspielmodus erzielt wurde. Dazu wurde eine statische 3D-Szene mit konstanter Polygonanzahl verwendet. Die maximale Bildwiederholrate wurde im Film auf 30 FPS begrenzt, da es bei höheren Raten zu Abstürzen oder Fehlfunktionen bei der Wiedergabe im Projektor kam. Das nachfolgende Diagramm zeigt zuerst immer die erreichte Bildwiederholrate und mit dem nächsten Balken die dazu gehörige CPU-Belastung. Das heißt, Balken in einem ähnlichen Farbton basieren auf dem selben Rendermodus. Insgesamt wurden alle vier Rendermodi mit allen Wiedergabeumgebungen kombiniert.

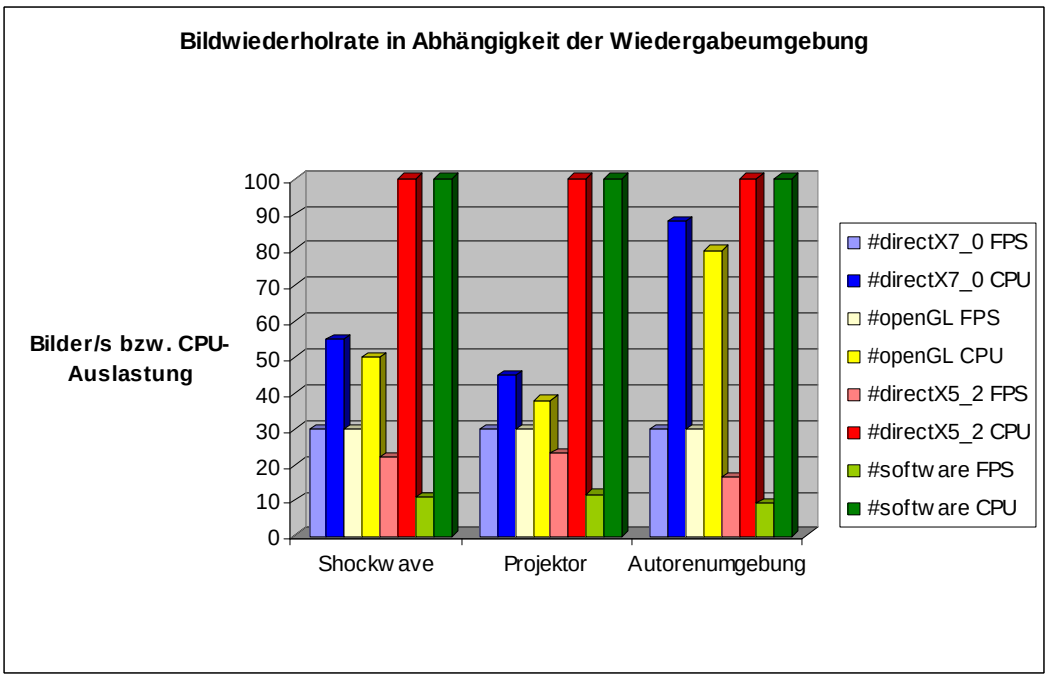



Abb. 6.15 Performanceuntersuchung nach Wiedergabeumgebung

Wie in Abb. 6.15 erkennbar ist, wird in allen drei Wiedergabeumgebungen im Rendermodus `#directX7_0` und `#openGL` die vorgegebene Framerate von 30 Bildern pro Sekunde erreicht. Allerdings unterscheiden sich die dafür benötigten Prozessorleistungen voneinander. Die 3D-Testszene im Browser abgespielt benötigt 10-15% mehr Prozessorleistung als dies bei einer Darstellung im Projektormodus der Fall ist. Die Werte im Autorenmodus sind schon deshalb höher, weil Director selbst eine Prozessorlast von mehr als 30% auf dem Testsystem erzeugt.

Werden die Rendermodis `#directX5_2` oder `#software` verwendet, so wird die erreichbare Bilderwiederholrate von der CPU-Leistung begrenzt. Bei maximaler Auslastung der CPU wird die Zielframerate von 30 FPS nicht mehr erreicht. Im Softwaremodus ist die Bildwiederholrate noch einmal etwa 50% geringer als beim `#directX5_2`-Rendermodus.

Die Unterschiede in der erreichten Framerate in der Shockwave- bzw. Projektorumgebung sind bei diesen beiden Rendermodi nicht sehr ausgeprägt (<1FPS). Nachfolgende Darstellung ( s. Abb. 6.16) soll den Zusammenhang zwischen Rendermodus und CPU-Auslastung noch einmal veranschaulichen. Die Zahlen sind dabei wie folgt den einzelnen Rendermodi zugeordnet:

(1) `#software` (3) `#directX7_0`

(2) `#directX5_2` (4) `#openGL`

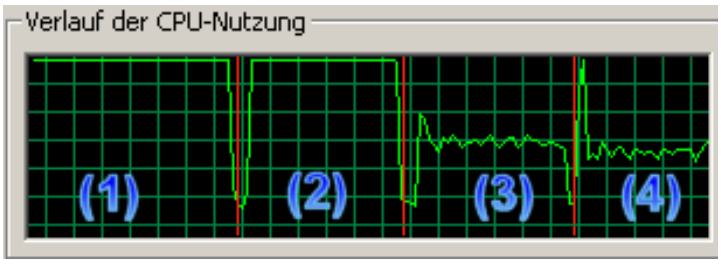



Abb. 6.16 Verlauf der CPU-Belastung bei Shockwavedarstellung

## 6.6. Zusammenfassung

Die Ergebnisse der Untersuchungen können wie folgt zusammengefasst werden:

- Die Autorenumgebung bietet im Vergleich zur Shockwave- und Projektordarstellung die schlechteste Performance (Ursache: siehe Abschnitt *6.2. Renderperformance in Abhängigkeit der Geometrieauflösung*). Dieser Nachteil spielt jedoch in Bezug auf die letztendliche Präsentationsform nur eine untergeordnete Rolle.
- Ein Directorfilm wird im Projektormodus bezogen auf die Präsentation im Webbrowser ca. 10-15% schneller wiedergegeben (bzw. die Prozessorlast ist bei gleicher FPS-Rate entsprechend geringer).
- Die Wertigkeit der Rendermodi wird allgemein für die Windows(9x,2000,XP)-Plattform in der Reihenfolge `#directX7_0, #directX5_2, #openGL, #software` angegeben. Die verwendete ATI 7500-Grafikkarte bietet jedoch die beste Performance im `#openGL-Modus`. Dieses Ergebnis ist jedoch hardwarespezifisch und spricht für die Qualität der Karte und besonders für deren OpenGL-Treiber.
- Die Performance im Softwaremodus ist im Vergleich zu den hardwareunterstützten Modi deutlich schlechter.
- Die Performance der einzelnen Systeme war im `#software`-Modus etwa gleich. Dies erklärt sich dadurch, dass in diesem Fall die annähernd gleiche CPU- Performance der einzige leistungsbestimmende Faktor ist.
- Die Vodoo3-Grafikkarte bestand die Hardware-Rendering-Tests nicht, es kam zu Abstürzen der Testpräsentation. Manchmal stürzte auch das Betriebssystem bei dem Versuch ab, einen nicht unterstützten Rendermodus zu erzwingen.
- Die Performanceverluste bei aktiviertem Antialiasing verbieten fast von selbst die Nutzung dieses Features in veränderlichen 3D-Szenen.
- Das Testobjekt, mit seinen wesentlich mehr Polygonen als in der Wandelhof-Szene, lies sich auf dem Testsystem 1 in allen Rendermodi ohne zeitliche Verzögerung drehen und zoomen. Ein Indiz dafür, dass die Performanceprobleme im Projekt zum großen Teil auf die komplexen und zyklisch aller 50 ms durchgeführten Algorithmen der Kamerasteuerung zurückzuführen sind.

Die Problematik von Renderperformance und Optimierung der Darstellungsqualität ist ein sehr vielschichtiges Thema, das viel praktische Erfahrung im Umgang mit Director3D und dem entsprechenden Modellierungswerkzeug erfordert. Die Bedeutung konnte hier nur auszugsweise dargestellt werden. Nützliche praktische Hinweise bieten unter anderem die Quellen [BIE02] und [GRO00].

# 7. Schlussbetrachtungen

## 7.1. Was bringt Director MX?

Auch wenn zum Zeitpunkt der Erstellung noch keine deutsche Trialversion des Autorenwerkzeuges zum Download auf der Webseite von Macromedia angeboten wurde, kann ein Eindruck von den neuen Features mit der englischsprachigen Testversion gewonnen werden.

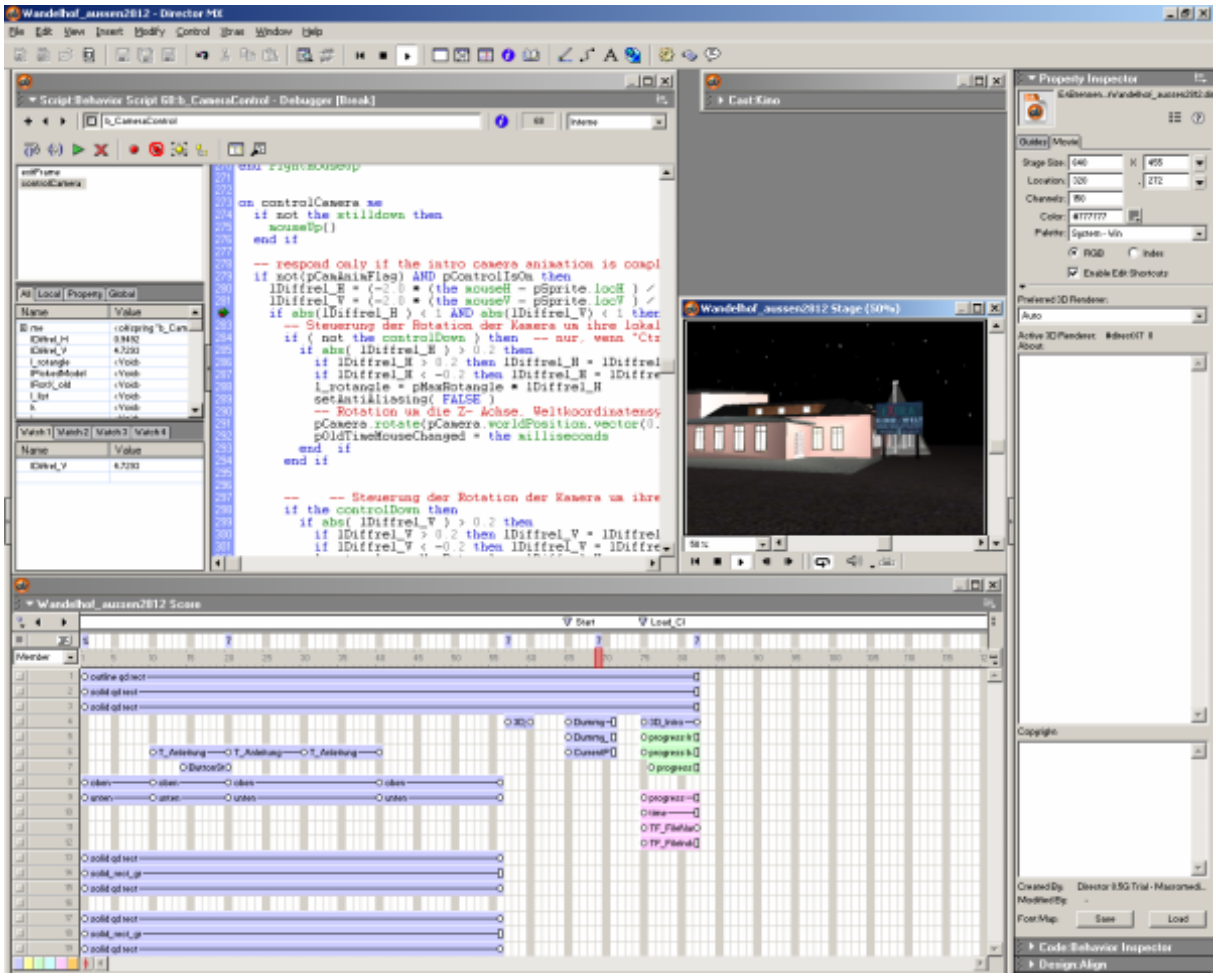


Abb. 7.1 Das neue Userinterface von Director MX

Von Director MX ist hinsichtlich der 3D-Unterstützung und der Darstellungsqualität keine Verbesserung zu erwarten. Es sind lediglich einige Xtras, die Unterstützung von Flash MX und Mac OS X sowie ein angepasstes Userinterface mit verbesserten Debuggingmöglichkeiten hinzugekommen. An der Renderengine hat es nach Aussage von Thomas Biendorf keine Veränderungen gegeben:

„Alles in allem enttäuscht Director MX in mehreren Punkten: Die NT-Unterstützung aufzugeben halte ich für falsch: wer im e-learning Bereich tätig ist, wird häufig für NT entwicklen müssen. Die Aufgabe des MU-Server ist sehr traurig und bringt viele Entwickler in Erklärungsnot gegenüber dem Kunden bzgl. der Investitionssicherheit des gewählten Produktes. Ob der Flash Communication Server den MU Server ersetzen kann wird die Zeit zeigen. Das neue Interface ist gewöhnungsbedürftig aber funktional, wenn auch in einigen Punkten sehr umständlich und sehr Platz verschwendend! Drei Monitore werden wohl nötig sein ;-).

Das allerschlimmste Manko aber ist: Die Player-Engine wurde nicht überarbeitet, also alt bekannte Bugs wurden weitergeführt. Im Herzen ist Director MX ein um das Userinterface und einige Xtras aufgebohrter Director 8.5.1 - ob das einen Versionsprung rechtfertigt??“[www55]

Die kritische Einstellung der Directornutzer bezüglich der neuen Version ist auch in den Diskussionen innerhalb der Director-MailingList [www47] erkennbar:
„Anscheinend ist Macromedias Engagement für 3D bestenfalls halbherzig. Das merkt man auch daran, dass man 3D-Cast-Member immer noch nicht abspeichern kann, und dass es immer noch kein Drehbuch o. ä. für 3D-Cast-Member gibt. Der 3DPI bleibt unerlässlich (ThanksUllala!).“ (Auszug aus einer Mail von Stefan Ladstaetter vom 23.01. 2003 an Director-MailingList)
Ob und wie der Support und die Entwicklung von Director, sowie speziell der 3D-Engine zukünftig auch in Hinblick auf Microsofts Interesse an der Flashtechnologie aussehen wird, bleibt abzuwarten. Allerdings wäre es wünschenswert, wenn dieses mächtige und interessante Autorenwerkzeug auch in Zukunft seine Reputation als multimediales Werkzeug behält.

## 7.2. Fazit

Im Rahmen der Diplomarbeit wurden die Möglichkeiten von Director zum Teil sehr intensiv getestet und verwendet. Neben der 3D-Implementation, die im Mittelpunkt der Arbeit stand, wurden verschiedene weitere Aspekte von Director berührt. So mussten u. a. auch Teillösungen realisiert werden, die auf NetLingo, Imaging-Lingo oder dem XML-Parsing-Xtra beruhen.

Der größte Teil der 3D-Philosophie von Director wurde in den Kapiteln zuvor beschrieben. Verzichtet wurde aus Prioritätsgründen auf eine eingehende schriftliche Darstellung von Knochenanimationen, die in dem Projekt keine Rolle spielten, auf Keyframeanimationen, die u. a. für die Animation der Türen verwendet wurden und auf die Beschreibung von Emitterprimitiven, die ebenfalls keine Verwendung fanden. Außerdem fehlen in der Ausarbeitung Ausführungen hinsichtlich verschiedener nicht verwendeter Standardmodifier (`Inker-, Toon-, LevelOfDetail`-Modifier).

Um die prototypische Entwicklung in eine reale Nutzung überführen zu können, sind neben der weiteren inhaltlichen Ausstattung der Szenerie zusätzliche Arbeiten bezüglich der Fehlerrobustheit, der Performanceoptimierung, der Plattformunabhängigkeit (abweichendes Downloadverhalten auf dem Mac) und der Browserproblematik notwendig.

Prinzipiell konnte nachgewiesen werden, dass auch komplexe Aufgabenstellungen, die ihren Schwerpunkt bei der dreidimensionalen Darstellung setzen, mit Director realisiert werden können.

Projekte, die auf einer bestehenden 3D-Datenbasis aufbauen und nur Standardinteraktionen (Objekte drehen und zoomen) abdecken müssen, können mit relativ geringem Aufwand unter Verwendung von mitgelieferten Verhaltensskripten internettauglich realisiert werden. Falls auf lange gerade Kanten und relativ große Dreiecksflächen in der Szene verzichtet wird und die fehlende Schattendarstellung kein großer subjektiver Nachteil ist, kann der Shockwave3D-Darstellung eine gute Qualität (und Performance) bescheinigt werden.

Kann nicht auf eine komplett fertig exportierte Szene aus einem 3D-Werkzeug zurückgegriffen werden, so ist neben Director ein weiteres, (teures) Softwarepaket für eine komplexe Szenengestaltung notwendig. Die Modellierungsmöglichkeiten von Director sind, im Vergleich zu anderen Web3D-Autorenwerkzeugen (Cult3D-Editor, Adobe Athmosphere-Editor), die wenigstens grundlegende 3D-Modellierungen unterstützen, sehr begrenzt.

Fehlende digitalisierte Konstruktionsunterlagen, unzureichende technische Zeichnungen und der damit verbundene Mehraufwand für die Erstellung der Wandelhof-Szene haben die Arbeit erschwert und nicht eingeplante zeitliche Ressourcen erfordert. Eine weitere, nur auf Fotos basierende, detaillierte Nachgestaltung zusätzlicher Räumlichkeiten, die zum Teil sehr komplexe Konstruktionen beinhalten (freies Fachwerk in der Kellerbar, verwinkelte Gänge, Treppen usw.) ist mit den Erfahrungen dieser Diplomarbeit als (fast) unmöglich einzuschätzen. Mit dieser Aussage soll nicht die Entscheidung zu einer Diplomarbeit mit Director3D in Frage gestellt werden. Nur die Wahl des darzustellenden Objektes war etwas unglücklich.

Vor die nochmalige Wahl eines Diplomthemas gestellt, würde ich wieder die 3D-Funktionalitäten von Director analysieren wollen, vielleicht dann aber weniger unter dem Aspekt der Szenenmodellierung und Navigation als vielmehr der Steuerung der Szene unter physikalischen Gesetzmäßigkeiten. Das Havok-Xtra-Konzept zur physikalischen Simulation ist eines der spannendsten und anspruchvollsten Themen, die im Zusammenhang mit Director und 3D auf eine diplomwürdige Aufgabenstellung warten.

# Anhang

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Literaturverzeichnis

Verzeichnis der Internet-Links

Glossar

Selbständigkeitserklärung

# Listings

zugehörige Erläuterungen hier

```
#VRML V2.0 utf8
# Der Kommentar der ersten Zeile ist zwingend vorgeschrieben
# Einfaches Hello World Programm das einen String und ein 3D VRML Modell
# darstellt animiert und einen Link ins World Wide Web darauf legt
Background{
   skyColor 0 0 1
}
Anchor{
   children
      [
      DEF SCHRIFT  Transform{
         children
            [
            Inline { url "tux.wrl" }
            Shape{
               geometry Text{
                  string ["Hello World"]
               }
            }
            ]
      }
      ]
   description "Hello World"
   url "http://www.csv.ica.uni-stuttgart.de/vrml/vrml_csv.html"
}
DEF ROTOTATOR OrientationInterpolator{
   key [0, 0.5, 1]
   keyValue [ 0 1 0 0, 0 1 0 3.14, 0 1 0 6.28 ]
}
DEF TIMER TimeSensor{
   cycleInterval 10
   loop TRUE
}
ROUTE TIMER.fraction_changed TO ROTOTATOR.set_fraction
ROUTE ROTOTATOR.value_changed TO SCHRIFT.set_rotation
```

Listing 41 Aufbau einer einfachen VRML-Szene, aus [www56]

zugehörige Erläuterungen hier

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D>
   <Scene>
      <Transform Parameter>
         <Shape>
            <Appearance>
               <Material Parameter>
               </Material>
               <ImageTexture Parameter>
               </ImageTexture>
            </Appearance>
            <IndexedFaceSet Paramter & Felder>
               <Coordinate Feld>
               </Coordinate>
               <TextureCoordinate Feld>
               </TextureCoordinate>
            </IndexedFaceSet>
         </Shape>
      </Transform>
   </Scene>
</X3D>
```

Listing 42 Pseudo-Code einer X3D-Szene [GÖT02]

zugehörige Erläuterungen hier

```
sphereList=[]
positionList=[1,1,1,-1,1,1,1,-1,1,-1,-1,1,1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1]
repeat with i=1 to 8
  sphereList.add(pMember.newModel("Sphere"&i,mr_sphere))
  sphereList[i].translate(positionList[i*3-2]*\
                        length,positionList[i*3-1]*length,positionList[i*3]*length)
end repeat
```

Listing 43 Erstellung und Translation der Szenenmodelle

zugehörige Erläuterungen hier

```
 lCount = pMember.model.count
 repeat with i= 1 to lCount
   if pMember.model[i].name contains "Ebene_Fenster" then
     lModel = pMember.model[i].clone("BG_"& pMember.model[i].name)
     lModel.shaderList[1] = pShaderLightOn
   end if
 end repeat
```

Listing 44 Duplizierung der Fensterebenen

zugehörige Erläuterungen hier

```
lModel = pMember.model("Ebene_Fenster00")
lTexture=pMember.newTexture("WndTexture_alpha")
lTexture.member = member("Fenster_mAlpha")
lModel.shaderList[1].texture = lTexture
```

Listing 45 Definition der Fenstertexturen

zugehörige Erläuterungen hier

```
-- Maske für Licht der Fenster erstellen
w = member("Fenster_mAlpha").width
h = member("Fenster_mAlpha").height
im = image(w,h,32,8)
am_neg = image(w,h,8)
am = member("Fenster_mAlpha").image.extractAlpha()
repeat with i = 1 to w
  repeat with j = 1 to h
    if am.getPixel(i,j) = paletteIndex(255) then
      am_neg.setPixel(i,j, paletteIndex(1))
    else
      am_neg.setPixel(i,j, paletteIndex(255))
    end if
  end repeat
end repeat
im.setAlpha(am_neg)
lTxr = pMember.newTexture("txr_wndLight", #fromImageObject, im)
pShaderLightOn = pMember.newShader("shLightOn",#standard)
pShaderLightOn.emissive = rgb( 255, 251, 240 )
pShaderLightOn.texture = lTxr
pShaderLightOff = pMember.newShader("shLightOff",#standard)
pShaderLightOff.emissive = rgb( 0, 0, 0 )
pShaderLightOff.texture = lTxr
pShaderLightOff.blend = 25  -- Opazität 25%
```

Listing 46 Erzeugen und Zuweisen der Alphakanalmasken

zugehörige Erläuterungen hier

```
on initOrbitCam()
deltaX=pOrbitCamera.transform.position.xpMember.model("Sky").transform.position.x
deltaY=pOrbitCamera.transform.position.ypMember.model("Sky").transform.position.y
pCamOrbitAngle=atan(1.0*deltaY/deltaX)
if pCamOrbitAngle >= 0 then
  if pOrbitCamera.transform.position.y < 0 then -- 3. Quadrant
      pCamOrbitAngle = pCamOrbitAngle + PI
    end if
  else -- 2. oder 4. Quadrant
    if pOrbitCamera.transform.position.y < 0 then -- 4. Quadrant
       pCamOrbitAngle = pCamOrbitAngle + 2.0*PI
    else
      pCamOrbitAngle = pCamOrbitAngle + PI -- 2. Quadrant
    end if
  end if
  pCamOrbitOffset = point(pMember.model("Sky").transform.position.x,\
          pMember.model("Sky").transform.position.y )
  pOrbitRadius = power((power( deltaX, 2 ) + power( deltaY, 2 )), 0.5 )
end initOrbitCam
```

Listing 47 Initialisierung für kreisförmige Kamerafahrt

zugehörige Erläuterungen hier

```
on animateCamera
…
else
  pCamOrbitAngle = pCamOrbitAngle - 2* PI / 120.0
  lx = cos(pCamOrbitAngle) * pOrbitRadius + pCamOrbitOffset.locH
  ly = sin(pCamOrbitAngle) * pOrbitRadius + pCamOrbitOffset.locV
  pOrbitCamera.transform.position = vector(lx,ly,pOrbitCamera.transform.position.z)
  pOrbitCamera.pointAt(pCamOrbitLookAt,vector(0,0,1))
end if
end animateCamera
```

Listing 48 Transformation der Orbitkamera auf Kreisbahn

zugehörige Erläuterungen hier

```
if pStartCamPathInterpol then -- increment the internal counter for the animation
  pCamAnimInfo.count = pCamAnimInfo.count + 1
  -- determine the interpolation percentage
  pctg = 100.0 * (pCamAnimInfo.count / 50.0)
  -- if pctg is greater than 100 end animation sequence
  if (pctg > 100) then
    pctg = 100
    pCamAnimFlag = FALSE
    pStartCamPathInterpol = FALSE
    pSprite.camera = pCamera
  end if
  -- determine the new interpolated camera transform
  t = pCamAnimInfo.initT.interpolate (pCamAnimInfo.finalT,pctg)
  -- apply that transform to the camera
  pOrbitCamera.transform = t
  …
```

Listing 49 Positionpathinterpolation

zugehörige Erläuterungen hier

```
if pMouseDown OR pRightMouseDown then
  -- Berechne Höhe Camera über Fussboden / Treppe und korrigiere ev. Kamerahöhe
  l_list=pMember.modelsUnderRay(pCamera.transform.position,vector(0,0,-1),3,#detailed)k = 1
  repeat with i = 1 to pIgnoreCollisionList.count
    repeat with j = 1 to l_list.count
      if l_list[j][1].name = pIgnoreCollisionList[i]  then
        k = k + 1
       end if
    end repeat
  end repeat
  sendsprite( me.spriteNum, #checkTrigger, l_list)
  if l_list.count() > 0 then
    if l_list[k].distance <> pCamOldDistance then
      lCamDeltaZ = pCamOldDistance - l_list[1].distance
      pCamera.transform.position.z = pCamera.transform.position.z + lCamDeltaZ
        …
```

Listing 50 Korrektur der Höhe der Kamera und Aufruf Triggertest

zugehörige Erläuterungen hier

```
case (TRUE) of
  (pMouseDown): -- left mouse down, cast ray forward
     tList=pMember.modelsUnderRay(pCamera.worldPosition,pCamera.transform.zAxis,#detailed)
     -- if there are models in front of the camera check for collisions
        if (tList.count) then
          me.checkForCollision(tList[1])
        end if
  (pRightMouseDown): -- right (control+) mouse down, cast ray backward
     tList=pMember.modelsUnderRay(pCamera.worldPosition,pCamera.transform.zAxis,#detailed)
     -- if there are models in back of the camera check for collisions
        if (tList.count) then
          me.checkForCollision(tList[1])
        end if
end case
…
```

Listing 51 Test, ob Objekte vor bzw. hinter der Kamera sind

zugehörige Erläuterungen hier

```
on checkForCollision (me, thisData)
  repeat with i=1 to pIgnoreCollisionList.count
    if thisData.model.name contains( pIgnoreCollisionList[i] ) then return
  end repeat
  -- grab the distance value
  dist = thisData.distance
  -- if distance is smaller than bounding radius resolve collision
  if (dist < pCameraSphere.resource.radius) then
    -- get distance of penetration
    diff = pCameraSphere.resource.radius - dist
    -- calculate vector *perpendicular* to the wall's surface to move the camera
    tVector = thisData.isectNormal * diff
    -- move the camera in order to resolve the collision
    pCamera.translate(tVector,#world)
   end if
end checkForCollision
```

Listing 52 Test auf Kollision, ggf. Auflösung

zugehörige Erläuterungen hier

```
on new me, aPropList
  pURL = aPropList.pURL
  pSpriteNum = aPropList.pSpriteNum
  pCallBackObj = aPropList.getaProp(#pCallBackObj)
  pCallBackFkt = aPropList.pCallBack
  pMember = aPropList.pMember
  pNetID= preLoadNetThing(pURL)
  return me
end
on stepFrame me
  if not voidp( pNetID ) then
    if netDone( pNetID ) then
      lState = getStreamStatus( pNetID ).state
      pNetID = void
      (the actorList).deleteOne(me)
      if not voidP( pCallBackFkt ) AND pSpriteNum<>void then
        sendSprite( pSpriteNum, pCallBackFkt )
      else
        if pSpriteNum = void then AND pCallBackObj <> void then
            call(pCallBackFkt, pCallBackObj, lState) --falls Funktion in einem
        end if                                        -- Objekt gerufen werden muss
      end if
    end if
  end if
end stepFrame
```

Listing 53 Auszug aus dem Parentskript des Loaderobjektes

zugehörige Erläuterungen hier

```
parserObject = new(xtra "XMLParser")
parserScriptObject = new(script "ps_ParserScript", parserObject, p3DObj )
if the runMode contains "Plugin" then
  errorCode=parserObject.parseURL(the moviepath&pURL_FilmInfo,#parseDone,
parserScriptObject)
  else
    errorCode = parserObject.parseString( member("XML_Info").text )
  end if
  errorString = parserObject.getError()
  if not voidP(errorString) then
    alert "Sorry, there was an error " & errorString
    -- Exit from the handler
    exit
  else
    parserScriptObject.parseDone()
  end if
```

Listing 54 Initialisierung XML-Parsing, Ausschnitt aus Behavior b_Cinema

zugehörige Erläuterungen hier

```
on parseDone me
  if voidP(myParserObject.getError()) then
    lFilmList = []
    Child1_SubList = myParserObject.child[1].makeSubList()
    repeat with i= 1 to Child1_SubList[1].count -1
      lFilmInfoList = [:]
      Child1_1_SubList = myParserObject.child[1].child[i].makeSubList()
      repeat with j = 2 to Child1_1_SubList[1].count
        lFilmInfoList.addProp( symbol( Child1_1_SubList[1].
                              getOne(Child1_1_SubList[1][j])),-- Name des Attributes\
                              Child1_1_SubList[1][j][2])      -- Wert des Attributes
      end repeat
      lFilmList.add(lFilmInfoList)
    end repeat
    sendAllSprites(#xmlParsing_successful, lFilmList)
  else
    put "Parse error:"
    put "   " & myParserObject.getError()
  end if
  (the actorList).deleteOne(me)
end
```

Listing 55 Umwandlung der XML-Propertyliste

zugehörige Erläuterungen hier

```
on createTexture aPropList
  gIsready = FALSE
  p3DMember = member( aPropList.p3DMemberName )
  targetMember = member( aPropList.pBitmapName )
  if targetMember.memberNum = -1 then
    targetMember = new(#bitmap)
    targetMember.name = aPropList.pBitmapName
  end if
  overlayImage = member(aPropList.pSourceName).image
  if voidP(overlayImage) then return
  if gUseAlpha then
    baseImage = image(aPropList.pWidth,aPropList.pHeight,24,8)
    baseImage.fill(0,0,aPropList.pWidth,aPropList.pHeight,rgb(255,255,255))
    -- holen des Image des Textdarsteller, das in das Bild hineinkopiert werden soll
    -- overlayImage = member(aTextMemberName).image
    --  hole  Alphakanal  des  Textdarstellers,  damit  dieser  mit  Antialiasing  in  das  Bild\
    --kopiert werden kann
    maskImg = overlayImage.extractAlpha()
    -- Positionierung der Schrift mittig auf Ziel- Bitmap
    targetRect = rect(0,0,0,0)
    targetRect.left = ( baseImage.width / 2 ) - ( overlayImage.width / 2 )
    targetRect.top = ( baseImage.height / 2 ) - ( overlayImage.height / 2 )
    targetRect.right = targetRect.left + overlayImage.width
    targetRect.bottom = targetRect.top + overlayImage.height
    --anlegen eines neuen ImagesObjektes, in das Bild und Text hineinkopiert werden
    targetImage = image( baseImage.width, baseImage.height, 32 )
```

```
    targetImage.copyPixels(baseImage, baseImage.rect, baseImage.rect )
    -- Beim Kopieren des Textes die Alphamaske mit angeben
    targetImage.copyPixels(overlayImage, targetRect, overlayImage.rect,[maskImage: maskImg])
    targetImage.setAlpha(maskImg)
    targetImage.useAlpha = TRUE
    targetMember.image = PowerOf2Dimensions( targetImage )
    theShader = member(1, 1).shader("sh_Box")
    theShader.textureTransformList[2].position = vector( 0.000, 0.000, 0.0000 )
    theShader.textureTransformList[2].scale = vector( 1.0000, 1.0000, 1.0000 )
  else
    targetImage = overlayImage
    targetMember.image = targetImage
    theShader = member(1, 1).shader("sh_Box")
    theShader.textureTransformList[2].position = vector( 0.2400, 0.2500, 0.0000 )
    theShader.textureTransformList[2].scale = vector( 0.5000, 0.5000, 1.0000 )
    theShader.textureRepeatList[2] = 0
  end if
  if voidP( p3DMember ) then return
  if not voidP(aPropList) then
    if not voidP(aPropList.pTexture) then
      aPropList.pTexture.member = targetMember
      gIsready = TRUE
      return
    end if
  end if
  lTexture = p3DMember.Texture( "tx_AlphaText" )
  if voidp( lTexture ) then
    lTexture = p3DMember.newTexture( "tx_AlphaText" )
  end if
  lTexture.member = targetMember
  gIsready = TRUE
  return lTexture
end createTexture

on PowerOf2Dimensions ( pImage )
  W = getNearestPowerGE( 2, pImage.width )
  H = getNearestPowerGE( 2, pImage.height )
  if W = pImage.width and H = pImage.height then
    pImage.setAlpha(getNegativ(getNegativ(pImage)))
    return pImage
  end if

  pUseAlpha = pImage.UseAlpha
  lColor = pImage.getPixel(0,0)
  nImage = image( W, H, pImage.depth )
  nImage.fill( 0, 0, W, H, lColor )
  lOffset = point( (W - pImage.width) / 2, (H - pImage.height) / 2 )
  lDestRect = rect( lOffset, point(pImage.rect.right, pImage.rect.bottom) + lOffset)
  nImage.copyPixels( pImage, lDestRect, pImage.rect )
  if pUseAlpha then
    nImage.setAlpha(getNegativ(getNegativ(nImage)))
```

```
    nImage.useAlpha = TRUE
  end if
  pImage = 0
  return nImage
end PowerOf2Dimensions

on getNearestPowerGE base, value
  -- Returns the power of the given base that is greater
  -- than or equal to the given value
  return integer( power( base, integer( log( value ) / log( base ) - 0.50000001 ) + 1 ) )
end getNearestPowerGE

on getNegativ( pImage )
  lRect = pImage.rect
  imgBlk = image( pImage.width,pImage.height,8,#grayscale )
  imgBlk.fill(0,0,pImage.width,pImage.height,rgb(0,0,0))
  imgWht = image( pImage.width,pImage.height,8,#grayscale )
  imgWht.fill(0,0,pImage.width,pImage.height,rgb(255,255,255))
  imgWht.copyPixels(pImage, rect(0,0,pImage.width,pImage.height),
                            rect(0,0,pImage.width,pImage.height),[#ink:4])
  return imgWht
end getNegativ
```

Listing 56 Imaging-Lingo für optimale Darstellungsqualität von Textinformation

zugehörige Erläuterungen hier

```
on mouseDown me
  lList = sprite(me.SpriteNum).camera.modelsUnderLoc(the mouseLoc, 1, #detailed)
  fID= lList[1].faceID
  -- liefert die PolygonIDput fID
  -- liefert eine Liste bestehend aus 3 Indizes der VertexListe
  put pModel.meshdeform.mesh[1].face[fID]
  -- liefert Liste der 3 Vektoren des Polygons
  put lList[1].vertices
end
```

Listing 57 Bestimmung des gepickten Polygons und dessen Scheitelpunkte

zugehörige Erläuterungen hier

```
lFaceList = [2,5,12,17,18,20] -- Liste der Flächenindizes der Fl. die nach vorn zeigen
  mrMesh = pMember.newMesh("mrMesh",6,8,0,0,0) -- 6 Polygone, 8 Scheitelpunkte
  lVertexList=[vector(128.2970,-13.2721,0.0), vector(349.4987,-13.2721,0.0),\
              vector(349.4987,-13.2721,286.0876), vector(-349.4987,-13.2721,286.0876),\
              vector(-349.4987,-13.2721,0), vector(-202.0309,-13.2721,0),\
              vector(-202.0309,-13.2721,176.9614vector(128.2970,-13.2721,176.9614)]
  mrMesh.vertexList = lVertexList
  mrMesh.face[1].vertices = [1,2,3]
  mrMesh.face[2].vertices = [3,8,1]
  mrMesh.face[3].vertices = [3,4,8]
  mrMesh.face[4].vertices = [4,7,8]
  mrMesh.face[5].vertices = [4,6,7]
  mrMesh.face[6].vertices = [4,5,6]
  mrMesh.generateNormals(#smooth)
  mrMesh.build()
--  ltmpModel = pMember.newModel("Test",mrMesh)
--  ltmpModel.shaderList = pMember.shader("Material #1")
```

Listing 58 Erstellung Referenzobjekt für Untersuchung der Darstellungsqualität

zugehörige Erläuterungen hier

```
pAASupported = FALSE
lVersion = the environment.productVersion
--Stringseparation
lTmpDelim = the itemDelimiter
the itemDelimiter = "."
if integer(lVersion.item[1]) > 9 then pAASupported = TRUE
  -- Dummyfkt. Muss für Dir MX angepasst werden
else
  if lVersion.item.count > 2 then pAASupported = TRUE -- 8.5.1 hat drei Items
end if
```

Listing 59 Abfrage auf Antialiasingsupport

zugehörige Erläuterungen hier

```
on setAntiAliasing aStatus
  if pAASupported AND pAllowAA then
    if pSprite.antiAliasingSupported =TRUE then
      if pSprite.antiAliasingEnabled <> aStatus then
         pSprite.antiAliasingEnabled = aStatus
…
```

Listing 60 Aktivieren bzw. Deaktivieren des Antialiasings

# Abbildungsverzeichnis

Abb. 0.7.1 3D-Ansichten des Entertainmentkomplexes ..... 5
Abb. 0.7.2 2D-Navigationsplan, Diashow ..... 6
Abb. 1.1 Relative Verteilung der Internet-Nutzertypen [www18] ..... 10
Abb. 1.2 Bedienungsanleitung einer Kamera [www24] ..... 13
Abb. 1.3 Interaktive 3D-Darstellung eines Schädels [www25] ..... 14
Abb. 1.4 Darstellung der ISS-Raumstation [www26] ..... 14
Abb. 1.5 Beispiel eines 3D-Chatsystems [www31] ..... 15
Abb. 1.6 3D-Rendering als Hilfsmittel für Designentscheidungen [www20] ..... 16
Abb. 1.7 Beispiel eines 3D-Entwurfes für ein Bebauungsprojekt [www20] ..... 16
Abb. 1.8 Beispiel einer 3D-Reliefdarstellung [www20] ..... 17
Abb. 1.9 Erweiterte Profile der X3D-Architektur [GÖT02] ..... 19
Abb. 1.10 Allgemeiner Aufbau einer X3D-Szene nach [GÖT02] ..... 20
Abb. 1.11 Quelle [www14] ..... 21
Abb. 1.12 Quelle [www40] ..... 22
Abb. 1.13 Quelle [www41] ..... 22
Abb. 1.14 Aus selbst-entwickelter Testszene ..... 23
Abb. 1.15 Quelle [www29] ..... 24
Abb. 1.16 Produktpräsentation mit 3Danywhere [www43] ..... 25
Abb. 1.17 Quelle [www44] ..... 25
Abb. 1.18 Aus Java 3D-Beleg ..... 26
Abb. 1.19 Quelle [www45] ..... 27
Abb. 1.20 Quelle [www46] ..... 27
Abb. 1.21 Allgemeiner Aufbau eines Java 3D-Szenegraphen ..... 29
Abb. 2.1 Der Director-Arbeitsbereich ..... 31
Abb. 2.2 Director-Arbeitsbereich – die Bühne ..... 32
Abb. 2.3 Director-Arbeitsbereich – die Besetzung ..... 32
Abb. 2.4 Director-Arbeitsbereich – das Drehbuch ..... 33
Abb. 2.5 Director-Arbeitsbereich – der Eigenschaftsinspektor ..... 33
Abb. 2.6 Director-Arbeitsbereich – das Skriptfenster ..... 34
Abb. 2.7 Director-Arbeitsbereich – das Nachrichtenfenster ..... 34
Abb. 2.8 Nachrichtenhierarchie in Director ..... 36
Abb. 2.9 Die Einheitsmatrix ..... 40
Abb. 2.10 Die Translationsmatrix ..... 40
Abb. 2.11 Die Skalierungsmatrix ..... 40
Abb. 2.12 Matrizen für Rotation um x-,y- und z-Achse ..... 40
Abb. 2.13 Das Preloaderelement ..... 41
Abb. 2.14 Internes Organisationsprinzip einer 3D-Darstellung ..... 42
Abb. 2.15 Aufbau eines W3D- Files [BIE02] ..... 44

Abb. 2.16 Koordinatensystem und Ausrichtung ..... 45
Abb. 2.17 Definition positive Drehrichtung ..... 46
Abb. 2.18 Würfel mit Standardshader ..... 47
Abb. 2.19 Übersicht Modelleigenschaften im 3DPI ..... 49
Abb. 2.20 Die einzelnen Kategorien des 3DPI ..... 50
Abb. 3.1 Ansicht Haupteingang Wandelhof Schwarzheide ..... 51
Abb. 3.2 Hilfsmittel Polygonzähler in 3D Studio MAX ..... 52
Abb. 3.3 3D Studio MAX-Renderausgabe ..... 53
Abb. 3.4 ohne Segmentierung ..... 53
Abb. 3.5 10x10x1 Segmente ..... 53
Abb. 3.6 Laterne mit simulierten Lichtkegel in 3D Studio MAX ..... 54
Abb. 3.7 Darstellungsfehler an Wandecken ..... 54
Abb. 3.8 Problem der sichtbaren Flächen und Kanten ..... 55
Abb. 3.9 n-seitige Pyramiden als Abschluss ..... 57
Abb. 3.10 Drahtgittermodell des 5-seitigen Infoobjektes ..... 57
Abb. 3.11 Beispiel für Flatshading ..... 59
Abb. 3.12 Das fertig texturierte Infoobjekt ..... 59
Abb. 3.13 Prinzipskizze Rotation des Kinoinformationsobjektes, Draufsicht ..... 60
Abb. 4.1 Prinzip Kameraanimation ..... 68
Abb. 4.2 Funktionsschema maussensitiver Bereiche ..... 69
Abb. 4.3 Schema Kollisionserkennung und -auflösung ..... 71
Abb. 5.1 Eine Szene mit importiertem 3D-Schriftzug ..... 77
Abb. 5.2 Textdarstellung mittels Overlay-Technik ..... 78
Abb. 5.3 Beispiel Multitexturing ..... 79
Abb. 5.4 Schlechte Visualisierungsqualität einer Textinformation ..... 79
Abb. 5.5 Verbesserte Darstellungsqualität durch Skalierung der Textur ..... 80
Abb. 5.6 Bestmögliche erreichbare Darstellungsqualität, realisiert mit Imaging-Lingo ..... 80
Abb. 5.7 Anzeige Downloadstatus auf Doppeltür ..... 81
Abb. 5.8 Anzeige der erreichten FPS-Rate innerhalb der 3D-Szene ..... 82
Abb. 5.9 Ausschnitt der Bitmap für die Texturen ..... 82
Abb. 5.10 komplette Bitmap als Textur ..... 83
Abb. 5.11 Teil der Bitmap als Textur ..... 83
Abb. 6.1 Ansicht nach W3D-Export ..... 85
Abb. 6.2 Flatshading in Director3D ..... 85
Abb. 6.3 Fehlerhafte Darstellung des Referenzobjektes ..... 86
Abb. 6.4 Gouraud-Shading am Beispiel des geboolten Objektes ..... 86
Abb. 6.5 Prinzip der Kanten- und Flächeninterpolation ..... 87
Abb. 6.6 Geboolte L-Extrusion ohne SDS-Modifier ..... 87
Abb. 6.7 Fehlerhafte Geometrie nach Anwendung SDS-Modifier ..... 87
Abb. 6.8 Testmodell ohne SDS-Modifier ..... 88

Abb. 6.9 Testmodell nach Anwendung des SDS-Modifiers ..... 88
Abb. 6.10 Testreihe zur Simulation eines typischen Spotlichtes ..... 88
Abb. 6.11 Zweite Testreihe zur Spotlichtsimulation mit veränderter Basisgeometrie ..... 89
Abb. 6.12 Maximale Bildwiederholrate bei veränderter Polygonanzahl ..... 90
Abb. 6.13 Vergleich Szenenausschnitt ohne / mit Antialiasing ..... 91
Abb. 6.14 Bildwiederholraten mit/ ohne Kantenglättung ..... 92
Abb. 6.15 Performanceuntersuchung nach Wiedergabeumgebung ..... 93
Abb. 6.16 Verlauf der CPU-Belastung bei Shockwavedarstellung ..... 93
Abb. 7.1 Das neue Userinterface von Director MX ..... 95

# Tabellenverzeichnis

Tabelle 1 Ausgewählte, maximal erreichbare Downloadraten ..... 10
Tabelle 2 Zusammenfassung Viewpoint Experience Technology ..... 21
Tabelle 3 Zusammenfassung Puls3D-Technologie ..... 22
Tabelle 4 Zusammenfassung Cult3D-Technologie ..... 22
Tabelle 5 Übersicht Macromedia Director3D ..... 23
Tabelle 6 Übersicht zu Adobe Atmosphere ..... 24
Tabelle 7 Zusammenfassung der Web3D-Technologie 3Danywhere ..... 25
Tabelle 8 Zusammenfassung der Web3D-Technologie Shout3D ..... 25
Tabelle 9 Übersicht zu Java 3D ..... 26
Tabelle 10 Übersicht zum Cortona VRML-Viewer ..... 27
Tabelle 11 Zusammenfassung der Web3D-Technologie Blaxxun Contact ..... 27
Tabelle 12 Mögliche Nachrichten für Primary-Event-Handler ..... 36
Tabelle 13 Komponenten eines 3D-Darstellers und ihre Funktionen ..... 43
Tabelle 14 Generierungsparameter des Infoobjektes ..... 56
Tabelle 15 Auflistung der zu realisierenden Funktionalitäten bezüglich der Kamerasteuerung ..... 62
Tabelle 16 Übersicht der Standardmodifikatoren ..... 63
Tabelle 17 Eigenschaften des Collision-Modifiers und deren Bedeutung ..... 64
Tabelle 18 Notwendige Eigenschaftsvariablen für die Kamerasteuerung ..... 66
Tabelle 19 Anzahl der Scheitelpunkte bei unterschiedlichen Schriftfonts ..... 76
Tabelle 20 Benötigte Parameter für die Texturerstellung ..... 80
Tabelle 21 Relevante Hardwareausstattung der 3 Testsysteme ..... 84

# Abkürzungsverzeichnis

| | |
|---|---|
| API | Application Programming Interface |
| ASCII | American Standard Code for Information Interchange |
| CERN | Conseil Européen de Recherches Nucléaires |
| CPU | Central Processing Unit |
| DTD | Document-Type-Definition |
| FPS | Frame per Second, Bilder pro Sekunde |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ILS | Initial Loader Segment |
| ISO | International Organization for Standardization |
| ISS | International Space Station |
| kB | KiloByte (= 1024 Byte) |
| LKS | Lokales Koordinatensystem |
| MB | Megabyte |
| MPEG | Moving Picture Expert Group |
| NCSA | National Centre for Supercomputing Applications |
| OBJ | Wavefront Object |
| OpenGL | Open Graphics Library |
| QT | QuickTime |
| QTVR | QuickTime Virtual Reality |
| SDS | Subdivision Surfaces |
| SGI | Silicon Graphics, Inc. |
| SGML | Standard Generalized Markup Language |
| SWF | Shockwave Flash Format |
| TGS | Template Graphics Software |
| URL | Uniform Resource Locator |
| UTF-8 | Unicode Text Format |
| VAG | VRML Architecture Group |
| VET | Viewpoint Experience Technology |
| vgl. | vergleiche |
| VR | Virtual Reality |
| VRML | Virtual Reality Modelling Language |
| VRML NG | VRML Next Generation |
| W3C | World Wide Web Consortium |
| Web3DC | Web3D Consortium |
| WKS | Weltkoordinatensystem |
| X3D | Extensible 3D |

# Literaturverzeichnis


[BIE02] Biedorf, Thomas et al.: 3D-Programmierung mit Director. Bonn: Galileo Press GmbH, 2002.

[GIL00] Gillmaier, Gerd, Gola, Joachim: Director8 Workshop, München: Addison-Wesley Verlag, 2000

[GÖT02] Götz, Frank: 3D Grafik im Web. Exemplarische Analyse aktueller Technologien. Diplomarbeit. Paderborn: Universität-Gesamthochschule Paderborn, 2002

[GRO00] Gross, Phil; Gross, Mikel: Macromedia Director8.5 ShockwaveStudio für 3D. Das offizielle Trainingshandbuch. München: Markt+Technik Verlag, 2002

[IMM99] Immler, Christian: Das große Buch 3D Studio Max. Düsseldorf: DATA Becker GmbH & Co. KG, 1999

[JUR02] Jurk, Andrea: Ergonomische Gestaltung und prototypische Umsetzung einer kindgerechten Lernsoftware für den Geografieunterricht. Diplomarbeit. Dresden: Hochschule für Technik und Wirtschaft Dresden (FH), 2002

[KLO00] Kloss, Martin: Lingo objektorientiert. Bonn: Galileo Press GmbH, 2000

[KOE00] Koenigsmarck, von, Arndt: 3D Design. Zürich: Midas Verlag AG, 2000

[LAM02] Lamprecht, Frank: 3D für das Web. München: Addison-Wesley Verlag. 2002

[NAU01] Naumann, Sven: Charakteranimation in Shockwave3D – Evaluation und Entwurf eines Produktionsprozesses. Diplomarbeit. Dresden: Hochschule für Technik und Wirtschaft Dresden: 2001

[RUK01] Rukzio, Enrico: Formate, Technologien und Architekturkonzepte für 3D-Web-Applikationen. Belegarbeit. Dresden: Technische Universität Dresden, 2001

[STR00] Strippgen, Simone: Java3D Programmierung. Vorlesungsskript zur Lehrveranstaltung. Dresden: Hochschule für Technik und Wirtschaft Dresden, 2000

[WEI01] Weingärtner, Monika: Publizieren im World Wide Web. Hannover: Regionales Rechenzentrum für Niedersachsen / Universität Hannover, 2001

# Verzeichnis der Internet- Links


[www01] Kernforschungszentrum Cern
http://www.cern.ch
Conseil Européen de Recherches Nucléaires, gefunden September 2002

[www02] National Center for Supercomputing Applications
http://www.ncsa.uiuc.edu
University of Illinois at Urbana-Champaign, gefunden September 2002

[www03] Netscape, Browsername und Entwicklungsfirma des Browsers
http://www.netscape.com/
Netscape, gefunden September 2002

[www04] Microsoft, Softwareentwicklungsfirma
http://www.microsoft.com/
Microsoft Corporation, gefunden September 2002

[www05] World Wide Web Consortium: Web-Standardisierungsgremium
http://www.w3c.org
W3 Consortium, gefunden August 2002

[www06] Pesce, Mark D.; Kennard, Peter; Parisi, Anthony S.:Cyberspace
http://www.hyperreal.org/~mpesce/
http://www.web3d.org/www-vrml/concepts/pesce-www.html
First International Conference on the World Wide Web, in Geneva, Switzerland,1994,
gefunden September 2002

[www07] SGI
http://www.sgi.com
Silicon Graphics Inc., gefunden September 2002

[www08] TGS, Open Inventor-Format
http://www.tgs.com/
Template Graphics Software, gefunden Oktober 2002

[www09] Homepage Sony Coporation of America
http://www.sony.com
Sony Corporation, gefunden Oktober 2002

[www10] Java 3D API
http://java.sun.com/products/java-media/3D/
Sun Microsystems Inc., gefunden Oktober 2002.

[www11] Web3D Consortium
http://www.web3d.org
Web3D Consortium, gefunden Oktober 2002

[www12] Adobe
http://www.adobe.com
Adobe Systems Inc., gefunden September 2002

[www13] Macromedia, Hersteller von Director, Flash u. a. Programmen
http://www.macromedia.com
Macromedia, gefunden Juli 2002

[www14] Viewpoint Rich Media Solutions, Web3D-Technologieanbieter
http://www.viewpoint.com
Viewpoint, gefunden September 2002

[www15] DirectX, Multimedia-API
http://www.microsoft.com/directx/
Microsoft Corporation, gefunden September 2002

[www16] internationale Vergleichsstudie zur Informationsgesellschaft
http://www.bitkom.org
Bundesverband Informationswirtschaft, Telekommunikation und Neue Medien e.V., gefunden Januar 2003

[www17] Internetnutzung in Europa – ein Puzzle mit 1000 Teilen? (14.03.2002)
http://www.ecin.de/marktbarometer/
Electronic Commerce Info Net, gefunden September 2002

[www18] Diverse Studien, Untersuchungen und Statistiken verschiedener kommerzieller Bereiche
http://www.gfk.de/
GfK - Gesellschaft für Konsum-, Markt- und Absatzforschung, gefunden September 2002

[www19] Pressemitteilung vom 30.01.2001 zum ersten deutschen 3D-eCommerce-Tag
http://www.hrz.uni-paderborn.de/universitaet-aktuell/presse/presse-2001/presse-30-01-01.htm
3D-eCommerce-Tag, gefunden Januar 2003

[www20] Zukunftsfabrik Kommunikation Hannover
http://www.3d-economy.de/
Portal zur wirtschaftlichen Bedeutung von 3DWeb, August 2002

[www21] Webseite zu Apples QuickTimeVR
http://www.apple.com/quicktime/qtvr/
gefunden Februar 2003

[www22] Shareware-Tool zur Erstellung von QuickTimeVR-Movies,
http://www.panaview.com
gefunden Oktober 2002

[www23] Weiteres Tool zur Erstellung von QuickTimeVR-Szenen
http://www.vrtoolbox.com/
gefunden Februar 2003

[www24] Beispiel einer Online-Bedienungsanleitung
http://www.parallelgraphics.com/showroom/solutions/distance-training/olympus/
ParallelGraphics, gefunden September 2002

[www25] Erstellung anatomischer Modelle für Lehrzwecke, Web3D-Präsentation eines Schädels
http://www.3bscientific.com/content/index.cfm?Country=Germany
3B Scientific Unternehmensgruppe, gefunden Januar 2002

[www26] Interaktives Web3D-Modell der Internationalen Raumstation ISS
http://www.discovery.com/highspeed/discovery/spacestation/frameset.htm
Discovery Communications Inc., gefunden Oktober 2002

[www27] Cybercore Entrance, W3D-Technologie, spezialisiert auf Online-Gaming
www.cycosys.com
CyCo Systems, gef. September 2002

[www28] Entwickler von Web3D-Spielen
http://www.3dgroove.com
The Groove Alliance Inc., gefunden Dezember 2002

[www29] Adobe Atmosphere, Betaversion eines Web3D-Plug-ins
http://www.adobe.com/products/atmosphere
Adobe, gefunden September 2002

[www30] Pulse3D Player, Web3D-Plug-in
http://www.pulse3d.com
Pulse, gefunden September 2002

[www31] 3D-Online-Chat
http://www.cybertown.com
IVN/Cybertown Inc., gefunden September 2002

[www32] Jay Leno – Tonight Show
http://nbctv.nbci.com.tonightshow/virtualjay/special.html
NBC, gefunden Februar 2003

[www33] VizStream, kollaborationsorientiertes Web3D-Plug-in
http://realitywave.com
RealityWave, gefunden September 2002

[www34] Pressemitteilung vom 26.01.2001 zum ersten deutschen 3D-eCommerce-Tag
http://hrz.uni-paderborn.de/universitaet-aktuell/presse/presse-2001/presse-26-01-01.htm
3D-eCommerce-Tag, gefunden Januar 2003

[www35] 3D-Technologiebeispiele der Firma Salient GmbH
http://www.salient.de/web3d/tec_any.html
Salient GmbH, gefunden Februar 2003

[www36] Java-Programmiersprache
http://www.java.sun.com
Sun Microsystems Inc., gefunden August 2002

[www37] VRML Consortium:The Virtual Reality Modeling Language - International Standard
ISO/IEC 14772-1:1997
http://www.web3d.org/Specifications/VRML97/
VRML Consortium Inc., gefunden Februar 2003

[www38] Xj3D Open Source Browser
http://www.web3d.org/TaskGroups/source/xj3d.html
Web3D Consortium, gefunden Februar 2002.

[www39] Koordinationsgremium für die Entwicklung X3D-fähiger Browser/ Player
http://www.web3d.org/x3d/browserwg_charter.html
Web3D Consortium: Browser Working Group, gefunden Februar 2003

[www40] 3D- Personal Information Manager, erstellt mit Pulse3D
http://www.wapme.net/
Wapme Systems AG, gefunden Dezember 2002

[www41] 3D-Präsentation des Fiat-Stilo, erstellt mit Cult3D
http://www.fiat.co.uk/stilo/eng/index.htm
FIAT AUTO U.K. Ltd., gefunden Februar 2003

[www42] Aufschlüsselung der kontinentalen Installation des Shockwave-Plug-ins
http://www.macromedia.com/software/player_census/shockwaveplayer/version_penetration.html
Macromedia, gefunden August 2002

[www43] 3di, Entwickler von Web3D-Technolgien
http://www.3danywhere.com/
3di Ltd., gefunden November 2002

[www44] 3D-Modepräsentation
http://www.eyematic.com/Fashion/index.html
Excite Inc. and Macy's, gefunden August 2002

[www45] ParallelGraphics, Web3D-Technologie-Anbieter, Beispiel einer Produktpräsentation
http://www.parallelgraphics.com/full/showroom/solutions/product-presentations/olympus
ParallelGraphics, gefunden November 2002

www46] Online-3D-Welt / Community, erstellt von Blaxxun
http://www.icity.co.il/
Suny Electronics Group, gefunden Februar 2003

[www47] hilfreiche deutsche Mailingliste zu Director, incl. Mailarchiv
http://www.startmovie.net
Direct@r_List, gefunden Juli 2002

[www48] Intel® Architecture Labs
http://developer.intel.com/ial/
Intel Corporation, gefunden September 2002

[www49] Intel® 3D Software Technologies – Intel Architekture Labs
http://www.intel.com/ial/3dsoftware/
Intel Corporation, gefunden September 2002

[www50] Homepage der Programmiererin des 3D-Propertyinspektors,
Downloadmöglichkeit der aktuellsten Version
http://ullala.at/3DPI/
Ulla Gusenbauer, gefunden Juli 2002

[www51] 3D Studio Max
http://www.discreet.com/
Autodesk, Inc., gefunden Juli 2002

[www52] Kamerasteuerung per Lingo (von Tom Higgins), im Projekt angepasst und erweitert
http://www.directordev.com/learning/demo_movies/3D/general/tunnels/default.htm
Macromedia Developer Team, gefunden August 2002

[www53] Havok, physikalische 3D-Simulation, Xtra für Director
http://www.havok.com/
Havok, gefunden Dezember 2002

[www54] Liste der nicht von Director unterstützten Grafikkarten
http://macromedia.com/support/Director/ts/documents/3d_rendering/udl.htm
Macromedia, gefunden August 2002

[www55] Auszug aus Newsletter der Director-List, vom “ListenPapa” Thomas Biedorf
http://www.startmovie.net/newsanzeigen.php?track=20021125101937
Director-List, gefunden 25.12.2002

[www56] Beispiel einer einfachen VRML-Szene
http:// www.csv.ica.uni-stuttgart.de/vrml/hello.wrl
Universität Stuttgart, gefunden Februar 2003

# Glossar

| | |
|---|---|
| **3**D Studio Max | 3D-Modellierungswerkzeug der Firma Discreet |
| **A**bspielkopf | Element der Anwendung Macromedia Director, markiert die Abspielposition |
| ActiveX | Entwicklung der Firma Microsoft, welche die Freigabe von Informationen zwischen Anwendungen erleichtert und die Einbettung beliebiger Objekte (Video, Sound) in fremden Dokumenten erlaubt |
| Animation | Veränderung einer Darstellung über die Zeit. Eindruck einer Bewegung, die entsteht, wenn durch sequentielles Anzeigen/ Abspielen von Bildern eine Bildwiederholrate von etwa 15-20 Bildern pro Sekunde erreicht wird |
| API | Application Programming Interface, Programmier- und Anwendungsschnittstelle |
| Avatar | Figur, die in 3D-Chatwelten und Spielen als Repräsentant eines Gesprächsteilnehmers bzw. Spielers verwendet wird |
| **B**esetzung | Element von Macromedia Director, verwaltet alle erstellten oder importierten Medien |
| Behavior | Skript, um einem Objekt ein bestimmtes Verhaltensmuster zuzuordnen, es bestimmt, welches Objekt wann und wie auf ein bestimmtes Ereignis reagiert |
| Bibliothek | Element von Macromedia Director, stellt meistens vordefinierte Skripte zur Verfügung, jede Besetzung im Libs-Ordner wird in die Bibliothek aufgenommen |
| Bitmap | Rastergrafik, bei der das Bild in unabhängig voneinander kontrollierbare Einzelpunkte aufgelöst wird |
| BoundingSphere | Nicht sichtbare Kugel, welche die Geometrie eines Körpers komplett umschließt, wird für Ereignisauslösung, z. B. Kollisionserkennung verwendet |
| Browser | Programm, welches Daten aus dem weltweiten Netz abruft und dann am Computer verarbeitet und anzeigt |
| Bühne | Element von Director, Fläche, auf der alle Sprites platziert werden, in der endgültigen Online- oder Offline-Präsentation der sichtbare Bereich |
| **D**arsteller | Element der Anwendung Macromedia Director, Mediadaten (Bilder, grafiken, Audio, Video, Text, 3D-Szenen) die in der Besetzung verwaltet werden |
| Debuggen | Fehlersuche im Programmcode, vom engl. Bug = Wanze |
| Director | Siehe Macromedia Director |
| Director-Film | komplette Anwendung (Datei), die in Macromedia Director erstellt wurde |
| DirectX | Microsofts interaktive Medien-Technologie für die Windows-Plattformen, enthält u. a. auch die Direct3D-API, die von Spielen und Web3D-Technologien genutzt wird |
| Download | Bezeichnung für das Herunterladen von Daten aus einem Kommunikationssystem wie dem Internet, dabei werden Programme oder Dateien auf den eigenen Computer übertragen |
| Drehbuch | Element von Macromedia Director, Instanzen der Darsteller werden dort in einer Zeitleiste angeordnet |
| DTD | Document-Type-Definition, deklariert die Struktur und Tags eines Dokumenttyps, |

| | |
|---|---|
| | DTD legt die Syntax und Semantik einer Auszeichnungssprache, z. B. von XML, fest |
| **E**chtzeit | Unmittelbares Reagieren auf Nutzerinteraktionen und Eingaben, Verzögerungszeit zwischen dem externen Ereignis und der Reaktion des Programms/ Systems ist dabei möglichst gering |
| Ereignisskript auch Event-Handler, Event-Handler | Element von Director, Behandlung bestimmter Ereignisse (z. B. Benutzereingaben) mit Hilfe der directoreigenen Programmiersprache Lingo siehe Ereignisskript |
| **F**all-off | Bereich, der die Lichtabnahme eines Spotlichtkegels definiert |
| Farbtiefe | Informationsmenge, mit der die Farbe eines Bildpunktes beschrieben wird |
| Filmskript | Element von Director, Skript, dessen Lingo-Programmcode Prozeduren global für alle anderen Skripte bereitstellt und das Verhalten des Filmes steuert |
| **G**rafikkarte | spezielle Erweiterungskarte für die Ansteuerung des Monitord und die Darstellung von Grafikfunktionen |
| **H**ardware | Alle „harten“ Bestandteile des Computers und seiner Peripherie, d. h. alle Geräte und Geräteteile vom Prozessor über Speicher und Datenträger usw. |
| Hypertext | Neben normalem Text existieren in einem Hypertextdokument Querverweise (Hyperlinks) zu andern Dokumenten oder Textstellen |
| **I**maging Lingo | Teil der Director-Lingo-Programmierung, Befehlssammlung zur Bildmanipulation auf Pixelebene |
| Interaktive 3D-Grafik | Bildsequenzen werden erst zum Zeitpunkt des Abspielens in Abhängigkeit von Nutzerinteraktionen in Echtzeit erzeugt |
| Interaktivität | Beschreibt dialogorientierten Informationsaustausch zwischen Anwender und Computer |
| Internet | Weltweit größtes Computernetzwerk, das aus miteinander verbundenen Netzwerken besteht |
| **L**ingo | Programmiersprache von Macromdia Director |
| **M**acromedia | Entwicklungsfirma verschiedener Grafik-, Web- und Multimedia-Entwicklungswerkzeuge (Director, Flash, Freehand, Dreamweaver) |
| Macromedia Director | Authoring-Werkzeug der Firma Macromedia |
| Maske | Es wird eine Farbe, ein Farb- oder ein Bildbereich vor Veränderungen geschützt oder speziell für die Bearbeitung ausgewählt |
| Map | Andere Bezeichnung für Textur |
| Mapping | Andere Bezeichnung für Texture Mapping |
| Modell | Geometrisches Objekt im Shockwave3D-Raum |
| Modellressource | Beschreibt die geometrischen Eigenschaften sowie die Obenflächeneigenschaften eines Modells |
| Modifikator oder Modifier | 3D-Studio Max: Modifikatoren bieten verschiedene Funktionen zur ObjektmanipulationDirector3D: Funktionen zur Erweiterung oder Manipulation von Objekteigenschaften (Animationen, Geometrieveränderungen, u. a.) |

| | |
|---|---|
| Mouseover | bezeichnet Vorgang, bei dem die Maus über sensitive Bereiche (Buttons, Links o. ä.) bewegt wird und dabei dieses Ereignis auslöst |
| Multimedia | Bezeichnung für die Aufzeichnung, Wiedergabe und Integration verschiedener Medien wie Video, Animation oder Audio |
| **O**pazität | Maß für die Undurchsichtigkeit einer Fläche mit Werten von 0 (völlig transparent) und 1 (total undurchsichtig).; wird auch als Alpha bezeichnet |
| OpenGL | Abkürzung für Open Graphics Library 3D-Software-Schnittstelle (3D-API), die ab Windows NT fester Bestandteil von Windows ist. OpenGL basiert auf Iris GL von Silicon Graphics (SGI). |
| **P**arameter | Werte, die an Prozeduren übergeben werden |
| Parser | Parser sind Software-Module, dir Dokumente oder Quelltexte nach vorgegebenen Kriterien (grammatikalischer Struktur) syntaktisch analysieren und für die Weiterverarbeitung aufbereiten |
| PlayList | Liste, die eine Abfolge der Animationen eines Director3D-Objektes beinhaltet |
| PlugIn | Hilfsprogramm, das die Funktionalität eines Browser oder anderer Programme erweitert |
| Polygon | Vieleck, definiert geometrische Struktur von Objekten, unterschiedliche qualitative Auslegung; 3D Studio Max: 1Polygon besteht aus zwei Dreiecken<br>Director3D: 1Polygon besteht aus einem Dreieck |
| Preloader | Kleiner Flash- oder Directorfilm, der den Benutzer motiviert, den eigentlichen, (größeren) Film weiterhin (im Hintergrund) laden zu lassen |
| Property | Eigenschaftsvariable in Director-Verhaltensskripten, der Gültigkeitsbereich ist auf die Skriptinstanz beschränkt |
| Prototyp | noch nicht fertig gestellte Software, Zwischenstand bei der Entwicklung |
| **R**endering | Zusammenfassung aller Rechenoperationen, die nötig sind, eine vorhandene Szenenbeschreibung in ein 2D-Pixelbild umzusetzen |
| **Q**uickTime | Systemerweiterung für das Windows-, Windows NT-, Macintosh- und Silicon Graphics-Betriebssystem zum Aufnehmen, Bearbeiten und Wiedergeben von Videos auf dem PC |
| QuickTime VR | 3D-Technologie, erzeugt einen räumlichen Eindruck durch geschickte Anordnung und Verzerrung von Einzelbildfolgen oder unter Nutzung von Panoramabildern. Sehr gute Darstellungsqualität, große Dateigröße, Interaktionen möglich |
| **S**hading | Flächen von Objekten werden Farbwerte zur Simulation von Oberflächeneigenschaften zugewiesen |
| Shockwave | Dateiformat von Director und Flash, mit dem multimediale Inhalte im Web präsentiert werden können, benötigt jeweils das entsprechende Plug-in auf der Zielplattform |
| Sprite | Instanz eines Darstellers auf der Bühne |
| Spritekanal | Sprites werden im Drehbuch in Spritekanälen organisiert, diese definieren die Tiefenanordnung der Sprites |

| | |
|---|---|
| Startup-Latenz | Zeitraum, der vergeht bis erste Interaktionen möglich sind, nachdem die Web3D-Szene angefordert wurde |
| Streaming | Dateien, die noch nicht vollständig geladen wurden, können bereits, soweit sie geladen sind, genutzt werden (Audio-, Video, 3D-Streaming) |
| Szene | (auch 3D-Szene, 3D-Welt) Bezeichnung für einen dreidimensionalen Raum, in dem 3D-Objekte existieren |
| Szenegraph | Hierarchische, baumähnliche Struktur zur Verwaltung von Objekten einer Szene |
| **T**extur | 2D-Grafik, die den Oberflächen von 3D-Objekten zugewiesen wird |
| Texture Mapping | Vorgang des Zuweisens einer 2D-Grafik zu einer Fläche unter Beachtung der korrekten perspektivischen (und ggf. geometrischen) Verzerrung |
| Tool | Engl. für Dienst- oder Hilfsprogramm, Anwendung, Werkzeug |
| Transparenz | Durchsichtigkeit, Teile eine Bitmap, einer Textur können Informationen besitzen, die bestimmen, welcher Teil wie stark durchsichtig dargestellt werden soll |
| Transformation | Zusammenfassung von Verschiebung, Rotation und Skalierung, bestimmt räumliche Position und Ausrichtung eines Objektes |
| **U**nicode | System zur Darstellung von Zeichen und Elementen aller bekannten Schriftkulturen und Zeichensystem, verwendet 4-Byte-Codierung |
| Usability | Engl. Brauchbarkeit, Nutzbarkeit, im Kontext der Informatik: Nutzerfreundlichkeit von Anwendungen, interaktiven Darstellungen, Hardwareprodukten |
| UTF | Unicode Text Format; Codierungsalgorithmus für Unicode-Zeichen |
| **V**ektor | Vektoren werden durch einen Anfangs und einen Endpunkt festgelegt, beschreiben eine Richtung und besitzen eine definierten Betrag, wird durch seine Länge repräsentiert |
| Vertex | Knoten-, Scheitelpunkt, dieses geometrische Element ist Anfangs-/ Endpunkt einer Kante (engl. edge) eines Gittermodells |
| Vertices | Plural von Vertex |
| Virtuell | Franz. künstlich, simuliert |
| Virtual Reality | Begriff, der eine durch Computer geschaffene künstliche Realität bezeichnet |
| **W**eb3D | Bezeichnung für alle offenen Standards und proprietären Lösungen, die interaktive 3D-Grafiken im WWW darstellen können |
| WorldWideWeb (WWW) | weltweites Netz, bekanntester Teil/ Dienst des Internets, Zugriff/ Darstellung von Informationen aus dem Internet erfolgt über Browser |
| Xtras | Erweitern Director um zusätzliche Funktionalität |
| **Z**oomen | Vergrößern oder Verkleinern von Bildausschnitten |

# Selbständigkeitserklärung

Ich versichere hiermit, dass ich die Diplomarbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel genutzt habe.

Ralf Halgasch